# Sparsity in Machine Learning: Theory and Practice

*Zakria Hussain*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

**Doctor of Philosophy**

of the

**University of London.**

The Centre for Computational Statistics and Machine Learning

Department of Computer Science

University College London

2008

# Abstract

The thesis explores sparse machine learning algorithms for supervised (classification and regression) and unsupervised (subspace methods) learning. For classification, we review the set covering machine (SCM) and propose new algorithms that directly minimise the SCMs sample compression generalisation error bounds during the training phase. Two of the resulting algorithms are proved to produce optimal or near-optimal solutions with respect to the loss bounds they minimise. One of the SCM loss bounds is shown to be incorrect and a corrected derivation of the sample compression bound is given along with a framework for allowing asymmetrical loss in sample compression risk bounds. In regression, we analyse the kernel matching pursuit (KMP) algorithm and derive a loss bound that takes into account the dual sparse basis vectors. We make connections to a sparse kernel principal components analysis (sparse KPCA) algorithm and bound its future loss using a sample compression argument. This investigation suggests a similar argument for kernel canonical correlation analysis (KCCA) and so the application of a similar sparsity algorithm gives rise to the sparse KCCA algorithm. We also propose a loss bound for sparse KCCA using the novel technique developed for KMP. All of the algorithms and bounds proposed in the thesis are elucidated with experiments.

# Acknowledgements

A PhD can at times be a lone battle and at others an academic partnership between student and supervisor. I was fortunate enough to have met my supervisor, John Shawe-Taylor, during my undergraduate studies. Before becoming John's student I was unaware of how well known and respected he is in his chosen discipline. I feel privileged to have had his supervision during my PhD and would like to take this opportunity to thank him for all the wisdom, insights and knowledge of machine learning he has passed on to me and will never forget his most important lesson to me that a hack can never beat a good theory. Long may our research collaboration continue.

During the PhD I have had the pleasure of being at two institutions. Having started at the University of Southampton I would like to thank everybody at the ISIS lab who helped during my PhD. Thanks to Anthony Demco, Steve Gunn, David R. Hardoon, Andriy Kharechko, Alain Lehmann, Emilio Parrado-Hernandez and Sandor Szedmak. Special mentions for David who became a flatmate and a good friend, Sandor who was always happy to help and Emilio for hosting me at Carlos III in Madrid and always being at hand to help. A big thank you to Ollie Blacklock who introduced me to the Accies cricket team, and a fresh outlook to Southampton. I would also like to take this opportunity to thank Mario Marchand and Francois Laviolette for their help and support with the first part of Chapter 3.

At UCL I would like to thank the 8th floor lab for being so quiet and considerate towards their fellow colleague. A special mention to Cédric Archambeau, Tom Diethe and Jan Rupnik who I have had many fruitful discussions with and who have become good friends (and a bad influence on my coffee drinking habit).

I would like to thank friends for being extremely supportive and always there to ask whether I had "finished yet?". This did not always help, but how were you to know? Thank you, to all the friends I made outside the lab in Glen Eyre and Monti halls in Southampton, with whom I shared some great times with.

Finally I would like to thank my family for always being there and dedicate this thesis to my mother and father.

# Nomenclature

| | |
|---|---|
| $\mathcal{X}$ | input space |
| $(\mathbf{x}, y)$ | input-output pair |
| $\mathcal{E}(\cdot)$ | true error |
| $\hat{\mathcal{E}}(\cdot)$ | empirical error |
| $\mathcal{P}$ | set of $\mathcal{P}$ examples |
| $\mathcal{N}$ | set of $\mathcal{N}$ examples |
| $S$ | sample |
| $\mathcal{H}$ | hypothesis space |
| $\in$ | exists in (within) |
| $f \in \mathcal{H}$ | hypothesis (function) in hypothesis space |
| $\mathcal{A}$ | learning algorithm |
| $\|\| \cdot \|\|$ | $L_2$ norm (length of a vector) |
| $\|\| \cdot \|\|^2_{\text{Frob}}$ | Frobenius norm |
| $\mathbf{K}$ | kernel matrix |
| $\mathbf{X}$ | data (input) matrix |
| $\kappa(\cdot, \cdot)$ | kernel function |
| $d(\cdot, \cdot)$ | distance function |
| $m$ | number of examples |
| $n$ | number of dimensions |
| $\mathbb{I}(\cdot)$ | indicator function |
| $\mathbb{R}$ | the set of real numbers |
| $diag\{\cdot\}$ | diagonal elements of a matrix |
| $\mathbf{e}_i$ | the $i$th unit vector |
| $\mathbf{1}$ | all one vector |
| $\mathcal{D}$ | probability distribution generating the data |
| Pr | probability |
| iid | independently and identically distributed |
| $\mathbb{E}$ | expectation |
| $d$ | sparsity parameter |
| $\mathbf{i}$ | index vector |
| $\mathbf{w}$ | primal weight vector |
| $\boldsymbol{\alpha}$ | dual weight vector |

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

*Machine learning is an attempt at creating computer systems capable of improving their performance automatically over time, allowing us to tackle problems that are unlikely to be solved using more classical computer science approaches. Obvious examples are image recognition, fraud detection, spam filtering and automatic car driving systems. Clearly, the realisation of such adaptive systems technology would certainly enhance many areas of human life.*

*It is important to note that computers have several distinct advantages over humans – most notably the ability to carry out very fast numerical calculations and database storage. However, humans excel in areas such as vision, speech, creation, etc. and so machine learning may be viewed as trying to bridge the gap between computational power and human learning.*

*In this chapter we identify the problem statement of the entire thesis by walking the reader through a typical machine learning problem, describing the main attributes and terminologies involved and then proceeding to a possible solution. After this we describe the main focus of the thesis, sparsity and describe it within the context of the thesis. We are not mathematically precise in this chapter and leave such rigour for the future. We conclude the chapter by discussing the contributions of the thesis and outlining the remaining chapters.*

## 1.1 Motivation

Machine learning is an attempt at making computers intelligent, allowing us to solve problems that would seem otherwise infeasible. For instance, consider learning to classify spam from genuine e-mails. Understanding the nature of the e-mails would allow us to classify them as spam or non-spam. However, how can we program a computer to automatically carry out this task to distinguish between the two?

A naive approach is to set flags that alert the email client of potential spam attacks. For example, ensuring e-mails contain full names of the recipient, or that the sender exists in the recipients e-mail address book. However the problem with this approach is that a genuine sender may use formal wording and only refer to the recipient using their surname. Also, the second flag would judge genuine senders as spam if there address was not already listed in the recipient's address book. Clearly setting flags as outlined above is not a practical solution to the problem and has serious drawbacks. We would like a more sophisticated solution in order to recognise e-mails that are spam, including those e-mails not previously encountered. This is exactly the type of problem machine learning tackles.

A machine learning approach to this problem would be the following. We may associate every e-mail with a label of whether or not it is spam. Potentially every e-mail user would be exposed to large amounts of data in this context, containing a large collection of e-mails together with their classifications (these classifications would need to be labelled manually but we will assume this has already been done). Given these e-mails and their corresponding labels (spam/non-spam) can we construct a "general rule" that decides whether or not a new e-mail is spam?

We may tackle this problem in the following way: create a distance measure between e-mails so that they lie at some distance from one another. Hopefully non-spam e-mails would lie very far away from those defined as spam. Next, by generating the smallest ball (for instance) that encloses all of the e-mails that are non-spam we can have a general rule that states that every e-mail within the ball is genuine and everything outside is spam. Therefore, computing the distance measure for every new e-mail received and using the general rule, we can predict whether or not an e-mail is spam. Notice that we have not explicitly defined attributes or features to look for in e-mails (as in the flag example given above) but simply addressed the similarity of e-mails using some arbitrary distance measure. This very simple (and not precisely defined) algorithm illustrates some important machine learning concepts.

In machine learning we use the term *examples* to denote the e-mails and *outputs* to denote the labelling of spam or non-spam. The problem of spam detection is known as *classification*; where we would like to predict which class an example belongs to. When the number of classes is two we refer to it as *binary classification*. When the number of classes is greater than two it is known as *multi classification*. Also, when the outputs are not restricted to discrete classes but numbers on the real line we call this *regression*.

All of these types of problems are described under the framework of *supervised learning* – where for every example we are given an output. In analogy to a teacher giving a student a question together with its answer during the learning phase, but only a question during the testing phase, the student must learn from the questions and solutions presented in order to generalise in the future. The construction of the general rule from the examples and labels is a phase known as *training/learning*. Unsurprisingly the prediction stage of our general rule is known as *testing*. The general rule is known as the *hypothesis* and is simply a *function* that (in the supervised learning case) maps examples to outputs. For classification (regression) we sometimes also refer to this function as the *classifier* (*regressor*), respectively.

Another framework for learning is *unsupervised learning* – where we are only given examples, perhaps because their exist no outputs or because the outputs are too costly to be supplied. In the unsupervised learning setting the learning algorithm attempts to find some pattern that will distinguish examples from one another. Hopefully, in the context of e-mails we may find a pattern that creates two distinct regions, one for spam and another for non-spam. However, because we have no outputs, this may be overly ambitious. Instead, more realistically, we may expect to generate several regions that perhaps partition the e-mails into categories of sender, date, length, etc.

A framework that amalgamates the two frameworks mentioned above is called *semi-supervised learning* – where we are given examples and a limited number of outputs, because examples are cheaper to acquire than labels. In the e-mail example, the number of e-mails that we may have access to could be huge. However, labelling all of them may be too costly. This is where supervised and unsupervised learning can be combined effectively to create powerful learning rules that makes use of large amounts of (cheap) unlabelled data.

Other learning frameworks exist but the thesis will focus on supervised and unsupervised learning and so we will limit ourselves to these methodologies throughout the thesis.

The algorithm we described above for classifying spam used all of the e-mails in order to construct the ball or hypothesis. Now let us consider the situation where every example-label (input-output) pair comes with an associated cost. We would like to classify correctly as many (all) of the e-mail messages that we may receive in the future but minimise the number of input-output pairs used in the construction of the hypothesis. We try to minimise this problem throughout the thesis, attempting to create parsimony in the learnt functions in the sense that they rely only on a small subset of examples. We refer to this property as *sparsity* throughout the remaining work. Sparsity forms the backdrop of the entire thesis and we show in the frameworks of supervised and unsupervised learning that we can indeed have such sparse learning algorithms along with theoretical guarantees of future generalisation ability.

Before commencing with a background chapter we give a brief historical overview of machine learning and then proceed with an outline and contribution of the thesis.

## 1.2 Brief history of machine learning

We give a brief overview of machine learning history with a bias on the topics from the thesis.

One of the first machine learning algorithms was the *perceptron* developed by Rosenblatt [1958]. It was biologically inspired and was proposed as the analogy to a neuron found in the brain. The algorithm looked for linear relationships in the data and was later shown, in a famous book called 'Perceprons' [Minsky and Papert, 1969], to be incapable of learning the XOR problem. This caused a decade of decline in machine learning research. However, in the 1980's several authors showed that multi-layer perceptrons could indeed solve more difficult problems (including the XOR).

Throughout this time several Russian authors were also developing their own theories for learning. Most notably, Vapnik and Chervonenkis [Vapnik and Chervonenkis, 1971] introduced the concept of the VC-dimension, which quantified the power of a classifier. They proposed upper bounds on linear classifiers using the VC-dimension and helped develop the field of *statistical learning theory.*

The theory of kernels were first proposed in the 1940s and one of the main exponents called Aronszajn [1950] published a paper that described their attributes. However, it wasn't until the work of Aizerman et al. [1964] that kernels were introduced into the machine learning literature. They were sporadically used with neural networks but were yet to take off as a new force in machine learning. The first step towards the kernel methods framework was taken in the Computational Learning Theory conference of 1992 when Boser et al. [1992] introduced their *support vector machine (SVM)* algorithm. The algorithm looked for linear relationships in the kernel defined feature space and hence showed that the use of kernels could help keep the simplistic and well understood linear functions in a setting that allowed them to be applied to more complex and non-linear data. Several years later Cortes and Vapnik [1995] introduced penalty terms into the SVM that allowed misclassifications to be accounted for (see also Cristianini and Shawe-Taylor [2000] for more discussion on the SVM).

During this period several researchers were working on the problem of devising a *computational learning theory.* It would be fair to say that this work started with the seminal paper of Valiant [1984] who looked to formalise a mathematical model for learning, much like the earlier work of Vapnik and Chervonenkis [1971] but with the difference that there was a more computer science standpoint, *i.e.*, Valiant also required a polynomial time learning algorithm in order to ensure 'learnability' in his model that was later coined the *probably approximately correct (PAC)* learning model. To demonstrate the effectiveness of this model Valiant [1984] proposed a learning algorithm for learning conjunctions or disjunctions of monomials. Several years later Haussler [1988] showed that Valiant's *standard monomial learning* algorithm (see Anthony and Biggs [1992] for details) could be viewed as the problem of the minimum set cover, and solved (approximately) with the greedy set cover algorithm [Chvátal, 1979].

In the late 1980's there was a real drive towards constructing PAC learning bounds for a larger class of algorithms. The work of Blumer et al. [1989] gave the first PAC guarantees using the VC-dimension. Also, in an unpublished manuscript Littlestone and Warmuth [1986] proved that PAC learnability was possible if you could find a small number of training samples that could be used (reconstructed) to relabel the training data. These compression bounds became known as *sample compression bounds.*

Kernel methods really came to prominence after the papers by Schölkopf et al. [1996, 1998] where they showed that a well-known (linear) statistical algorithm called principal components analysis could be computed in kernel defined feature space and hence tackle non-linear data. It is fair to say that most of the work that came after the kernel principal components analysis publication, in the kernel methods domain, tried to take linear algorithms and make then non-linear in this way. It may be attributed to starting the area of kernel methods (although SVMs came earlier, Schölkopf et al. [1996] demonstrated that the kernel trick was not restricted to just SVMs). For good introductions to kernel methods see the books by Schölkopf and Smola [2002] and Shawe-Taylor and Cristianini [2004].

Valiant and Haussler's work had not been applied to real world data sets because of the definition of monomials, which were restricted to Boolean-valued data sets, and their algorithms made no provisions for misclassifications. Marchand and Shawe-Taylor [2001] fixed these problems and proposed the *set covering machine (SCM)*, a general purpose learning algorithm capable of learning on real world data sets with sample compression bounds guaranteeing its future success. This is where we begin our journey for the classification algorithm.

After the Schölkopf et al. [1998] paper and several kernel algorithms later, machine learners were now concerned with working with larger data sets. But the fact the number of data points determined the dimensions of the kernel meant that larger data sets could not be stored in computer memory. Therefore, there was a drive towards constructing low rank matrix approximations and sparse variants of kernel algorithms. Smola and Schölkopf [2000] created a *sparse kernel principal components analysis (SKPCA)* algorithm which gave a low rank approximation of the kernel matrix. Their algorithm used principles from a well known algorithm in the signal processing community called matching pursuit [Mallat and Zhang, 1993]. An algorithm that pursues parsimonious solutions to find the line of best fit in a least squares sense. The algorithm was suboptimal in the sense that only the last element of the weight vector was updated, however Pati et al. [1993] and Davis et al. [1994] proposed an extension called *orthogonal matching pursuit* that fixed this issue. The algorithms were greedy in nature and could be constructed by only using dot products. This led Vincent and Bengio [2002] to propose *kernel matching pursuit (KMP)*, which was a sparse version of kernel least squares regression. These two algorithms are very much related and use the same principles for constructing sparse solutions. We will use these two algorithms as the basis of the regression and subspace methods work.

## 1.3 Outline of thesis

The thesis is structured as follows. Chapter 2 gives a brief overview of the main machine learning principles to be discussed. It will describe the main building blocks needed, including the set covering machine, the kernel least squares regression, kernel principal components analysis and basic learning theory including Vapnik-Chervonenkis (VC) theory, probably approximately correct (PAC) learning theory and sample compression theory. The thesis is structured so that the chapters are all self contained. The novel contributions of the thesis are given in Chapters 3 and 4, with Chapter 3 beginning with the problem of classification. It discusses a new generalisation error bound for the set covering machine that bounds separately the error on the positive and negative class of examples and also proposes several new algorithms that use the bounds to drive the SCMs optimisation criteria. We provide theoretical results to prove that the algorithms presented produce optimal (or near optimal) solutions. The final part of the chapter investigates the regression problem in a sparse setting, by describing the kernel matching pursuit algorithm. We propose a generalisation error bound for kernel matching pursuit that takes into account the sparse solutions delivered. The bound uses a novel technique of upper bounding KMP's loss by combining sample compression schemes together with VC theory.

Chapter 4 is dedicated to subspace methods (unsupervised learning) and starts with a description of the sparse kernel principal components analysis. We prove that sparse KPCA can be viewed as a compression scheme and hence propose the first sample compression bound for a subspace method. From this analysis and the investigation into sparse KPCA, we are able to propose a sparse version of kernel canonical correlation analysis that can help tackle problems not previously tractable with the KCCA algorithm. We upper bound its future loss using the same method derived for KMP.

All the novel work proposed in the thesis is illuminated with empirical results.

The main contributions can be stated as follows:

- Chapter 3
  - A new generalisation error bound for the SCM
  - A sample compression bound for asymmetric loss
  - The bound set covering machine (BSCM)
  - The branch and bound set covering machine (BBSCM)
  - The $\tau$-branch and bound set covering machine (BBSCM($\tau$))
  - Theoretical justification of the BBSCM and BBSCM($\tau$)
  - Sparse generalisation error bound for KMP
- Chapter 4
  - Sample compression bound for sparse KPCA

- Sparse kernel canonical correlation analysis (fast)
- Sparse kernel canonical correlation analysis (faster)
- Sparse generalisation error bound for sparse KCCA

## 1.4 Contributions

The work contained in this thesis was published or submitted for publication in the following papers.

- Zakria Hussain and John Shawe-Taylor. Using generalization error bounds to train the set covering machine. In *Proceedings of International Conference of Neural Infromation Processing*, 2007.
- Zakria Hussain, Francois Laviolette, Mario Marchand, John Shawe-Taylor, Spencer Charles-Brubaker and Matthew Mullin. Revised Loss Bounds for the Set Covering Machine and Sample-Compression Loss Bounds for Class Imbalance. *Journal of Machine Learning Research*, 8:2533–2549, 2007.
- Zakria Hussain, John Shawe-Taylor, Charanpal Dhanjal and David R. Hardoon. Theoretical analysis of matching pursuit in machine learning and sparse kernel canonical correlation analysis. *To be submitted to the Journal of Machine Learning Research.*

# Chapter 2

# Background

*The previous chapter gave a simple example of a typical supervised learning problem without being mathematically precise. In this chapter we make all the preliminary definitions needed and survey the most common methods in machine learning and learning theory in the context of the thesis. There are several strands of the learning methodology that we wish to introduce; Classification, regression and subspace methods.*

*Classification is typically encountered in the two class case, where examples are drawn from two different classes, most frequently referred to as the positive and negative classes. By training on these examples and finding a general pattern we would like to predict the class of a new example. The classification algorithm we describe is the set covering machine (SCM). Regression is similar to classification but with the difference that the predicted functions are no longer restricted to two classes but numbers on the real line. For this line of work we describe a non-linear regression algorithm called kernel least squares regression. Finally, subspace methods look to approximate complete subspaces from which the original data may come from, by finding the most important directions in the data. This is commonly used as a pre-learning tool to map the data into a lower dimensional space before training a learning algorithm in this lower dimensional space. The non-linear subspace method we discuss in this chapter is the kernel principal components analysis algorithm.*

*Learning theory gives worse case guarantees on learning protocols, upper bounding the future loss that may be incurred, resulting in the ability to judge the quality of learning that an algorithm has achieved. We review three learning theory methodologies, namely, probably approximately correct (PAC) learning theory, Vapnik-Chervonenkis (VC) theory and sample compression theory.*

## 2.1 Machine learning

Machine learning is the study of making computers "learn". Extracting information from the data set, inferring rules and making predictions in the future. These data sets are generally composed of objects in the form of vectors known as *training examples.* Other structures such as graphs, strings, trees, etc. are also considered but we restrict ourselves to vectors here. The following definition makes the distinction between the supervised and unsupervised learning *data sets* (containing multiple numbers of training examples) that we will use throughout the thesis.

**Definition 2.1** (Supervised learning data set). *Let* $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ *be a member of an input space* $\mathcal{X}$ *and let* $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ *be a sample (data set) of* $m$ *input-output pairs* $(\mathbf{x}, y)$. *The* classification *sample considers the output values* $\{-1, +1\}$ *and the* regression *sample considers outputs in* $\mathbb{R}$.

In the unsupervised learning case we have the following definition for a data set.

**Definition 2.2** (Unsupervised learning data set). *Let* $\mathcal{X}$ *be the input space and* $\mathbf{x}$ *be a member of* $\mathcal{X}$. *The unsupervised learning data set is given by a sample* $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ *of inputs only. In the case when we have a second input space* $\mathcal{Y}$ *we have a paired sample* $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)\}$ *of* $\text{input}_{\mathcal{X}}$*-*$\text{input}_{\mathcal{Y}}$ *pairs.*

Data can often be non-linear and in this case a combination of linear functions can be combined to tackle non-linearity. This technique is used for instance in multi-layer perceptrons. Another method for tackling non-linear data is to map the input data into a higher dimensional space and find a single linear function in this space.

**Definition 2.3.** *Given a vector* $\mathbf{x} = (x_1, \ldots, x_n)$ *in* $n$*-dimensional space we can map it into a higher* $N$*-dimensional feature space* $\mathcal{F}$ *using the mapping* $\phi$

$$\phi : \mathbf{x} \mapsto \phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_N(\mathbf{x})) \in \mathcal{F}$$

*where* $N > n$.

In the algorithms we discuss, we will use the mapped input data to carry out learning. However mapping into higher dimensions (the *feature space*) and computing inner products to find linear functions (in feature space) is more expensive than simply using the input data, but this computation can be made considerably more efficient by making use of kernels[Aronszajn, 1950, Aizerman et al., 1964].

## 2.2 Kernel methods

The theory of kernels dates back to the work of Aronszajn [1950] and Aizerman et al. [1964]. However, the practical significance of kernels in machine learning was not realised until almost 30 years later in the development of the support vector machine (SVM) [Boser et al., 1992, Vapnik, 1998, Cristianini and Shawe-Taylor, 2000]. The SVM works by first mapping the input data into a higher dimensional feature space and then looking for a linear function (hyperplane) that creates a large separation between the two classes of examples. This large separation is known as the *margin* and created for the points closest to the hyperplane known as *support vectors*. The kernel trick allows one to work in feature space without having to explicitly carry out this mapping, but simply works with the input data using kernel functions.

**Definition 2.4** (Aizerman et al. [1964]). *A kernel is a function $\kappa$ that for all $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ satisfies*

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

*where $\phi$ is a mapping from $\mathcal{X}$ to an (inner product) feature space $\mathcal{F}$*

$$\phi : \mathbf{x} \mapsto \phi(\mathbf{x}) \in \mathcal{F}.$$

The kernel function allows us to take full advantage of working in higher dimensional feature space but without the computational burden of computing the mapping. We make use of kernel algorithms for the regression and subspace methods algorithms described later, as they look to find a linear function. However, before moving onto these kernel algorithms we first describe the set covering machine (SCM) which does not necessarily rely on a kernel function[1] as it is an ensemble method and can simply be computed using some $L_p$ distance metric.

## 2.3 Algorithms

In this section we discuss all the algorithms that we use as the basis of the thesis. In future chapters we either propose a theoretical analysis of the algorithms using sparsity arguments (regression and subspace methods), new theoretical analysis (all chapters) and new learning algorithms (classification and subspace methods). Therefore the algorithms of this section are the only prerequisites to reading and understanding the remainder of the work presented in the thesis. We start the discussion in the order of the chapters of the thesis, and therefore begin with a description of the classification algorithm known as the set covering machine (SCM).

Before we begin with classification we would like to make some general definitions. We will be given a sample $S$ that contains vectors of our data inputs and our (usually scalar) outputs for the case of classification and regression. The input space will be denoted by $\mathcal{X}$ and the output space by $\mathcal{Y}$. In the unsupervised learning setting (namely for canonical correlation analysis) we will use $\mathcal{Y}$ to denote an input (see Definition 2.2). So the following

input-output

pairing is only for the supervised learning case. For the unsupervised learning case we will have

input

[1]It is possible to use a kernel, however, we choose not to make this unecessary computation.

or

$$\text{input}_{\mathcal{X}}\text{-input}_{\mathcal{Y}}$$

pairings. All these definitions will be made clearer in their respective sections.

### 2.3.1 Classification: the set covering machine (SCM)

The classification problem can be described as follows. Find a function (hypothesis) $f : \mathbf{x} \mapsto y$ that maps examples $\mathbf{x}$ to labels $y \in \{-1, 1\}$ given a classification sample $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$.

Consider a set $\mathcal{B} = \{h_i(\mathbf{x})\}_{i=1}^{|\mathcal{B}|}$ of Boolean-valued functions $h_i(\mathbf{x})$ that each map examples $\mathbf{x}$ belonging to $\mathcal{X}$ onto $\{0, 1\}$ and assume that we have found a small subset $\mathcal{R} \subset \mathcal{B}$ of these functions. Given $\mathcal{R}$, and an arbitrary example $\mathbf{x}$, then the function $f(\mathbf{x})$ of the set covering machine (SCM) is defined as

$$f(\mathbf{x}) = \begin{cases} \bigwedge_{h \in \mathcal{R}} h(\mathbf{x}) & \text{(conjunction)} \\ \bigvee_{h \in \mathcal{R}} h(\mathbf{x}) & \text{(disjunction).} \end{cases}$$

The conjunction function $f(\mathbf{x})$ outputs True (1) if all $h(\mathbf{x}) \in \mathcal{R}$ are True and False otherwise. The disjunction function $f(\mathbf{x})$ outputs False (0) if all $h(\mathbf{x}) \in \mathcal{R}$ are False and True otherwise. In the context of this thesis we will use the value of $-1$ to denote the output of 0 (False) for the conjunction or disjunction of the function $f(\mathbf{x})$. Furthermore we define the number of Boolean-valued functions contained in $\mathcal{R}$ or $f$ by $|\mathcal{R}|$ or $|f|$, respectively. With these notations we can make the following definition.

**Definition 2.5.** *Let $\mathcal{P}$ be the set of positive $(+1)$ and $\mathcal{N}$ the set of negative $(-1)$ training examples when the SCM is constructing a* conjunction *of Boolean-valued functions. Similarly, let $\mathcal{P}$ be the set of negative $(-1)$ and $\mathcal{N}$ the set of positive $(+1)$ training examples when the SCM is constructing a* disjunction *of Boolean-valued functions.*

We describe the SCM for the case when the set of Boolean-valued functions is a set of functions constructed from the data, known as *data-derived (decision) functions.* The set of data derived functions we use throughout the thesis is the set $\mathcal{B}$ consisting of the following *data-dependent balls.*

**Definition 2.6.** *For a training example $\mathbf{x}_i \in \mathcal{X}$ with label $y_i \in \{-1, 1\}$ and (real-valued) radius $\rho = d(\mathbf{x}_i, \mathbf{x}_j) \pm \alpha$ where $d(\mathbf{x}_i, \mathbf{x}_j)$ denotes the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, $\mathbf{x}_j \in \mathcal{P}$ is a* border *point and $\alpha$ is a small positive fixed real number, let $h_{i,\rho}$ be the following* data-dependent ball *centred at $\mathbf{x}_i$:*

$$h_{i,\rho}(\mathbf{x}) = \begin{cases} y_i & \textit{if } d(\mathbf{x}_i, \mathbf{x}) \leq \rho \\ \bar{y}_i & \textit{otherwise} \end{cases}$$

*where $\bar{y}_i$ is the complement of $y_i$, $\rho = d(\mathbf{x}_i, \mathbf{x}_j) + \alpha$ if $\mathbf{x}_i \in \mathcal{P}$ and $\rho = d(\mathbf{x}_i, \mathbf{x}_j) - \alpha$ if $\mathbf{x}_i \in \mathcal{N}$.*

We will often switch, without loss of generality, between the subscript notation $h_{i,\rho}$ to the more simple $h$ to denote a data-dependent ball when $i$ and $\rho$ are clear from the context. Consider a sample $S$ consisting of pairs $(\mathbf{x}, y)$, a ball $h$ from the set of data-dependent balls $\mathcal{B}$, then $\nu_{\mathcal{N}}(h)$ denotes the set of pairs $(\mathbf{x}, y) \in \mathcal{N}$ correctly classified by $h$ and $\pi_{\mathcal{P}}(h)$ denotes the set of pairs $(\mathbf{x}, y) \in \mathcal{P}$ misclassified by $h$. Given these definitions the *usefulness* of a data-dependent ball can be defined as follows.

**Definition 2.7.** *The* usefulness *(or* utility*)* $U_{\mathcal{N},\mathcal{P}}$ *of a data-dependent ball* $h$ *is expressed as:*

$$U_{\mathcal{N},\mathcal{P}}(h) = |\nu_{\mathcal{N}}(h)| - p|\pi_{\mathcal{P}}(h)| \tag{2.1}$$

*where* $p$ *is a small positive real number.*

When we discuss a $\mathcal{P}$ example we will mean an example from the set of $\mathcal{P}$. Similarly, an $\mathcal{N}$ example will refer to an example from the set of $\mathcal{N}$ examples.

The SCM algorithm uses a *greedy* approach to try and completely classify the set of $\mathcal{N}$ examples whilst misclassifying zero (or a small number) of $\mathcal{P}$ examples. Let $N$ contain the set of $\mathcal{N}$ examples yet to be covered and let $P$ contain the set of $\mathcal{P}$ examples that have been misclassified. We would like to find a subset of balls $\mathcal{R} \subset \mathcal{B}$ . Therefore, initially $\mathcal{R} \leftarrow \emptyset$ , $N \leftarrow \mathcal{N}$ and $P \leftarrow \emptyset$. At the first iteration the SCM algorithm looks for ball $h_{i,\rho}$ that maximises the utility value $U_{N,\mathcal{P}\setminus P}(h_{i,\rho})$. After the ball $h_{i,\rho}$ with the highest usefulness is found, then the subset $\mathcal{R} \leftarrow \mathcal{R} \cup \{h_{i,\rho}\}$ is updated together with $N \leftarrow N \setminus \nu_N(h_{i,\rho})$ and $P \leftarrow P \cup \pi_{\mathcal{P}\setminus P}(h_{i,\rho})$. This is repeated until $N = \emptyset$ is empty or until the early (soft) stopping criterion $|\mathcal{R}| \geq s$ is satisfied (where $s \in \mathbb{N}$ is a positive integer).

**Algorithm 1**: The set covering machine (SCM)

```
Input: empty hypothesis $\mathcal{R} \leftarrow \emptyset$ , the set of data-dependent balls $\mathcal{B}$, soft stopping parameter $s$
    and penalty parameter $p$.
1: for $i = 1$ to $s$ do
2:     find ball $h \in \mathcal{B}$ that maximises

           $U_{N,\mathcal{P}\setminus P}(h) = |\nu_N(h)| - p|\pi_{\mathcal{P}\setminus P}(h)|$

3:     update $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{h\}$, $N \leftarrow N \setminus \nu_N(h)$ and $P \leftarrow P \cup \pi_{\mathcal{P}\setminus P}(h)$
4: end for
Output: subset of data-dependent balls $\mathcal{R}$
```

Clearly the algorithm is greedy as it only adds ball $h$ to the subset $\mathcal{R}$ if it maximises the utility $U_{N,\mathcal{P}\setminus P}(h)$. Once the SCM has output $\mathcal{R}$ then we can construct a hypothesis $f$ consisting of data-dependent balls $\mathcal{R} \subset \mathcal{B}$ in order to make predictions.

Let $\mathbf{i} = (i_1, \ldots, i_s)$ denote an index vector that points to training examples in $S$, $s \leq |S|$. Therefore, given $\mathcal{R} = \{h_{\mathbf{i}_1,\rho}, \ldots, h_{\mathbf{i}_s,\rho}\}$ consisting of data-dependent balls we can define a function $f_T(\mathbf{x})$ to classify example $\mathbf{x}$ like so:

$$f_T(\mathbf{x}) = \begin{cases} y_T & \text{if } h(\mathbf{x}) = y_T \quad \forall h \in \mathcal{R} \\ \bar{y}_T & \text{otherwise} \end{cases} \tag{2.2}$$

where, from Definition 2.5, a conjunction $(T = c)$ SCM defines $y_c = 1$, $\bar{y}_c = -1$ and a disjunction $(T = d)$ SCM defines $y_d = -1$, $\bar{y}_d = 1$.

### 2.3.2 Regression: kernel least squares (KLSR)

The regression problem aims to find a hypothesis $f : \mathbf{x} \mapsto y$ that maps examples $\mathbf{x}$ to outputs $y \in \mathbb{R}$ given a regression sample $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$.

Least squares is a regression algorithm that only looks for linear relationships in the data. This is fine if the function can be approximated by a linear combination of the features, however, as pointed out earlier data sets are often non-linear. Therefore, algorithms reliant on the inner product, can in a very simple and natural manner be transformed into a non-linear algorithm by using a kernel.

We denote a matrix of examples by $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)^\top$ and a vector of real valued outputs as $\mathbf{y} = (y_1, \ldots, y_m)^\top$, where $\top$ denotes the transpose of a matrix or vector. Given a pair $(\mathbf{x}, y)$ we would like a function $f(\mathbf{x})$ that predicts the value of $y$. Because $y$ is no longer a discrete value the difference between the prediction and the actual value is a real value known as the *residual*. We would like to minimise the sum of squares of the residual by,

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2, \tag{2.3}$$

where $\mathbf{w}$ is known as a *weight vector* and forms the linear function. This is known as the primal least squares regression. The kernel (dual) least squares regression can be defined by writing the primal weight vector in terms of the training examples so that $\mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha}$, where $\boldsymbol{\alpha}$ is known as the *dual* weight vector. Making this substitution into the above minimisation problem gives the kernel least squares minimisation problem

$$\min_{\boldsymbol{\alpha}} \|\mathbf{y} - \mathbf{X}\mathbf{X}^\top \boldsymbol{\alpha}\|^2.$$

The solution of the above equation is given by

$$\boldsymbol{\alpha} = \left(\mathbf{X}\mathbf{X}^\top \mathbf{X}\mathbf{X}^\top\right)^{-1} \mathbf{X}\mathbf{X}^\top \mathbf{y},$$

where the inverse of $\left(\mathbf{X}\mathbf{X}^\top \mathbf{X}\mathbf{X}^\top\right)$ exists. Otherwise the pseudoinverse can be used. By making a further substitution to create the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$ we get,

$$\boldsymbol{\alpha} = \left(\mathbf{K}^\top \mathbf{K}\right)^{-1} \mathbf{K}\mathbf{y} = \mathbf{K}^{-1}\mathbf{y},$$

where as before we assume $\mathbf{K}^{-1}$ exists. This $\boldsymbol{\alpha}$ helps create a function that is linear in higher dimensional feature space – however making the substitution $\mathbf{w} = \mathbf{X}^\top\boldsymbol{\alpha}$ yields a non-linear function in the input space. Therefore, we can make predictions using a function $f(\mathbf{x}_i)$ on example $\mathbf{x}_i$ like so

$$\begin{aligned} f(\mathbf{x}_i) &= \mathbf{w}^\top\mathbf{x}_i && \text{(primal)} \\ &= \boldsymbol{\alpha}^\top\mathbf{K}[:,i] && \text{(dual)} \end{aligned}$$

where $\mathbf{K}[:,i] = (\kappa(\mathbf{x}_1,\mathbf{x}_i),\ldots,\kappa(\mathbf{x}_m,\mathbf{x}_i))^\top$ is defined as the kernel functions between all the training examples in $\mathbf{X} = (\mathbf{x}_1,\ldots,\mathbf{x}_m)^\top$ and $\mathbf{x}_i$. We will also refer to this as a *kernel basis vector*. For the remainder of the thesis we will only concern ourselves with dual problems.

### 2.3.3 Subspace methods: principal components analysis (PCA)

The problem of subspace methods such as principal components analysis can be described as finding a projection function $f$ that finds a low dimensional representation (a subspace) of the input space $\mathcal{X} \subset \mathbb{R}^n$ given a sample $S = \{\mathbf{x}_1,\ldots,\mathbf{x}_m\}$.

Principal components analysis (PCA) looks to find the directions in which $S = \{\mathbf{x}_i\}_{i=1}^m$ has maximal variance. This results in a low dimensional representation of $S$ that has the unimportant dimensions removed. We use the convention $\mathbf{X} = S$ to denote the training sample as rows of examples. The PCA problem is formulated as follows:

$$\max_{\mathbf{w}_x} \frac{\mathbf{w}_x^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}_x}{\mathbf{w}_x^\top\mathbf{w}_x}, \tag{2.4}$$

where $\mathbf{w}_x$ is the primal weight vector (eigenvector) that maximises the expression of Equation (2.4). This quotient is known as the *Rayleigh quotient* and the PCA maximisation problem is solved for the eigenvector $\mathbf{w}_x$ that has the largest corresponding eigenvalue $\lambda$. We can move on from this simple linear PCA problem to its dual formulation by mapping the training examples into higher dimensional feature spaces via the kernel trick. This equates to finding the primal weight vectors as a linear combination of the training examples and the dual weight vectors, hence, giving us the following identity:

$$\mathbf{w}_x = \mathbf{X}^\top\boldsymbol{\alpha}_x, \tag{2.5}$$

where $\boldsymbol{\alpha}_x$ is a dual weight vector. Making this substitution into Equation (2.4) gives us the dual PCA problem

$$\max_{\boldsymbol{\alpha}_x} \frac{\boldsymbol{\alpha}_x^\top\mathbf{X}\mathbf{X}^\top\mathbf{X}\mathbf{X}^\top\boldsymbol{\alpha}_x}{\boldsymbol{\alpha}_x^\top\mathbf{X}\mathbf{X}^\top\boldsymbol{\alpha}_x}, \tag{2.6}$$

and defining the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$ we have the kernel PCA problem

$$\max_{\boldsymbol{\alpha}_x} \frac{\boldsymbol{\alpha}_x^\top \mathbf{K}^\top \mathbf{K} \boldsymbol{\alpha}_x}{\boldsymbol{\alpha}_x^\top \mathbf{K} \boldsymbol{\alpha}_x}. \tag{2.7}$$

In a similar way as we had done with the kernel least squares regression we can of course use the projection function $f$ to map new examples $\mathbf{x}_i$ into the low dimensional subspace, like so:

$$f(\mathbf{x}_i) = \frac{\boldsymbol{\alpha}_x^\top \mathbf{K}[:, i]}{\|\boldsymbol{\alpha}_x\|}.$$

## 2.4 Learning theory

The goal of a learning theory is to formulate the learning phenomenon into a mathematical model that quantifies the level of learning that is possible. Two of the earliest pioneers of learning theory were Vapnik and Chervonenkis [1971] who developed what is now commonly referred to as Vapnik-Chervonenkis (VC) theory. This and related theories are also referred to as Statistical Learning Theory. Given some learning algorithm that generates a hypothesis the theory looks to prove upper bounds on the number of mistakes the hypothesis may incur in the future. The hope is that these bounds will hold with high probability and will be close to the true number of mistakes actually made by the hypothesis. A parallel development saw the definition of Probably Approximately Correct (PAC) [Valiant, 1984] learning that also further asserts that a polynomial time algorithm must exist. Perhaps, because of this very computer science requirement, it can be attributed to the development of what is known as Computational Learning Theory (CoLT). Note, that the extra requirement of a polynomial time algorithm is the only differentiating factor between PAC theory and VC theory.

The initial paper on PAC learning used a simple counting argument for hypothesis spaces in order to generate bounds on the generalisation error. However, many learning algorithms do not work in this restrictive regime and so several years later, Blumer et al. [1989] showed that a finite VC-dimension (see definition below) could also be used in order to prove learnability in the PAC sense. This work allowed the upper bounding of many more learning algorithms whose power (capacity/complexity) could be explained using the VC-dimension of the hypothesis class. Around the same time, Littlestone and Warmuth [1986] showed that data compression could also be related to PAC learnability and introduced sample compression theory. This theory, relates the level of sparsity that an algorithm achieves and simply counts the cardinality of the training examples (compression set) used in order to derive loss bounds. As with the VC-dimension, the size of the compression set can be viewed as the power/capacity of the hypothesis class. It is an open question if the VC-dimension of $d$ for a function class implies a sample compression set of size $d$.

Before we commence with some definitions and notations we would like to point out that we will give sketch proofs of all the results in this section as they are well known and because we will provide more elaborate proofs of all the bounds proposed later on in the thesis, which make use of the bounds presented here. Throughout the thesis we assume some probability distribution $\mathcal{D}$ generates a sample $S$ from a joint space $\mathcal{X} \times \mathcal{Y}$ in an independently and identically distributed (iid) manner. In this section, we assume $\mathcal{Y} = \{-1, 1\}$ is the output space. Given such an $S$ we would like to find a hypothesis $f$ that correctly computes $f(\mathbf{x}) = y$ with high probability for any $(\mathbf{x}, y) \sim \mathcal{D}$. We need a measure that calculates whether or not we have been erroneous in our prediction. Hence, we would like to define the probability of $f$ making a loss on future unseen data points. This is a very important quantity in machine learning and known as the *true error* (generalisation error).

**Definition 2.8** (true error). *Given a hypothesis $f$ and a probability distribution $\mathcal{D}$, the true error* $\mathrm{er}(\cdot)$ *of $f$ is the following probability,*

$$\mathrm{er}(f) \quad = \quad \Pr_{(\mathbf{x},y)\sim\mathcal{D}} \{f(\mathbf{x}) \neq y\}.$$

However, as we have pointed out above, the true error is a future loss and unobservable as the labels (and sometimes even the data points) have not yet been received. Before moving onto a quantity that is observable we would like to raise some points about the true error. Firstly, it is a quantity that all machine learning algorithms should try to minimise – however this is impossible to carry out directly because it is defined for data points whose labels are unknown (also referred to as *test data*). As its definition suggests, minimising the generalisation error would lead to hypotheses that have good generalisation in the future, a sound goal for all learning algorithms. There are two main principles that indirectly try to achieve this goal, called the Empirical Risk Minimisation (ERM) principle and the Structural Risk Minimisation (SRM) principle, both of which we will address in due course.

The fact that we cannot minimise the true error directly seems somewhat disappointing. However, there does exist a quantity that suggests an indirect minimisation of the generalisation error. Given a training sample $S$ and assuming that the data points in $S$ arrive iid from the same probability distribution $\mathcal{D}$ then we can define the following *empirical error*[2] as the (probability of) misclassification on the observed training sample $S$.

**Definition 2.9** (empirical error). *Given a hypothesis $f$ and a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, the empirical error* $\hat{\mathrm{er}}(\cdot)$ *of $f$ is the following quantity:*

$$\hat{\mathrm{er}}(f) \quad = \quad \Pr_{(\mathbf{x},y)\sim S} \{f(\mathbf{x}) \neq y\} = \frac{1}{m}\sum_{i=1}^{m} \mathbb{I}\left(f(\mathbf{x}_i) \neq y_i\right), \tag{2.8}$$

*where $\mathbb{I}$ is the indicator function.*

The empirical/training error is based on the training data that our learning machines have access to. By assuming that the training and testing data are generated from the same

[2]Empirical here indicates that the input-output pairs have been observed in the training data. This is also why it is commonly referred to as the training error.

distribution we are expecting that a learning protocol will only need to use the training data in order to generalise well in the future. This is true for many learning algorithms. The main goal, as mentioned above, is to minimise the true error. However, as this is not possible and from the assumptions we have made on the distribution generating the data we can try and minimise the empirical error. This is known as the *Empirical Risk Minimisation (ERM)* principle.

**Definition 2.10** (Empirical Risk Minimisation). *Given a hypothesis space $\mathcal{H}$, the Empirical Risk Minimisation principle wants to find a hypothesis $f^* \in \mathcal{H}$ having the smallest empirical error. This equates to the following expression:*

$$ERM(f^*) \quad = \quad \arg\min_{f\in\mathcal{H}} \{\hat{\text{er}}(f)\} .$$

Given a training sample $S$, any learning algorithm will want to minimise the true error. The ERM principle is an indirect route to achieving this goal. By making the assumption that the training and testing data come from the same distribution we can look to carry out the ERM in hope that it will also minimise the true error. However, an important issue is that simply minimising the empirical error could lead to overftting, finding functions that are only capable of correctly classifying the training data but poor at generalising in the future. This is a danger of the ERM principle but can be avoided using regularisation. This is where we would, for example, solve a more approximate problem such as in ridge regression where the ridge parameter acts as a "smoother" of the function generated by least squares regression. This smoothing in effect makes sure that we find functions that are more general. Another way to think about this is that it is an attempt at striking a balance between the complexity of the function learnt and the accuracy on the training data. This leads us onto what is known as the Structural Risk Minimisation (SRM) principle.

**Definition 2.11** (Structural Risk Minimisation). *Given a hypothesis space $\mathcal{H}$, a complexity measure $V(f)$ of a hypothesis $f$ then the Structural Risk Minimisation principle wants to find a hypothesis $f^* \in \mathcal{H}$ whose empirical error plus complexity is minimal. This equates to the following expression:*

$$SRM(f^*) \quad = \quad \arg\min_{f\in\mathcal{H}} \{\hat{\text{er}}(f) + \lambda V(f)\} . \tag{2.9}$$

The complexity (structure) term tries to avoid the situation of over/under-fitting the data. It is not rigourously defined above, as there are several different complexity terms that can and have been used in learning theory. We will develop these terms below and show that in sparsity this complexity term simply translates to the parsimony of the hypothesis sought. However, before moving onto specifics of capacity terms we now commence with a discussion about the PAC learning framework, that allows us to formulate upper bounds for learning algorithms. After this we go on to discuss more complex bounds, in the PAC framework, that take into account the structure of the hypotheses found.

### 2.4.1 Probably Approximately Correct (PAC) theory

PAC learning is an acronym for "Probably Approximately Correct". Simply translated as saying that we can find a hypothesis with low error (approximately correct) with high probability (probably). These bounds rely on using the information available to us and not making any assumptions about the distribution that may have generated the data. This is why they are sometimes referred to as distribution-free bounds.

**Definition 2.12** (Probably Approximately Correct). *Given a hypothesis space $\mathcal{H}$ with input space $\mathcal{X}$, a learning algorithm $\mathcal{A}$ is probably approximately correct if for all distributions $\mathcal{D}$ on $\mathcal{X}$, for all $\epsilon, \delta \in \mathbb{R}$ $(0 \leq \epsilon, \delta < 1)$, there is a sufficient sample size $m_0$ such that if $m \geq m_0$, then*

$$\Pr\{S\colon \exists f \in \mathcal{H}\colon \hat{\mathrm{er}}(f) = 0, \mathrm{er}(f) > \epsilon\} < \delta,$$

*where $S$ is an $m$-sample generated iid according to $\mathcal{D}$.*

We will use the notation $\Pr\{S : \ldots\}$ to denote the probability $P^m\{\ldots\}$ over $m$ points from the sample $S$ generated iid. An interpretation of the PAC bound is that the probability of finding a hypothesis $f \in \mathcal{H}$ whose training error $\hat{\mathrm{er}}(f) = 0$ and true error $\mathrm{er}(f) < \epsilon$ is lower bounded with high confidence (greater than a probability of $1 - \delta$). If we can show that an algorithm will find a hypothesis that is provably PAC then with very high probability we will have an error in the future that is less than $\epsilon$. A practical PAC bound looks to find the number $\epsilon$ which in turn is an upper bound for the true error $\mathrm{er}(f)$ of the function $f$. We now discuss a PAC bound that uses the size of the hypothesis space as the main bounding principle to find a value for $\epsilon$.

Given a sample $S$ consisting of $m$ examples and a hypothesis space $\mathcal{H}$, then the probability of finding a function $f \in \mathcal{H}$ that has error greater than some $\epsilon \in \mathbb{R}$ $(0 \leq \epsilon < 1)$ will be greater than $1 - \epsilon$. If we have $m$ independent samples then this will be $(1-\epsilon)^m \leq \exp(-\epsilon m)$. Applying the union bound we have the following upper bound

$$\Pr\{S : \hat{\mathrm{er}}(f) = 0, \mathrm{er}(f) > \epsilon\} < |\mathcal{H}| \exp(-\epsilon m). \qquad (2.10)$$

The bound uses the size of the hypothesis space $\mathcal{H}$ and the number of samples $m$ in its calculation. There is no sign of the distribution that generated the data and so we can give a PAC bound in terms of the sample complexity.

**Theorem 2.1** (Sample complexity bound). *Let $\mathcal{H}$ be the hypothesis space and $m$ the size of the sample. Then with probability $1 - \delta$, given a function $f \in \mathcal{H}$ with zero training error $\hat{\mathrm{er}}(f) = 0$, we can upper bound the true error $\mathrm{er}(f)$ by,*

$$\mathrm{er}(f) \leq \frac{1}{m}\left[\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right] \qquad (2.11)$$

*Proof.* Set rhs of Equation (2.10) to $\delta$ and solve for $\epsilon$. □

This bound can only be formed by counting the size of the hypothesis space and hence implies the hypothesis space must be finite. However, in most situations this is not the case. For example, the set of hyperplanes in the SVM algorithm is an infinite set. In the case when we have an infinite hypothesis space we can use a complexity term known as the VC-dimension to help upper bound the future loss of an algorithm.

### 2.4.2 Vapnik-Chervonenkis (VC) theory

As was mentioned at the start of the previous section, Vapnik and Chervonenkis have been pioneers of learning theory. They helped develop the theory of generalisation error bounds and proposed the VC dimension. We start the discussion with a quantity that is related to the VC dimension known as the growth function.

**Definition 2.13** (Growth function). *Given a hypothesis space $\mathcal{H}$, a set of $m$ examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ then the growth function $\Pi_{\mathcal{H}}(m)$ can be defined like so:*

$$\Pi_{\mathcal{H}}(m) = \max_{(\mathbf{x}_1,\ldots,\mathbf{x}_m)\in\mathcal{X}^m} \left|\{(f(\mathbf{x}_1),\ldots,f(\mathbf{x}_m)) : f \in \mathcal{H}\}\right|$$

In other words, the growth function is the maximum number of distinct binary vectors that can be formed from a set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ of $m$ input points using every possible hypothesis in $\mathcal{H}$. As we are dealing with a classification problem and each example can only take one of two predictions $-1$ or $+1$, then we can have a maximum of $2^m$ such distinct vectors. If these $2^m$ distinct vectors can be realised then we say that a set of $m$ points is *shattered* by $\mathcal{H}$.

**Definition 2.14** (Vapnik-Chervonenkis dimension). *The Vapnik-Chervonenkis (VC) dimension $VCdim(\mathcal{H})$ is the* maximum *number of points $d$ shattered by $\mathcal{H}$. If such sets of all sizes exist then $VCdim(\mathcal{H}) = \infty$.*

The VC-dimension quantifies the power of a hypothesis space with a single number $d$. If $\forall m$, not all $2^m$ realisations can be achieved then the true error of an algorithm defining an infinite hypothesis space can be upper bounded. This is done using the *double sample trick.* For details of the proof see [Blumer et al., 1989, Anthony and Biggs, 1992].

**Theorem 2.2** (Blumer et al. [1989]). *Let $\mathcal{H}$ be a hypothesis space, $S$ a training sample of size $m$ and $\epsilon \in \mathbb{R}$. Then for sample $S$, the probability of $f \in \mathcal{H}$ observing zero empirical error $\hat{\mathrm{er}}(f)$ and true error $\mathrm{er}(f)$ greater than some $\epsilon$ can be upper bounded by,*

$$\Pr\{S : \hat{\mathrm{er}}(f) = 0, \mathrm{er}(f) > \epsilon\} \leq 2\Pi_{\mathcal{H}}(2m)2^{-\epsilon m/2}.$$

The growth function can be upper bounded using the VC-dimension and hence an upper bound given for learning algorithms whose hypothesis spaces can be quantified by a finite VC-dimension.

**Theorem 2.3** (VC theory bound). *Let $\mathcal{H}$ be a hypothesis space having finite VC-dimension $d$. For any probability distribution $\mathcal{D}$ on $\mathcal{X} \times \{-1, 1\}$, with probability $1 - \delta$ over $m$ random* *examples $S$, any hypothesis $f \in \mathcal{H}$ that has* $\hat{\text{er}}(f) = 0$ *has true error* $\text{er}(f)$ *upper bounded by,*

$$\text{er}(f) \leq \frac{2}{m}\left[d\log\left(\frac{2em}{d}\right) + \log\left(\frac{2}{\delta}\right)\right]$$

*Proof.* By making use of Sauers lemma and the VC dimension $d$ we can upper bound the growth function for $m > d$ by,

$$\Pi_{\mathcal{H}}(m) \leq \left(\frac{em}{d}\right)^d,$$

which gives polynomial growth in exponent $d$. Making this substitution into the rhs of the probability in Theorem 2.2, setting it equal to $\delta$ and solving for $\epsilon$ gives the desired result. $\square$

The VC bound is data independent in the sense that it does not take into account the information from the data points but simply needs to find a finite VC-dimension hypothesis class in order for the learning algorithm to subsume PAC learnability. By data independent we mean that the VC-dimension is calculated from whether the hypothesis space $\mathcal{H}$ can shatter some set of $m$ points not just those given in the sample $S$. However, the information gained from the hypothesis constructed from the data can give us more powerful and general complexity measures that ultimately yield tighter upper bounds on the generalisation error. This is known as a *data-dependent bound* and is possible in the sample compression framework.

### 2.4.3 Sample compression theory

In 1986, Littlestone and Warmuth wrote a technical report that was never published. Its influence allowed several learning algorithms to be bounded in terms of the so-called *compression set*. A publication in 1995 by Sally Floyd and Manfred Warmuth on "Sample compression, leanability and the Vapnik-Chervonenkis dimension" addressed the earlier work of Littlestone and Warmuth [1986] and also showed relationships between the size of the compression set and the VC-dimension. We show later that there is a one-to-one correspondence (in some cases) that helps bound regression style algorithms in a sample compression setting. The relationship between these two numbers is still a topic of research and beyond the scope of this thesis. The interested reader is referred to the reference Floyd and Warmuth [1995]. Our interest for the moment with sample compression theory is that it can help bound learning algorithms in a data-dependent fashion and also in terms of the level of sparsity achieved by the functions – giving us a complexity measure other than that defined by the VC dimension. We commence with a definition of a sample compression scheme.

**Definition 2.15.** *The* **compression function** $\Lambda$ *induced by a sample compression algorithm $\mathcal{A}$ on training set $S$ is the map*

$$\Lambda : S \longmapsto \Lambda(S)$$

*such that the* compression set $\Lambda(S) \subset S$ *is returned by $\mathcal{A}$.*

*A* **reconstruction function** $\Phi$ *is a mapping from a compression set $\Lambda(S)$ to a set $\mathcal{H}$ of* *hypotheses*

$$\Phi : \Lambda(S) \longmapsto f \in \mathcal{H}.$$

With these ingredients we can now define a compression scheme.

**Definition 2.16.** *Let $\mathcal{A}(S)$ be the function output by learning algorithm $\mathcal{A}$ on training set $S$. A sample compression scheme is a reconstruction function $\Phi$ mapping a compression set $\Lambda(S)$ to some set of functions $\mathcal{H}$ such that*

$$\mathcal{A}(S) = \Phi(\Lambda(S)).$$

*If $\mathcal{H}$ is the set of Boolean-valued functions then the sample compression scheme is said to be a classification algorithm. If $\mathcal{H}$ is the set of Real-valued functions then the sample compression scheme is a regression algorithm.*

This definition suggests that a compression scheme and a learning algorithm are equivalent if and only if the learning algorithm can construct its hypothesis from a small subset of the data (the compression set).

An example of a compression scheme (see Herbrich [2002]) is the support vector machine algorithm $\mathcal{A}_{svm}$ which produces a large margin separating hyperplane $\mathcal{A}_{svm}(S)$ from the set of support vectors. Observe that the SVM will produce the same hypothesis by only using the set of support vectors - we can view this as the compression set $\Lambda_{svm}(S)$. Furthermore, we can define the notion of a reconstruction function $\Phi_{svm}$ by rerunning the SVM using only this set $\Lambda_{svm}(S)$ and producing the same hyperplane $\mathcal{A}_{svm}(S)$ as the SVM, *i.e.*, $\mathcal{A}_{svm}(S) = \Phi_{svm}(\Lambda_{svm}(S))$.

We can bound the performance of learning algorithms using the size of the compression set (*i.e.*, $|\Lambda(S)|$) in a PAC sense using the following simple counting argument. Let us denote by $d$ the cardinality of the compression set $\Lambda(S)$. We showed for the VC bound that the error of any function $f \in \mathcal{H}$ could be upper bounded by assuming a double sample and requiring a finite VC dimension. In the sample compression theory the double sample is in fact implicit in the definition of a compression scheme. The second sample $\hat{S}$ that acted as a test set in the VC argument is in fact the set of points that remain outside of the compression set in the sample compression argument. The size of the compression set acts like the VC dimension *i.e.*, as a single number defining the complexity of the classifier learnt. However, the size of the compression set is computed after the hypothesis has been constructed and is in fact data-dependent unlike the VC dimension. Therefore, we can bound learning algorithms using the size of the compression set.

Recall that the error of the function $f$ is at most $(1-\epsilon)^m$ for $m$ points given a second sample $\hat{S}$. Therefore, if the second sample is defined over the $m - d$ points outside of the compression set then we have a probability of $(1-\epsilon)^{m-d} \leq \exp(\epsilon(m-d))$. Using this trick we have saved on the 2 that appears in the VC bound. Next we would like to bound the error for every possible hypothesis that could be constructed of size $d$ from a sample of size $m$. This is the binomial coefficient $\binom{m}{d}$ and so a union bound over the number of errors that can occur in all possible ways gives the following bound for a sample compression scheme making zero training error.

**Theorem 2.4** (Sample compression bound [Bartlett and Shawe-Taylor, 1999]). *Consider a compression scheme $\Phi(\Lambda(S))$. For any probability distribution $\mathcal{D}$ on $\mathcal{X} \times \{-1, 1\}$, with probability $1 - \delta$ over $m$ random examples $S$, the true error* $\mathrm{er}(f)$ *of a hypothesis $f \in \mathcal{H}$ defined by a compression set of size $d$ can be upper bounded by,*

$$\mathrm{er}(f) \;\leq\; \frac{1}{m-d}\left[d \ln\left(\frac{em}{d}\right) + \ln\left(\frac{m}{\delta}\right)\right].$$

*Proof.* We can upper bound the probability of making zero training error $\hat{\mathrm{er}}_{\hat{S}}(f) = 0$ on the set $\hat{S} = S \setminus \Lambda(S)$, but large true error $\mathrm{er}(f) > \epsilon$, by the following quantity,

$$\Pr\left\{S\colon \hat{\mathrm{er}}_{\hat{S}}(f) = 0, \mathrm{er}(f) > \epsilon\right\} < \binom{m}{d} \exp(-\epsilon(m-d)). \tag{2.12}$$

Therefore the result follows by upper bounding the Binomial coefficient using Sauer's lemma *i.e.*, $\binom{m}{d} \leq \left(\frac{em}{d}\right)^d$, setting the rhs of the above probability to $\delta$ and multiplying it by $1/m$ for each possible choice of $d$, and solving for $\epsilon$. □

**Remark 2.1.** *The sample compression bound can take into account misclassifications on the set $\hat{S} = S \setminus \Lambda(S)$ by counting the number of ways of choosing the errors $k$ from the $m - d$ points and multiplying the rhs of probability (2.12) with the Binomial coefficient $\binom{m-d}{k}$.*

**Remark 2.2.** *The sample compression scheme bounds given here only need the information from the compression set in order to recreate the hypotheses found by algorithm $\mathcal{A}$. However, in some situations we also need a* message string *that defines extra information needed in order to make this reconstruction. This type of extra information is needed in the set covering machine (SCM) where it is required to distinguish between border points and centre points found in the compression set. This will be explored further in the next chapter.*

The data-dependent bound we have just given for sample compression schemes implies that we will learn well if the size of the compression set is small and the error is also small (in the case when we allow misclassifications). This is a structural risk minimisation style bound and helps to avoid the problem of overfitting by guarding against classifiers that are too complex. The SCM algorithm contains sample compression bounds and looks to trade-off the complexity of the classifier against the empirical error. We show in Chapter 3 that we can directly minimise such error bounds in order to train the set covering machine algorithm. But first, we give a detailed and elaborated proof of a sample compression bound that accounts for the loss on both classes of examples, a property referred to as asymmetrical loss.

Chapter 3

# Sparsity in supervised learning

*In this chapter we discuss supervised learning algorithms for classification and regression. For classification we propose a sample compression bound for the set covering machine (SCM) that allows the user the ability to specify different costs (weights) for the misclassification of different classes (asymmetric loss). Next we look to apply the generalisation error bounds for the SCM (including the novel bound proposed) in order to train the algorithm. Recall that the SCM contains two parameters that need tuning during training, the penalty parameter* $p$ *and the soft stopping parameter* $s$*. By noting that the non-trivial bounds are tight enough for model selection we propose bound-driven learning algorithms for the SCM that remove both of these parameters. The first variant is called the bound set covering machine (BSCM) that greedily adds data-dependent balls to the hypothesis. However, this is a sub-optimal algorithm and so we extend our result to compute optimal solutions, and call it the branch and bound set covering machine (BBSCM). By using a branch and bound approach we prove that the BBSCM will produce the hypothesis with the smallest generalisation error bound. Therefore, we also propose a final algorithm called the BBSCM(*$\tau$*) that produces (i) classifiers whose generalisation error bounds are a factor* $\tau$ *from the optimal and (ii) trades off time against the quality of the solution.*

*In the regression section we turn our attention to a sparse non-linear variant of least squares regression called kernel matching pursuit (KMP) . We describe the algorithm and show that unlike the SCM it does not form a compression scheme and hence cannot take advantage of sample compression theory. We show that although this is the case, we can view the kernel defined feature space defined by the KMP algorithm as a sample compression scheme and make use of VC theory to help bound its future loss. Therefore, we propose the first (to our knowledge) upper bound for KMP that jointly uses VC theory and sample compression theory in order to construct loss bounds. We end the chapter with a plot of the KMP bound against its test error on a Boston housing data set and show that the lowest bound value actually coincides with the smallest test error for KMP.*

## 3.1 Introduction

The following bound proposed by Marchand and Shawe-Taylor [2001] bounds the performance of the SCM using sample compression theory.

**Theorem 3.1** (Marchand and Shawe-Taylor [2001]). *Suppose an SCM finds a hypothesis $f$ given by a set $\mathcal{R} \subset \mathcal{B}$ of data-dependent balls with $d = d(\mathcal{R}) = |\mathcal{R}|$ balls, $c_p = c_p(\mathcal{R})$ of which are centred around $\mathcal{P}$ examples, $k_p = k_p(\mathcal{R})$ are the number of $\mathcal{P}$ training errors and $k_n = k_n(\mathcal{R})$ the number of $\mathcal{N}$ training errors on a sample of size $m > 2d + k_p + k_n$.*[1] *Then with probability $1 - \delta$ over random draws of training sets, the generalisation error* $\mathrm{er}(f)$ *of the resulting classifier $f$ can be bounded by*

$$\begin{aligned} \mathrm{er}(f) \leq \epsilon(m, d, c_p, k_p, k_n, \delta) &= 1 - \exp\left\{ \frac{-1}{m - 2d - k_p - k_n} \left( \ln \binom{m}{2d} + \ln \binom{2d}{c_p} \right.\right. \\ &\left.\left. + \ln \binom{m - 2d}{k_p + k_n} + \ln \frac{2m^2 d}{\delta} \right) \right\} \end{aligned}$$

This bound is only specialised to the SCM and is not general enough to be applied to any other sample compression based algorithms that rely on extra sources of information[2] in order to make the reconstruction of the hypothesis. The second bound was originally constructed for the Decision List Machine (DLM) Marchand and Sokolova [2005] and is a more general result that can be applied to different sample compressed learning algorithms, and is slightly tighter than Theorem 3.1 (for the SCM).

**Theorem 3.2** (Marchand and Sokolova [2005]). *Suppose an SCM finds a solution given by a set $\mathcal{R}$ of data-dependent balls with $c_p = c_p(\mathcal{R})$ centred around $\mathcal{P}$ examples, $c_n = c_n(\mathcal{R})$ centred around $\mathcal{N}$ examples and $b_p = b_p(\mathcal{R})$ borders defined only by $\mathcal{P}$ examples, with $k_p = k_p(\mathcal{R})$ the number of $\mathcal{P}$ training errors and $k_n = k_n(\mathcal{R})$ the number of $\mathcal{N}$ training errors. Then with probability $1-\delta$ over random draws of training sets, the generalisation error* $\mathrm{er}(f)$ *of the resulting classifier $f$ can be bounded by*

$$\begin{aligned} \mathrm{er}(f) \leq \epsilon(d, c_p, b_p, k_p, k_n, \delta) &= 1 - \exp\left\{ \frac{-1}{m - d - k} \left( \ln \binom{m}{d} + \ln \binom{m - d}{k} \right.\right. \\ &\left.\left. + \ln \binom{c_p + b_p}{b_p} + \ln \left( \frac{1}{\zeta(d)\zeta(k)\zeta(b_p)\delta} \right) \right) \right\} \end{aligned}$$

*where $d = c_n + c_p + b_p$, $k = k_p + k_n$, and $\zeta(a) = 6\pi^{-2}(a+1)^{-2}$.*

Although, this bound is more general it does not take into account the individual losses on the two classes of examples. However, to obtain a loss bound that reflects more accurately the performance of classifiers trained on imbalanced data sets, Marchand and Shawe-Taylor [2002]

[1] Strictly speaking we use $d$ to define the size of the compression set throughout this chapter, however, in this instance we make a slight abuse of this fact and use it to denote the number of balls. However, this is only done to simplify notation in the work that will be presented in Section 3.5 and to avoid using lots of different notations.

[2] in the SCM this relates to whether the points in the compression set are ball borders.

have proposed a SCM loss bound that depends on the observed fraction of positive examples in the training set and on the fraction of positive examples used for the compression set of the final classifier. Hussain [2003] showed that this loss bound is incorrect. Therefore we propose, in Section 3.3, a general loss bound which is valid for any sample compression learning algorithm (including the SCM) and that depends on the observed fraction of positive examples and on what the classifier achieves on the positive training examples. This loss bound is applicable to a broader class of compression schemes.

We begin our discussion with preliminary definitions and terminology. Let the input space $\mathcal{X}$ be a set of $n$-dimensional vectors of $\mathbb{R}^n$ and let $\mathbf{x}$ be a member of $\mathcal{X}$. A positive example will be referred to as a $\mathcal{P}$-example and a negative example as a $\mathcal{N}$-example. Given a training set $S = S_{\mathcal{P}} \cup S_{\mathcal{N}}$ of examples, the set of positive training examples will be denoted by $S_{\mathcal{P}}$ and the set of negative training examples by $S_{\mathcal{N}}$.

Any learning algorithm that constructs a conjunction can be transformed into an algorithm constructing a disjunction just by exchanging the role of the positive and negative examples (see Definition 2.5). For the rest of this chapter we will assume, without loss of generality, that the SCM always produces a conjunction.

In this chapter, we use the set of data-depedent balls $\mathcal{B}$ defined in Definition 2.6. Hence, the subset $\mathcal{R} \subset \mathcal{H}$ of data-dependent balls found by the SCM gives us a set of ball centres and a set of ball borders. The union of these two sets gives us the *compression set* of the SCM (see definition 2.15). We refine further this notion, for the SCM, in Section 3.3.

We adopt the PAC model where it is assumed that each example $(\mathbf{x}, y)$ is drawn independently at random according to a fixed (but unknown) distribution. In this chapter, we consider the probabilities of events taken separately over the $\mathcal{P}$-examples and the $\mathcal{N}$-examples. We will therefore denote by $\Pr_{(\mathbf{x},y)\sim\mathcal{P}}\{a(\mathbf{x}, y)\}$ the probability that predicate $a$ is true on a random draw of an example $(\mathbf{x}, y)$, given that this example is positive. Hence, the error probability of classifier $f$ on $\mathcal{P}$-examples and on $\mathcal{N}$-examples, that we call respectively the *expected $\mathcal{P}$-loss* and the *expected $\mathcal{N}$-loss*, are given by

$$\begin{aligned} \mathrm{er}_{\mathcal{P}}(f) &= \Pr_{(\mathbf{x},y)\sim\mathcal{P}}\{f(\mathbf{x}) \neq y\}, \\ \mathrm{er}_{\mathcal{N}}(f) &= \Pr_{(\mathbf{x},y)\sim\mathcal{N}}\{f(\mathbf{x}) \neq y\}. \end{aligned}$$

Similarly, let $\widehat{\mathrm{er}}_{\mathcal{P}}(f, S)$ denote the proportion of examples in $S_{\mathcal{P}}$ misclassified by $f$ and let $\widehat{\mathrm{er}}_{\mathcal{N}}(f, S)$ denote the proportion of examples in $S_{\mathcal{N}}$ misclassified by $f$. Hence

$$\begin{aligned} \widehat{\mathrm{er}}_{\mathcal{P}}(f, S) &= \Pr_{(\mathbf{x},y)\sim S_{\mathcal{P}}}\{f(\mathbf{x}) \neq y\} = \mathbb{E}_{(\mathbf{x},y)\sim S_{\mathcal{P}}}\{\mathbb{I}(f(\mathbf{x}) \neq y))\}, \\ \widehat{\mathrm{er}}_{\mathcal{N}}(f, S) &= \Pr_{(\mathbf{x},y)\sim S_{\mathcal{N}}}\{f(\mathbf{x}) \neq y\} = \mathbb{E}_{(\mathbf{x},y)\sim S_{\mathcal{N}}}\{\mathbb{I}(f(\mathbf{x}) \neq y))\}. \end{aligned}$$

The probability of occurrence of a positive example will be denoted by $p_{\mathcal{P}}$. Similarly, $p_{\mathcal{N}}$ will denote the probability of occurrence of a negative example. We will consider the general case where the loss $l_{\mathcal{P}}$ of misclassifying a positive example can differ from the loss $l_{\mathcal{N}}$ of misclassifying a negative example. We will denote by $\mathcal{A}(S)$ the classifier returned by the learning algorithm $\mathcal{A}$ trained on a set $S$ of examples. In this case, the expected loss $\mathcal{E}[\ell(\mathcal{A}(S))]$ of classifier $\mathcal{A}(S)$ is defined as

$$\mathcal{E}[\ell(\mathcal{A}(S))] \;=\; l_{\mathcal{P}} \cdot p_{\mathcal{P}} \cdot \mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)] \;+\; l_{\mathcal{N}} \cdot p_{\mathcal{N}} \cdot \mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)] \tag{3.1}$$

## 3.2 Incorrect Bound

Theorem 5 of Marchand and Shawe-Taylor [2002] gives the following loss bound for the SCM with the symmetric loss case of $l_{\mathcal{P}} = l_{\mathcal{N}} = 1$.

*Given the above definitions, let $A$ be any learning algorithm that builds a SCM with data-dependent balls with the constraint that the returned function $A(S)$ always correctly classifies every example in the compression set. Then, with probability $1-\delta$ over all training sets $S$ of $m$ examples,*

$$\mathcal{E}[\ell(\mathcal{A}(S))] \;\leq\; 1-\exp\left\{-\frac{1}{m-c_p-b-c_n-k_p-k_n}\left(\ln B + \ln\frac{1}{\delta_0}\right)\right\},$$

*where*

$$\begin{aligned}\delta_0 &= \left(\frac{\pi^2}{6}\right)^{-5} \cdot ((c_p+1)(c_n+1)(b+1)(k_p+1)(k_n+1))^{-2} \cdot \delta \\ B &= \binom{m_p}{c_p}\binom{m_p-c_p}{b}\binom{m_n}{c_n}\binom{m_p-c_p-b}{k_p}\binom{m_n-c_n}{k_n},\end{aligned}$$

*and where $k_p$ and $k_n$ are the number of misclassified positive and negative training examples by classifier $A(S)$. Similarly, $c_p$ and $c_n$ are the number of positive and negative ball centres contained in classifier $A(S)$ whereas $b$ denotes the number of ball borders*[3] *in classifier $A(S)$. Finally $m_p$ and $m_n$ denote the number of positive and negative examples in training set $S$.*

Let us take the $B$ expression only and look more closely at the number of ways of choosing the errors on $S_{\mathcal{P}}$ and $S_{\mathcal{N}}$:

$$\binom{m_p-c_p-b}{k_p}\binom{m_n-c_n}{k_n}.$$

As was pointed out by Hussain [2003] the bound on the expected loss given above will be small only if each factor is small. However, each factor can be small for a small number of training errors (desirable) or a large number of training errors (undesirable). In particular, the product of these two factors will be small for a small value of $k_n$ (say, $k_n = 0$) and a large value of $k_p$ (say, $k_p = m_p - c_p - b$). In this case, the denominator in the exponential part of the bound

[3]As explained in Marchand and Shawe-Taylor [2002], the ball borders are always positive examples.

given above will become

$$m - c_p - b - c_n - k_p - k_n = m_n - c_n \, ,$$

and will be large whenever $m_n \gg c_n$. Consequently, the bound given by Theorem 5 of Marchand and Shawe-Taylor [2002] will be small for classifiers having a small compression set and making a large number of errors on $S_{\mathcal{P}}$ and a small number of errors on $S_{\mathcal{N}}$. Clearly, this is incorrect as it implies a classifier with good generalisation ability and so exposes an error in the proof. In order to derive a loss bound where the issue of imbalanced misclassifications can be handled, the errors for positive and negative examples must be bounded separately.

## 3.3 Sample Compression Loss Bounds for Imbalanced Data

Recall that $\mathcal{X}$ denotes the input space. Let $X = (\mathcal{X} \times \{-1, 1\})^m$ be the set of training sets of size $m$ with inputs from $\mathcal{X}$. We consider any learning algorithm $\mathcal{A}$ having the property that, when trained on a training set $S \in X$, $\mathcal{A}$ produces a classifier $\mathcal{A}(S)$ which can be identified solely by a subset $\Lambda = \Lambda_{\mathcal{P}} \cup \Lambda_{\mathcal{N}} \subset S$, called the *compression set*, and a *message string* $\sigma$ that represents some additional information required to obtain a classifier. Here $\Lambda_{\mathcal{P}}$ represents a subset of positive examples and $\Lambda_{\mathcal{N}}$ a subset of negative examples. More formally, this means that there exists a *reconstruction function* $\Phi$ that produces a classifier $f = \Phi(\Lambda, \sigma)$ when given an arbitrary compression set $\Lambda$ and message string $\sigma$. We can thus consider that the learning algorithm $\mathcal{A}$, trained on $S$, returns a compression set $\Lambda(S)$ and a message string $\sigma(S)$. The classifier is then given by $\Phi(\Lambda(S), \sigma(S))$.

For any training sample $S$ and compression set $\Lambda$, consisting of a subset $\Lambda_{\mathcal{P}}$ of positive examples and a subset $\Lambda_{\mathcal{N}}$ of negative examples, we use the notation $\Lambda(S) = (\Lambda_{\mathcal{P}}(S), \Lambda_{\mathcal{N}}(S))$. Any further partitioning of the compression set $\Lambda$ can be performed by the message string $\sigma$. For example, in the set covering machine, $\sigma$ specifies for each point in $\Lambda_{\mathcal{P}}$, whether it is a ball centre or a ball border (not already used as a centre). As explained by Marchand and Shawe-Taylor [2002], this is the only additional information required to obtain a SCM consistent with the compression set.

We will use $d_p$ to denote the number of examples present in $\Lambda_{\mathcal{P}}$. Similarly, $d_n$ will denote the number of examples present in $\Lambda_{\mathcal{N}}$. To simplify the notation, we will use the $\mathbf{m}_{\mathcal{P}}$ and $\mathbf{m}_{\mathcal{N}}$ vectors defined as

$$\mathbf{m}_{\mathcal{P}} = (m, m_p, m_n, d_p, d_n, k_p) \qquad (3.2)$$

$$\mathbf{m}_{\mathcal{N}} = (m, m_p, m_n, d_p, d_n, k_n) \, , \qquad (3.3)$$

and

$$\begin{aligned}
\mathbf{m}_{\mathcal{P}}(S,\mathcal{A}(S)) &= (|S|,|S_{\mathcal{P}}|,|S_{\mathcal{N}}|,|\Lambda_{\mathcal{P}}(S)|,|\Lambda_{\mathcal{N}}(S)|,\hat{\text{er}}_{\mathcal{P}}(\mathcal{A}(S),S)\times m_p) &(3.4)\\
\mathbf{m}_{\mathcal{N}}(S,\mathcal{A}(S)) &= (|S|,|S_{\mathcal{P}}|,|S_{\mathcal{N}}|,|\Lambda_{\mathcal{P}}(S)|,|\Lambda_{\mathcal{N}}(S)|,\hat{\text{er}}_{\mathcal{N}}(\mathcal{A}(S),S)\times m_n)\,. &(3.5)
\end{aligned}$$

Hence, the predicate $\mathbf{m}_{\mathcal{P}}(S,\mathcal{A}(S)) = \mathbf{m}_{\mathcal{P}}$ means that $|S| = m$, $|S_{\mathcal{P}}| = m_p$, $|S_{\mathcal{N}}| = m_n$, $|\Lambda_{\mathcal{P}}(S)| = d_p$, $|\Lambda_{\mathcal{N}}(S)| = d_n$, $\hat{\text{er}}_{\mathcal{P}}(\mathcal{A}(S),S) = \frac{k_p}{m_p}$. We use a similar definition for predicate $\mathbf{m}_{\mathcal{N}}(S,\mathcal{A}(S)) = \mathbf{m}_{\mathcal{N}}$. We will also use $B_{\mathcal{P}}(\mathbf{m}_{\mathcal{P}})$ and $B_{\mathcal{N}}(\mathbf{m}_{\mathcal{N}})$ defined as

$$\begin{aligned}
B_{\mathcal{P}}(\mathbf{m}_{\mathcal{P}}) &= \binom{m_p}{d_p}\binom{m_n}{d_n}\binom{m_p-d_p}{k_p} &(3.6)\\
B_{\mathcal{N}}(\mathbf{m}_{\mathcal{N}}) &= \binom{m_p}{d_p}\binom{m_n}{d_n}\binom{m_n-d_n}{k_n}\,. &(3.7)
\end{aligned}$$

The proposed loss bound will hold uniformly for all possible messages that can be chosen by $\mathcal{A}$. It will thus loosen as we increase the set $\mathcal{M}$ of possible messages that can be used. To obtain a smaller loss bound, we will therefore permit $\mathcal{M}$ to be *dependent* on the compression set chosen by $\mathcal{A}$. In fact, the loss bound will depend on a *prior* distribution $P_\Lambda(\sigma)$ of message strings over the set $\mathcal{M}_\Lambda$ of possible messages that can be used with a compression set $\Lambda$. We will see that the only condition that $P_\Lambda$ needs to satisfy is

$$\sum_{\sigma\in\mathcal{M}_\Lambda} P_\Lambda(\sigma) \le 1\,.$$

Consider, for example, the case of a SCM conjunction of balls. Given a compression set $\Lambda = (\Lambda_{\mathcal{P}},\Lambda_{\mathcal{N}})$ of size $(|\Lambda_{\mathcal{P}}|,|\Lambda_{\mathcal{N}}|) = (d_p,d_n)$, recall that each example in $\Lambda_{\mathcal{N}}$ is a ball centre whereas each example in $\Lambda_{\mathcal{P}}$ can either be a ball border or a ball centre. Hence, to specify a classifier given $\Lambda$, we only need to specify the examples in $\Lambda_{\mathcal{P}}$ that are ball borders[4]. This specification can be used with a message string containing two parts. The first part specifies the number $b \in \{0,\ldots,d_p\}$ of ball borders in $\Lambda_{\mathcal{P}}$. The second part specifies which subset, among the set of $\binom{d_p}{b}$ possible subsets, is used for the set of ball borders. Consequently, if $b(\sigma)$ denotes the number of ball borders specified by message string $\sigma$, we can choose

$$P_\Lambda(\sigma) = \zeta(b(\sigma))\cdot\binom{d_p}{b(\sigma)}^{-1} \quad \text{(SCM case)}\,, \tag{3.8}$$

where, for any non-negative integer $b$, we define

$$\zeta(b) = \frac{6}{\pi^2}(b+1)^{-2}\,. \tag{3.9}$$

[4]For a SCM making no error with $\Lambda$, we can pair each centre with its border in the following way. For each negative centre, we choose the closest border. For each positive centre, we choose the furthest border.

Indeed, in this case, we clearly satisfy

$$\sum_{\sigma \in \mathcal{M}_\Lambda} P_\Lambda(\sigma) = \sum_{b=0}^{d_p} \zeta(b) \sum_{\sigma : b(\sigma)=b} \binom{d_p}{b(\sigma)}^{-1} \leq 1 .$$

The proposed loss bound will make use of the following functions:

$$\epsilon_{\mathcal{P}}(\mathbf{m}_{\mathcal{P}}, \beta) = 1 - \exp\left(-\frac{1}{m_p - d_p - k_p}\left[\ln\left(B_{\mathcal{P}}(\mathbf{m}_{\mathcal{P}})\right) + \ln\frac{1}{\beta}\right]\right) \tag{3.10}$$

$$\epsilon_{\mathcal{N}}(\mathbf{m}_{\mathcal{N}}, \beta) = 1 - \exp\left(-\frac{1}{m_n - d_n - k_n}\left[\ln\left(B_{\mathcal{N}}(\mathbf{m}_{\mathcal{N}})\right) + \ln\frac{1}{\beta}\right]\right) . \tag{3.11}$$

**Theorem 3.3.** *Given the above definitions, let $\mathcal{A}$ be any learning algorithm having a reconstruction function that maps compression sets and message strings to classifiers. For any prior distribution $P_\Lambda$ of messages and for any $\delta \in (0, 1]$:*

$$\Pr\left\{S \in X \,:\, \mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)] \leq \epsilon_{\mathcal{P}}\Big(\mathbf{m}_{\mathcal{P}}(S, \mathcal{A}(S)), g_{\mathcal{P}}(S)\delta\Big)\right\} \geq 1 - \delta$$

$$\Pr\left\{S \in X \,:\, \mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)] \leq \epsilon_{\mathcal{N}}\Big(\mathbf{m}_{\mathcal{N}}(S, \mathcal{A}(S)), g_{\mathcal{N}}(S)\delta\Big)\right\} \geq 1 - \delta ,$$

*where $\mathbf{m}_{\mathcal{P}}(S, \mathcal{A}(S))$ and $\mathbf{m}_{\mathcal{N}}(S, \mathcal{A}(S))$ are defined by Equation 3.4 and Equation 3.5, and*

$$g_{\mathcal{P}}(S) = \zeta(d_p(S)) \cdot \zeta(d_n(S)) \cdot \zeta(k_p(S)) \cdot P_{\Lambda(S)}(\sigma(S)) \tag{3.12}$$

$$g_{\mathcal{N}}(S) = \zeta(d_p(S)) \cdot \zeta(d_n(S)) \cdot \zeta(k_n(S)) \cdot P_{\Lambda(S)}(\sigma(S)) . \tag{3.13}$$

Note that Theorem 3.3 directly applies to the SCM when we use the distribution of messages given by Equation 3.8.

*Proof.* To prove Theorem 3.3, it suffices to upper bound by $\delta$ the following probability

$$\begin{aligned} P &= \Pr\left\{S \in X \,:\, \mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)] \geq \epsilon\Big(\mathbf{m}_{\mathcal{P}}(S, \mathcal{A}(S)), \Lambda(S), \sigma(S)\Big)\right\} \\ &= \sum_{\mathbf{m}_{\mathcal{P}}} \Pr\left\{S \in X \,:\, \mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)] \geq \epsilon\Big(\mathbf{m}_{\mathcal{P}}, \Lambda(S), \sigma(S)\Big), \mathbf{m}_{\mathcal{P}}(S, \mathcal{A}(S)) = \mathbf{m}_{\mathcal{P}}\right\} , \end{aligned}$$

where $\epsilon(\mathbf{m}_{\mathcal{P}}, \Lambda(S), \sigma(S))$ denotes a risk bound on $\mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)]$ that depends (partly) on the compression set $\Lambda(S)$ and the message string $\sigma(S)$ returned by $\mathcal{A}(S)$. The summation over $\mathbf{m}_{\mathcal{P}}$ stands for

$$\sum_{\mathbf{m}_{\mathcal{P}}}(\cdot) = \sum_{m_p=0}^{m} \sum_{d_p=0}^{m_p} \sum_{d_n=0}^{m-m_p} \sum_{k_p=0}^{m_p-d_p} (\cdot) .$$

Note that the summation over $k_p$ stops at $m_p - d_p$ because, as we will see later in the proof, we can upper bound the risk of a sample-compressed classifier only from the training errors it makes on the examples that are not used for the compression set.

We will now use the notation $\mathbf{i} = (i_1, \ldots, i_d)$ for a sequence (or a vector) of strictly increasing indices, $0 < i_1 < i_2 < \cdots < i_d \leq m$. Hence there are $2^m$ distinct sequences $\mathbf{i}$ of any length $d$, including the empty sequence. We will also use $|\mathbf{i}|$ to denote the length $d$ of a sequence $\mathbf{i}$. Such sequences (or vectors) of indices will be used to identify subsets of $S$. For $S \in X$, we define $S_\mathbf{i}$ as

$$S_\mathbf{i} = ((x_{i_1}, y_{i_1}), \ldots, (x_{i_d}, y_{i_d})) \ .$$

Under the constraint that $\mathbf{m}(S, \mathcal{A}(S)) = \mathbf{m}$, we will denote by $\mathbf{i}_p$ any sequence (or vector) of indices where each index points to an example of $S_\mathcal{P}$. We also use an equivalent definition for $\mathbf{i}_n$. If, for example, $\mathbf{i}_n = (2,3,6,9)$, then $S_{\mathbf{i}_n}$ will denote the set of examples consisting of the second, third, sixth, and ninth $\mathcal{N}$-example of $S$. Therefore, given a training set $S$ and vectors $\mathbf{i}_p$ and $\mathbf{i}_n$, the subset $S_{\mathbf{i}_p,\mathbf{i}_n}$ will denote a compression set. We will also denote by $\mathcal{I}_{m_p}$ the set of all the $2^{m_p}$ possible vectors $\mathbf{i}_p$ under the constraint that $|S_\mathcal{P}| = m_p$. We also use an equivalent definition for $\mathcal{I}_{m_n}$. Using these definitions, we will now upper bound $P$ uniformly over all possible realizations of $\mathbf{i}_p$ and $\mathbf{i}_n$ under the constraint $\mathbf{m}_\mathcal{P}(S, \mathcal{A}(S)) = \mathbf{m}_\mathcal{P}$. Thus

$$\begin{aligned} P &\leq \sum_{\mathbf{m}_\mathcal{P}} \Pr\Big\{ S \in X \colon \exists \mathbf{i}_p \in \mathcal{I}_{m_p}, \exists \mathbf{i}_n \in \mathcal{I}_{m_n}, \exists \sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \\ &\qquad \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma\Big), \mathbf{m}_\mathcal{P}(S, \mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \Big\} \\ &\leq \sum_{\mathbf{m}_\mathcal{P}} \sum_{\mathbf{i}_p \in \mathcal{I}_{m_p}} \sum_{\mathbf{i}_n \in \mathcal{I}_{m_n}} \Pr\Big\{ S \in X \ : \ \exists \sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \\ &\qquad \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma\Big), \mathbf{m}_\mathcal{P}(S, \mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \Big\} \ , \end{aligned}$$

where $\Phi(S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma)$ denotes the classifier obtained once, $S, \mathbf{i}_p, \mathbf{i}_n$, and $\sigma$ have been fixed. The last inequality comes from the union bound over all the possible choices of $\mathbf{i}_p \in \mathcal{I}_{m_p}$ and $\mathbf{i}_n \in \mathcal{I}_{m_n}$. Let

$$P' = \Pr\Big\{ S \in X \colon \exists \sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n}, \sigma\Big), \mathbf{m}_\mathcal{P}(S, \mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \Big\} .$$

We now make explicit how the positive and negative examples are interleaved in the training sequence $S$ by introducing a new variable $\mathbf{b}$, which is a bit-string of length $m$ such that $S_i$ is a positive example if and only if $\mathbf{b}_i = 1$. Let $B_{m_p}$ denote the set of possible $\mathbf{b}$ vectors that we can have under the constraint that $|S_\mathcal{P}| = m_p$. We then have

$$\begin{aligned}
P' &= \sum_{\mathbf{b}\in B_{m_p}} \Pr\Big\{ S \in X \;:\; \exists\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
&\qquad \mathbf{m}_\mathcal{P}(S,\mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \mid \mathbf{b}(S) = \mathbf{b}\Big\} \Pr\Big\{ S \in X \;:\; \mathbf{b}(S) = \mathbf{b}\Big\} \\
&= \sum_{\mathbf{b}\in B_{m_p}} \Pr\Big\{ S \in X \;:\; \exists\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
&\qquad \mathbf{m}_\mathcal{P}(S,\mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \mid \mathbf{b}(S) = \mathbf{b}\Big\} p_\mathcal{P}^{m_p}(1-p_\mathcal{P})^{m-m_p}.
\end{aligned}$$

$$\begin{aligned}
P' &\leq \binom{m}{m_p} p_\mathcal{P}^{m_p}(1-p_\mathcal{P})^{m-m_p} \sup_{\mathbf{b}\in B_{m_p}} \Pr\Big\{ S \in X \;:\; \exists\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \\
&\qquad \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \mathbf{m}_\mathcal{P}(S,\mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \mid \mathbf{b}(S) = \mathbf{b}\Big\}
\end{aligned}$$

Under the condition $\mathbf{b}(S) = \mathbf{b}$, index vectors $\mathbf{i}_p$ and $\mathbf{i}_n$ are now pointing to specific examples in $S$. Consequently, under this condition, we can compute the above probability by first conditioning on the compression set $S_{\mathbf{i}_p,\mathbf{i}_n}$ and then performing the expectation over $S_{\mathbf{i}_p,\mathbf{i}_n}$. Hence

$$\Pr\Big\{ S \in X \colon (\cdot) \Big| \mathbf{b}(S) = \mathbf{b}\Big\} \;=\; \mathbb{E}_{S_{\mathbf{i}_p,\mathbf{i}_n}|\mathbf{b}} \Pr\Big\{ S \in X \colon (\cdot) \Big| \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\}.$$

By applying the union bound over $\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}}$, we obtain

$$\begin{aligned}
&\Pr\Big\{ S \in X \colon \exists\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}} : \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
&\qquad\qquad \mathbf{m}_\mathcal{P}(S,\mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \mid \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\} \\
&\quad \leq \sum_{\sigma\in\mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}}} \Pr\Big\{ S \in X \colon \mathrm{er}_\mathcal{P}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_\mathcal{P}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
&\qquad\qquad\qquad \mathbf{m}_\mathcal{P}(S,\mathcal{A}(S)) = \mathbf{m}_\mathcal{P} \mid \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\}
\end{aligned}$$

We will now stratify this last probability by the set of possible errors that classifier $\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)$ can perform on the training examples that are not in the compression set $S_{\mathbf{i}_p,\mathbf{i}_n}$. Note that we do not force here the learner to produce a classifier that does not make errors on $S_{\mathbf{i}_p,\mathbf{i}_n}$. However, the set of message strings needed by $\Phi$ to identify a classifier $h$ might be larger when $h$ can err on $S_{\mathbf{i}_p,\mathbf{i}_n}$. To perform this stratification, let $\hat{\mathbf{er}}(f, S_\mathcal{P})$ be the vector of indices pointing to the examples of $S_\mathcal{P}$ that are misclassified by $f$. Moreover, let $\mathcal{I}_{m_p}(\mathbf{i}_p)$ denote the set of all vectors $\mathbf{j}_p \in \mathcal{I}_{m_p}$ for which no index $i \in \mathbf{j}_p$ is also in $\mathbf{i}_p$. In other words, for all $\mathbf{i}_p \in \mathcal{I}_{m_p}$

and all $\mathbf{j}_p \in \mathcal{I}_{m_p}(\mathbf{i}_p)$, we have $\mathbf{j}_p \cap \mathbf{i}_p = \emptyset$. Therefore

$$\begin{aligned}
\Pr\Big\{ S \in X \colon \mathrm{er}_{\mathcal{P}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \mathbf{m}_{\mathcal{P}}(S,\mathcal{A}(S)) = \mathbf{m}_{\mathcal{P}} \mid \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\} \\
= \sum_{\mathbf{j}_p \in \mathcal{I}_{m_p}(\mathbf{i}_p)} \Pr\Big\{ S \in X \colon \mathrm{er}_{\mathcal{P}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
\hat{\mathbf{er}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma), S_{\mathcal{P}}] = \mathbf{j}_p, \mathbf{m}_{\mathcal{P}}(S, A(S)) = \mathbf{m}_{\mathcal{P}} \mid \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\} \\
= \sum_{\mathbf{j}_p \in \mathcal{I}_{m_p}(\mathbf{i}_p)} \Pr\Big\{ S \in X \colon \mathrm{er}_{\mathcal{P}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
\hat{\mathbf{er}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma), S_{\mathcal{P}}] = \mathbf{j}_p \mid \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\},
\end{aligned}$$

where the last equality comes from the fact that the condition $\mathbf{m}_{\mathcal{P}}(S,\mathcal{A}(S)) = \mathbf{m}_{\mathcal{P}}$ is obsolete when $\mathbf{b}(S) = \mathbf{b}$ with fixed vectors $\mathbf{i}_p, \mathbf{i}_n, \mathbf{j}_p$. Now, under the condition $\mathbf{b}(S) = \mathbf{b}$ with a fixed compression set $S_{\mathbf{i}_p,\mathbf{i}_n}$, this last probability is obtained for the random draws of the training examples that are not in $S_{\mathbf{i}_p,\mathbf{i}_n}$. Consequently, this last probability is at most equal to the probability that a fixed classifier, having $\mathrm{er}_{\mathcal{P}} \geq \epsilon(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)$, makes no errors on $m_p - d_p - k_p$ positive examples that are not in the compression set $S_{\mathbf{i}_p,\mathbf{i}_n}$. Note that the probability space created by the conditioning specifies only the positions of the positive examples but places no further restrictions on them. They can therefore be viewed as independent draws from the distribution of positive examples. This makes it possible to bound the probability of the event by the probability that $m_p - d_p - k_p$ independent draws are all correctly classified. Hence, we have

$$\begin{aligned}
\Pr\Big\{ S \in X \colon \mathrm{er}_{\mathcal{P}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)] \geq \epsilon\Big(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma\Big), \\
\hat{\mathbf{er}}[\Phi(S_{\mathbf{i}_p,\mathbf{i}_n},\sigma), S_{\mathcal{P}}] = \mathbf{j}_p \mid \mathbf{b}(S) = \mathbf{b}, S_{\mathbf{i}_p,\mathbf{i}_n}\Big\} \\
\leq \Big(1 - \epsilon(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)\Big)^{m_p - d_p - k_p}.
\end{aligned}$$

By regrouping the previous results, we get

$$\begin{aligned}
P &\leq \sum_{\mathbf{m}_{\mathcal{P}}} \binom{m}{m_p} p_{\mathcal{P}}^{m_p}(1-p_{\mathcal{P}})^{m-m_p} \sum_{\mathbf{i}_p \in \mathcal{I}_{m_p}} \sum_{\mathbf{i}_n \in \mathcal{I}_{m_n}} \\
&\quad \mathbb{E}_{S_{\mathbf{i}_p,\mathbf{i}_n}|\mathbf{b}} \sum_{\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}}} \sum_{\mathbf{j}_p \in \mathcal{I}_{m_p}(\mathbf{i}_p)} \Big(1 - \epsilon(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)\Big)^{m_p - d_p - k_p} \\
&= \sum_{m_p=0}^{m} \binom{m}{m_p} p_{\mathcal{P}}^{m_p}(1-p_{\mathcal{P}})^{m-m_p} \sum_{d_p=0}^{m_p} \binom{m_p}{d_p} \sum_{d_n=0}^{m-m_p} \binom{m_n}{d_n} \sum_{k_p=0}^{m_p-d_p} \binom{m_p - d_p}{k_p} \\
&\quad \mathbb{E}_{S_{\mathbf{i}_p,\mathbf{i}_n}|\mathbf{b}} \sum_{\sigma \in \mathcal{M}_{S_{\mathbf{i}_p,\mathbf{i}_n}}} \Big(1 - \epsilon(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)\Big)^{m_p - d_p - k_p}.
\end{aligned}$$

By using

$$\left(1-\epsilon(\mathbf{m}_{\mathcal{P}}, S_{\mathbf{i}_p,\mathbf{i}_n},\sigma)\right)^{m_p-d_p-k_p} = P_{S_{\mathbf{i}_p,\mathbf{i}_n}}(\sigma)\cdot\frac{1}{B_{\mathcal{P}}(\mathbf{m}_{\mathcal{P}})}\cdot\zeta(k_p)\cdot\zeta(d_n)\cdot\zeta(d_p)\cdot\delta\,,$$

we get $P \le \delta$ as desired. Similarly, we have

$$\Pr\left\{S\in X\ :\ \mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)] \ge \epsilon_{\mathcal{N}}\left(\mathbf{m}_{\mathcal{N}}(S,\mathcal{A}(S)), g_{\mathcal{N}}(S)\delta\right)\right\} \le \delta\,,$$

which completes the proof. □

**Remark 3.1.** *This Theorem can be viewed in a standard form by using the inequality* $1-\exp(-x)\le x$, *for* $x\ge 0$. *To see this, we simply need to substitute Equations 3.10 and 3.11 into each probability given in Theorem 3.3 and weaken them with the above inequality. Doing so yields the following bounds,*

$$\Pr\left\{S\in X\ :\ \mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)] \le \frac{1}{m_p-d_p-k_p}\left[\ln\left(B_{\mathcal{P}}(\mathbf{m}_{\mathcal{P}})\right)+\ln\frac{1}{\beta}\right]\right\} \ge 1-\delta,$$

$$\Pr\left\{S\in X\ :\ \mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)] \le \frac{1}{m_n-d_n-k_n}\left[\ln\left(B_{\mathcal{N}}(\mathbf{m}_{\mathcal{N}})\right)+\ln\frac{1}{\beta}\right]\right\} \ge 1-\delta\,.$$

*However, each probability is separately bounding the error on the positive and negative examples and so will not (in the final bound) hold with probability* $1-\delta$ *but with probability* $1-4\delta$ *(to be shown) as the expected loss will rely on four bounds simultaneously holding true (i.e., from Equation 3.1 we would like to upper bound* $\mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)]$, $\mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)]$, $p_{\mathcal{P}}$ *and* $p_{\mathcal{N}}$*).*

Now that we have a bound on both $\mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)]$ and $\mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)]$, to bound the expected loss $\mathcal{E}[\ell(\mathcal{A}(S))]$ of Equation 3.1 we now need to upper bound the probabilities $p_{\mathcal{P}}$ and $p_{\mathcal{N}}$. For this task, we could use a well-known approximation of the binomial tail such as the additive Hoeffding bound or the multiplicative Chernoff bound. However, the Hoeffding bound is known to be very loose when the probability of interest (here $p_{\mathcal{P}}$ and $p_{\mathcal{N}}$) is close to zero. Conversely, the multiplicative Chernoff bound is known to be loose when the probability of interest is close to 1/2. In order to obtain a tight loss bound for both balanced and imbalanced data sets, we have decided to use the binomial distribution without any approximation.

Recall that the probability $\mathrm{Bin}(m,k,p)$ of having at most $k$ successes among $m$ Bernoulli trials, each having probability of success $p$, is given by the binomial tail

$$\mathrm{Bin}(m,k,p) = \sum_{i=0}^{k}\binom{m}{i}p^i(1-p)^{m-i}\,.$$

Following Langford [2005], we now define the *binomial tail inversion* $\overline{\mathrm{Bin}}\,(m,k,\delta)$ as the largest value of probability of success such that we still have a probability of at least $\delta$ of observing at most $k$ successes out of $m$ Bernoulli trials. In other words,

$$\overline{\mathrm{Bin}}\,(m,k,\delta) \;=\; \sup\Big\{p : \mathrm{Bin}\,(m,k,p) \geq \delta\Big\}\,. \tag{3.14}$$

From this definition, it follows that $\overline{\mathrm{Bin}}\,(m,m_n,\delta)$ is the *smallest* upper bound on $p_{\mathcal{N}}$, which holds with probability at least $1-\delta$, over the random draws of $m$ examples.

$$\Pr\Big\{S \in X : p_{\mathcal{N}} \leq \overline{\mathrm{Bin}}\Big(m,m_n,\delta\Big)\Big\} \geq 1-\delta\,. \tag{3.15}$$

From this bound (applied to both $p_{\mathcal{P}}$ and $p_{\mathcal{N}}$), and from the previous Theorem, the following predicates hold simultaneously with probability $1-\delta$ over the random draws of $S$:

$$\begin{aligned}
\mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)] &\leq \epsilon_{\mathcal{P}}\Big(\mathbf{m}_{\mathcal{P}}, g_{\mathcal{P}}(S)\frac{\delta}{4}\Big)\\
\mathrm{er}_{\mathcal{N}}[\mathcal{A}(S)] &\leq \epsilon_{\mathcal{N}}\Big(\mathbf{m}_{\mathcal{N}}, g_{\mathcal{N}}(S)\frac{\delta}{4}\Big)\\
p_{\mathcal{N}} &\leq \overline{\mathrm{Bin}}\Big(m,m_n,\frac{\delta}{4}\Big)\\
p_{\mathcal{P}} &\leq \overline{\mathrm{Bin}}\Big(m,m_p,\frac{\delta}{4}\Big)
\end{aligned}$$

where $\mathbf{m}_{\mathcal{P}} = \mathbf{m}_{\mathcal{P}}(S,\mathcal{A}(S))$ and $\mathbf{m}_{\mathcal{N}} = \mathbf{m}_{\mathcal{N}}(S,\mathcal{A}(S))$. Consequently, we have the next theorem.

**Theorem 3.4.** *Given the above definitions, let $\mathcal{A}$ be any learning algorithm having a reconstruction function that maps compression sets and message strings to classifiers. With probability $1-\delta$ over the random draws of a training set $S$, we have*

$$\begin{aligned}
\mathcal{E}[\ell(\mathcal{A}(S))] \;\leq\; & l_{\mathcal{P}}\cdot\overline{\mathrm{Bin}}\Big(m,m_p,\frac{\delta}{4}\Big)\cdot\epsilon_{\mathcal{P}}\Big(\mathbf{m}_{\mathcal{P}}, g_{\mathcal{P}}(S)\frac{\delta}{4}\Big)\\
&+ l_{\mathcal{N}}\cdot\overline{\mathrm{Bin}}\Big(m,m_n,\frac{\delta}{4}\Big)\cdot\epsilon_{\mathcal{N}}\Big(\mathbf{m}_{\mathcal{N}}, g_{\mathcal{N}}(S)\frac{\delta}{4}\Big)\,,
\end{aligned}$$

*where $\mathbf{m}_{\mathcal{P}} = \mathbf{m}_{\mathcal{P}}(S,\mathcal{A}(S))$ and $\mathbf{m}_{\mathcal{N}} = \mathbf{m}_{\mathcal{N}}(S,\mathcal{A}(S))$ are defined by Equations 3.4 and 3.5.*

We can now improve the loss bound given by Theorem 3.4 in the following way. Consider the frequencies $\hat{p}_{\mathcal{P}} = m_p/m$ and $\hat{p}_{\mathcal{N}} = m_n/m$. Let us simply denote by $\epsilon_{\mathcal{P}}$ and $\epsilon_{\mathcal{N}}$ some upper bounds on $\mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)]$ and $\mathrm{er}_{\mathcal{P}}[\mathcal{A}(S)]$. Let us also denote by $\overline{p}_{\mathcal{P}}$ and $\overline{p}_{\mathcal{N}}$ some upper bounds on $p_{\mathcal{P}}$ and $p_{\mathcal{N}}$. Let us first assume that $l_{\mathcal{N}}\epsilon_{\mathcal{N}} \geq l_{\mathcal{P}}\epsilon_{\mathcal{P}}$. Then we have

$$\begin{aligned}
\mathcal{E}[\ell(\mathcal{A}(S))] &\leq p_{\mathcal{P}} l_{\mathcal{P}} \epsilon_{\mathcal{P}} + p_{\mathcal{N}} l_{\mathcal{N}} \epsilon_{\mathcal{N}} \\
&= l_{\mathcal{P}} \epsilon_{\mathcal{P}} + p_{\mathcal{N}} (l_{\mathcal{N}} \epsilon_{\mathcal{N}} - l_{\mathcal{P}} \epsilon_{\mathcal{P}}) \\
&\leq l_{\mathcal{P}} \epsilon_{\mathcal{P}} + \overline{p}_{\mathcal{N}} (l_{\mathcal{N}} \epsilon_{\mathcal{N}} - l_{\mathcal{P}} \epsilon_{\mathcal{P}}) \\
&= \hat{p}_{\mathcal{P}} l_{\mathcal{P}} \epsilon_{\mathcal{P}} + \hat{p}_{\mathcal{N}} l_{\mathcal{N}} \epsilon_{\mathcal{N}} + (\overline{p}_{\mathcal{N}} - \hat{p}_{\mathcal{N}})(l_{\mathcal{N}} \epsilon_{\mathcal{N}} - l_{\mathcal{P}} \epsilon_{\mathcal{P}}) .
\end{aligned}$$

Likewise, if $l_{\mathcal{P}} \epsilon_{\mathcal{P}} \geq l_{\mathcal{N}} \epsilon_{\mathcal{N}}$, we have

$$\mathcal{E}[\ell(\mathcal{A}(S))] \leq \hat{p}_{\mathcal{P}} l_{\mathcal{P}} \epsilon_{\mathcal{P}} + \hat{p}_{\mathcal{N}} l_{\mathcal{N}} \epsilon_{\mathcal{N}} + (\overline{p}_{\mathcal{P}} - \hat{p}_{\mathcal{P}})(l_{\mathcal{P}} \epsilon_{\mathcal{P}} - l_{\mathcal{N}} \epsilon_{\mathcal{N}}) .$$

Consequently, we have the following theorem.

**Theorem 3.5.** *Given the above definitions, let $\mathcal{A}$ be any learning algorithm having a reconstruction function that maps compression sets and message strings to classifiers. For any real numbers $a, b, c$, let*

$$\Psi(a; b; c) \stackrel{\text{def}}{=} \begin{cases} a \cdot |c| & \text{if } c \geq 0 \\ b \cdot |c| & \text{if } c \leq 0 . \end{cases}$$

*Then, with probability $1 - \delta$ over the random draws of a training set $S$, we have*

$$\begin{aligned}
\mathcal{E}[\ell(\mathcal{A}(S))] \leq\ & \frac{m_p}{m} \cdot l_{\mathcal{P}} \cdot \epsilon_{\mathcal{P}}\Big(\mathbf{m}_{\mathcal{P}}, g_{\mathcal{P}}(S)\frac{\delta}{4}\Big) + \frac{m_n}{m} \cdot l_{\mathcal{N}} \cdot \epsilon_{\mathcal{N}}\Big(\mathbf{m}_{\mathcal{N}}, g_{\mathcal{N}}(S)\frac{\delta}{4}\Big) \\
& + \Psi\Bigg(\overline{\mathrm{Bin}}\Big(m, m_p, \frac{\delta}{4}\Big) - \frac{m_p}{m} \,;\, \overline{\mathrm{Bin}}\Big(m, m_n, \frac{\delta}{4}\Big) - \frac{m_n}{m} \,; \\
& \qquad l_{\mathcal{P}} \epsilon_{\mathcal{P}}\Big(\mathbf{m}_{\mathcal{P}}, g_{\mathcal{P}}(S)\frac{\delta}{4}\Big) - l_{\mathcal{N}} \epsilon_{\mathcal{N}}\Big(\mathbf{m}_{\mathcal{N}}, g_{\mathcal{N}}(S)\frac{\delta}{4}\Big)\Bigg)
\end{aligned}$$

*where $\mathbf{m}_{\mathcal{P}} = \mathbf{m}_{\mathcal{P}}(S, \mathcal{A}(S))$ and $\mathbf{m}_{\mathcal{N}} = \mathbf{m}_{\mathcal{N}}(S, \mathcal{A}(S))$ are defined by Equations 3.4 and 3.5.*

To compare the bound given by Theorem 3.5 with the bound given by Theorem 3.4, let us assume that $l_{\mathcal{N}} \epsilon_{\mathcal{N}} \geq l_{\mathcal{P}} \epsilon_{\mathcal{P}}$. Using our shorthand notation, the bound of Theorem 3.5 is given by

$$l_{\mathcal{P}} \hat{p}_{\mathcal{P}} \epsilon_{\mathcal{P}} + l_{\mathcal{N}} \hat{p}_{\mathcal{N}} \epsilon_{\mathcal{N}} + (\overline{p}_{\mathcal{N}} - \hat{p}_{\mathcal{N}})(l_{\mathcal{N}} \epsilon_{\mathcal{N}} - l_{\mathcal{P}} \epsilon_{\mathcal{P}}) .$$

Whereas the bound of Theorem 3.4 is given by

$$l_{\mathcal{P}} \overline{p}_{\mathcal{P}} \epsilon_{\mathcal{P}} + l_{\mathcal{N}} \overline{p}_{\mathcal{N}} \epsilon_{\mathcal{N}} .$$

The bound of Theorem 3.4 minus the bound of Theorem 3.5 then gives

$$\begin{aligned}
&(l_{\mathcal{P}}\overline{p}_{\mathcal{P}}\epsilon_{\mathcal{P}} + l_{\mathcal{N}}\overline{p}_{\mathcal{N}}\epsilon_{\mathcal{N}}) - (l_{\mathcal{P}}\hat{p}_{\mathcal{P}}\epsilon_{\mathcal{P}} + l_{\mathcal{N}}\hat{p}_{\mathcal{N}}\epsilon_{\mathcal{N}} + (\overline{p}_{\mathcal{N}} - \hat{p}_{\mathcal{N}})(l_{\mathcal{N}}\epsilon_{\mathcal{N}} - l_{\mathcal{P}}\epsilon_{\mathcal{P}}))\\
&\quad = (\overline{p}_{\mathcal{P}} - \hat{p}_{\mathcal{P}})l_{\mathcal{P}}\epsilon_{\mathcal{P}} + (\overline{p}_{\mathcal{N}} - \hat{p}_{\mathcal{N}})l_{\mathcal{N}}\epsilon_{\mathcal{N}} - (\overline{p}_{\mathcal{N}} - \hat{p}_{\mathcal{N}})(l_{\mathcal{N}}\epsilon_{\mathcal{N}} - l_{\mathcal{P}}\epsilon_{\mathcal{P}})\\
&\quad = (\overline{p}_{\mathcal{P}} - \hat{p}_{\mathcal{P}} + \overline{p}_{\mathcal{N}} - \hat{p}_{\mathcal{N}})l_{\mathcal{P}}\epsilon_{\mathcal{P}}\\
&\quad = (\overline{p}_{\mathcal{P}} + \overline{p}_{\mathcal{N}} - 1)l_{\mathcal{P}}\epsilon_{\mathcal{P}}\,.
\end{aligned}$$

Since $l_{\mathcal{P}}\epsilon_{\mathcal{P}} > 0$ and $\overline{p}_{\mathcal{P}} + \overline{p}_{\mathcal{N}} > 1$, we have an improvement using Theorem 3.5.

**Example 3.1.** *If $l_{\mathcal{P}} = l_{\mathcal{N}} = 1, \delta = 0.05, m = 100, m_p = 40, m_n = 60, \epsilon_{\mathcal{P}} = 0.3, \epsilon_{\mathcal{N}} = 0.4$, we get 0.439 for the bound of Theorem 3.4 and only 0.371 for the bound of Theorem 3.5. Hence, the bound of Theorem 3.5 can be significantly better than the the bound of Theorem 3.4.*

## 3.4 Discussion and Numerical Comparisons with Other Bounds

Let us first discuss the bounds that we have proposed and make explicit some of the details and consequences. In general, risk bounds are simply upper bounds of the true error calculated from the (overall) error achieved during training. There is no distinction made between the positive and negative class. The results of the current chapter are bounds on the error achieved separately on the positive and negative examples, hence, making the distinction between the two classes explicit. Furthermore, the risk bound on one class depends on what the classifier achieves on the training examples of that class, thus, making the bound more data-dependent than the usual bounds on the true error. This strong data-dependence also allows the user to take into account the observed number of positive and negative examples in the training sample as well as the flexibility of specifying different losses for each class. This is known as asymmetric loss and is not possible with current sample-compression loss bounds.

Note also that the proposed bounds are data dependent bounds for which there are no corresponding lower bounds. A small compression scheme is evidence of simplicity in the structure of the classifier, but one that is related to the training distribution rather than a priori determined.

Any algorithm that uses a compression scheme can use the bounds that we have proposed and take advantage of asymmetrical loss and cases of imbalanced data sets. However, the tightness of the bound relies on the sparsity of the classifiers (*e.g.*, the size of the compression set). Hence, it may not be advantageous to use algorithms that do not possess levels of sparsity similar (or comparable) to the SCM. This is one reason why we will provide a numerical comparison of various sample-compression bounds for the case of the SCM.

In order to show the merits of our bound we must now compare numerically against more common sample compression bounds and the bound found to be incorrect. In doing so we point out when our bound can be smaller and when it can become larger. All the compared bounds are specialized to the set covering machine compression scheme that uses data-dependent balls. Here each ball is constructed from two data points—one that defines the centre of the ball and another that helps define the radius of the ball (known as the border point). Hence to build a classifier from the compression set, we also need an informative message string to discriminate between the border points and the centres.

Let us now discuss the experimental setup, including a list of all the bounds compared, and then conclude with a review of the results.

### 3.4.1 Setup

From Example 3.1 of Section 3.3, it is clear that using Theorem 3.5 is more advantageous than Theorem 3.4. Hence, all experiments will be conducted with the bound of Theorem 3.5. The first bound we compare against is taken from the original set covering machine paper by Marchand and Shawe-Taylor [2001] and is similar to the Littlestone and Warmuth [1986] bound but with more specialization for the SCM compression set defined from the set of data-dependent balls. The second generalisation error bound is adapted from Marchand and Sokolova [2005] and is a slight modification of the Marchand and Shawe-Taylor [2001] result. All these bounds will also be compared against the incorrect bound given in Marchand and Shawe-Taylor [2002].

Please note that traditional sample compression bounds, such as that given by Theorem 6.1 of Langford [2005], cannot be used with the set covering machine as it does *not* allow the inclusion of any side information in the reconstruction of the classifier. The SCM, however, stores both the centre and border points in order to construct its hypotheses. This implies the need for side information to discriminate between centres and border points, something that traditional sample compression bounds do not cater for. Therefore, we cannot give numerical comparisons against these types of bounds.

All generalisation error bounds detailed below will make use of the following definitions: $d_n = c_n$, $d_p = c_p + b$, $d = d_p + d_n$ and $k = k_p + k_n$. For completeness, we give the definitions of all risk bounds not already stated and, to avoid repetition, we only give references to the bounds described earlier.

- **new bound** (Theorem 3.5). When applied to the SCM, the new bound uses the distribution of messages given by Equation 3.8 and Equations 3.9, 3.10, 3.11, 3.12, 3.13, and 3.14.

- **incorrect bound** (Theorem 5 of Marchand and Shawe-Taylor [2002]). This bound can also be found in Section 3.2 of the current chapter.

- **MS01 bound** (Theorem 5.2 of Marchand and Shawe-Taylor [2001]). This bound is stated in Theorem 3.1.

- **MS05 bound** (Equation 10 of Marchand and Sokolova [2005]). This bound is stated in Theorem 3.2.

### 3.4.2 Discussion of results

The numerical comparisons of these four bounds (*new bound, incorrect bound, MS01 bound and MS05 bound*) are shown in Figure 3.1 and Figure 3.2. Each plot contains the number of positive examples $m_p$, the number of negative examples $m_n$, the number of positive centres $c_p$, the number of negative centres $c_n$ and the number of borders $b$. The number of negative misclassifications $k_n$ was fixed for all plots and these values can be found in the x-axis label (either 0 or 500). The number of positive examples was varied and its quantity was set to those values given by the x-axis of the plot. For example, in the left hand side plot of Figure 3.1, the number of negative misclassifications $k_n$ was 0 and the number of positive misclassifications $k_p$ varied from 1 to 2000. The y-axis give the bound values achieved. Finally, the empirical error was also included in each plot—which is simply the number of examples misclassified divided by the number of examples, *i.e.*, $(k_p + k_n)/(m_p + m_n)$.

Figure 3.1 shows the case where the number of positive and negative examples is approximately the same. We clearly see that the incorrect bound becomes erroneous when the number $k_p$ of errors on the positive training examples approaches the total number $m_p$ of positive training examples. We also see that the new bound is tighter than the MS01 and MS05 bounds when the $k_p$ differs greatly from $k_n$. However, the latter bound is slightly tighter than the new bound when $k_p = k_n$.

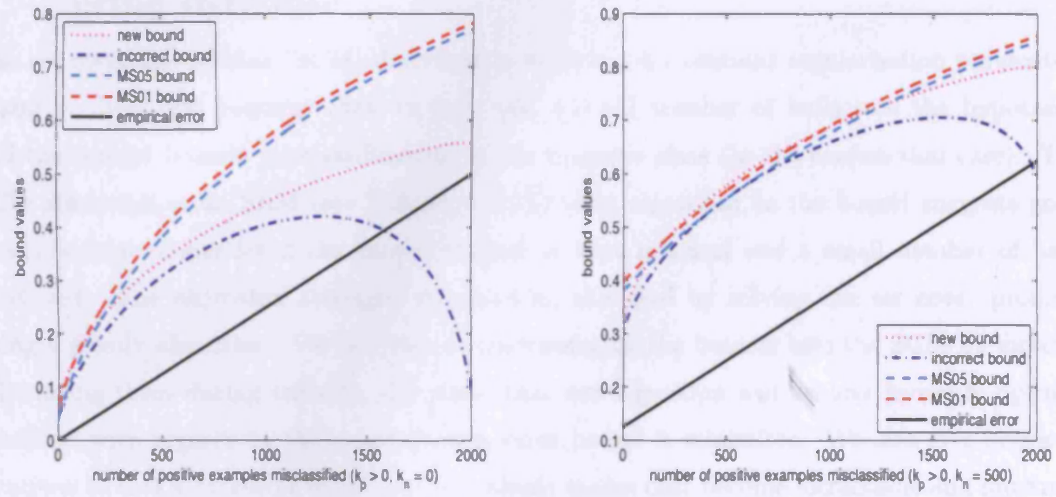

Figure 3.1: Bound values for the SCM when $m_p = 2020, m_n = 1980, c_p = 5, c_n = 5, b = 10$.

Figure 3.2 depicts the case where there is an imbalance in the data set ($m_n \gg m_p$), implying greater possibility of imbalance in misclassifications. However, the behavior is similar as the one found in Figure 3.1. Indeed, the MS01 and MS05 loss bounds are slightly smaller than the new bound when $k_p/m_p$ is similar to $k_n/m_n$, but the new bound becomes smaller when these two quantities greatly differ. This is where the new bound is most advantageous—in the case when there is an imbalance in misclassifications. As we would expect, the new bound is smaller when one class of examples is more abundant than the other.

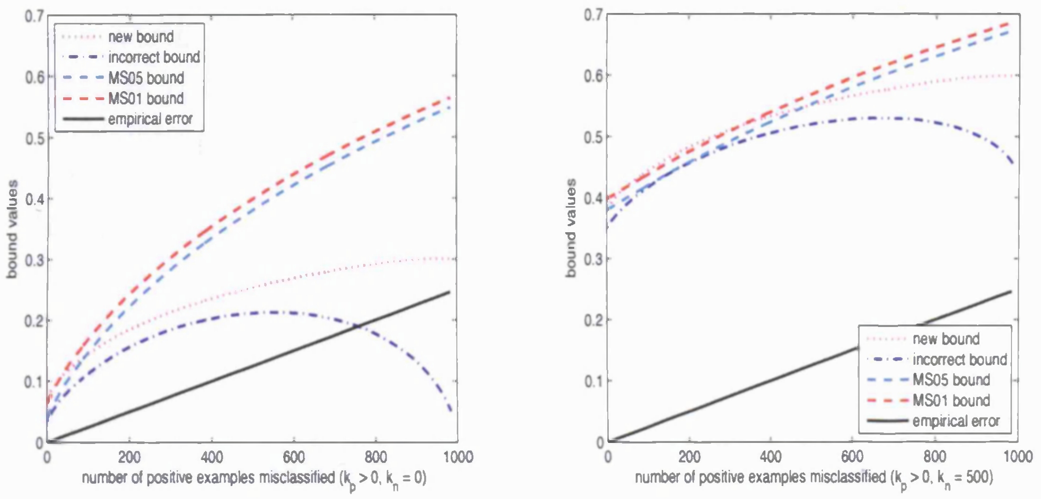

Figure 3.2: Bound values for the SCM when $m_p = 1000, m_n = 3000, c_p = 5, c_n = 5, b = 10$.

Now that we have described tight sample compression bounds for the SCM we can look to minimise the bounds in practice and integrate them more closely into the workings of the algorithm.

## 3.5 Using generalisation error bounds to train the set covering machine

The set covering machine (SCM) described in Section 2.3.1 contains regularisation parameters $s$ and $p$. The first requires that we only add a small number of balls into the hypothesis and the second bounds misclassifications of the negative class (in the conjunction case). The SCM algorithm is an SRM (see Definition 2.11) style algorithm as the bound suggests good generalisation properties if the empirical error is kept minimal and a small number of balls produced. The algorithm attempts at achieving this goal by solving the set cover problem using a greedy algorithm. We now aim at incorporating the bounds into the SCM by directly minimising them during training. We show that our algorithm will achieve provably optimal solutions with regards to the generalisation error bound it minimises. We also give heuristic solutions to tackle problems with large hypothesis spaces that become intractable and illustrate the efficacy of the approaches by conducting experiments on real world data sets.

Interestingly, note that all the bounds presented so far in this chapter are non-trivial (*i.e.* always less than 1) and are expected to be small whenever the SCM builds a classifier consisting of a small number of balls. Also Marchand and Shawe-Taylor [2002] showed that model selection using the loss bound was competitive against traditional cross-validation model selection techniques. Exploiting this fact we apply the generalisation error bounds directly to obtain classifiers for the SCM and, with it, remove the need for parameter estimation in the SCM. Because of its greedy application of the bound, we call this first heuristic the bound set covering machine (BSCM).

## 3.6 The bound set covering machine

In this variant of the SCM we allow the algorithm to be driven by one of the generalisation error bounds given by Theorem 3.1, Theorem 3.2 or Theorem 3.5. However, for simplicity and to save space we only describe the algorithm with Theorem 3.1, although Theorem 3.2 and Theorem 3.5 can also be applied using the same reasoning.

Recall from Theorem 3.1 that the generalisation error $\text{er}(f)$ for the classifier $f$ found by the SCM can be upper bounded by the quantity $\epsilon(f) = \epsilon(m, d, c_p, k_p, k_n, \delta)$, where $d, c_p, k_p$ and $k_n$ are computed from the set of balls in the hypothesis $f$ and $m$ and $\delta$ are fixed. Therefore, given any hypothesis $f = \{h_{i_1}, \ldots, h_{i_d}\}$ containing $d$ balls, we can calculate the risk bound of adding any new ball $h$ to $f$ as $\epsilon(f \cup \{h\}) = \epsilon(m, d+1, c_p + r, k_p + q, k_n - w, \delta)$ where $0 \leq r, q, w \in \mathbb{N}$. Therefore the bound set covering machine (BSCM) algorithm can be described as follows. Initially hypothesis $f \leftarrow \emptyset$, $N \leftarrow \mathcal{N}$, $P \leftarrow \emptyset$ and best bound $\epsilon^* \leftarrow 1$. At the first iteration, the BSCM algorithm looks for ball $h_i$ that minimises the generalisation error bound $\epsilon(f \cup \{h_i\})$ when added to hypothesis $f$. Ball $h_i$ that maximally minimises $\epsilon(f \cup \{h_i\})$ is added to the hypothesis $f \leftarrow f \cup \{h_i\}$, $N \leftarrow N \setminus \{\nu_N(h_i)\}$, $P \leftarrow P \cup \{\pi_{\mathcal{P} \setminus P}(h_i)\}$ and best bound $\epsilon^* \leftarrow \epsilon(f \cup \{h_i\})$. This is repeated until no new loss bound $\epsilon(f \cup \{h\})$, for any new ball $h$ can be found such that $\epsilon(f \cup \{h\}) < \epsilon^*$. After this the resulting hypothesis $f$ can be used to classify new test examples using (2.2). See Algorithm 2 for details.

**Algorithm 2**: The bound set covering machine (BSCM)

```
Input: empty hypothesis $f \leftarrow \emptyset$, $N \leftarrow \mathcal{N}$, $P \leftarrow \emptyset$ and error bound parameter $\epsilon^* \leftarrow 1$.
1: found ← true
2: while found do
3:   Call bscm-addball($f, \epsilon^*, N, P$, found)
4: end while
Output: A conjunction or disjunction of balls $f$.
```

**Function** bscm-addball($f$,$\epsilon^*$,$N$,$P$,found)

```
1: Construct the set of data-dependent balls $\mathcal{B} = \{h_1, \ldots, h_{|\mathcal{B}|}\}$ from the examples in
   $N \cup (\mathcal{P} \setminus P)$
2: found ← false
3: for $i = 1, \ldots, |\mathcal{B}|$ do
4:   if $\epsilon(f \cup \{h_i\}) < \epsilon^*$ then
5:     $i^* \leftarrow i$
6:     $\epsilon^* \leftarrow \epsilon(f \cup \{h_i\})$
7:     found ← true
8:   end if
9: end for
10: if found then
11:   update $f \leftarrow f \cup \{h_i\}$, $N \leftarrow N \setminus \{\nu_N(h_i)\}$ and $P \leftarrow P \cup \{\pi_{\mathcal{P} \setminus P}(h_i)\}$
12: end if
```

The BSCM does not involve the soft stopping parameter $s$, as was the case for the SCM, since if the addition of a new ball in the current hypothesis causes the bound to become worse, then the algorithm is terminated. This is a better and more theoretically motivated stopping criterion than stopping the algorithm early. Also, the BSCM no longer requires the penalty parameter $p$ used to allow misclassifications on the set $\mathcal{N}$ because the function to be minimised is now the generalisation error bound and not the utility function given by equation (2.1). Hence, we have eradicated the need for *both* parameters $s$ and $p$ that were present in the SCM.

The BSCM algorithm is greedy in the same spirit as the SCM but with the difference that it minimises the loss of a hypothesis using the generalisation error bound rather than maximising the utility function for the addition of a single data-dependent ball. However it is well known that a greedy algorithm will not always deliver globally optimal solutions and so we now turn our attention to tackling this problem. By using a branch and bound approach we prove that our algorithm will return solutions that are globally optimal with respect to the loss bounds it minimises.

## 3.7 The branch and bound set covering machine

The branch and bound algorithm for solving combinatorial optimisation problems was first introduced in Land and Doig [1960]. The idea is to first partition the solution space into subproblems, and then to optimise individually over each in turn. This method implicitly enumerates all possibilities of a solution space and allows many solutions to be discarded[5] without explicitly looking at them. Clearly this is advantageous when the search space is particularly large as it only requires the algorithm to search a subset of the entire space. Many large instances of combinatorial problems have been solved to optimality using branch and bound algorithms. Therefore from the fact that we can enumerate the entire hypothesis space for the set covering machine and compute bounds for any new ball added to the current hypothesis, we can solve the set covering machine in this way.

We will use a branch and bound approach to enumerate all possible hypotheses that can be generated from the set of data-dependent balls. This is done by evaluating the bound every time a new ball can be added to the current hypothesis. If this bound is not smaller than the best bound currently found then there is no need to include it or its descendants, *i.e.*, balls that may be constructed from this hypothesis. Therefore, we can consider the set of hypotheses as a tree, where each subtree contains all the balls that can be constructed from the current set $N$ and $\mathcal{P} \setminus P$. Furthermore, pruning balls without explicitly visiting them at their depths of the tree can dramatically reduce the number of balls that must be visited and hence speed up the algorithm. As mentioned earlier, the motivation for a branch and bound strategy for solving the set covering machine is that if the function to be minimised is the generalisation error bound then we are guaranteed to find the hypothesis with the smallest generalisation error bound. Hence, if the estimate (upper bound) of the true error is a good approximation to the test error then we can be confident that the hypothesis produced will generalise well.

The algorithm works in the following way. We first define the notion of a best possible (bp) generalisation error bound as the computation of a bound that covers the remaining number

[5]Only solutions that will never achieve improved results over the current best solution are dismissed.

of negative examples and misclassifies no further positive examples (see below for an exact definition). Initially we compute all the hypotheses possible with a single ball and order them according to their generalisation error bounds. Next we look at the hypotheses, in turn, and their bp bounds and disregard further inspection of hypotheses if their bp generalisation error bounds are larger than the current smallest loss bound. If this is not the case then we compute a table of values that indicates whether we can achieve a smaller risk bound by sacrificing sparsity. These two pruning strategies allow a full enumeration of the entire hypothesis space. Therefore, termination of the algorithm implies that all hypotheses have been considered.

### 3.7.1 Algorithm

The algorithm relies on functions `addball` and `createtable`. We detail below each function and also include pseudocode. Before we describe the functions in detail we will give some notation. Let $S$ be a sample containing $m$ input-output pairs from $\mathcal{X} \times \{-1, 1\}$, $\mathcal{B}$ the set of data-dependent balls computed from $S$ and $T$ the SCM machine type, which can either be a *conjunction* or *disjunction* of balls. As earlier, let $\pi_{\mathcal{P} \setminus P}(h)$ be the set of examples from $\mathcal{P} \setminus P$ misclassified by $h$ and $\nu_N(h)$ be the set of examples from $N$ correctly classified by $h$.

For any hypothesis $f$ and any ball $h_i \notin f$ let $f_i = f \cup \{h_i\}$. From earlier definitions, the generalisation error bound of $f$ is given by $\epsilon(f)$ where,

$$\epsilon(f) = \epsilon(m, d, c_p, k_p, k_n, \delta),$$

also for the same hypothesis $f$ the best potential (bp) generalisation error bound $\eta(f)$ is given by,

$$\eta(f) = \epsilon(m, d + 1, c_p, k_p, 0, \delta).$$

The bp generalisation error bound $\eta(f)$ is the bound $\epsilon(f_i)$ if a single ball $h_i$ can be added to hypothesis $f$ such that *all* of the remaining $\mathcal{N}$ examples are covered and *none* of the remaining $\mathcal{P}$ examples misclassified. Contrast this to the generalisation error bound $\epsilon(f)$ which is simply the bound given for hypothesis $f$.

#### Algorithm BBSCM

The input of the algorithm is sample $S$ containing input-output pairs from the sets $\mathcal{P}$ and $\mathcal{N}$, machine type $T$ and the set of data-dependent balls $\mathcal{B}$.

The algorithm contains local variables $f$,$N$,$P$,$\epsilon^*$ and global variable $\ell$. Initially $f \leftarrow \emptyset$ is the empty hypothesis, $N$ is the set of $\mathcal{N}$ training examples still to be covered, $P$ is empty because no $\mathcal{P}$ examples have been misclassified, $\epsilon^*$ is the best generalisation error bound found so far for any hypothesis $f$ and initially set to 1. Global variable $\ell$ is set to the number of balls $|\mathcal{B}|$ possible for current $\mathcal{B}$.

Using the above inputs and variables, algorithm BBSCM calls recursive function `addball(`$f$`,`$\epsilon^*$`,`$N$`,`$P$`)` (see below).

Finally the output of algorithm BBSCM is a conjunction or disjunction of balls (classifier/hypothesis) $f^*$ that can be used for classification using equation (2.2).

```
Input  : $S$,$T$,$\mathcal{B}$
1: $f \leftarrow \emptyset$;
2: $N \leftarrow \mathcal{N}$;
3: $P \leftarrow \emptyset$;
4: $\epsilon^* \leftarrow 1$;
5: $\ell \leftarrow |\mathcal{B}|$;
Call   : addball($f$,$\epsilon^*$,$N$,$P$)
Output: A $T$ =‘conjunction’ or ‘disjunction’ of data-dependent balls $\mathcal{R}^* \subseteq \mathcal{B}$
```

**Algorithm 4**: BBSCM($S, T, \mathcal{B}$)

## Function addball

This function adds each possible ball $h_i$ to the current hypothesis so that $f_i = f \cup \{h_i\}$ and checks to see if its generalisation error bound $\epsilon_i$ is smaller than the best generalisation error bound $\epsilon^*$ found so far. If so, then the value of $f^*$ is replaced with $f_i$ and the best risk bound $f^*$ is replaced with $\epsilon_i$ (line 8). Also at this stage function `createtable` is called (line 9) to get `table` (see description of function `createtable` below).

```
Function addball($f$,$\epsilon^*$,$N$,$P$)
Data:
1: Consider all $h \in \mathcal{B} \setminus f$ ;
2: Order according to $\epsilon(\{h\} \cup f) \rightarrow (h_1, \epsilon_1, \eta_1), \ldots, (h_\ell, \epsilon_\ell, \eta_\ell)$ ;
3: for $i = 1, \ldots, \ell$ do
4:   $ftemp \leftarrow f$, $Ntemp \leftarrow N$, $Ptemp \leftarrow P$
5:   if $\eta_i < \epsilon^*$ then
6:     $f_i \leftarrow f \cup \{h_i\}$, $N \leftarrow N \setminus \{\nu_N(h_i)\}$, $P \leftarrow P \cup \{\pi_{\mathcal{P} \setminus P}(h_i)\}$
7:     if $\epsilon_i < \epsilon^*$ then
8:       $f^* \leftarrow f_i$, $\epsilon^* \leftarrow \epsilon_i$
9:       call createtable($\epsilon^*$, $m$)
10:    end if
11:    found $\leftarrow$ false
12:    d $\leftarrow |f_i|$
13:    A $\leftarrow -|N|$
14:    while $\neg$found do
15:      d $\leftarrow$ d + 1
16:      A $\leftarrow$ A + $|Ntemp| - |N|$
17:      kn $\leftarrow$ table(d, $|P|$)
18:      if kn $= -1$ or A $\geq -$kn then
19:        found $\leftarrow$ true
20:      end if
21:    end while
22:    if kn $\neq -1$ then
23:      call addball($f_i$,$\epsilon^*$,$N$,$P$)
24:    end if
25:  end if
26: end for
```

On line 17 if `table(d,|P|)` returns kn $= -1$ then this indicates that there is no bound for d and $|P|$ that is smaller than $\epsilon^*$. If `table(d,|P|)` is a positive integer kn, then there is a possibility of finding a ball to add to the current hypothesis that will give a smaller risk bound than $\epsilon^*$ provided there exists a set of d additional balls that leave no more than **kn** $\mathcal{N}$ examples uncovered and no additional $\mathcal{P}$ examples misclassified. If **kn** $\geq 0$, then line 18 checks whether a larger number of $\mathcal{N}$ examples can be covered using d balls (see Lemma 3.2 and equation (3.20) from proof of main theorem ). If so, then the procedure calls itself recursively (line 23) until all balls in $\mathcal{B}$ have been enumerated.

### Function `createtable`

Local variable `table` is an $m \times m$ matrix whose elements are all initially set to $-1$. Function `createtable` calculates for d balls and **kp** misclassifications (on the $\mathcal{P}$ examples) the number of $N$ examples that can be covered without creating a bound that is larger than the best bound $\epsilon^*$ found so far (line 14). This function returns `table`.

```
Function createtable($\epsilon^*$,$m$)
1: Initialize table ← −1 ;
2: $c_p$ ← 0;
3: kpfound ← true
4: kp ← −1
5: while kpfound do
6:    kpfound ← false
7:    kp ← kp + 1
8:    dfound ← true
9:    d ← 0
10:   while dfound do
11:      dfound ← false
12:      d ← d + 1
13:      kn ← 0
14:      while $\epsilon(m, d, c_p, kp, kn, \delta) < \epsilon^*$ do
15:         kn ← kn + 1
16:      end while
17:      if kn > 0 then
18:         kpfound ← true
19:         dfound ← true
20:      end if
21:      table(d, kp) ← kn − 1
22:   end while
23: end while
```

## 3.8 BBSCM($\tau$)

BBSCM($\tau$) allows a trade-off between the accuracy of the classifier and speed. In function `createtable`($\epsilon^*, m$) the **while** condition computes $\epsilon(m, d, c_p, kp, kn, \delta) < \epsilon^*$. In the BBSCM($\tau$) heuristic this **while** condition becomes $\epsilon(m, d, c_p, kp, kn, \delta) < \tau \cdot \epsilon^*$. Allowing the BBSCM algorithm to ignore solutions whose generalisation error bounds are not a factor $\tau$ from the optimal found so far.

Clearly setting $\tau = 1$ returns the BBSCM algorithm – however as mentioned above this algorithm is too slow for data sets with $m > 100$ training examples. Therefore we would like to set $\tau < 1$. Setting $\tau$ close to 1 may cause the heuristic to be too slow but create hypotheses that have small generalisation error bounds similar to those for $\tau = 1$. Setting $\tau$ close to 0 will speed up the solution but may not create a large enough search space in order for BBSCM($\tau$) to find hypotheses with relatively small generalisation error bounds. Hence, setting $\tau$ is unlike setting a regularisation parameter since it is trading accuracy against time - bigger $\tau$ is always better in terms of generalisation error bounds, but costs more in terms of computational time.

## 3.9 Theory

Now that we have described the BBSCM algorithm we are in a position to describe its theory. The two pruning steps make use of the generalisation error bounds and their properties. The first theorem is a direct consequence of the first pruning step from function addball (line 5). The second theorem includes the ideas behind the second pruning step found in function addball (line 11-21) and relies on lemma 3.2 and 3.3 to help justify its claim.

Before looking at the first theorem we would like to present the following lemma that allows the theorem to be proved. Note that all the results here are given for Theorem 3.1 although as mentioned earlier they can also be obtained with Theorem 3.2 and 3.5.

**Lemma 3.1.** *The generalisation error bound given by equation (3.1) (Theorem 3.1) is monotonically increasing in the second parameter if* $m \geq 4d + 2 - c_p + k_p + k_n$.

*Proof.* We would like to show for varying $d$ with fixed $m > 0$, $c_p \leq d$, $k_p, k_n \geq 0$, $0 \leq \delta < 1$, that $\epsilon(m, d+1, c_p, k_p, k_n, \delta) \geq \epsilon(m, d, c_p, k_p, k_n, \delta)$. Writing out the bounds in full we get

$$\begin{aligned}\epsilon(m, d+1, c_p, k_p, k_n, \delta) \;=\;& 1 - \exp\left\{ - \overbrace{\frac{1}{m - 2(d+1) - k_p - k_n}}^{(i)} \left( \overbrace{\ln \binom{m}{2(d+1)}}^{(ii)} + \overbrace{\ln \binom{2(d+1)}{c_p}}^{(iii)} \right.\right. \\ & \left.\left. + \underbrace{\ln \binom{m - 2(d+1)}{k_p + k_n}}_{(iv)} + \underbrace{\ln \frac{2m^2(d+1)}{\delta}}_{(v)} \right) \right\}\end{aligned} \tag{3.16}$$

and

$$\begin{aligned}\epsilon(m, d, c_p, k_p, k_n, \delta) \;=\;& 1 - \exp\left\{ - \overbrace{\frac{1}{m - 2d - k_p - k_n}}^{(i')} \left( \overbrace{\ln \binom{m}{2d}}^{(ii')} + \overbrace{\ln \binom{2d}{c_p}}^{(iii')} + \overbrace{\ln \binom{m-2d}{k_p + k_n}}^{(iv')} \right.\right. \\ & \left.\left. + \underbrace{\ln \frac{2m^2 d}{\delta}}_{(v')} \right) \right\}.\end{aligned} \tag{3.17}$$

We know that the function $f(x) = 1 - \exp\{-x\}$ is a monotonically increasing function because its first derivative $f'(x) = \exp\{-x\} > 0$.

Looking at only the binomial coefficients, let $\ln(z(d+1)) = (ii) + (iii) + (iv)$ and let $\ln(z(d)) = (ii') + (iii') + (iv')$. Subtracting we get $\ln\left(\frac{z(d+1)}{z(d)}\right)$. Ignoring the ln, we would like to show:

$$\frac{z(d+1)}{z(d)} = \frac{\binom{m}{2(d+1)}\binom{2(d+1)}{c_p}\binom{m-2(d+1)}{k_p+k_n}}{\binom{m}{2d}\binom{2d}{c_p}\binom{m-2d}{k_p+k_n}} \geq 1. \tag{3.18}$$

Using the definition of a binomial coefficient $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ and simplifying, equation (3.18) becomes:

$$\frac{(m-2d-k_p-k_n)(m-2d-1-k_p-k_n)}{(2d+2-c_p)(2d+1-c_p)} \geq 1, \tag{3.19}$$

which is true if $m - 2d - k_p - k_n \geq 2d + 2 - c_p \implies m \geq 4d + 2 - c_p + k_p + k_n$. This holds for all the experiments we have conducted using the bounds. We also know that $(i) > (i')$ and $(v) > (v')$. Therefore it follows that (3.16) $\geq$ (3.17). □

**Theorem 3.6.** *Let $\epsilon^*$ be the smallest generalisation error bound currently found for hypothesis $f^*$ by the BBSCM algorithm. For any hypothesis $f_i$ that gives $\eta(f_i) > \epsilon^*$ there exists no extension $\hat{f} \supseteq f_i$ such that the generalisation error bound for $\hat{f}$ satisfies $\epsilon(\hat{f}) \leq \epsilon^*$.*

*Proof.* Let $\hat{f} = f_i \cup \{h_1, \ldots, h_n\}$ where $n$ is the number of balls added to hypothesis $f_i$. After $n$ balls the best generalisation error bound for $\hat{f}$ is $\epsilon(\hat{f})$ which is $\epsilon(m, d+n, c_p, k_p, 0, \delta)$. We know by Lemma 3.1 this bound is $\geq \epsilon(m, d+1, c_p, k_p, 0, \delta) = \eta(f_i)$ which from the statement in the Theorem is greater than $\epsilon^*$. Hence for any hypothesis $f_i$ with $\eta(f_i) > \epsilon^*$ there is no extension $\hat{f} \supseteq f_i$ such that $\epsilon(\hat{f}) \leq \epsilon^*$. □

This theorem states that for any hypothesis $f_i$ if the bp generalisation error bound $\eta_i$ is worse than the best bound $\epsilon^*$ then there is no need to try and cover any more $N$ examples from this ball as there will never be a smaller bound than the best $\epsilon^*$ found so far.

**Lemma 3.2.** *Let $U$ be a set covered by $A_1, \ldots, A_k$. For any $V \subseteq U$ $\exists\, j$ such that $\frac{|A_j \cap V|}{|V|} \geq \frac{1}{k}$.*

*Proof.* We know for $i = 1, \ldots, k$ that $|V| = |\cup (A_i \cap V)| \leq \sum_{i=1}^{k} |A_i \cap V|$ which implies $\sum_{i=1}^{k} \frac{|A_i \cap V|}{|V|} \geq 1$. Therefore by using the pigeon hole principle we know $\exists\, j$ such that $\frac{|A_j \cap V|}{|V|} \geq \frac{1}{k}$. □

**Lemma 3.3.** *Suppose* `createtable`$(\epsilon^*, m)$ *has been executed and* $\mathrm{kn} = \mathtt{table}(\mathrm{d}, \mathrm{kp}) \geq 0$. *It follows that* $\epsilon(m, \mathrm{d}, \mathrm{c_p}, \mathrm{kp}, \mathrm{kn}+1, \delta) > \epsilon^*$ *for* $\mathrm{c_p} \geq 0$.

*Proof.* In function `createtable`$(\epsilon^*, m)$ **while** the condition $\epsilon(m, \mathrm{d}, \mathrm{c_p}, \mathrm{kp}, \mathrm{kn}, \delta) < \epsilon^*$ holds we increment $\mathrm{kn} = \mathrm{kn} + 1$ for fixed values of $m > 0, \mathrm{d} > 0, \mathrm{c_p} = 0, \mathrm{kp} \geq 0$ and $\delta$. When this condition fails we have $\mathtt{table}(\mathrm{d}, \mathrm{kp}) = \mathrm{kn} - 1$ for $\mathrm{kn} \geq 0$. Therefore if $\mathrm{kn} = \mathtt{table}(\mathrm{d}, \mathrm{kp}) \geq 0$ for fixed d and kp then $\epsilon(m, \mathrm{d}, 0, \mathrm{kp}, \mathrm{kn}+1, \delta) > \epsilon^*$. □

Using Lemma 3.2 and 3.3 we can now prove that the BBSCM($\tau$) algorithm will only disregard a ball for inclusion if it cannot lead to a hypothesis with generalisation error bound smaller than that already found by BBSCM($\tau$).

**Theorem 3.7** (main theorem). *If algorithm BBSCM($\tau$) outputs hypothesis $f^*$ with generalisation error bound $\epsilon^*$ then there exists no hypothesis $\hat{f}$ such that $\epsilon(\hat{f}) < \tau \cdot \epsilon^*$.*

*Proof.* We will prove this result by contradiction. Assume $\hat{f}$ has the smallest generalisation error bound $\epsilon(\hat{f}) < \tau \cdot \epsilon^*$. Let $f = \{b_1, \ldots, b_k\}$ be the hypothesis returned by the BBSCM algorithm. In this setting we know that $f$ has generalisation error bound $\epsilon(f) = \epsilon^* > \frac{1}{\tau}\epsilon(\hat{f})$. Let $f^\dagger = \{b_1, \ldots, b_n\}$ be any maximal hypothesis in the search tree of the algorithm such that $f^\dagger \subset \hat{f}$. Let the number of $\mathcal{N}$ examples misclassified for $\hat{f}$ be given by $\hat{k}_n = k_n(\hat{f})$, similarly the number of $\mathcal{P}$ examples misclassified $\hat{k}_p = k_p(\hat{f})$, and the number of balls $\hat{d} = d(\hat{f})$. Also for $f^\dagger$ let the number of $\mathcal{P}$ examples misclassified be defined as $k_p^\dagger = k_p(\mathcal{B}^\dagger)$. Lemma 3.2 implies that there exists a ball $b_j \in \hat{f}, b_j \notin f^\dagger$ such that at node $f^\dagger$,

$$|\nu_N(b_j) \cap Ntemp| \geq \frac{|Ntemp| - \hat{k}_n}{\hat{d} - |f^\dagger| + 1}\,. \tag{3.20}$$

We claim that this implies $b_j$ was acceptable at node $f^\dagger$ and should have been added to hypothesis $f^\dagger$. However $b_j$ was not chosen by algorithm BBSCM to be added to $f^\dagger$, a contradiction.

To prove the claim assume that the last call to `createtable`$(\tau \cdot \tilde{\epsilon}, m)$ before $b_j$ was considered at node $f^\dagger$ was made with $\tau \cdot \tilde{\epsilon} \geq \tau \cdot \epsilon^*$. By lemma 3.3 we must have

$$\mathbf{kn} = \texttt{table}(\hat{d}, k_p^\dagger) \geq \hat{k}_n \tag{3.21}$$

since otherwise $\hat{k}_n \geq \mathbf{kn} + 1$ and so

$$\begin{aligned}
\epsilon(\hat{f}) &= \epsilon(m, \hat{d}, \hat{c}_p, \hat{k}_p, \hat{k}_n, \delta) \\
&\geq \epsilon(m, \hat{d}, \hat{c}_p, k_p^\dagger, \hat{k}_n, \delta) \\
&\geq \epsilon(m, \hat{d}, \hat{c}_p, k_p^\dagger, \mathbf{kn} + 1, \delta) \\
&\geq \epsilon(m, \hat{d}, 0, k_p^\dagger, \mathbf{kn} + 1, \delta) > \tau \cdot \tilde{\epsilon} \geq \tau \cdot \epsilon^*
\end{aligned}$$

contradicting $\epsilon(\hat{f}) < \tau \cdot \epsilon^*$. Substituting (3.21) into (3.20) we get

$$|\nu_N(b_j) \cap Ntemp|(\hat{d} - |f^\dagger| + 1) \geq |Ntemp| - \mathbf{kn}$$

setting $|\nu_N(b_j) \cap Ntemp| = |Ntemp| - |N|$ gives

$$\begin{aligned}
(|Ntemp| - |N|)(\hat{d} - |f^\dagger| + 1) - |Ntemp| &\geq - \mathbf{kn} \\
|Ntemp|(\hat{d} - |f^\dagger|) - |N|(\hat{d} - |f^\dagger| + 1) &\geq -\mathbf{kn}
\end{aligned}$$

Now we need to show that $\mathtt{A} = |Ntemp|(\hat{d} - |f^\dagger|) - |N|(\hat{d} - |f^\dagger| + 1)$ in function `addball`. Initially before the **while** loop in `addball`$(f^\dagger, \epsilon^*, N, P)$ we have $d = |f^\dagger|$ and $\mathtt{A} = -|N|$ and so $d + 1 = \hat{d}$

on the $\hat{d} - |f^\dagger|$ iteration implying

$$\begin{aligned} \mathtt{A} &= -|N| + (\hat{d} - |f^\dagger|)(|Ntemp| - |N|) \\ &= |Ntemp|(\hat{d} - |f^\dagger|) - |N|(\hat{d} - |f^\dagger| + 1) \end{aligned}$$

hence the test $\mathtt{A} \geq -\mathtt{kn}$ succeeds. Proving the claim. □

**Theorem 3.8.** *If algorithm BBSCM outputs a hypothesis $f^*$ then its generalisation error bound $\epsilon^*$ will be globally optimal (i.e. the smallest bound possible).*

*Proof.* Apply Theorem 3.7 with $\tau = 1$. □

From these theoretical results we have shown that the BBSCM is guaranteed to find the global optimal hypothesis $f^*$ with the smallest generalisation error bound $\epsilon^*$.

## 3.10 Experiments

Experiments were conducted on seven standard UCI repository data sets D.J. Newman and Merz [1998] described in table 3.1. All examples with contradictory labels and whose attributes contained unknown values were removed (this reduced considerably the Votes data set).

| Data Set | # of examples | | | # of |
|---|---|---|---|---|
| | pos | neg | total | features |
| BreastW | 239 | 444 | 683 | 9 |
| Votes | 18 | 34 | 52 | 16 |
| Pima | 269 | 499 | 768 | 8 |
| Haberman | 219 | 75 | 294 | 3 |
| Bupa | 145 | 200 | 345 | 6 |
| Glass | 87 | 76 | 163 | 9 |
| Credit | 296 | 357 | 653 | 15 |

Table 3.1: Description of data sets

Initially we test the BBSCM algorithm for the Votes data set as this was the only data set for which we could obtain results in a reasonable time (for BBSCM). We show that a full branch and bound search (full search) and the BBSCM yield the same hypothesis but that the BBSCM obtains it more quickly. The full search equates to removing line 9 and lines 11-21 from the `addball` function. This carries out a full exhaustive search of the hypothesis space. Similarly the BBSCM algorithm also carries out a full enumeration but with enumerating a smaller number of balls, as the pruning step removes a larger percentage of the search space.

We report these results in Table 3.2 – 3.7 where the first table is a full search and the remaining tables report the results for BBSCM for varying values of $\tau$ as $\tau$ goes from 1 down to 0.1. In each table we report the results for 10 fold cross-validation and give machine type $T$, which indicates a conjunction with a 'c' and a disjunction with a 'd'. The column 'thm' denotes the bound used, 'time' is the number of seconds needed for all folds, '# balls' is the average number of balls per fold, 'error %' is the overall error over the folds, 'std (error)' is the standard

| $T$ | thm | time (secs) | # balls | error % | std (error) | bound | std (bound) |
|---|---|---|---|---|---|---|---|
| c | 3.1 | 7.7 | 1.1 | 7 | 0.09 | 0.48 | 0.05 |
| d | 3.1 | 31.67 | 1.9 | 10.83 | 0.18 | 0.48 | 0.05 |
| c | 3.2 | 28.14 | 1.7 | 10.67 | 0.12 | 0.45 | 0.06 |
| d | 3.2 | 28.86 | 2.1 | 10.83 | 0.18 | 0.43 | 0.05 |
| c | 3.5 | 0.16 | 1 | 8.33 | 0.16 | 0.64 | 0.02 |
| d | 3.5 | 0.24 | 1 | 10.33 | 0.16 | 0.65 | 0.02 |

Table 3.2: 10 fold cross-validation on Votes data set using a full search.

deviation of the error for the 10 folds, 'bound' is the average bound value output for each of the classifiers and 'std (bound)' is the standard deviation of the bound values.

As we would expect, Table 3.2 and Table 3.3 are almost identical, with the difference being in the time column, which shows speed-ups for several of the Theorems and machines. Note that when the number of balls is close to 1 then there does not seem to be a large (or any) speed-up as the search space is pretty small because the branch and bound tree is shallow in depth. This is due to the fact that both algorithms will be searching the same paths of the branch and bound tree. The speed-ups are more prominent for the Theorems which produce more depth to the branch and bound tree. Meaning that the BBSCM is able to prune away some redundant subtrees, whereas a full search is required to visit them all. We see from the disjunction case of Theorem 3.1 and both machines of Theorem 3.2 that we can achieve almost two times the speed-up of an exhaustive search. Also, note that although the bound proposed in this chapter has a larger generalisation error it seems to produce classifiers with smaller test errors, when compared to Theorem 3.1 and Theorem 3.2 (except in the case of the conjunction machine of Theorem 3.1).

| $T$ | thm | time (secs) | # balls | error % | std (error) | bound | std (bound) |
|---|---|---|---|---|---|---|---|
| c | 3.1 | 7.08 | 1.1 | 7 | 0.09 | 0.48 | 0.05 |
| d | 3.1 | 17.16 | 1.9 | 10.83 | 0.18 | 0.48 | 0.05 |
| c | 3.2 | 16.06 | 1.7 | 10.67 | 0.12 | 0.45 | 0.06 |
| d | 3.2 | 15.47 | 2.1 | 10.83 | 0.18 | 0.43 | 0.05 |
| c | 3.5 | 0.16 | 1 | 8.33 | 0.16 | 0.64 | 0.02 |
| d | 3.5 | 0.24 | 1 | 10.33 | 0.16 | 0.65 | 0.02 |

Table 3.3: 10 fold cross-validation on Votes data set using BBSCM with $\tau = 1$.

We also report results for the BBSCM($\tau$) when $\tau = 0.8, 0.7, 0.6, 0.1$ in Tables 3.4 – 3.7. We do not report results for $\tau = 0.9$ because they were identical to $\tau = 1$, and also for $\tau = 0.5 - 0.2$ because they showed no difference from $\tau = 0.6$. We can see that as $\tau$ becomes smaller then the bound values become larger, because $\tau$ removes the smaller bounds from the search space. However, from all tables of varying $\tau$ we can see that Theorem 3.5 does not change. This is because only 1 ball is ever added meaning that the search space is small and the bound is too loose initially to allow any more balls to produce hypotheses that are smaller. Although as the speed increases ($\tau$ becomes smaller) we see that we still achieve the optimal solution (*i.e.*, for $\tau = 1$) for this particular generalisation error bound. The advantage of smaller $\tau$ is the increase in time complexity and the sparser solutions. We see from Table 3.4 and Table 3.5 that although the BBSCM with $\tau = 0.8, 0.7$ obtains the (overall) smallest test error for all data sets, it creates loss bounds that are greater than 0.5. This points to the fact that maybe the sample compression bounds of the set covering machine (SCM) tend to focus more towards minimising the empirical error and not so much towards the level of sparsity, *i.e.*, maybe the SRM principle is not being demonstrated accurately enough within the bounds. Clearly, the sparser solutions are generalising better (for the Votes data set) but the bounds are not reflecting this as well as perhaps they could, *i.e.*, the smallest bounds are not (always) corresponding to the smallest test errors. Finally, note for instance, the time given in Table 3.5, for the disjunction machine of Theorem 3.2 is almost four times faster than a full search (see Table 3.2) using the same bound and machine, resulting in a sparser solution and a smaller test error than the test error constructed by the smallest bound found using a full search.

| $T$ | thm | time (secs) | # balls | error % | std (error) | bound | std (bound) |
|---|---|---|---|---|---|---|---|
| c | 3.1 | 4.62 | 1 | 7 | 0.09 | 0.54 | 0.16 |
| d | 3.1 | 7.49 | 1.8 | 9.5 | 0.13 | 0.56 | 0.16 |
| c | 3.2 | 12.43 | 1.6 | 10.67 | 0.12 | 0.52 | 0.17 |
| d | 3.2 | 6.06 | 1.9 | 7.5 | 0.13 | 0.5 | 0.18 |
| c | 3.5 | 0.16 | 1 | 8.33 | 0.16 | 0.64 | 0.02 |
| d | 3.5 | 0.24 | 1 | 10.33 | 0.16 | 0.65 | 0.02 |

Table 3.4: 10 fold cross-validation on Votes data set using BBSCM with $\tau = 0.8$.

| $T$ | thm | time (secs) | # balls | error % | std (error) | bound | std (bound) |
|---|---|---|---|---|---|---|---|
| c | 3.1 | 4.46 | 1 | 7 | 0.09 | 0.54 | 0.16 |
| d | 3.1 | 5.03 | 1.8 | 9.5 | 0.13 | 0.56 | 0.15 |
| c | 3.2 | 11.6 | 1.6 | 10.67 | 0.12 | 0.52 | 0.17 |
| d | 3.2 | 5.64 | 1.9 | 7.5 | 0.13 | 0.5 | 0.18 |
| c | 3.5 | 0.16 | 1 | 8.33 | 0.16 | 0.64 | 0.02 |
| d | 3.5 | 0.24 | 1 | 10.33 | 0.16 | 0.65 | 0.02 |

Table 3.5: 10 fold cross-validation on Votes data set using BBSCM with $\tau = 0.7$.

| $T$ | thm | time (secs) | # balls | error % | std (error) | bound | std (bound) |
|---|---|---|---|---|---|---|---|
| c | 3.1 | 3.26 | 1 | 7 | 0.09 | 0.54 | 0.16 |
| d | 3.1 | 3.96 | 1.6 | 16.7 | 0.15 | 0.57 | 0.15 |
| c | 3.2 | 11.32 | 1.6 | 10.67 | 0.12 | 0.52 | 0.17 |
| d | 3.2 | 4.78 | 1.9 | 7.5 | 0.13 | 0.5 | 0.18 |
| c | 3.5 | 0.16 | 1 | 8.33 | 0.16 | 0.64 | 0.02 |
| d | 3.5 | 0.24 | 1 | 10.33 | 0.16 | 0.65 | 0.02 |

Table 3.6: 10 fold cross-validation on Votes data set using BBSCM with $\tau = 0.6$.

As mentioned earlier, the BBSCM was not practical for any of the data sets other than Votes. Therefore, we now turn our attention to the BSCM algorithm that is essentially a greedy algorithm that minimises the loss bound. First we give results for the SVM and SCM on all 7 data sets. For the SVM we used the Gaussian kernel and evaluated the $\gamma$ and $C$ parameters

| $T$ | thm | time (secs) | # balls | error % | std (error) | bound | std (bound) |
|---|---|---|---|---|---|---|---|
| c | 3.1 | 3.24 | 1 | 7 | 0.09 | 0.54 | 0.16 |
| d | 3.1 | 2.13 | 1 | 20.17 | 0.16 | 0.63 | 0.13 |
| c | 3.2 | 3.59 | 1 | 9 | 0.1 | 0.54 | 0.1 |
| d | 3.2 | 3 | 1 | 18.16 | 0.14 | 0.61 | 0.14 |
| c | 3.5 | 0.16 | 1 | 8.33 | 0.16 | 0.64 | 0.02 |
| d | 3.5 | 0.24 | 1 | 10.33 | 0.16 | 0.65 | 0.02 |

Table 3.7: 10 fold cross-validation on Votes data set using BBSCM with $\tau = 0.1$.

| Data Set | SVM | | | SCM | | | |
|---|---|---|---|---|---|---|---|
| | SVs | time | error % | $T$ | # balls | time | error % |
| Votes | 18.5 | 8.8 | 10.33 | c | 1 | 5.4 | 11.54 |
| Glass | 92.4 | 183.8 | 23.83 | c | 3.8 | 57.9 | 23.31 |
| Haberman | 136.4 | 5129.2 | 24.13 | d | 13 | 272.5 | 27.89 |
| Bupa | 199.4 | 3083.1 | 26.94 | d | 32.6 | 128 | 34.78 |
| Credit | 325.9 | 7415.1 | 25.45 | d | 3.7 | 831.2 | 31.7 |
| BreastW | 79.2 | 506.1 | 3.5 | c | 2 | 658.4 | 3.95 |
| Pima | 398.1 | 11071.9 | 24.6 | c | 5.9 | 1091.9 | 27.21 |

Table 3.8: SVM and SCM model-selection using 10 fold cross-validation.

using 10 fold cross-validation. We report in Table 3.8 the number of support vectors 'SVs' found by the SVM, the time taken over 10 fold cross-validation (including the parameter tuning stage), and the 'error'. For the SCM we used the $L_2$ norm to construct the set of data-dependent balls and report the best machine type $T$ equal to 'c' for a conjunction and 'd' for a disjunction that gave the smallest cross-validation error, the average number of balls found '# balls', the 'time' taken in seconds and the test 'error' found for cross validation. In Table 3.9 we report 10 fold cross validation results for the bound set covering machine (BSCM) algorithm. The table reports the results for all the machines and theorems presented so far. We see that compared to the results of the SCM and SVM in Table 3.8 the BSCM is much faster (as there are no parameters to tune) and also produces sparser solutions. Note that the results of the BSCM for the Votes data set are optimal (see BBSCM and full search Tables 3.2 and 3.3 above) with respect to the generalisation error bound of Theorem 3.5. We bold font the rows of Table 3.9 to indicate the classifiers that would be chosen, in a model selection strategy, by picking the hypothesis with the smallest generalisation error bound for a particular choice of machine type and loss bound. When compared to Table 3.8 we see that these classifiers do not obtain the smallest test errors compared to the SVM or SCM, except for Haberman and BreastW which are competitive. On the other hand we have made rows in Table 3.9 italic to signify those classifiers producing the smallest test error. In this scenario, the test error results for Votes, Haberman, BreastW and Pima are better than for the SCM and also competitive with the SVM. Note that the BSCM results are considerably sparser than the SVM and SCM and the running times are at least 10 times faster.

| data set | $T$ | thm | time (secs) | # balls | error % | std (error) | bound |
|---|---|---|---|---|---|---|---|
| Votes | c | 3.1 | 0.09 | 1 | 13.67 | 0.1732 | 0.4808 |
| Votes | d | 3.1 | 0.12 | 1 | 13.67 | 0.1732 | 0.4839 |
| Votes | c | 3.2 | 0.12 | 1.6 | 15.67 | 0.1671 | 0.4654 |
| **Votes** | **d** | 3.2 | **0.15** | **1** | **13.67** | **0.1732** | **0.4644** |
| *Votes* | *c* | 3.5 | *0.07* | *1* | *8.33* | *0.162* | *0.6354* |
| Votes | d | 3.5 | 0.12 | 1 | 10.33 | 0.1629 | 0.6498 |
| **Glass** | **c** | 3.1 | **1.03** | **2** | **25.68** | **0.1511** | **0.5299** |
| Glass | d | 3.1 | 1.65 | 1 | 26.38 | 0.1211 | 0.5658 |
| Glass | c | 3.2 | 2.17 | 2.2 | 27.01 | 0.1578 | 0.5339 |
| Glass | d | 3.2 | 1.82 | 1 | 26.38 | 0.1211 | 0.5707 |
| Glass | c | 3.5 | 1.07 | 1.2 | 36.41 | 0.1063 | 0.6331 |
| Glass | d | 3.5 | 0.81 | 1 | 40.53 | 0.0702 | 0.6381 |
| Haberman | c | 3.1 | 7.85 | 1 | 25.85 | 0.0772 | 0.5311 |
| Haberman | d | 3.1 | 3.4 | 1 | 26.2 | 0.0771 | 0.5327 |
| Haberman | c | 3.2 | 14.23 | 1.4 | 25.86 | 0.0772 | 0.5338 |
| Haberman | d | 3.2 | 4.9 | 1 | 26.2 | 0.0771 | 0.5355 |
| *Haberman* | *c* | 3.5 | *17.32* | *1* | *25.14* | *0.0251* | *0.3859* |
| **Haberman** | **d** | 3.5 | **5.75** | **1** | **25.83** | **0.0133** | **0.3804** |
| Bupa | c | 3.1 | 6.72 | 1.2 | 38.85 | 0.0795 | 0.6606 |
| Bupa | d | 3.1 | 14.15 | 3.4 | 38.55 | 0.1017 | 0.6503 |
| Bupa | c | 3.2 | 10.75 | 1.1 | 38.55 | 0.0795 | 0.6645 |
| Bupa | d | 3.2 | 27.12 | 3.6 | 38.28 | 0.1006 | 0.653 |
| **Bupa** | **c** | 3.5 | **11.2** | **1** | **39.73** | **0.0401** | **0.5369** |
| *Bupa* | *d* | 3.5 | *20.5* | *1* | *37.97* | *0.0393* | *0.538* |
| *Credit* | *c* | 3.1 | *24.94* | *1* | *32.47* | *0.0708* | *0.5991* |
| Credit | d | 3.1 | 40.44 | 2.5 | 32.63 | 0.0605 | 0.5991 |
| *Credit* | *c* | 3.2 | *42.22* | *1* | *32.47* | *0.0708* | *0.6009* |
| Credit | d | 3.2 | 69.63 | 2.3 | 32.63 | 0.0605 | 0.5977 |
| Credit | c | 3.5 | 45.71 | 1 | 42.27 | 0.0203 | 0.5293 |
| **Credit** | **d** | 3.5 | **66.35** | **1** | **41.96** | **0.0243** | **0.5292** |
| BreastW | c | 3.1 | 25.3 | 1.5 | 3.64 | 0.0207 | 0.1374 |
| BreastW | d | 3.1 | 36.63 | 1.1 | 3.65 | 0.0217 | 0.1396 |
| **BreastW** | **c** | 3.2 | **33.69** | **1.5** | **3.64** | **0.0207** | **0.1333** |
| BreastW | d | 3.2 | 26.9 | 1.1 | 3.65 | 0.0217 | 0.1351 |
| *BreastW* | *c* | 3.5 | *16.89* | *1* | *2.77* | *0.023* | *0.1653* |
| BreastW | d | 3.5 | 67.4 | 1 | 3.06 | 0.0259 | 0.1682 |
| *Pima* | *c* | 3.1 | *36.98* | *2.8* | *25.4* | *0.0445* | *0.5448* |
| Pima | d | 3.1 | 56.67 | 2 | 29.06 | 0.0556 | 0.5632 |
| Pima | c | 3.2 | 69.24 | 2.7 | 25.53 | 0.046 | 0.5472 |
| Pima | d | 3.2 | 101.58 | 2.2 | 29.32 | 0.0539 | 0.5649 |
| **Pima** | **c** | 3.5 | **50.4** | **1** | **33.47** | **0.0205** | **0.4189** |
| Pima | d | 3.5 | 110.53 | 1 | 33.07 | 0.0152 | 0.4206 |

Table 3.9: 10 fold cross-validation on several UCI repository data sets using BSCM.

## 3.11 Kernel matching pursuit

The kernel least squares regression algorithm described in Section 2.3.2 uses all of the training data (the full kernel matrix $\mathbf{K}$) in order to construct a prediction function of the form

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i),$$

where $\boldsymbol{\alpha}$ is the dual weight vector and $m$ is the number of examples observed in the training sample $S$. We now describe a sparse kernel least squares regression algorithm that only uses a small number of kernel functions to construct a prediction function of the form

$$f(\mathbf{x}) = \sum_{i=1}^{|\mathbf{i}|} \alpha_{\mathbf{i}_i} \kappa(\mathbf{x}, \mathbf{x}_{\mathbf{i}_i}), \qquad \text{(sparse)} \tag{3.22}$$

where $\mathbf{i} = (i_1, \ldots, i_d)$ is an index vector pointing to $d << m$ examples in $S$. We use the notation $\mathbf{i}_j$ to denote the $j$th element of $\mathbf{i}$. Also, recall that we denote a kernel basis vector using the notation $\mathbf{K}[:, j] = (\kappa(\mathbf{x}_1, \mathbf{x}_j), \ldots, \kappa(\mathbf{x}_m, \mathbf{x}_j))^\top$.

## 3.11.1 Algorithm

We can induce sparsity in the dual by defining the primal weight vector $\mathbf{w}$ from a small number of training examples such that $\mathbf{w} = \mathbf{X}[\mathbf{i}, :]^\top \tilde{\boldsymbol{\alpha}}$, where $|\tilde{\boldsymbol{\alpha}}| = |\mathbf{i}|$. Substituting this into Equation (2.3) from Chapter 2 we get

$$\min_{\tilde{\boldsymbol{\alpha}}} \|\mathbf{y} - \mathbf{X}\mathbf{X}[\mathbf{i}, :]^\top \tilde{\boldsymbol{\alpha}}\|^2,$$

as the sparse dual least squares minimisation problem. The sparse kernel $\mathbf{K}[:, \mathbf{i}] = \mathbf{X}\mathbf{X}[\mathbf{i}, :]^\top$ is the set of kernel basis vectors defined by the index set $\mathbf{i}$ and yields the sparse kernel least squares regression problem as,

$$\min_{\tilde{\boldsymbol{\alpha}}} \|\mathbf{y} - \mathbf{K}[:, \mathbf{i}]\tilde{\boldsymbol{\alpha}}\|^2. \tag{3.23}$$

If we differentiate with respect to $\tilde{\boldsymbol{\alpha}}$ and then set the resulting equation (known as the normal equations) to zero and solve for $\tilde{\boldsymbol{\alpha}}$ we retrieve the optimal solution of Equation (3.23) for

$$\tilde{\boldsymbol{\alpha}} = \left(\mathbf{K}[:, \mathbf{i}]^\top \mathbf{K}[:, \mathbf{i}]\right)^{-1} \mathbf{K}[:, \mathbf{i}]^\top \mathbf{y}. \tag{3.24}$$

We assume that $\left(\mathbf{K}[:, \mathbf{i}]^\top \mathbf{K}[:, \mathbf{i}]\right)$ is invertible as each basis vector added is orthogonal to all others, hence making the set of sparse bases linearly independent. We are now left with the problem of finding an appropriate set $\mathbf{i}$ of kernel basis vectors that minimises the loss of (3.23). A simple method to find $\mathbf{i}$ is the following greedy procedure. Let $\mathbf{j} = \{1, \ldots, m\}$ be the set of indices pointing to every training sample in $S$. Initially assign $\mathbf{i} = \emptyset$ to empty and add in turn an index $j \in \mathbf{j}$ to $\mathbf{i} = \mathbf{i} \cup \{j\}$ and evaluate expression (3.23). After this evaluation, update with the $i_1 = j$ that minimises (3.23) and repeat for $i_2, \ldots, i_d$ until $d$ basis vectors have been added to $\mathbf{i} = \{i_1, \ldots, i_d\}$. As simple as this algorithm is it is very inefficient as it requires the computation of a $d \times d$ matrix inverse for each trial and yields a time complexity of $\mathcal{O}\left(d^3 m\right)$ to add one basis. Another, more efficient method for solving this problem is by using kernel matching pursuit (KMP) [Vincent and Bengio, 2002], which also greedily constructs the set $\mathbf{i}$ but does not compute the inverse of a matrix in order to find the optimal $\tilde{\boldsymbol{\alpha}}$. This avoids the cubic time complexity of the greedy strategy discussed above. The pseudocode for the KMP algorithm is given in Algorithm 7.

**Algorithm 7**: Kernel matching pursuit with prefitting [Vincent and Bengio, 2002]

```
Input: Kernel $\mathbf{K}$, sparsity parameter $d > 0$.
1: initialise $\mathbf{A} = [\,]$ and $\mathbf{i} = [\,]$
2: for $i = 1$ to $d$ do
3:     Find improvement $= \left| \frac{(\mathbf{Ky})}{\sqrt{(\mathbf{K}^{\cdot 2})^\top \mathbf{1}}} \right|$
4:     Set $\mathbf{i}_i$ to the index of $max\{$improvement$\}$
5:     $\tilde{\alpha}_i = \left(\mathbf{K}[:, \mathbf{i}_i]^\top \mathbf{K}[:, \mathbf{i}_i]\right)^{-1} \mathbf{K}[:, \mathbf{i}_i]^\top \mathbf{y}$
6:     $(\tilde{\alpha}_1, \ldots, \tilde{\alpha}_{i-1})^\top = (\tilde{\alpha}_1, \ldots, \tilde{\alpha}_{i-1})^\top - \tilde{\alpha}_i \mathbf{A}[:, \mathbf{i}_i]$ (if $\mathbf{A}$ is non-empty)
7:     Set $\boldsymbol{\tau} = \mathbf{K}[:, \mathbf{i}_i]$ and $\mathbf{p} = \left(\frac{\boldsymbol{\tau}^\top \mathbf{K}}{\boldsymbol{\tau}^\top \boldsymbol{\tau}}\right)$, then update (if $\mathbf{A}$ is non-empty)
           $\mathbf{A} = \mathbf{A} - \mathbf{A}[:, \mathbf{i}_i]\mathbf{p}$
8:     Deflate kernel matrix
           $\mathbf{K} = \mathbf{K} - \mathbf{K}[:, \mathbf{i}_i]\mathbf{p} = \left(\mathbf{I} - \frac{\boldsymbol{\tau}\boldsymbol{\tau}^\top}{\boldsymbol{\tau}^\top \boldsymbol{\tau}}\right)\mathbf{K}$
9:     Add row vector $\mathbf{p}$ to $\mathbf{A}$ matrix
           $\mathbf{A} = \begin{pmatrix} \mathbf{A} \\ \mathbf{p}_1, \ldots, \mathbf{p}_m \end{pmatrix}$
10:    Set $\mathbf{K}[:, \mathbf{i}_i] = \mathbf{0}$
11: end for
Output: Index vector $\mathbf{i}$ and sparse dual weight vector $\tilde{\alpha}$
```

The algorithm works in the following way. Firstly we look for the largest improvement that can be made to Equation (3.23) with a single kernel basis vector. Hence we would like to minimise for all $i = 1, \ldots, m$ the following expression,

$$\begin{aligned} \|\mathbf{y} - \mathbf{K}[:, i]\tilde{\alpha}_i\|^2 &= \left\| \mathbf{y} - \mathbf{K}[:, i] \frac{\mathbf{K}[:, i]^\top \mathbf{y}}{\|\mathbf{K}[:, i]\|^2} \right\|^2 \\ &= \|\mathbf{y}\|^2 - 2\mathbf{K}[:, i]^\top \mathbf{y} \frac{\mathbf{K}[:, i]^\top \mathbf{y}}{\|\mathbf{K}[:, i]\|^2} + \left( \frac{\mathbf{K}[:, i]^\top \mathbf{y}}{\|\mathbf{K}[:, i]\|^2} \right)^2 \|\mathbf{K}[:, i]\|^2 \\ &= \|\mathbf{y}\|^2 - \left( \frac{\mathbf{K}[:, i]^\top \mathbf{y}}{\|\mathbf{K}[:, i]\|} \right)^2, \end{aligned}$$

where we have made the substitution of $\tilde{\alpha}_i$ using Equation 3.24 when $\mathbf{i} = \{i\}$. Therefore we would like to maximise the final component of the last line, like so,

$$\arg\max_i \left| \frac{\mathbf{K}[:, i]^\top \mathbf{y}}{\|\mathbf{K}[:, i]\|} \right| \qquad \forall i = 1, \ldots, m,$$

which corresponds to finding the kernel basis vector that is most collinear with the regression output vector $\mathbf{y}$. This computation is carried out in Line 3 of Algorithm 7. We use the notation

$\mathbf{X}.\mathbf{X} = \mathbf{X}^{\cdot 2}$ to denote element (component) wise multiplication of the matrices $\mathbf{X}$. Similar notation will hold for vectors. The division of vectors (matrices) will always be assumed component wise division.

After this we can update the last $\tilde{\alpha}_i$ using Equation (3.24). This leaves the remaining $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_{i-1}$ unchanged and so to reflect the new chosen basis vector $\mathbf{K}[:, i]$ we would need to update these values. This is carried out in Line 5 of Algorithm 7. Finally, the algorithm orthogonalises the kernel matrix such that the remaining points are chosen from the orthogonal subspace defined by the set $\mathbf{i}$, and that the indices in $\mathbf{i}$ can never be considered again. This step is carried out in Line 8 of Algorithm 7. For more information about the algorithm see [Vincent and Bengio, 2002].

### 3.11.2 A generalisation error bound for kernel matching pursuit

Kernel matching pursuit does not form a compression scheme (see definition 2.16) as it relies on all of the information in the training data in order to construct the regressor given by Equation (3.22) because $\tilde{\alpha}$ is computed using kernel columns $\mathbf{K}[:, j] = (\kappa(\mathbf{x}_1, \mathbf{x}_j), \ldots, \kappa(\mathbf{x}_m, \mathbf{x}_j))^\top$ which, by definition, use all of the $m$ training examples $\mathbf{x}_1, \ldots, \mathbf{x}_m$ present in $S$. Therefore, we cannot apply a sample compression bound similar to Theorem 2.4 in order to upper bound the loss of KMP. However, we can view the sparsity in dual space constructed by KMP for defining the feature space as a compression scheme and use a VC argument in order to upper bound KMP's future risk.

VC bounds as mentioned in subsection 2.4.2 have commonly been used to bound learning algorithms whose hypothesis spaces are infinite. The problem with these results is that the VC-dimension can sometimes be infinite even in cases where learning is successful. However, for KMP we can avoid this issues by making use of the fact that the VC-dimension of the set of linear threshold functions is simply the dimension of the function class. In KMP this directly translates into the number of basis vectors chosen and results in a standard VC argument.

The KMP algorithm (or any regression algorithm) can be upper bounded using a regression loss function or a classification loss function. The former requires the need for a pseudo-dimension and extra machinery not presented in this thesis. The latter method simply requires the use of the standard VC bounds for classification (see 2.4.2). Therefore we must first map the regression loss suffered by KMP into a corresponding classification loss in the following way.

**Definition 3.1.** *Let $S \sim \mathcal{D}$ be a regression training sample generated iid from a fixed but unknown probability distribution $\mathcal{D}$. Given the error* $\mathrm{er}(f) = \Pr_{S\sim\mathcal{D}}\{|f(\mathbf{x}) - y|\}$ *for a regression function $f$ between training example $\mathbf{x}$ and regression output $y$ we can define, for some fixed positive scalar $\alpha \in \mathbb{R}$, the corresponding true classification loss (error) as*

$$\mathrm{er}_\alpha(f) = \Pr_{(\mathbf{x},y)\sim\mathcal{D}} \left\{ |f(\mathbf{x}) - y| > \alpha \right\}.$$

*Similarly, we can define the corresponding empirical classification error as*

$$\begin{aligned}\hat{\mathrm{er}}_\alpha(f) = \mathrm{er}_\alpha(f, S) &= \Pr_{(\mathbf{x},y)\sim S}\{|f(\mathbf{x}) - y| > \alpha|\}\\ &= \mathbb{E}_{(\mathbf{x},y)\sim S}\{\mathbb{I}(|f(\mathbf{x}) - y| > \alpha)\},\end{aligned}$$

*where $\mathbb{I}$ is the indicator function and $S$ is supressed when clear from context.*

Now that we have a loss function that is discrete we can make a simple sample compression argument, that counts the number of possible subspaces, together with a traditional VC style bound given by Theorem 2.2 to upper bound the expected loss of KMP.

To help keep the notation consistent with earlier definitions we will denote the indices of the chosen basis vectors by $\mathbf{i}$. The indices of $\mathbf{i}$ are chosen from the training sample $S$ and we denote $S_\mathbf{i}$ to be those samples indexed by the vector $\mathbf{i}$. Given these definitions and Theorem 2.2 we can upper bound the true loss of KMP as follows.

**Theorem 3.9.** *Fix $\alpha \in \mathbb{R}$, $\alpha > 0$. Let $\mathcal{A}$ be the regression algorithm of KMP, $m$ the size of the training set $S$ and $d$ the size of the chosen basis vectors $\mathbf{i}$. Let $S$ be reordered so that the last $m - d$ points are outside of the set $\mathbf{i}$ and let $t = \sum_{i=m-d}^{m} \mathbb{I}(|f(\mathbf{x}_i) - y_i| > \alpha)$ be the number of errors for those points in $S \setminus S_\mathbf{i}$. Then with probability $1 - \delta$ over the generation of the training set $S$ the expected loss $\mathcal{E}[\ell(\cdot)]$ of algorithm $\mathcal{A}$ can be bounded by,*

$$\begin{aligned}\mathcal{E}[\ell(\mathcal{A}(S))] \le \frac{2}{m-d-t}\Bigg[&(d+1)\log\left(\frac{4e(m-d-t)}{d+1}\right) + d\log\left(\frac{em}{d}\right)\\ &+t\log\left(\frac{e(m-d)}{t}\right) + \log\left(\frac{2m^2}{\delta}\right)\Bigg].\end{aligned} \qquad (3.25)$$

*Proof.* First consider a fixed size $d$ for the compression set and number of errors $t$. Let $S_1 = \{\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_d}\}$ be the set of $d$ training points chosen by the KMP regressor, $S_2 = \{\mathbf{x}_{i_{d+1}}, \dots, \mathbf{x}_{i_{d+t}}\}$ the set of points erred on in training and $\bar{S} = S \setminus (S_1 \cup S_2)$ the points outside of the compression set ($S_1$) and training error set ($S_2$). Suppose that the first $d$ points form the compression set and the next $t$ are the errors of the KMP regressor. Since the remaining $m - d - t$ points $\bar{S}$ are drawn independently we can apply Theorem 2.2 to the $\mathrm{er}_\alpha$ loss to obtain the bound

$$\Pr\left\{\bar{S} : \hat{\mathrm{er}}_\alpha(f) = 0, \mathrm{er}_\alpha(f) > \epsilon\right\} \le 2\left(\frac{4e(m-d-t)}{d+1}\right)^{d+1} 2^{-\epsilon(m-d-t)/2},$$

where we have made use of a bound on the number of dichotomies that can be generated by parallel hyperplanes due to Anthony [2004], which is $\sum_{i=0}^{d+n-1}\binom{mn-1}{i}$ which is $\le \left(\frac{e(mn-1)}{d+n-1}\right)^{d+n-1}$, where $n$ is the number of parallel hyperplanes and equals 2 in our case. We now need to consider all of the ways that the $d$ basis vectors and $t$ error points might have occurred and apply the union bound over all of these possibilities. This gives the bound

$$\Pr\left\{S : \exists\, f \in \mathrm{span}\{S_1\} \text{ s.t. } \mathrm{er}_\alpha(f, S_2) = 1, \mathrm{er}_\alpha(f, \bar{S}) = 0, \mathrm{er}_\alpha(f) > \epsilon\right\}$$
$$\leq \binom{m}{d}\binom{m-d}{t} 2 \left(\frac{4e(m-d-t)}{d+1}\right)^{d+1} 2^{-\epsilon(m-d-t)/2}. \tag{3.26}$$

Finally we need to consider all possible choices of the values of $d$ and $t$. The number of these possibilities is clearly upper bounded by $m^2$. Setting $m^2$ times the rhs of (3.26) equal to $\delta$ and solving for $\epsilon$ gives the result. □

The bound is determined by the level of sparsity together with a VC-style argument making it the first such analysis of this type. It can and will be adapted for the sparse kernel canonical correlation analysis that we propose in Chapter 4. Note that this is the first upper bound on the generalisation error for KMP that we are aware of and as such we cannot compare the bound against any others.

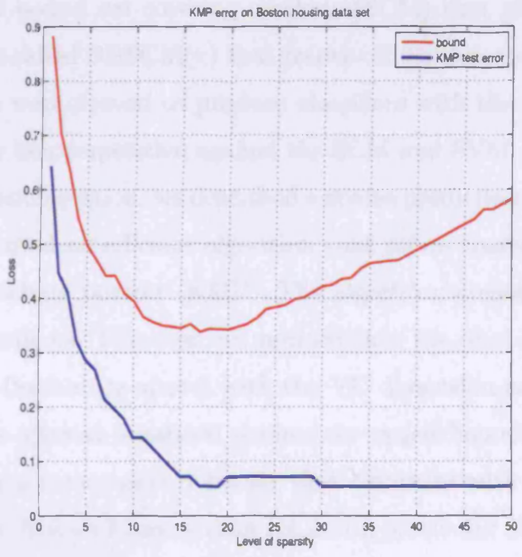

Figure 3.3: Plot of KMP bound against its test error. We used 450 examples for training and the 56 for testing. Bound was scaled down by a factor of 5.

Figure 3.3 plots the KMP test error against the loss bound given by Theorem 3.9. The bound value has been scaled by 5 in order to get the correct pictorial representation of the two plots. It is in actual fact trivial but as Figure 3.3 shows, its minima directly coincides with the lowest test error (after 17 basis vectors). This motivates a training algorithm for KMP that would use the bound as the minimisation criteria and stop once the bound fails to become smaller. This is a similar strategy as the one proposed for the set covering machine (SCM) in Section 3.5. This type of training regime can help make learning algorithms more automated as they need less human intervention for the model selection phase and also tie together machine learning theory and practice.

## 3.12 Summary

We have observed that the SCM loss bound proposed by Marchand and Shawe-Taylor [2002] is incorrect and, in fact, becomes erroneous in the limit where the number of errors on the positive training examples approaches the total number of positive training examples. We then proposed a new loss bound, valid for any sample compression learning algorithm (including the SCM), that depends on the observed fraction of positive examples and on what the classifier achieves on them. This new bound captures the spirit of Marchand and Shawe-Taylor [2002] with very similar tightness in the regimes in which the bound could hold. This is shown in numerical comparisons of the loss bound proposed in this chapter with all of the earlier bounds that can be applied to the SCM.

Using the bound proposed and two previously existing bounds for the SCM we applied them directly into the workings of the SCM. Initially we proposed the bound set covering machine (BSCM), which greedily minimises the loss bound and terminates after no risk bound can be found that is smaller, hence removing the need for regularisation parameters. Furthermore, we proposed the branch and bound set covering machine (SCM) that globally minimises the loss bound and gave a variant called BBSCM($\tau$) that trades-off time complexity against accuracy of solution. The algorithms were proved to produce classifiers with the smallest loss bounds and experimentally showed to be competitive against the SCM and SVM.

Finally, for the regression section, we described a sparse prediction function for least squares regression and then presented an efficient algorithm that solves sparse kernel least squares regression called kernel matching pursuit (KMP). The algorithm creates dual sparsity and does not form a compression scheme. However, we noticed that the dual sparsity could be viewed as a compression scheme (in feature space) with the VC dimension acting synonymously with the compression set. This allowed a natural method for upper bounding the loss of KMP (for classification loss) and in practice gave a bound that coincides with the minimum test error achieved by KMP (for the Boston housing data set and a particular choice of $\alpha$).

Chapter 4

# Sparsity in unsupervised learning

*Section 3.11 gave a sample compression analysis in the feature space of KMP. In this chapter, we attempt to make similar analyses for kernel principal components analysis (KPCA) and kernel canonical correlation analysis (KCCA). For KPCA we analyse a sparse variant that uses matching pursuit in order to construct sparse subspaces. We show that this algorithm does form a compression scheme (unlike KMP) and that its loss can be upper bounded using sample compression theory defined in Section 2.4.3 of Chapter 2. We test our proposed bound against the KPCA bound of Shawe-Taylor et al. [2005] and show that it is significantly tighter.*

*Furthermore, we propose a matching pursuit algorithm for sparse kernel canonical correlation analysis (SKCCA) and propose a generalisation error bound for SKCCA using a similar approach taken for KMP in Section 3.11. We show that the difference between the projections can be viewed as a regression problem and state the bound in the same form as the KMP bound. We test the bound for a real world data set and show that although the bound is trivial its minima coincides with the smallest test error. Finally we conclude the chapter with experimental results for document retrieval tasks that show little deterioration in test error but large improvements in time complexity when compared to KCCA.*

## 4.1 Sparse kernel principal components analysis

Recall from Chapter 2 that the kernel principal components analysis (KPCA) finds a small subspace of the training data in feature space using the entire kernel matrix. As we had done in the previous chapter for regression, we now turn our attention to computing KPCA with only a small subset of the kernel basis vectors. This is known as sparse kernel principal components analysis (SKPCA) and can be formulated by restricting the definition of the primal weight vectors to only using a small number of training examples.

### 4.1.1 Algorithm

In a sparse PCA algorithm we may want to find a sparsely represented vector $\mathbf{w}_x = \mathbf{X}[\mathbf{i},:]^\top \tilde{\boldsymbol{\alpha}}_x$, that is a linear combination of a small number of training examples indexed by vector $\mathbf{i}$. This corresponds to projections being made onto a sparse subspace maximising the variance of the data. Therefore by making the substitution $\mathbf{w}_x = \mathbf{X}[\mathbf{i},:]^\top \tilde{\boldsymbol{\alpha}}_x$ into Equation (2.4) we have the following sparse dual PCA maximisation problem,

$$\max_{\tilde{\boldsymbol{\alpha}}_x} \frac{\tilde{\boldsymbol{\alpha}}_x^\top \mathbf{X}[\mathbf{i},:]\mathbf{X}^\top \mathbf{X}\mathbf{X}[\mathbf{i},:]^\top \tilde{\boldsymbol{\alpha}}_x}{\tilde{\boldsymbol{\alpha}}_x^\top \mathbf{X}[\mathbf{i},:]\mathbf{X}[\mathbf{i},:]^\top \tilde{\boldsymbol{\alpha}}_x}, \tag{4.1}$$

which is equivalent to sparse kernel PCA (SKPCA) with $\mathbf{K}[:,\mathbf{i}] = \mathbf{X}\mathbf{X}[\mathbf{i},:]^\top$,

$$\max_{\tilde{\boldsymbol{\alpha}}_x} \frac{\tilde{\boldsymbol{\alpha}}_x^\top \mathbf{K}[:,\mathbf{i}]^\top \mathbf{K}[:,\mathbf{i}]\tilde{\boldsymbol{\alpha}}_x}{\tilde{\boldsymbol{\alpha}}_x^\top \mathbf{K}[\mathbf{i},\mathbf{i}]\tilde{\boldsymbol{\alpha}}_x}, \tag{4.2}$$

where $\tilde{\boldsymbol{\alpha}}_x$ is a sparse vector of length $d = |\mathbf{i}|$. Clearly maximising the quantity above will lead to the maximisation of the generalised eigenvalues corresponding to $\tilde{\boldsymbol{\alpha}}_x$ – and hence a sparse subset of the original PCA problem. This analysis assumes that $\mathbf{i}$ has already been chosen. We now consider how to choose the set of vectors indexed by $\mathbf{i}$.

The procedure involves choosing basis vectors that maximise the Rayleigh quotient without the set of eigenvectors, choosing basis vectors iteratively until some pre-specified number of $d$ vectors are chosen. An orthogonalisation of the kernel matrix at each step ensures future potential basis vectors will be orthogonal to those already chosen. This algorithm is equivalent to Smola and Schölkopf [2000]. The quotient to maximise is:

$$\max \rho_i = \frac{\mathbf{e}_i^\top \mathbf{K}^2 \mathbf{e}_i}{\mathbf{e}_i^\top \mathbf{K} \mathbf{e}_i}, \tag{4.3}$$

where $\mathbf{e}_i$ is the $i$th unit vector. Note, that this quotient looks similar to the KPCA formulation of Equation (2.7) but without the use of the eigenvectors $\tilde{\boldsymbol{\alpha}}_x$. Maximising Equation (4.3) will find the maximum variance in the feature space and not the projected space as is the case in traditional KPCA. This is fine to do in the first iteration of picking a basis vector but not for subsequent iterations. We need to orthogonalise (or deflate as it is more commonly referred to in the subspace method literature) the kernel matrix, creating a projection into the space orthogonal to the basis vectors chosen. This step ensures that we will find the maximum variance of the data in the projected space as is the case with KPCA.

The deflation step can be carried out as follows. Let $\boldsymbol{\tau} = \mathbf{K}[:, i] = \mathbf{X}\mathbf{X}^\top \mathbf{e}_i$ where $\mathbf{e}_i$ is the $i$th unit vector. We know that primal PCA deflation can be carried out with respect to the features in the following way:

$$\tilde{\mathbf{X}}^\top = \left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^\top}{\mathbf{u}^\top\mathbf{u}}\right)\mathbf{X}^\top,$$

where $\mathbf{u}$ is the projection directions defined by the chosen eigenvector and $\tilde{\mathbf{X}}$ is the deflated matrix. However, in sparse KPCA, $\mathbf{u} = \mathbf{X}^\top \mathbf{e}_i$ because the projection directions are simply the examples in $\mathbf{X}$. Therefore, for sparse KPCA we have:

$$\begin{aligned}
\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top &= \mathbf{X}\left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^\top}{\mathbf{u}^\top\mathbf{u}}\right)\left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^\top}{\mathbf{u}^\top\mathbf{u}}\right)\mathbf{X}^\top \\
&= \mathbf{X}\left(\mathbf{I} - \frac{\mathbf{u}\mathbf{u}^\top}{\mathbf{u}^\top\mathbf{u}}\right)\mathbf{X}^\top \\
&= \mathbf{X}\mathbf{X}^\top - \frac{\mathbf{X}\mathbf{X}^\top\mathbf{e}_i\mathbf{e}_i^\top\mathbf{X}\mathbf{X}^\top}{\mathbf{e}_i^\top\mathbf{X}\mathbf{X}^\top\mathbf{e}_i} \\
&= \mathbf{K} - \frac{\mathbf{K}[:, i]\mathbf{K}[:, i]^\top}{\mathbf{K}[i, i]}.
\end{aligned}$$

Therefore, given a kernel matrix $\mathbf{K}$ the deflated kernel matrix $\hat{\mathbf{K}}$ can be computed as follows:

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{\boldsymbol{\tau}\boldsymbol{\tau}^\top}{\mathbf{K}[\mathbf{i}_d, \mathbf{i}_d]} \tag{4.4}$$

where $\boldsymbol{\tau} = \mathbf{K}[:, \mathbf{i}_d]$ and $\mathbf{i}_d$ denotes the latest element in the vector $\mathbf{i}$.

**Algorithm 8**: A matching pursuit algorithm for sparse kernel principal components analysis

```
Input: Kernel $\mathbf{K}$, sparsity parameter $d > 0$.
1: initialise $\mathbf{i} = [\,]$
2: for $j = 1$ to $d$ do
3:     Find improvement = $\frac{(\mathbf{K}^{\cdot 2})^\top \mathbf{1}}{diag\{\mathbf{K}\}}$
4:     Set $\mathbf{i}_j$ to the index of $max\{$improvement$\}$
5:     set $\boldsymbol{\tau} = \mathbf{K}[:, \mathbf{i}_j]$
6:     deflate kernel matrix like so:
           $\mathbf{K} = \mathbf{K} - \frac{\boldsymbol{\tau}\boldsymbol{\tau}^\top}{\mathbf{K}[\mathbf{i}_j, \mathbf{i}_j]}$
7: end for
8: Compute approximation $\tilde{\mathbf{K}}$ using $\mathbf{i}$ and Equation (4.6)
Output: Output sparse matrix approximation $\tilde{\mathbf{K}}$
```

This algorithm is presented in Algorithm 8 and is equivalent to the algorithm proposed by Smola and Schölkopf [2000], as they iteratively add a basis vector by maximising the Rayleigh quotient and then deflate to guarantee the chosen vectors are orthogonal to all other vectors in the kernel matrix. Line 6 of Algorithm 8 is equivalent to the Smola and Schölkopf [2000] orthogonalisation procedure. However, their motivation comes from the stance of finding a low rank matrix approximation of the kernel matrix. They proceed by looking for an approximation $\tilde{\mathbf{K}} = \mathbf{K}[:, \mathbf{i}]T$ for a set $\mathbf{i}$ such that the Frobenius norm between the trace residuals $\mathrm{tr}\{\mathbf{K} - \mathbf{K}[:, \mathbf{i}]T\} = \mathrm{tr}\{\mathbf{K} - \tilde{\mathbf{K}}\}$ is minimal. Their algorithm finds the set of indices $\mathbf{i}$ and the projection matrix $T$. However, the use of $T$ in computing the low rank matrix approximation seems to imply the need for additional information from outside of the chosen basis vectors in order to construct this approximation. However, we show that a projection into the space defined solely by the chosen indices is enough to reconstruct the kernel matrix and does not require any extra information.[1] The projection is the well known Nyström method [Williams and Seeger, 2001].

An orthogonal projection $P_{\mathbf{i}}(\phi(\mathbf{x}_j))$ of a feature vector $\phi(\mathbf{x}_j)$ into a subspace defined only by the set of indices $\mathbf{i}$ can be expressed as:

$$P_{\mathbf{i}}(\mathbf{x}_j) = \tilde{\mathbf{X}}^\top(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top)^{-1}\tilde{\mathbf{X}}\phi(\mathbf{x}_j),$$

where $\tilde{\mathbf{X}} = \mathbf{X}[\mathbf{i}, :]$ are the $\mathbf{i}$ training examples from data matrix $\mathbf{X}$. It follows that,

$$\begin{aligned} P_{\mathbf{i}}(\mathbf{x}_j)^\top P_{\mathbf{i}}(\mathbf{x}_j) &= \phi(\mathbf{x}_j)^\top \tilde{\mathbf{X}}^\top(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top)^{-1}\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top)^{-1}\tilde{\mathbf{X}}\phi(\mathbf{x}_j) \\ &= \phi(\mathbf{x}_j)^\top \tilde{\mathbf{X}}^\top(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top)^{-1}\tilde{\mathbf{X}}\phi(\mathbf{x}_j) \\ &= \mathbf{K}[\mathbf{i}, j]\mathbf{K}[\mathbf{i}, \mathbf{i}]^{-1}\mathbf{K}[j, \mathbf{i}], \end{aligned} \tag{4.5}$$

with $\mathbf{K}[\mathbf{i}, j]$ denoting the kernel entries between the index set $\mathbf{i}$ and the feature vector $\phi(\mathbf{x}_j)$, giving us the following projection into the space defined by $\mathbf{i}$:

$$\tilde{\mathbf{K}} = \mathbf{K}[:, \mathbf{i}]\mathbf{K}[\mathbf{i}, \mathbf{i}]^{-1}\mathbf{K}[:, \mathbf{i}]^\top. \tag{4.6}$$

Hence, we simply need the chosen basis vectors $\mathbf{i}$ to make this reconstruction. This is very important for the theoretical analysis that we carry out in the next section, as it implies the following claim.

**Claim 4.1.** *The sparse kernel principal components analysis algorithm is a compression scheme.*

*Proof.* Given a data point $\phi(\mathbf{x}_j)$ in feature space and a set of chosen indices $\mathbf{i}$ we can reconstruct the projection using Equation (4.5), *i.e.*, $\mathbf{K}[\mathbf{i}, j]\mathbf{K}[\mathbf{i}, \mathbf{i}]^{-1}\mathbf{K}[j, \mathbf{i}]$. Therefore, we only require kernel evaluations between the training examples indexed by $\mathbf{i}$ and the data point $\phi(\mathbf{x}_j)$ in order to make this reconstruction. Hence, $\mathbf{i}$ forms a compression set. □

**Remark 4.1.** *This claim proves that sparse kernel PCA forms a sample compression scheme. The only information needed for the reconstruction function is the data in the compression set from which the matrix approximation (given by Equation 4.6) can be created.*

[1] In their book, Smola and Schölkopf redefined their kernel approximation in the same way as we have done [Schölkopf and Smola, 2002].

We can now prove that Algorithm 8 is equivalent to Algorithm 2 of Smola and Schölkopf [2000].

**Theorem 4.1.** *Agorithm 8 is equivalent to Algorithm 2 of Smola and Schölkopf [2000].*

*Proof.* Let $\mathbf{K}$ be the kernel matrix and let $\mathbf{K}[:, i]$ be the $i$th column of the kernel matrix. Assume $\mathbf{X}$ is the input matrix containing rows of vectors that have already been mapped into a higher dimensional feature space using $\phi$ such that $\mathbf{X} = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_m))^\top$. Smola and Schölkopf [2000] state in section 4.2 of their paper that their algorithm 2 finds a low rank approximation of the kernel matrix such that it minimises the Frobenius norm $\|\mathbf{X} - \tilde{\mathbf{X}}\|^2_{\text{Frob}} = \text{tr}\{\mathbf{K} - \tilde{\mathbf{K}}\}$ where $\tilde{\mathbf{X}}$ is the low rank approximation of $\mathbf{X}$. Therefore, we need to prove that Algorithm 8 also minimises this norm.

We would like to show that the maximum reduction in the Frobenius norm between the kernel $\mathbf{K}$ and its projection $\tilde{\mathbf{K}}$ is in actual fact the choice of basis vectors that maximise the Rayleigh quotient and deflate according to Equation 4.4. At each stage we deflate by,

$$\mathbf{K} = \mathbf{K} - \frac{\tau\tau^\top}{\mathbf{K}[\mathbf{i}_d, \mathbf{i}_d]}.$$

The trace $\text{tr}\{\mathbf{K}\} = \sum_{i=1}^m \mathbf{K}[i, i]$ is the sum of the diagonal elements of matrix $\mathbf{K}$. Therefore,

$$\begin{aligned}\text{tr}\{\mathbf{K}\} &= \text{tr}\{\mathbf{K}\} - \frac{\text{tr}\{\tau\tau^\top\}}{\mathbf{K}[\mathbf{i}_d, \mathbf{i}_d]} \\ &= \text{tr}\{\mathbf{K}\} - \frac{\text{tr}\{\tau^\top\tau\}}{\mathbf{K}[\mathbf{i}_d, \mathbf{i}_d]} \\ &= \text{tr}\{\mathbf{K}\} - \frac{\mathbf{K}^2[\mathbf{i}_d, \mathbf{i}_d]}{\mathbf{K}[\mathbf{i}_d, \mathbf{i}_d]}.\end{aligned}$$

The last term of the final equation corresponds exactly to the Rayleigh quotient of Equation 4.3. Therefore the maximisation of the Rayleigh quotient does indeed correspond to the maximum reduction in the Frobenius norm between the approximated matrix $\tilde{\mathbf{X}}$ and $\mathbf{X}$. □

Claim 4.1 proves that we have a compression scheme and so we can bound the sparse KPCA algorithm using the tools available to us from sample compression theory.

### 4.1.2 A generalisation error bound for sparse kernel principal components analysis

The results of the last section imply a sample compression analysis for sparse KPCA. We use the sample compression framework of Littlestone and Warmuth [1986] to bound the generalisation error of the sparse PCA algorithm. Note that kernel PCA bounds of [Shawe-Taylor et al., 2005] do not use sample compression in order to bound the true error. As pointed out above, we use the simple fact that this algorithm can be viewed as a *compression scheme.* No side information is needed in this setting and a simple application of Littlestone and Warmuth [1986] is all that is required. That said, the usual application of compression bounds has been for classification algorithms, while here we are considering a subspace method.

Recall that a sample compression scheme for classification is a learning algorithm that only needs to use a small subset of training points to construct a hypothesis. This hypothesis should be capable of labelling all of the training points not found in the compression set. As mentioned earlier, this implies that this type of learning algorithm and a compression scheme are equivalent. Therefore, we will sometimes refer to a compression scheme learning algorithm as a compression scheme and vice versa.

To bound the generalisation error of the sparse KPCA algorithm we make use of the following well-known theorem.

**Theorem 4.2** (Hoeffding's inequality). *If $X_1, \ldots, X_n$ are independent random variables satisfying $X_i \in [a_i, b_i]$ (meaning $X_i$ is in the interval between $a_i$ and $b_i$) and if we define the sum of these random variables as $S_n = \sum_{i=1}^n X_i$, then it follows that*

$$\Pr\{|S_n - \mathbb{E}[S_n]| \geq \epsilon\} \leq 2\exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right) \tag{4.7}$$

*where $\mathbb{E}[\cdot]$ denotes the expectation.*

This theorem upper bounds the probability of deviation between the sum of the random variables and the expected value of this sum. In a machine learning context it can provide an upper bound on the deviation of the observed error and the true error for a fixed function $f$ given for random draws of training sets. Together with the proof that sparse kernel PCA defines a compression scheme we are now in a position to present a sample compression bound for sparse KPCA.

**Theorem 4.3.** *Let $\mathcal{A}_d$ be any learning algorithm having a reconstruction function that maps compression sets to* regressors. *Let $m$ be the size of the training set $S$, let $d$ be the size of the compression set and let $\hat{\mathcal{E}}_{m-d}[\ell(\mathcal{A}_d(S))]$ be the empirical loss on the $m-d$ points outside of the compression set. Then with probability $1-\delta$, the expected loss $\mathcal{E}[\ell(\mathcal{A}_d(S))]$ of algorithm $\mathcal{A}_d$ given any training set $S$ can be bounded by,*

$$\mathcal{E}[\ell(\mathcal{A}_d(S))] \leq \min_{1 \leq t \leq d}\left[\hat{\mathcal{E}}_{m-t}[\ell(\mathcal{A}_t(S))] + \sqrt{\frac{R^2}{2(m-t)}\left[t\ln\left(\frac{em}{t}\right) + \ln\left(\frac{2m}{\delta}\right)\right]}\right],$$

*where $\ell(\cdot) \geq 0$ and $R = \sup \ell(\cdot)$.*

*Proof.* Consider the case where we have a compression set of size $d$. Then we have $\binom{m}{d}$ different ways of choosing the compression set. Given $\delta$ confidence we apply Hoeffding's bound to the $m-d$ points not in the compression set once for each choice by setting the right hand side of Equation (4.7) equal to $\frac{\delta}{\binom{m}{d}}$. From the definitions $\ell(\cdot) \geq 0$ and $R = \sup \ell(\cdot)$ we can set $b_i = 0$

and $a_i = \frac{R}{m-d}$ respectively, in Hoeffding's bound. Using these facts we get,

$$\begin{aligned}
2\exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{m-d}\left(\frac{R}{m-d}-0\right)^2}\right) &= \frac{\delta}{\binom{m}{d}} \\
2\exp\left(-\frac{2\epsilon^2}{(m-d)R^2/(m-d)^2}\right) &= \frac{\delta}{\binom{m}{d}} \\
2\exp\left(-\frac{(m-d)2\epsilon^2}{R^2}\right) &= \frac{\delta}{\binom{m}{d}}.
\end{aligned}$$

Solving for $\epsilon$ and further applying a factor $1/m$ to $\delta$ to ensure one application for each possible choice of $d$ we get:

$$\epsilon \leq \sqrt{\frac{R^2}{2(m-d)}\left[d\ln\left(\frac{em}{d}\right)+\ln\left(\frac{2m}{\delta}\right)\right]}.$$

Hence by Hoeffding's bound $\Pr\{\mathcal{E}[\ell(\mathcal{A}_d(S))] > \hat{\mathcal{E}}_{m-d}[\ell(\mathcal{A}_d(S))] + \epsilon\} \leq \delta$. This together with the fact that using more dimensions can only reduce the expected loss on test points gives the result. □

**Remark 4.2.** *It should be noted that this simple analysis could not be carried out with KMP as it requires additional information from outside of the compression set in order to reconstruct its regressors. This is not considered in the above theorem. Such an analysis is possible by taking into account the extra information needed in order to make the desired reconstruction. This can be performed by using the Vapnik–Chervonenkis (VC) argument we proposed in Section 3.11.*

We now consider the application of the above bound to sparse KPCA. Let the corresponding loss function be defined as

$$\ell(\mathcal{A}_t(S))(\mathbf{x}) = \|\mathbf{x} - P_{\mathbf{i}_t}(\mathbf{x})\|^2,$$

where $\mathbf{x}$ is a test point and $P_{\mathbf{i}_t}(\mathbf{x})$ its projection into the subspace determined by the set $\mathbf{i}_t$ of indices returned by $\mathcal{A}_t(S)$. Thus we can give a more specific loss bound in the case where we use a Gaussian kernel in the sparse kernel principal components analysis.

**Corollary 4.1** (Sample compression bound for SKPCA). *Using a Gaussian kernel and all of the definitions from Theorem 4.3, we get the following bound:*

$$\mathcal{E}[\ell(\mathcal{A}(S))] \leq \min_{1\leq t\leq d}\left[\frac{1}{m-t}\sum_{i=1}^{m-t}\|\mathbf{x}_i - P_{\mathbf{i}_t(\mathbf{x}_i)}\|^2 + \sqrt{\frac{1}{2(m-t)}\left[t\ln\left(\frac{em}{t}\right)+\ln\left(\frac{2m}{\delta}\right)\right]}\right], \tag{4.8}$$

Note that $R$ corresponds to the smallest radius of a ball that encloses all of the training points. Hence, for the Gaussian kernel $R$ equals 1. Figure 4.1 shows how the bound is optimising the structural risk minimisation principle (when we do not use the min in Theorem 4.3). The blue line represents the training residual computed as the loss of those points not in $\mathbf{i}$, the black line is the complexity term in the bound (the square root term in Equation (4.8)). The red line in the figure is the bound value and carries out a trade-off between accuracy on the training data and complexity of the function.

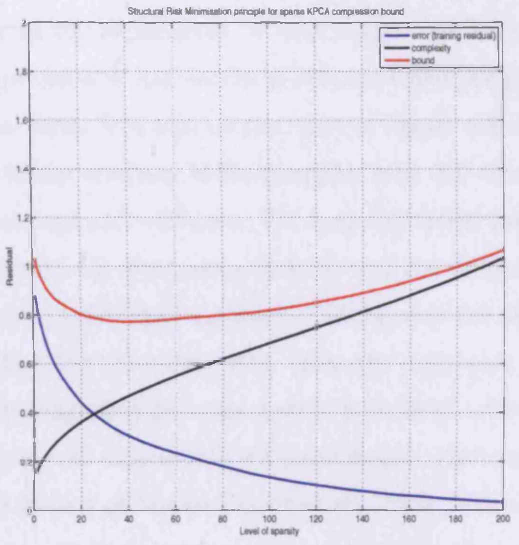

Figure 4.1: Sample compression bound plot for sparse KPCA including the error and complexity terms separately (SRM principle).

We now compare the sample compression bound proposed above for SKPCA with the KPCA bound introduced by Shawe-Taylor et al. [2005]:

**Theorem 4.4** (Shawe-Taylor et al. [2005]). *If we perform PCA in the feature space defined by a kernel $\kappa(\mathbf{x}, \mathbf{z})$ then with probability greater than $1-\delta$, for any $1 \leq d \leq m$, if we project new data onto the space $U_d$, the expected squared residual is bounded by*

$$\mathcal{E}\left[\|P_{U_d}^{\perp}(\phi(\mathbf{x}))\|^2\right] \quad \leq \quad \min_{1\leq t\leq d}\left[\frac{1}{m}\lambda^{>t}(S) + \frac{1+\sqrt{t}}{\sqrt{m}}\sqrt{\frac{2}{m}\sum_{i=1}^{m}\kappa(\mathbf{x}_i,\mathbf{x}_i)^2}\right] + R^2\sqrt{\frac{18}{m}\ln\left(\frac{2m}{\delta}\right)},$$

*where the support of the distribution is in a ball of radius $R$ in the feature space and $\lambda^{>t}(S) = \sum_{i=t+1}^{m}\lambda_i$ is the sum of the eigenvalues greater than $t$ computed from the training data in the feature space.*

We apply this bound with the Gaussian kernel which sets both $R$ and $\kappa(\mathbf{x}_i, \mathbf{x}_i)$ to 1. We compute it for sparse KPCA by simply adapting it to work with a sparse set of basis vectors rather than a full kernel matrix and its reduced dimensionality. Figure 4.2 shows plots for the test error residuals together with its upper bounds computed using Theorem 4.4 and the sample compression bound of Corollary 4.1. The sample compression bound is much tighter than the PCA bound and non-trivial. This is in stark contrast to the PCA bound, which is looser and trivial.

The bound is at its lowest point after 43 basis vectors have been added. We speculate that at this point the "true" dimensions of the data have been found and that all other dimensions correspond to "noise". Notice how, after approximately 43 dimensions the test error looks straighter in its decline – implying that a constant "noise" factor is helping to reduce the test error. We carry out an extra toy experiment to help assess whether or not this is true and to show that the sample compression bound can help indicate when the principal components have captured most of the actual data. The plot on the right of Figure 4.2 depicts the results of a toy experiment where we randomly sampled 1000 examples with 450 dimensions from a Gaussian distribution with zero mean and unit variance. We then multiplied the first 50 dimensions with large numbers (using exponentially decreasing factors) and the remaining 400 dimensions with a small constant noise factor (very small number). This caused the first 50 dimensions to have much larger eigenvalues than the remaining 400. From the right plot of Figure 4.2 we see that the test residual keeps dropping after 50 basis vectors have been added with small decrements. The compression bound picks 46 dimensions with the largest eigenvalues and fails to pick the final 4, however, the PCA bound of Shawe-Taylor et al. [2005] is much more optimistic and is at its lowest point only after 30 basis vectors, implying that it has captured most of the data in 30 dimensions. Therefore, as well as being tighter and non-trivial, the compression bound is much better at predicting the best choice for the number of dimensions to use with sparse KPCA. Note that we carry out this experiment without randomly permuting the projections into a subspace because SKPCA is rotation invariant and will always pick out the principal components with the largest eigenvalues.

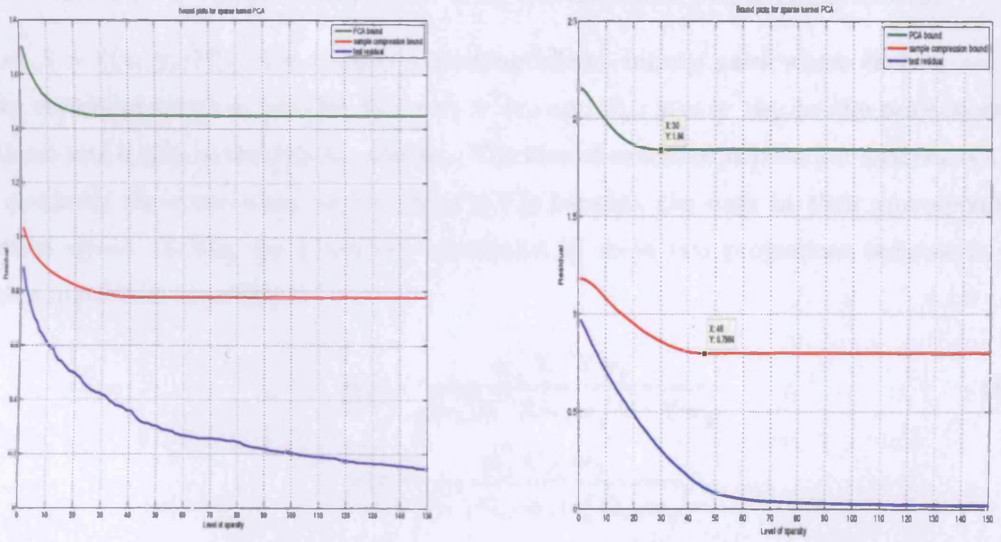

Figure 4.2: Bound plots for sparse kernel PCA comparing the sample compression bound proposed in this chapter and the already existing PCA bound.

## 4.2 Kernel canonical correlation analysis

The theoretical work we proposed in the earlier sections together with the practical implementations motivates this section.

In this section we introduce a sparse canonical correlation analysis algorithm in the dual. In doing so, we point out why we cannot use the theory of section 4.1 and must use the theory of section 3.11 to bound the generalisation error of this new KCCA variant. We also show that the algorithm is similar in spirit to KMP and sparse KPCA. Firstly, we discuss the work from a primal view and then proceed towards its dual counterpart.

We first derive a sparse version of kernel canonical correlation and present it as a function (see Function 9). Then we are left with the issue of how to choose the set of basis vectors **i**. The initial algorithm lays the foundations to become our "gold standard". However, its naive implementation leads to an inefficient algorithm that must be refined in order for it to be practical. Therefore, we improve the speed of the algorithm with a matching pursuit algorithm that corresponds to the Algorithm 8 of KPCA and also introduce an approximation method to gain extra computational advantage. We give a brief overview of canonical correlation analysis (CCA) and kernel canonical correlation analysis (KCCA), but start the discussion with a small example to motivate the reason why two views of the same data may be important.

In canonical correlation analysis we seek to find the maximum correlation between two different views of the same object. For instance, we may have the same book in English and French. The two views are the two different languages and the objective of CCA in this case would be to find the features that are maximally correlated between the two languages. In this way we would hope to extract features that bring out the underlying semantic content.

Let $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$ be a sample containing input$_\mathcal{X}$-input$_\mathcal{Y}$ pairs where each $\mathbf{x}_i$ and $\mathbf{y}_i$ are row vectors of length $n$, and let $P_x : \mathbf{x} \mapsto \mathbf{x}^\top \mathbf{w}_x$ and $P_y : \mathbf{y} \mapsto \mathbf{y}^\top \mathbf{w}_y$ be the projections of each input into spaces defined by $\mathbf{w}_x$ and $\mathbf{w}_y$. The idea of canonical correlation analysis (CCA) is to maximise the correlation $corr(P_x(\mathbf{X}), P_y(\mathbf{Y}))$ between the data in their corresponding projection space. Taking the maximum correlation of these two projections reduces to the following maximisation problem:

$$\begin{aligned}\rho &= \max_{\mathbf{w}_x, \mathbf{w}_y} \frac{\mathbf{w}_x^\top \mathbf{X}^\top \mathbf{Y} \mathbf{w}_y}{\sqrt{\mathbf{w}_x^\top \mathbf{X}^\top \mathbf{X} \mathbf{w}_x \mathbf{w}_y^\top \mathbf{Y}^\top \mathbf{Y} \mathbf{w}_y}} \\ &= \max_{\mathbf{w}_x, \mathbf{w}_y} \frac{\mathbf{w}_x^\top \mathbf{C}_{xy} \mathbf{w}_y}{\sqrt{\mathbf{w}_x^\top \mathbf{C}_{xx} \mathbf{w}_x \mathbf{w}_y^\top \mathbf{C}_{yy} \mathbf{w}_y}},\end{aligned} \tag{4.9}$$

where $\mathbf{C}_{xy} = \mathbf{C}_{yx}^\top$ is the covariance matrix between $\mathbf{X}$ and $\mathbf{Y}$ and $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$ the covariance matrices of $\mathbf{X}$ and $\mathbf{Y}$, respectively.

The above maximisation problem can be evaluated by solving a *generalised eigenproblem* of the form:

$$\mathbf{A}\mathbf{w} = \lambda \mathbf{B}\mathbf{w}, \tag{4.10}$$

where $(\mathbf{w}, \lambda)$ are the eigenvector-eigenvalue pair corresponding to the solution and $\mathbf{A}$, $\mathbf{B}$ are square matrices. The CCA generalised eigenproblem can be written as:

$$\begin{bmatrix} \mathbf{0} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w}_x \\ \mathbf{w}_y \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{C}_{xx} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{yy} \end{bmatrix} \begin{bmatrix} \mathbf{w}_x \\ \mathbf{w}_y \end{bmatrix}. \tag{4.11}$$

The fact that this problem only looks for linear relationships in data and does not tackle non-linear data sets limits its use in many contexts. By applying the kernel trick, several authors [Lai and Fyfe, 2000, Bach and Jordan, 2003] have proposed a kernel version of canonical correlation analysis in order to tackle non-linear relations.

Let us map each training example to a higher dimension using a feature mapping $\phi : \mathbf{x} \mapsto \phi(\mathbf{x})$. In the case of a linear kernel each $n$-dimensional vector $\mathbf{x}$ is mapped with the identity $\phi(\mathbf{x}) = \mathbf{x}$ that corresponds to the following kernel matrix $\mathbf{K}_x = \mathbf{X}\mathbf{X}^\top$. Therefore by making substitutions $\mathbf{w}_x = \mathbf{X}^\top \boldsymbol{\alpha}_x$ and $\mathbf{w}_y = \mathbf{Y}^\top \boldsymbol{\alpha}_y$ in Equation (4.9) and replacing $\mathbf{X}\mathbf{X}^\top$ with $\mathbf{K}_x$ and $\mathbf{Y}\mathbf{Y}^\top$ with $\mathbf{K}_y$ we get the following kernel CCA maximisation problem,

$$\rho = \max_{\boldsymbol{\alpha}_x, \boldsymbol{\alpha}_y} \frac{\boldsymbol{\alpha}_x^\top \mathbf{K}_x^\top \mathbf{K}_y \boldsymbol{\alpha}_y}{\sqrt{\boldsymbol{\alpha}_x^\top \mathbf{K}_x^\top \mathbf{K}_x \boldsymbol{\alpha}_x \boldsymbol{\alpha}_y^\top \mathbf{K}_y^\top \mathbf{K}_y \boldsymbol{\alpha}_y}},$$

which can be solved as the following generalised eigenproblem,

$$\begin{bmatrix} \mathbf{0} & \mathbf{K}_{xy} \\ \mathbf{K}_{yx} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_x \\ \boldsymbol{\alpha}_y \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{K}_x^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_y^2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_x \\ \boldsymbol{\alpha}_y \end{bmatrix}, \tag{4.12}$$

where $\mathbf{K}_{xy} = \mathbf{K}_x \mathbf{K}_y$, $\mathbf{K}_{yx} = \mathbf{K}_y \mathbf{K}_x = \mathbf{K}_{xy}^\top$, $\mathbf{K}_x^2 = \mathbf{K}_x \mathbf{K}_x$ and $\mathbf{K}_y^2 = \mathbf{K}_y \mathbf{K}_y$. Note that any kernel can be used in the above setting.

The solution of the above eigenproblem may lead to overfitting (see Hardoon et al. [2004], Shawe-Taylor and Cristianini [2004]) and to avoid this the following regularised version has been proposed,

$$\begin{bmatrix} \mathbf{0} & \mathbf{K}_{xy} \\ \mathbf{K}_{yx} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_x \\ \boldsymbol{\alpha}_y \end{bmatrix} = \lambda \begin{bmatrix} (1-\tau_x)\mathbf{K}_x^2 + \tau_x \mathbf{K}_x & \mathbf{0} \\ \mathbf{0} & (1-\tau_y)\mathbf{K}_y^2 + \tau_y \mathbf{K}_y \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_x \\ \boldsymbol{\alpha}_y \end{bmatrix}, \tag{4.13}$$

where $0 < \tau_x, \tau_y < 1$ are regularisation parameters that each penalise the norms of the weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$, respectively. The solution of the generalised eigenproblem is of order $\mathcal{O}(m^3)$ complexity. Another downside is that the amount of data needed is twice the size of the kernel matrices. These problems have been tackled by reducing to solving a standard eigenproblem which saves on memory but the overall saving in time complexity is not significant.

Also, Gram Schmidt orthogonalisation procedures have been used to tackle these problems. However, analysing data sets with more than 10,000 data points is typically a real challenge with any of these variants of KCCA. We now present methods that deliver dual sparsity which result in fast training/testing times and tractability for larger data sets.

### 4.2.1 Sparse kernel canonical correlation analysis

We would like to construct (dual) sparse kernel canonical correlation analysis (SKCCA) algorithms. By sparsity we mean using a small subset of basis vectors from the sample $(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, m$. The sparse index set of basis vectors will be contained in an index vector $\mathbf{i}$. Following earlier notation for sparse PCA the sparse CCA problem can be expressed as:

$$\rho = \max_{\mathbf{i},\tilde{\mathbf{w}}_x,\tilde{\mathbf{w}}_y} \frac{\tilde{\mathbf{w}}_x^\top \mathbf{X}^\top \mathbf{Y} \tilde{\mathbf{w}}_y}{\sqrt{\tilde{\mathbf{w}}_x^\top \mathbf{X}^\top \mathbf{X} \tilde{\mathbf{w}}_x \tilde{\mathbf{w}}_y^\top \mathbf{Y}^\top \mathbf{Y} \tilde{\mathbf{w}}_y}},$$

where $\mathbf{w}_x \in span\{\mathbf{X}[\mathbf{i},:]\}$ and $\mathbf{w}_y \in span\{\mathbf{Y}[\mathbf{i},:]\}$. This equation can be converted into its dual by taking advantage of the fact that the primal weight vectors can be written in terms of a linear combination of the training examples and the dual weight vectors:

$$\tilde{\mathbf{w}}_x = \mathbf{X}[\mathbf{i},:]^\top \tilde{\alpha}_x, \tag{4.14}$$

$$\tilde{\mathbf{w}}_y = \mathbf{Y}[\mathbf{i},:]^\top \tilde{\alpha}_y. \tag{4.15}$$

We can substitute these two expressions into the CCA problem:

$$\rho = \max_{\mathbf{i},\tilde{\alpha}_x,\tilde{\alpha}_y} \frac{\tilde{\alpha}_x^\top \mathbf{X}[\mathbf{i},:]\mathbf{X}^\top \mathbf{Y}\mathbf{Y}[\mathbf{i},:]^\top \tilde{\alpha}_y}{\sqrt{\tilde{\alpha}_x^\top \mathbf{X}[\mathbf{i},:]\mathbf{X}^\top \mathbf{X}\mathbf{X}[\mathbf{i},:]^\top \tilde{\alpha}_x \tilde{\alpha}_y^\top \mathbf{Y}[\mathbf{i},:]\mathbf{Y}^\top \mathbf{Y}\mathbf{Y}[\mathbf{i},:]^\top \tilde{\alpha}_y}}.$$

Furthermore, we have $\mathbf{K}_x[:,\mathbf{i}] = \mathbf{X}\mathbf{X}[\mathbf{i},:]^\top$ and $\mathbf{K}_y[:,\mathbf{i}] = \mathbf{Y}\mathbf{Y}[\mathbf{i},:]^\top$, therefore we have the sparse kernel CCA problem:

$$\rho = \max_{\mathbf{i},\tilde{\alpha}_x,\tilde{\alpha}_y} \frac{\tilde{\alpha}_x^\top \mathbf{K}_x[:,\mathbf{i}]^\top \mathbf{K}_y[:,\mathbf{i}] \tilde{\alpha}_y}{\sqrt{\tilde{\alpha}_x^\top \mathbf{K}_x^2[\mathbf{i},\mathbf{i}] \tilde{\alpha}_x \tilde{\alpha}_y^\top \mathbf{K}_y^2[\mathbf{i},\mathbf{i}] \tilde{\alpha}_y}},$$

where $\tilde{\alpha}_x$ and $\tilde{\alpha}_y$ are sparse dual eigenvectors. This leads to, for fixed $\mathbf{i}$, the sparse KCCA generalised eigenproblem of the form

$$\begin{bmatrix} \mathbf{0} & \mathbf{K}_{xy}[\mathbf{i},\mathbf{i}] \\ \mathbf{K}_{yx}[\mathbf{i},\mathbf{i}] & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{\alpha}_x \\ \tilde{\alpha}_y \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{K}_x^2[\mathbf{i},\mathbf{i}] & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_y^2[\mathbf{i},\mathbf{i}] \end{bmatrix} \begin{bmatrix} \tilde{\alpha}_x \\ \tilde{\alpha}_y \end{bmatrix}. \tag{4.16}$$

Therefore, the solution of Equation (4.16) will yield a sparse set of eigenvectors from which to make the projection into the common space. Function sparseKCCA describes how to solve the problem and is the sparse KCCA algorithm that we will use throughout the chapter, whenever we are given two kernels $\mathbf{K}_x$, $\mathbf{K}_y$ and a (small) index set $\mathbf{i}$. Notice the similarities between

Equation (4.16) and Equation (4.12) given for KCCA. Firstly, the forms are identical when **i** is the full set of training examples. However, as we will restrict **i** to be much smaller, we will only require the solution from square kernel matrices of size $d \times d$ where $d = |\mathbf{i}|$. Finally, note that our sparse eigenproblem given by Equation (4.16) is in the "un-regularised" form of the KCCA problem. However, we will show in the experiments that sparsity is in fact a more robust form of regularisation for KCCA as opposed to the standard method given in Equation (4.13).

**Function** sparseKCCA($\mathbf{K}_x, \mathbf{K}_y, \mathbf{i}$)

```
1: set $\tilde{\mathbf{X}} = \mathbf{K}_x[:, \mathbf{i}]^\top$ and $\tilde{\mathbf{Y}} = \mathbf{K}_y[:, \mathbf{i}]^\top$
2: create sparse matrices $\tilde{\mathbf{K}}_{xx} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top$, $\tilde{\mathbf{K}}_{yy} = \tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^\top$, $\tilde{\mathbf{K}}_{xy} = \tilde{\mathbf{X}}\tilde{\mathbf{Y}}^\top$ and $\tilde{\mathbf{K}}_{yx} = \tilde{\mathbf{K}}_{xy}^\top$
3: solve the following generalised eigenvalue problem:
   $$\begin{bmatrix} \mathbf{0} & \tilde{\mathbf{K}}_{xy} \\ \tilde{\mathbf{K}}_{yx} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{\alpha}_x \\ \tilde{\alpha}_y \end{bmatrix} = \lambda \begin{bmatrix} \tilde{\mathbf{K}}_{xx} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{K}}_{yy} \end{bmatrix} \begin{bmatrix} \tilde{\alpha}_x \\ \tilde{\alpha}_y \end{bmatrix}$$
Output: sparse eigenvectors $\tilde{\alpha}_x$, $\tilde{\alpha}_y$ and eigenvalues $\lambda$.
```

Now that we have described the sparseKCCA algorithm we turn our attention to finding the best sparse index set **i** of training examples to use in conjunction with the sparse KCCA.

### 4.2.2 Choosing the best i (slow method)

We describe an algorithm that chooses the best index set **i** from the entire set of basis vectors in a greedy fashion. We seek a low dimensional subspace as the span of a set **i** of training examples, using the corresponding subsets for the projections in the spaces $\mathcal{X}$ and $\mathcal{Y}$. The criterion for extending the subspace is to choose the example for which the correlation computed when projecting all of the data into the corresponding subspaces is maximal. This corresponds to a greedy optimisation of this criterion.

One approach is to look at the addition of each basis vector, in turn, and to run the sparseKCCA using this set of chosen basis vectors. A basis vector whose sparse KCCA solution yields the largest eigenvalue is added to the set and this process repeated up to a sparsity parameter $d$. This simple greedy strategy results in Algorithm 10.

This simple algorithm has one major problem: its complexity is cubic in the number of training points. This is a larger complexity than we are willing to accept. Therefore, we would like an approach similar to SKPCA and KMP when looking for these basis vectors. A more computationally efficient approach for choosing basis vectors can be given by looking at the idea behind the sparse KPCA algorithm of Algorithm 8.

### 4.2.3 Choosing the best i (fast method)

Using a similar strategy to Smola and Schölkopf [2000] we now show that maximising the quotient of the CCA problem leads to a fast method for choosing basis vectors. However, after picking a basis vector (unlike algorithm 10) we must project into a space orthogonal to it, and describe a deflation step that guarantees future basis vectors chosen are orthogonal to all others.

We would like to construct a very fast calculation that enables us to quickly find basis

**Algorithm 10**: A greedy algorithm for choosing basis vectors ("gold standard")

```
Input: two views $\mathbf{K}_x$, $\mathbf{K}_y$, sparsity parameter $d > 0$.
1: initialise $\mathbf{i} = [\,]$ and $\mathbf{j} = [1, \dots, m]$
2: for $i = 1$ to $d$ do
3:   for $j = 1$ to $|\mathbf{j}|$ do
4:     set $\mathbf{i}_i = \mathbf{j}_j$ such that $\mathbf{i}$ only contains the chosen indices
5:     run sparseKCCA($\mathbf{K}_x, \mathbf{K}_y, \mathbf{i}$) to find eigenvalues $\lambda$ in decreasing order
6:     if $\lambda_j$ is the largest found so far then
7:       set $best_i = j$
8:     end if
9:   end for
10:  update $\mathbf{i}_i = \mathbf{j}_{best_i}$ with the index of the best basis vector found
11:  remove index $\mathbf{j}_{best_i}$ from $\mathbf{j}$
12: end for
13: run sparseKCCA($\mathbf{K}_x, \mathbf{K}_y, \mathbf{i}$) with final $\mathbf{i}$ to find $\tilde{\alpha}_x$, $\tilde{\alpha}_y$ and $\lambda$
Output: eigenvectors $\tilde{\alpha}_{\mathbf{x}}$, $\tilde{\alpha}_{\mathbf{y}}$.
```

vectors. We take a similar approach to Algorithm 8 which implies the maximisation of the generalised Rayleigh quotient,

$$\max_i \rho_i = \frac{\mathbf{e}_i^\top \mathbf{K}_x \mathbf{K}_y \mathbf{e}_i}{\sqrt{\mathbf{e}_i^\top \mathbf{K}_x^2 \mathbf{e}_i \mathbf{e}_i^\top \mathbf{K}_y^2 \mathbf{e}_i}}, \tag{4.17}$$

where $\mathbf{e}_i$ is the $i$th unit vector. At each iteration we look to find the basis vector that maximises the quotient given by Equation (4.17). Once it has been chosen then the following orthogonality procedure (deflation) is carried out to make sure future chosen bases are sufficiently far (geometrically) from those already added to the set $\mathbf{i}$.

Initially, at the first step $j = 1$, let $\tilde{\mathbf{K}}_x^j = \mathbf{K}_x$ and $\tilde{\mathbf{K}}_y^j = \mathbf{K}_y$ denote the deflated kernel matrices at the $j$th iteration. To find the deflated matrices at step $j+1$ we use the KMP deflation given in line 8 of Algorithm 7:

$$\tilde{\mathbf{K}}_x^{j+1} = \left(\mathbf{I} - \frac{\tau_x \tau_x^\top}{\tau_x^\top \tau_x}\right) \tilde{\mathbf{K}}_x^j, \tag{4.18}$$

$$\tilde{\mathbf{K}}_y^{j+1} = \left(\mathbf{I} - \frac{\tau_y \tau_y^\top}{\tau_y^\top \tau_y}\right) \tilde{\mathbf{K}}_y^j, \tag{4.19}$$

where $\tau_x = \tilde{\mathbf{K}}_x^j[:, \mathbf{i}_j]$ and $\tau_y = \tilde{\mathbf{K}}_y^j[:, \mathbf{i}_j]$ such that $j = |\mathbf{i}|, j > 0$ and $\mathbf{i}_j$ is the last element added to vector $\mathbf{i}$ and $\mathbf{I}$ is the identity matrix. This procedure is repeated until $d$ basis vectors have been chosen. This protocol is described in Algorithm 11. Notice the similarities between this algorithm and Algorithm 8 which uses the same procedure of *quotient maximisation* and *deflation* in order to evaluate a sparse kernel principal components analysis.

### 4.2.4 A generalisation error bound for sparse kernel canonical correlation analysis

The compression bound for SKPCA cannot be applied in the SKCCA setting because the basis vectors chosen in order to produce the common subspace requires the full information of the

**Algorithm 11**: A fast greedy algorithm for choosing basis vectors

```
Input: Two views $\mathbf{K}_x$, $\mathbf{K}_y$, sparsity parameter $d > 0$.
1: initialise $\mathbf{i} = [\,]$
2: for $i = 1$ to $d$ do
3:     find improvement $= \frac{\left((\mathbf{K}_x.\mathbf{K}_y)^\top \mathbf{1}\right)^{.2}}{((\mathbf{K}_x^{.2})^\top \mathbf{1}).((\mathbf{K}_y^{.2})^\top \mathbf{1})}$
4:     set $\mathbf{i}_i$ to the index of $\max\{$improvement$\}$
5:     set $\boldsymbol{\tau}_x = \mathbf{K}_x[:, \mathbf{i}_i]$ and $\boldsymbol{\tau}_y = \mathbf{K}_y[:, \mathbf{i}_i]$
6:     deflate kernel matrices like so:
           $\mathbf{K}_x = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_x\boldsymbol{\tau}_x^\top}{\boldsymbol{\tau}_x^\top\boldsymbol{\tau}_x}\right)\mathbf{K}_x$
           $\mathbf{K}_y = \left(\mathbf{I} - \frac{\boldsymbol{\tau}_y\boldsymbol{\tau}_y^\top}{\boldsymbol{\tau}_y^\top\boldsymbol{\tau}_y}\right)\mathbf{K}_y$
7: end for
8: run sparseKCCA($\mathbf{K}_x$,$\mathbf{K}_y$,$\mathbf{i}$) with final $\mathbf{i}$ to find $\tilde{\boldsymbol{\alpha}}_x$, $\tilde{\boldsymbol{\alpha}}_y$ and $\lambda$
Output: index vector $\mathbf{i}$, and eigenvectors $\tilde{\boldsymbol{\alpha}}_x$, $\tilde{\boldsymbol{\alpha}}_y$
```

training set. However, from the type of analysis we made earlier for KMP we can upper bound the future loss of sparse KCCA.

To help keep the notation for the following bound consistent with the bound introduced for KMP we make the following definitions. We denote the input$_\mathcal{X}$ sample as $S^\mathcal{X}$ and similarly the input$_\mathcal{Y}$ sample as $S^\mathcal{Y}$. Therefore, two training samples consisting of paired data sets from the joint space $\mathcal{X} \times \mathcal{Y}$ will be denoted as $S^{\mathcal{X}\times\mathcal{Y}} = S^\mathcal{X} \cup S^\mathcal{Y}$. We denote the index set of the chosen basis vectors as $\mathbf{i}$ and also $S_\mathbf{i}^{\mathcal{X}\times\mathcal{Y}}$ will denote the paired samples indexed by vector $\mathbf{i}$. Earlier we denoted $P_x$ to be the projection from input$_\mathcal{X}$ but here use the notation $f_x$ to denote the same projection function. We make the same change in the notation of the $P_y$ function.

We would also like to remind the reader that the primal weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$ for each view are 1-dimensional weight vectors for some given dimension $i \in \{1, \ldots, d\}$. Finally, analogously to Definition 3.1 we now define the loss functional that will be used for the SKCCA bound.

**Definition 4.1.** *Let $S^{\mathcal{X}\times\mathcal{Y}} \sim \mathcal{D}$ be a paired training sample from a fixed but unknown distribution $\mathcal{D}$. Given the projection functions $f_x = f_x(\mathbf{x}) = \mathbf{w}_x^\top\mathbf{x}$ and $f_y = f_y(\mathbf{y}) = \mathbf{w}_y^\top\mathbf{y}$ and the error $\mathrm{er}(f_x, f_y) = |\mathbf{w}_x^\top\mathbf{x} - \mathbf{w}_y^\top\mathbf{y}| = |f_x(\mathbf{x}) - f_y(\mathbf{y})|$ for the paired data points $\mathbf{x}$ and $\mathbf{y}$ we can define, for some fixed positive scalar $\alpha \in \mathbb{R}$, the corresponding true classification loss as*

$$\mathrm{er}_\alpha(f_x, f_y) = \Pr_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\{|f_x(\mathbf{x}) - f_y(\mathbf{y})| > \alpha\}.$$

*Similarly, we can define the corresponding empirical classification loss as*

$$\begin{aligned}\hat{\mathrm{er}}_\alpha(f_x, f_y) = \mathrm{er}_\alpha^{S^{\mathcal{X}\times\mathcal{Y}}}(f_x, f_y) &= \Pr_{(\mathbf{x},\mathbf{y})\sim S^{\mathcal{X}\times\mathcal{Y}}}\{|f_x(\mathbf{x}) - f_y(\mathbf{y})| > \alpha\}\\ &= \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim S^{\mathcal{X}\times\mathcal{Y}}}\{\mathbb{I}(|f_x(\mathbf{x}) - f_y(\mathbf{y})| > \alpha)\},\end{aligned}$$

*where $\mathbb{I}$ is the indicator function and $S^{\mathcal{X}\times\mathcal{Y}}$ is suppressed when clear from context.*

Given this definition we would like to upper bound the true classification loss with the information gained from the empirical classification loss. We can proceed in much the same way as was done for KMP but with the difference that the weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$ are the vectors that allow a projection into a single dimension. Giving a bound on the loss of the two corresponding projections $P_x$ and $P_y$ onto each dimension found. By making this style of analysis we can simply adapt the KMP bound as follows.

**Theorem 4.5.** *Fix $\alpha \in \mathbb{R}$, $\alpha > 0$. Let $\mathcal{A}$ be the SKCCA algorithm, $m$ the size of the paired training sets $S^{\mathcal{X}\times\mathcal{Y}}$ and $d$ the cardinality of the set $\mathbf{i}$ of chosen basis vectors. Let $S^{\mathcal{X}\times\mathcal{Y}}$ be reordered so that the last $m-d$ points are outside of the set $\mathbf{i}$ and define $t = \sum_{i=m-d}^{m} \mathbb{I}(|f_x(\mathbf{x}_i) - f_y(\mathbf{y}_i)| > \alpha)$ to be the number of errors for those points in $S^{\mathcal{X}\times\mathcal{Y}} \setminus S_{\mathbf{i}}^{\mathcal{X}\times\mathcal{Y}}$. Then with probability $1-\delta$ over the generation of the paired training sets $S^{\mathcal{X}\times\mathcal{Y}}$ the expected loss $\mathcal{E}[\ell(\cdot)]$ of algorithm $\mathcal{A}$ can be bounded by,*

$$\mathcal{E}[\ell(\mathcal{A}(S))] \leq \frac{2}{m-d-t}\left[(d+1)\log\left(\frac{4e(m-d-t)}{d+1}\right) + d\log\left(\frac{em}{d}\right) \right.$$
$$\left. + t\log\left(\frac{e(m-d)}{t}\right) + \log\left(\frac{2m^2}{\delta}\right)\right].$$

*Proof.* We can treat the loss between $\mathbf{w}_x^\top \mathbf{x}$ and $\mathbf{w}_y^\top \mathbf{y}$ as a regression loss

$$\begin{pmatrix} \mathbf{w}_x & -\mathbf{w}_y \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{w}_x^\top \mathbf{x} - \mathbf{w}_y^\top \mathbf{y} \tag{4.20}$$

where we would want (4.20) to be 0 (or the *zero function*) for all $\mathbf{x}$ and $\mathbf{y}$ and every dimension projection $\mathbf{w}_x$ and $\mathbf{w}_y$. This corresponds to a regression loss and can be mapped it into its corresponding classification loss using Definition 4.1. Therefore by constraining the weight vectors to be 1-dimensional and using this loss functional we can proceed in the same manner as the proof of Theorem 3.9. □

This bound can only work on each dimension of the projections found. In order to get a bound for the entire subspace found by SKCCA we would require a union bound over the whole $d$ dimensional subspace – which would greatly loosen the bound. We do not include this bound here.

We have plotted the bound against the true classification loss found between test points $\mathbf{x}$ and $\mathbf{y}$ projected into 1-dimensional subspaces found by KCCA and averaged over all the dimensions. The bound plots are given in Figure 4.3, and each plot corresponds to a different split of the training and testing data. The bounds have also been scaled down by a factor of 5 in order to show both plots closer together (the bounds become trivial early on). As you can see all bound plots look very similar in shape (with some slight orientation) to the test error, with the plot on the the top left being the most similar.

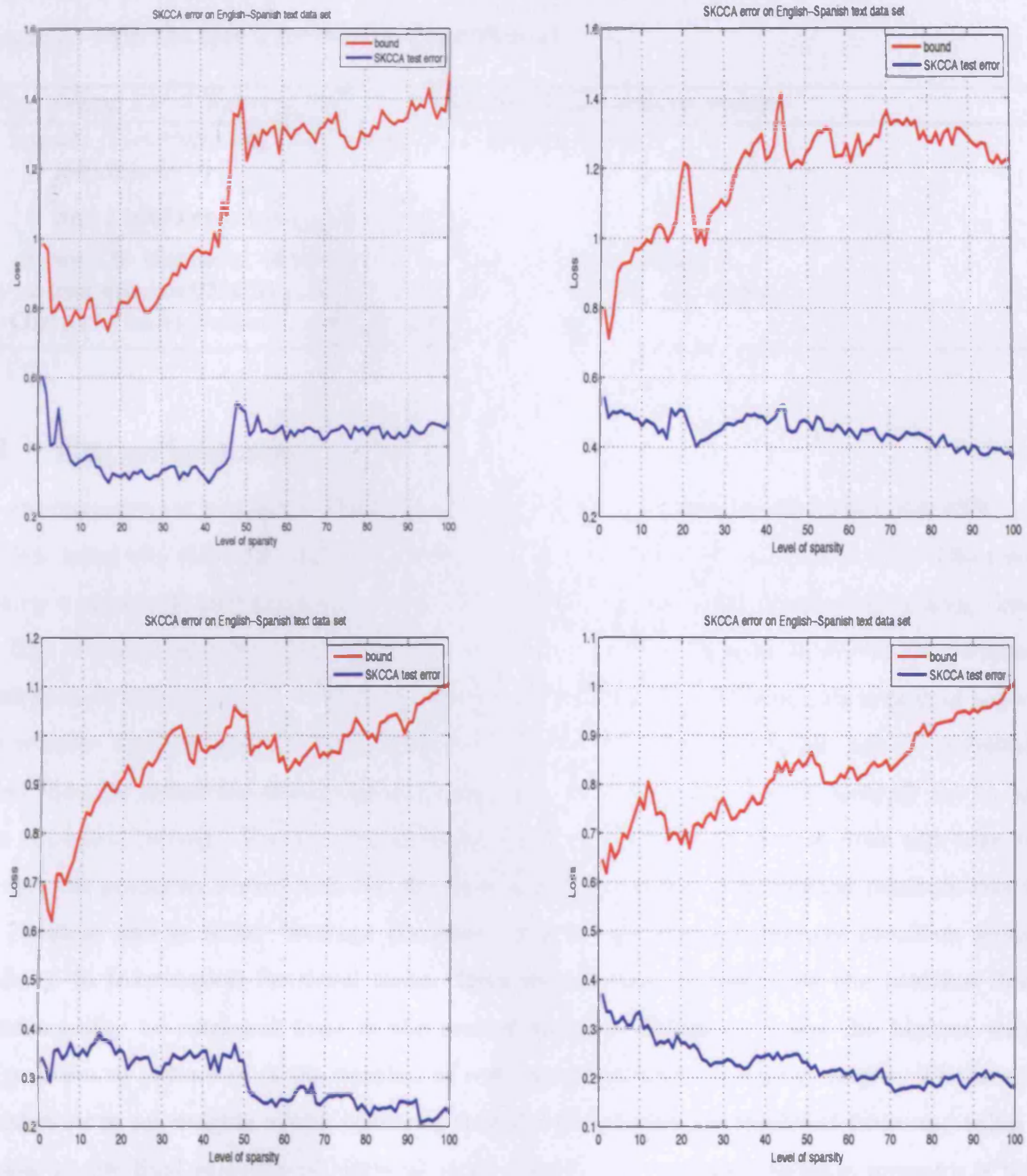

Figure 4.3: Plot of SKCCA bound against its test error for different splits of the data. We used 500 examples for training and the 300 for testing. Bound was scaled down by a factor of 5 and $\alpha = 0.00095$.

### 4.2.5 Choosing a set i (faster method)

Interestingly the bound of the previous section only requires a small subset of chosen basis vectors $\mathbf{i}$ in order to make the sparse reconstruction of the joint subspace in kernel canonical correlation analysis. This indicates that the bound is not algorithm-specific and that any set of chosen basis vectors $\mathbf{i}$ would give an upper bound on the generalisation error. We could randomly select basis vectors but this doesn't seem to generate good solutions.

Another alternative is to compute the Rayleigh quotient of Equation (4.17) once and then choose the $d$ indices generating the largest correlation values of the Rayleigh quotient. Clearly the bound still holds in this case and the algorithm has a constant complexity of $\mathcal{O}(m^2)$. This algorithm is described in Algorithm 12. We show in the next section that this algorithm competes in accuracy with the fast algorithm of Algorithm 11.

**Algorithm 12**: A faster greedy algorithm for choosing basis vectors

```
Input: Two views $\mathbf{K}_x$, $\mathbf{K}_y$, sparsity parameter $d > 0$.
1: initialise $\mathbf{i} = [\,]$
2: find improvement $= \frac{\left((\mathbf{K}_x.\mathbf{K}_y)^\top \mathbf{1}\right)^2}{((\mathbf{K}_x^2)^\top \mathbf{1}).\left((\mathbf{K}_y^2)^\top \mathbf{1}\right)}$
3: set $\mathbf{i}$ to the index of the largest $d$ values of improvement
4: run sparseKCCA($\mathbf{K}_x$,$\mathbf{K}_y$,$\mathbf{i}$) with final $\mathbf{i}$ to find $\tilde{\alpha}_x$, $\tilde{\alpha}_y$ and $\lambda$
Output: index vector $\mathbf{i}$, and eigenvectors $\tilde{\alpha}_x$, $\tilde{\alpha}_y$
```

## 4.3 Experiments

The experiments we conduct are for text retrieval tasks and compared against the KCCA algorithm using two different measures for small, medium and large sized data sets. The results indicate comparable performance to KCCA with the added advantage of faster running times.

The retrieval is assessed using mate retrieval. The first measure is called the "window" measure and counts the number of times documents can be retrieved using their pair as a query. The window size we use is 10 and if the pair can be retrieved within the top 10 correlation values then we count the document as being retrieved, otherwise it is considered not to have been retrieved (error). However, a problem with this measure is that it does not take into account the positions from which the documents were retrieved. The second measure rectifies this problem and is called "average precision" and is the standard average precision method employed in Information Retrieval tasks. Here we associate a weight for the position that a document may be retrieved from in the second language. Position 1 has the highest weight and position $m$ (where $m$ is the number of test examples) has the lowest weight. The average precision sums up weights of the positions that the documents are retrieved from and takes an average as the final measure of retrieval rate. Clearly, the average precision measure is more robust because it takes into account the position of the documents retrieved. This is in contrast with the window method that is content with a document that is contained in the top 10 (say) highest correlation values.

| Data Set | KCCA | | | SKCCA | | | SKCCA (faster) | | |
|---|---|---|---|---|---|---|---|---|---|
| | train | test | total | train | test | total | train | test | total |
| Small | 8 | 20 | 28 | 9 | 5 | 14 | 2 | 5 | 7 |
| Medium | 1707 | 1995 | 3702 | 334 | 224 | 558 | 59 | 224 | 283 |
| Large | 24693 | 27733 | 52426 | 5242 | 698 | 5940 | 1873 | 695 | 2568 |

Table 4.1: Training and test times in seconds for small, medium and large data set sizes (English-Spanish data)

As you can see from Figures 4.4, 4.5 and 4.6 the KCCA algorithm requires different parameter values in order to produce stable results. Also, as the data set sizes increase, so do the best parameter values, *i.e.*, from 0.1 in the small data set (Figure 4.4) to 0.75 in the larger

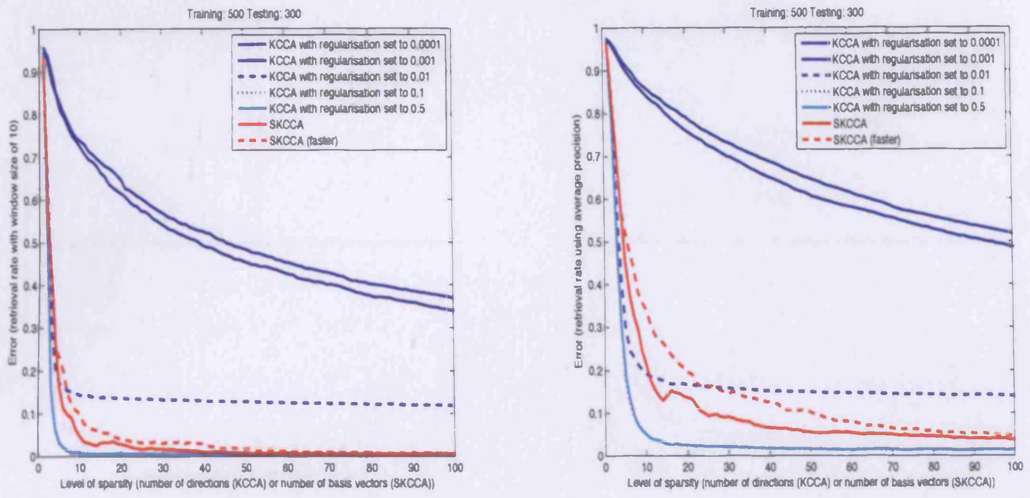

Figure 4.4: A small data set – average test error for retrieval over 5 different splits of the data. Left: average error using window (method) of size 10. Right: average error using average precision.

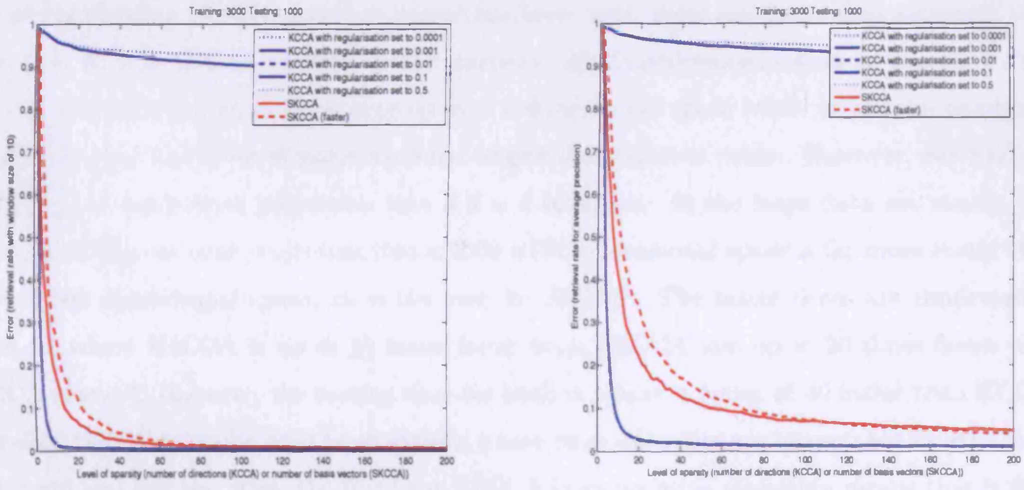

Figure 4.5: A medium sized data set – average test error for retrieval over 5 different splits of the data. Left: average error using window (method) of size 10. Right: average error using average precision.

data set (Figure 4.6). Moreover, we hypothesise that as the size of the data set increases, so to does the value of the regularisation parameter needed in order to achieve good generalisation. However, the upper bound of this parameter value is 1 and when this value is reached then KCCA may start to generate trivial solutions and overfit (as was shown for the smaller parameter values in all three plots). We cannot however show that this hypothesis is correct for very large data sets because it is not possible to train KCCA on data sets with more than 20000 training points (for instance). The experiments show that SKCCA is stable throughout the experiments with different sized data sets, meaning that sparsity is a robust and efficient

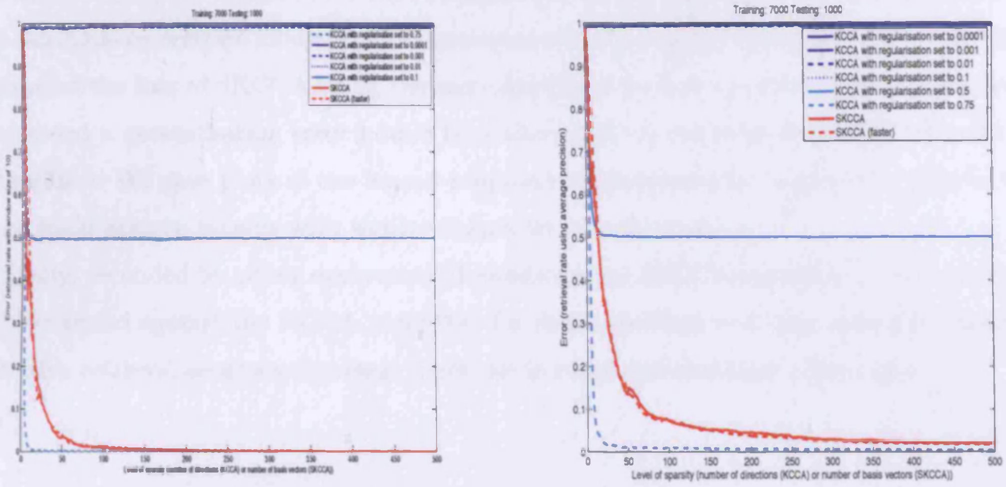

Figure 4.6: A large sized data set – average test error for retrieval over 2 different splits of the data. Left: average error using window (method) of size 10. Right: average error using average precision.

way of regularising KCCA. Another important issue with these results is that although they show that KCCA obtains slightly better accuracy this improvement comes at a cost, KCCA needs to compute the projections onto an $m \times d$ dimensional space where $m$ are the number of training samples and $d$ the dimension of the largest $d$ correlation values. However, our SKCCA only needs to compute a projection into a $d \times d$ subspace. In the large data set results (see Fig. 4.6) we can see that projecting into a $7000 \times 500$ dimensional space is far more costly than a $500 \times 500$ dimenisonal space, as is the case for SKCCA. The faster times are confirmed in Table 4.1 where SKCCA is up to 10 times faster than SKCCA and up to 20 times faster with SKCCA (faster). However, the testing time for both is almost a factor of 40 faster than KCCA. Therefore, SKCCA can be used in situations where large data sets are intractable for KCCA to both train and test on. Also, the fact that SKCCA is an iterative algorithm means that it does not rely on storing the full kernel matrix in memory and has very fast testing times once the sparse set of basis vectors have been chosen.

## 4.4 Summary

We proposed the first sample compression bound for a subspace method called sparse kernel principal components analysis (SKPCA) and proved that the algorithm is a sample compression scheme. We also tested the sample compression bound proposed against the bound of Shawe-Taylor et al. [2005] on the Boston housing data set and showed that it was considerably tighter and non-trivial. Also, a toy experiment demonstrated that the compression bound is a better candidate for model selection than the KPCA bound and is able to indicate when most of the true data has been captured with SKPCA.

Next we exploited the property that maximising the Rayleigh quotient and deflating the kernel matrix can speed up sparse KPCA to propose the first matching pursuit style algorithm for kernel canonical correlation analysis called sparse kernel canonical correlation analysis (SKCCA). We bounded the loss of SKCCA using the same argument we had made for KMP in Chapter 3 and provided a generalisation error bound that also relied on the level of sparsity achieved by the algorithm. We gave plots of the bound proposed which showed for a particular choice of $\alpha$ that we could achieve bounds with similar shapes to the test error.

Finally, we ended by giving experimental results for the SKCCA algorithm on text retrieval tasks. We tested against the KCCA algorithm for small, medium and large data sets, showing competitive retrieval accuracy but large increases in computational time complexity.

# Chapter 5

# Conclusions

We have taken the reader on a journey that looked at machine learning algorithms and techniques to help enforce sparsity. We promoted many of its merits through experimentation and theoretical analyses, and maintained a similar algorithmic approach across the different learning domains to help deliver sparsity *i.e.*, by using a greedy algorithm to build up predictive functions until some termination criteria was reached. This simple protocol was capable of producing very powerful learning algorithms to help tackle machine learning problems using a small number of training samples (kernel basis vectors). However, as is the nature of research the work presented is in no way complete or finished and there remain many areas for future work. We outline some of the more important issues we feel have been raised from the work conducted in this thesis.

## 5.1 Further work

The classification section described the set covering machine (SCM) algorithm and proposed variants called the BSCM, BBSCM and BBSCM($\tau$) and showed that they are considerably sparser than the well known support vector machine (SVM). However, even though the SCM has some very nice properties like tight generalisation error bounds and sparsity it does not seem to be very well known or popular as an alternative classification algorithm to the SVM. We feel that new application areas for the SCM are needed in order to show its merits on a wider class of problems – for instance using kernels, applying it to other problem domains such as Bioinformatics or in situations where we believe that the target function is made up of a small number of features. Showing that the algorithm is advantageous in different scenarios could potentially increase the popularity of the algorithm. Also, an implementation that is readily available is also a future research direction and something that we hope to carry out, by making the algorithms in the thesis available as a standalone C++ library together with a MATLAB interface. This would certainly allow more users access to the algorithms without having to program it explicitly. Another important issue is that for the SCM to be more popular it may need to tackle problems other than classification, such as novelty detection and regression. Hopefully, keeping all the SCM benefits and competing with other well known algorithms in these problem domains. There have been extensions and improvements made to the set covering machine (SCM) (see Marchand et al. [2003], Laviolette et al. [2006], Hussain et al. [2004]) but all the experimental work to date has been on the UCI data sets. Although this thesis made no attempt at addressing these issues we feel that it is now of paramount importance to try and tackle a larger class of problems with the SCM to help gauge its merits and advantages over other learning algorithms.

The BBSCM algorithm's experimental results have been disappointing. Although we proved theoretically that the algorithm would find the smallest bound, we also showed for the Votes data set that this does not indicate the smallest test error. In some cases, bounds with larger values gave the smallest test error. This certainly is not what we expected. The bounds clearly find functions that yield small test error but not in the situations where we would hope, *i.e.*, when the bounds were the smallest. The bound proposed in this thesis looked for sparser solutions (in most cases creating hypotheses with one ball) but the previous bounds (not relying on bounding the positive and negative examples separately) created less sparse solutions. In some cases the one ball hypotheses gave the smallest test error, in others it gave the worst. It would be interesting to analyse why these bounds have such behaviour. Perhaps they need some sort of interaction amongst one another? Maybe applying the bound proposed for the first ball chosen and then switching to one of the other bounds for the remaining choice of balls would help avoid the "one ball" situation of the bound of Theorem 3.5. Another more pressing issue is that the BBSCM algorithm, with all the pruning strategies we employed, was only capable of improving the running times (over a full search) by a factor of 2. This was not nearly enough to be able to tackle larger data sets. A future research direction for the BBSCM would be to prune further the search space, although we would need to be able to prove (as we did in the thesis) that the deletion of such balls would never help create a smaller generalisation error bound.

The regression section that described the KMP algorithm introduced a novel theoretical analysis justifying the performance of the algorithm in terms of the level of sparsity achieved. However, as these bounds become trivial early on, what have we gained? The point of generalisation error bounds can be two-fold. One is to understand the level of learning that is possible (*i.e.*, tight bounds) the second is to help in model selection (*i.e.*, follow the test error shape). The bounds certainly help in model selection situations but do not deliver tight bounds. A research direction would be to tighten these bounds by using a more natural loss function for regression such as least squares. Tighter bounds would certainly give us more confidence in the final regressors computed, such as in medical data analysis situations where we would like to be confident that our mistakes won't be worse than some (small) upper bound. This would also tighten the SKCCA bound proposed.

The KMP and SKCCA algorithms both look for dual sparsity and hence require the full set of training examples in order to compute the final prediction functions. This seems somewhat at odds with the sparsity property we desired at the start of the thesis. We required a small number of training points in order to reconstruct our functions but in the KMP and SKCCA case we need a small number of kernel basis vectors but *not* a small number of examples. In order to enforce this type of sparsity we could choose a small number of training examples and use the remaining points as a test set – much like the sample compression framework of learning. The bounds would certainly be tight and we would no longer require the full set of training data as is the case currently. However, the problem with this regime is that we could not take advantage of the "maximise a quotient and deflate" procedure we have been so dependent on throughout the second half of the thesis. This is because we would need to evaluate a loss on the points remaining outside of the compression set, which the maximisation of a quotient would not give us. Therefore, we would lose our natural speed-up for the sparse kernel algorithms we described in this thesis. Therefore, a method for achieving similar levels of computational speed-ups during training would certainly be a fruitful area of research.

We believe that the SKCCA algorithm can be applied to real world and very large data sets. One issue is storing the full kernel matrix in memory where for larger data sets this can become a problem. However because the SKCCA algorithm is an incremental algorithm that adds kernel basis vectors one at a time, we can attempt at tackling larger data sets. For instance, methods such as sub-sampling would greatly decrease the computational effort needed to store kernel matrices and compute KCCA on very large data sets and so we consider the experimentation on very large real world data sets for SKCCA an important future research direction.

As a final remark we would like to point out that the unsupervised learning and regression section all used the same framework of matching pursuit, where a greedy algorithm is used to build up functions in order to evaluate some target. Although KMP and SKPCA already existed, we provided a more coherent framework and clearer connections between the matching pursuit algorithms in machine learning. The main principle was to maximise some function (*i.e.*, Rayleigh quotient), deflate the kernel matrices and then repeat until some stopping criterion was reached. This simple framework allowed us to pose the KCCA algorithm in a matching pursuit format and also describe bounds on its generalisation error. We feel that the matching pursuit framework is not only restricted to these problem domains and a future research direction would be to apply the framework presented to a wider class of problem domains such as novelty detection and multi-view learning. We hope that the work presented in this thesis will help motivate research in this direction.

# Bibliography

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821 - 837, 1964.

Martin Anthony. Partitioning points by parallel planes. *Discrete Mathematics*, 282:17–21, 2004.

Martin Anthony and Norman Biggs. *Computational Learning Theory.* Cambridge University Press, Cambridge, 1992.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337 - 404, 1950.

Francis R. Bach and Michael I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2003.

P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54, Cambridge, MA, 1999. MIT Press.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of ACM*, 36(4):929–965, 1989.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, Cambridge, U.K., 2000.

G. Davis, S. Mallat, and Z. Zhang. Adaptive time-frequency approximations with matching pursuits. *Optical Engineering*, 33:7:2183–2191, 1994.

C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Sally Floyd and Manfred Warmuth. Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine Learning*, 21(3):269–304, 1995.

David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12):2639–2664, 2004.

David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

Ralf Herbrich. *Learning Kernel Classifiers.* MIT Press, Cambridge, Massachusetts, 2002.

Zakria Hussain. The set covering machine: Implementation, extensions and applications. Master's thesis, Royal Holloway, 2003.

Zakria Hussain, Sandor Szedmak, and John Shawe-Taylor. The linear programming set covering machine. *Technical Report*, 2004.

Pei Ling Lai and Colin Fyfe. Kernel and nonlinear canonical correlation analysis. *International Joint Conference on Neural Networks*, 4:4614, 2000.

A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

John Langford. Tutorial on practical prediction theory for classification. *Journal of Machine Learning Research*, 6:273–306, 2005.

François Laviolette, Mario Marchand, and Mohak Shah. A PAC-Bayes approach to the set covering machine. *Proceedings of the 2005 conference on Neural Information Processing Systems (NIPS 2005)*, 2006.

Nick Littlestone and Manfred K. Warmuth. Relating data compression and learnability. Technical report, University of California Santa Cruz, Santa Cruz, CA, 1986.

Stéphane Mallat and Zhifeng Zhang. Matching pursuit with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

Mario Marchand and John Shawe-Taylor. Learning with the set covering machine. *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 345–352, 2001.

Mario Marchand and John Shawe-Taylor. The set covering machine. *Journal of Machine Learning Reasearch*, 3:723–746, 2002.

Mario Marchand and Marina Sokolova. Learning with decision lists of data-dependent features. *Journal of Machine Learning Reasearch*, 6:427–451, 2005.

Mario Marchand, Mohak Shah, John Shawe-Taylor, and Marina Sokolova. The set covering machine with data-dependent half-spaces. *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pages 520–527, 2003.

M. Minsky and S. Papert. *Perceptrons: An introduction to Computational Geometry.* MIT Press, 1969.

Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–45, 1993.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

B. Schölkopf and A. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Max-Planck-Institut für biologische Kybernetik, 1996.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, Cambridge, U.K., 2004.

John Shawe-Taylor, Christopher K. I. Williams, Nello Cristianini, and Jaz 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 and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of 17th International Conference on Machine Learning*, pages 911–918. Morgan Kaufmann, San Francisco, CA, 2000.

L. G. Valiant. A theory of the learnable. *Communications of the Association of Computing Machinery*, 27(11):1134–1142, November 1984.

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

Vladimir N. Vapnik. *Statistical Learning Theory.* Wiley, New York, NY, 1998.

Pascal Vincent and Yoshua Bengio. Kernel matching pursuit. *Machine Learning*, 48:165–187, 2002.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, volume 13, pages 682–688. MIT Press, 2001.