Kernel Methods
Software, Algorithms and Applications

Dissertation

Zur Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften
an der Technischen Universität Wien

eingereicht bei:

1. Beurteiler:
Univ.-Prof. Dr. Kurt Hornik
Institut für Statistik und Mathematik
Wirtschaftsuniversität Wien

2. Beurteiler:
Univ.-Prof. Dr. Friedrich Leisch
Institut für Statistik
Ludwig-Maximilians-Universität München

von

Alexandros Karatzoglou

Wien, im Februar 2006

Kurzfassung

Die vorliegende Arbeit untersucht einen teilbereich des maschinellen Lernens, die Kernmethoden. Nach einer kurzen Präsentation der mathematischen Grundlagen in Kapitel 1 wird in Kapitel 2 das R Erweiterungspaket `kernlab` vorgestellt. Basierend auf dem S4-Klassen Konzept stellt es einen objektorientierten flexiblen Baukasten für Kernmethoden zur Verfügung, und noch dazu Implementierungen von bekannten Kernmethoden wie z.b. Support Vector Machines SVM, Spectral Clustering, und Kernel PCA. Im Kapitel 3 wird die SVM in `kernlab` mit drei anderen SVM-Implementierungen in R in Bezug auf Features und Effizienz verglichen.

Kapitel 4 beschreibt einen neuen Kern-basierten Algorithmus für on-line Training von SVMs in der die Schrittgröße des stochastischen Abstiegs dynamisch adaptiert wird. Eine Anwendung der Methode auf einem Standart-Datensatz des maschinellen Lernens zur Handschrifterkennung bestätigt die Leistungsteigung gegenüber ähnlichen Methoden, die die Schrittgröße nicht dynamisch anpassen. Kapitel 5 stellt Kern-basierte Clusterverfahren für die Gruppierung von Textdokumenten vor. Spectral Clustering und eine Kern-Version des bekannten k-means Verfahrens in zusammenhang mit den für Clusterung von Text speziell entwickelten Kern werden vorgestellt und miteinander verglichen.

Foreword

This thesis was written during the years I was part of the Institute of Statistics of the Vienna University of Technology in the Center for Computational Intelligence (CI).

In this small note I would like to thank those who one way or another have contributed to this thesis. I would first like to express my gratitude to my supervisor Kurt Hornik for his support, guidance and for giving me the freedom to choose my research subject. I am also grateful to Alex Smola for the chance he gave me to visit the Machine Learning Group at the Australian National University and NICTA and for introducing me into the research field of Kernel Methods.

Special thanks go to my second supervisor Friedrich Leisch for his helpful comments on the thesis and to my colleagues at the Institute of Statistics David Meyer, Achim Zeileis and Bettina Grüen. I would also like to thank Evgenia Dimitriadou for being a good colleague and friend all of these years and for her help and mental support especially at the first difficult years in Vienna.

I was privileged to make good friends in Canberra Australia during my visits at the ML group at the ANU. Cheng Soon Ong, Vishy Viswanathan, Greg Rawlings, Ah Young Park and Nic Schraudolph have always been the best of companies and made my stay in Australia very enjoyable.

This thesis was financially supported by the Austrian Science Foundation (FWF) under grand SFB010 “Adaptive Information Systems and Modeling in Economics and Management Science”, by the “Statistical Computing in R” grand of the Vienna University of Economics and Business Administration, and by grands of the National ICT Australia (NICTA) the Academic Society of Austria and the Austrian Research Association (ÖFG).

Last but not least, my gratitude to my parents and my brother for always being there for me, this is for you.

Abstract

This monograph intends to contribute to the area of kernel-based Machine Learning.

After a basic introduction to kernel-based Machine Learning we continue by introducing a software package for kernel-based learning in R. The package provides a range of kernel methods including various formulations of Support Vector Machines, Gaussian processes for classification and regression, a Spectral Clustering implementation, the Relevance Vector Machine for regression, and kernel PCA. The package includes infrastructure for developing kernel methods and to this purpose it also contains implementations of the many popular kernels and functions for fast calculation of kernel expressions along with a quadratic problem solver and a incomplete Cholesky decomposition method.

The second chapter of the thesis presents and compares the Support Vector Machines implementations contained in various R packages.

Chapter three introduces a novel kernel based on-line learning algorithm. The algorithm is derived by utilizing stochastic-meta-decent in order to calculate the learning rate of a simple kernel based stochastic gradient decent decent. We evaluate the algorithm on a character recognition data set.

The fourth chapter presents an application of kernel method on text clustering. We use the kernel k-means and a spectral clustering method along with a string kernel to cluster a set of text documents. The results are then compared to a standard text clustering method.

Contents

Chapter 1

Introduction

1.1 Machine Learning

Machine Learning (Mitchell, 1997) is a discipline closely related to statistical model fitting and is also by and large considered part of the Artificial Intelligence (AI) field (Russell and Norvig, 2002). The main goal in Machine Learning is to extract information from data by creating very flexible models characterized by large number of parameters and automatizing the processes of model fitting as much as possible. The terminology used stems from AI thus terms like "fitting" are replaced by terms like "learning". The field has seen a recent explosive increase in interest since the exponential increase in available computing power has opened many possibilities in the area of both machine learning and statistical modeling. The advent of new application areas like bioinformatics (Baldi and Brunak, 1998) and the increasing availability of large datasets both in size of observations and in dimensionality have inspired new techniques and algorithms, and given rise to successful commercial applications of machine learning, `Google` being the most obvious example.

Machine Learning has many diverse applications, from controlling robots to analyzing medical records but it is mostly the extraction of meaningful structure out of huge amounts of data also know as "data mining" (Witten and Frank, 2005) that drives a big part of the progress in the field. Some machine learning techniques look for a structural description of what is learned, descriptions that can become fairly complex and are typically expressed as sets of rules, as is the case in decision trees (Quinlan, 1993) or association rules (Agrawal and Srikant, 1994). Because these models can be understood by people these descriptions provide a very useful insight into the structure of the data. Other methods like neural networks (Ripley, 1996) or Support Vector Machines (SVM) (Vapnik, 1995) do not provide a structural description of the data which can be easily understood, but tend to provide very good performance on new data examples and can be used directly on structured data like text. Although the ability to produce a human readable model of the data is certainly an advantage for any Machine Learning method, there are applications as in bioinformatics where this is not required and other characteristics (as outright performance) of a learning machine are more important.

Early Machine Learning approaches where based on linear methods like the perceptron algorithm, while non-linear methods where considered complex and where not theoretically well-founded. Kernel methods (Schölkopf and Smola, 2002), (Shawe-Taylor and Christianini, 2004) combine the theoretical well-founded approach previously limited to linear systems with the flexibility and real world application performance typical of nonlinear methods, hence forming a remarkably powerful and robust class of Machine Learning techniques.

1.2 Kernel Methods

In the last ten years machine learning methods based on positive definite kernels have become quite popular. Kernel-based learning methods, use an implicit mapping of the input data into a high dimensional feature space defined by a kernel function, i.e., a function returning the inner product $\langle \Phi(x), \Phi(x') \rangle$ between the images of two data points x, x' in the feature space. The learning then takes place in the feature space, provided the learning algorithm can be expressed so that the data points only appear inside dot products with other points. This is often referred to as the "kernel trick" (Schölkopf and Smola, 2002). More precisely, if a projection $\Phi : X \to H$ is used, the inner product $\langle \Phi(x), \Phi(x') \rangle$ can be represented by a kernel function k

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle, \tag{1.1}$$

which is computationally simpler than explicitly projecting x and x' into the feature space H.

A simple classification algorithm on the data $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ where $\mathcal{Y} = \pm 1$ would be to calculate the mean of the data points belonging to class $\mathcal{Y} = 1$, $c_+ = \frac{1}{n_+} \sum_{i:y_i=+1} \Phi(x_i)$ and the mean of the points belonging to class $\mathcal{Y} = -1$, $c_- = \frac{1}{n_-} \sum_{i:y_i=-1} \Phi(x_i)$ where n_+ and n_- the number of data points in the positive and the negative class. A new point x is then assigned to the closest class through

$$y = \operatorname{sgn}(\langle \Phi(x), c_+ \rangle - \langle \Phi(x), c_- \rangle + b) \tag{1.2}$$

whith $b = \frac{1}{2}(\|c_-\|^2 - \|c_+\|^2)$, and substituting $c_\pm$ gives

$$y = \operatorname{sgn}\left(\frac{1}{n_+}\sum_{i:y_i=+1}\langle\Phi(x),\Phi(x_i)\rangle - \frac{1}{n_-}\sum_{i:y_i=-1}\langle\Phi(x),\Phi(x_i)\rangle + b\right) \tag{1.3}$$

substituting $\langle\Phi(x),\Phi(x_i)\rangle$ with $k(x,x_i)$ this becomes

$$y = \operatorname{sgn}\left(\frac{1}{n_+}\sum_{i:y_i=+1}k(x,x_i) - \frac{1}{n_-}\sum_{i:y_i=-1}k(x,x_i) + b\right) \tag{1.4}$$

where $b = \frac{1}{2}(\frac{1}{n_-^2}(\sum_{(i,j):y_i=y_j=-1}k(x_i,x_j) - \frac{1}{n_+^2}\sum_{(i,j):y_i=y_j=+1}k(x_i,x_j)))$.

A special case of this simple classifier, assuming that the class means have the same distance to the origin ($b = 0$) and that $k(.,x)$ is a density for all $x' \in \mathcal{X}$ and the two classes are equally likely and generated from two probability distributions that are correctly estimated by the Parzen window estimates

$$p_+ := \frac{1}{n_+}\sum_{i:y_i=+1}k(x,x_i), p_- := \frac{1}{n_-}\sum_{i:y_i=-1}k(x,x_i) \tag{1.5}$$

is the Bayes decision rule.

This simple classifier is quite similar to the famous Support Vector Machine, in both cases the data are separated by a hyperplane in the high dimensional feature space. The classifier is linear in the feature space and is represented by a kernel expansion in the input domain.

One interesting property of kernel-based systems is that, once a valid kernel function has been selected, one can practically work in spaces of any dimension without paying any computational cost, since feature mapping is never effectively performed. In fact, one does not even need to know which features are being used.

Another advantage is the that one can design and use a kernel for a particular problem that could be applied directly to the data without the need for a feature extraction process. This is particularly important in problems where a lot of structure of the data is lost by the feature extraction process (e.g., text processing). The inherent modularity of kernel-based learning methods allows one to use any valid kernel on a kernel-based algorithm.

1.2.1 Kernels

As we already mentioned a kernel is a function k which satisfies that for all $x, x' \in \mathcal{X}$

$$k(x,x') = \langle\Phi(x),\Phi(x')\rangle \tag{1.6}$$

where Φ is a mapping from $\mathcal{X}$ to an inner product feature space H, $\Phi : X \rightarrow H$. Given a kernel k and inputs $x_1, \ldots, x_n \in \mathcal{X}$ the $n \times n$ matrix

$$K := (k(x_i, x_j))_{ij} \tag{1.7}$$

is called the kernel matrix (or Gram matrix) of k with respect to $x_1, \ldots, x_n$. A real $n \times n$ matrix K_{ij} satisfying

$$\sum_{i,j} c_i c_j K_{ij} \geq 0 \tag{1.8}$$

for all $c_i \in \mathbb{R}$ is called positive definite. A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ which for all $n \in \mathbb{N}, x_i \in \mathcal{X}$ gives rise to a positive definite kernel matrix is called a positive definite kernel. Positive definite kernels are usually simply referred as kernels. Since

$$\sum_{i,j} c_i c_j \langle \Phi(x_i) \Phi(x_j) \rangle = \left\langle \sum_i c_i \Phi(x_i), \sum_j c_j \Phi(x_j) \right\rangle \geq 0 \tag{1.9}$$

kernels are positive definite for any choice of Φ. Kernels can thus be regarded as generalized inner products. Whilst they are not generally bilinear, they share important properties with dot products, such as the Caych-Schwartz inequality.

1.2.2 Kernel Classes

Due to the growing interest in kernel methods a large number of kernels have been conceived, we will only present some of the most commonly used and interesting families of kernel functions :

RBF kernels The most commonly used kernel class is the radial basis function (RBF) kernels which are kernels that can be written in the form :

$$k(x, x') = f(d(x, x')) \tag{1.10}$$

where $d(x, x')$ is a metric on $\mathcal{X}$ and f is a function in $\mathbb{R}$. Usually the metric arises from the inner product $d(x, x') = \|x - x'\|$ and in this case RBF kernels are also translation invariant i.e. $k(x, x') = k(x + x_0, x' + x_0)$ for all $x_0 \in \mathbb{R}$. The popular Gaussian kernel $k(x, x') = \exp(-\sigma \|x - x'\|^2)$ sugested by (Boser *et al.*, 1992), (Guyon *et al.*, 1993), (Vapnik, 1998) is an RBF kernel. The Gaussian kernels is one of the most commonly used kernels in kernel-based learning, other examples of RBF kernels include the Laplace kernel, $k(\mathbf{x}, \mathbf{x}') = \exp(-\sigma \|\mathbf{x} - \mathbf{x}'\|)$

and the Bessel function of first order kernel $k(\mathbf{x}, \mathbf{x}') = \frac{\mathrm{Bessel}^n_{(\nu+1)}(\sigma\|\mathbf{x}-\mathbf{x}'\|)}{(\|x-x'\|)^{-n(\nu+1)}}$. Most RBF kernels are general purpose kernels and provide good performance on many learning problems.

Polynomial kernels Kernels of the type $k(x, x') = \langle x, x'\rangle^p$ are positive definite for $p \in \mathbb{N}$. The corresponding feature map can be calculated analyticaly

$$\langle x, x'\rangle = \langle\sum_{j=1}^{d}[x]_j[x']_j\rangle^p =$$

$$\sum_{j\in[d]^p}[x]_{j1}, \dots, [x]_{jp} \cdot [x']_{j1}, \dots, [x']_{jp} = \langle C_p(x), C_p(x')\rangle \tag{1.11}$$

where C_p maps $x \in \mathbb{R}^d$ to the vector $C_p(x)$ whose entries are the p-th degree ordered products of the entries of x. The polynomial kernel of degree p computes a inner product in the space of all monomials of degree p of the input coordinates. A variation of this polynomial kernel is the inhomogeneous polynomial function $k(x, x') = (\langle x, x'\rangle + c)^p$ where $c \geq 0$.

Convolution kernels Kernels for structured data (Haussler, 1999), (Watkins, 2000) like strings and trees have gathered allot of attention recently. A data type is said to be structured if there is a natural way to decompose it into smaller parts, for example a string can be decomposed into substrings and a tree can be decomposed into subtrees. The idea behind convolution kernels is to compute the product of sub-kernels comparing the smaller parts before summing over the set of allowed decompositions. Mathematically the set of allowed decompositions can be represented as a relation $R(x_1, \dots, x_2)$ where $x_1, \dots, x_p$ constitute the composite object x. A kernel between composite objects can be defined by building on similarity measures that asses their respective parts, in other words kernels k_p defined on $\mathcal{X}_p \times \mathcal{X}_p$. The R convolution of $k_1, k_2, \dots k_p$ is defined as :

$$[k_1 \star \dots \star k_p] = \sum_{\bar{x}\in R(x), \bar{x'}\in\mathbb{R}(\curvearrowleft')}\prod_{p=1}^{P} k_p(\bar{x}_p, \bar{x}'_p) \tag{1.12}$$

where the sum runs over all possible ways $R(x)$ and $R(x')$ in which we can decompose x and x' into $\bar{x}_1, \dots, \bar{x}_P$ or $\bar{x'}_1, \dots, \bar{x'}_P$.

Specific examples of convolution kernels are ANOVA kernels (Vapnik, 1998). To construct an ANOVA kernel we consider $\mathcal{X} = S^N$ for some set S and kernels $k^{(i)}$ on $S \times S$ where $i = 1, \dots, N$. For $P = 1, \dots, N$ the ANOVA kernel of order P is defined as

$$k_p(x, x') = \sum_{1 \leq i_1 < \cdots < i_P \leq N} \prod_{p=1}^{P} k^{(i_p)}(x_{i_p}, {x'}_{i_p}) \qquad (1.13)$$

ANOVA kernels are performing well in multi-dimensional regression problems (Stitson *et al.*, 1997).

String kernels One of the most popular application of the SVM is text categorization. In many of these applications a so called "bag of words" or "document term matrix" representation is used (Joachims, 1998). In this representations a sparse vector where each component corresponds to the number of time a word occurs in the text is constructed for each document. Using a linear or RBF kernel and an efficient sparse representation, an inner product between the vectors can be computed very quickly. The main drawback of this technique is that it does not take into account the word ordering in the document. A more sophisticated way of dealing with string data was proposed in (Watkins, 2000) and (Haussler, 1999). The idea is based on the convolution kernels and is basicly comparing two string by how many substrings they have in common. Some string kernels like the subsequence kernel (Lodhi *et al.*, 2002) don't even require the substring to be contiguous, and impose a penalty on the distance of the first and the last element of the considered substrings. In (Vishwanathan and Smola, 2002) a fast, linear time implementation of string kernels of the type $k(x, x') = \sum_{c_s} \mathrm{num}_s(x)\mathrm{num}_s(x')c_s$ where $\mathrm{num}_s(x)$ is the number of times string s occurs in x is proposed. This is done by utilizing suffix-trees representations of the string vectors x. For inexact matching substrings a similar kernel has been proposed in (Leslie *et al.*, 2002) where $\mathrm{num}_s(x)$ is replaced by a mismatch function $\mathrm{num}_s(x, \epsilon)$ where ϵ controls the number of mismatches.

Other types of kernels include kernels on trees, since they can be decomposed into subtrees it is fairly simple to design a kernel on them. In Collins and Duffy (2001) a fast decomposition method is proposed mapping a tree into its its subtrees. The kernel is then defined as a weighted sum of all terms between both trees. Kernels on graphs can be defined either between two graphs Thomas Gärtner and Wrobel (2003) or between the vertices of a single graph. Also interesting is the Fisher kernel (Jaakkola and Haussler, 1999) which introduces a metric by constructing a probabilistic model of the data.

1.3 Kernel algorithms and Software

The Support Vector Machine is one of the early success stories of statistical learning theory and arguably the most well known kernel method. The early emergence of the Support vector machine and the following interest in its characteristics and performance contributed to it's success by generating a flow of publications and software implementations not seen since then on any other kernel method. The result of this early interest in the SVM was that a large number of optimized algorithms where conceived (e.g. chunking (Osuna *et al.*, 1997), SMO (Platt, 1998), simpleSVM (Vishwanathan *et al.*, 2003)) for the solution of the SVM optimization problem with very good performance in terms of speed scalability and space complexity. As a result although the original SVM implementation required the solution of a large quadratic problem, it is now used on large datasets without requiring a large computation facility (computer cluster etc.) and several SVM implementations exist on almost all major programming languages many of them providing excellent speed and performance.

A rather large number of new kernel methods exists today some of them with many advantages over the original SVM like direct computation of class-probabilities (Kernel Fisher Discriminant Analysis) (Mika *et al.*, 1999), better sparcenes (e.g. Relevance Vector Machine) (Tipping, 2001) or performance etc. but most of this methods still lack a fast iterative algorithm or a proper software implementation and thus receive little attention. It thus seems that the extensive research the SVM went through was only a product of its early appearance during the emergence of the field of statistical learning research.

One could claim that most new kernel methods do not benefit form this maturity processes the SVM wend through. This also apparent by the slow pace of software implementations for this methods. This is somehow natural, since nowadays many more kernel based algorithms are available and many older linear methods have been kernelized creating a considerable amount of available kernel methods for classification regression dimensionality reduction, ranking and clustering. Some of these methods get a fair share of attention but the research interest in machine learning seems also to be slightly shifting from generating yet another kernel algorithm to taking advantage of the characteristics of kernel methods and dealing with problems directly on structured data (e.g. text documents, HTML data, DNA sequences)

One of the aims of this thesis is to contribute to the maturing processes of some kernel methods through the introduction of a framework for easy and flexible kernel methods development and through the implementation of some of the never kernel methods using this framework.

1.4 Thesis Outline

This monograph aims to contributing to the field of kernel based machine learning research. It consists of four chapters:

Chapter 1 is dedicated to the description of the **kernlab** R package, (Karatzoglou *et al.*, 2005a) which aims to provide a framework for developing kernel-based machine learning methods, provides some basic tools and also includes some implementations of algorithms for classification, clustering, ranking, and dimensionality reduction.

Chapter 2 provides a qualitative study of the SVM software included in various R packages. The aim is to give a clear picture of the relative advantages of the various SVM implementations. This chapter mainly covers Karatzoglou *et al.* (2006)

Chapter 3 introduces a novel on-line learning algorithm which utilizes stochastic meta decent in order to estimate an optimal learning rate at each iteration. This allows the algorithm to outperform standard on-line algorithms particularly on non-stationary streams of data. This chapter mainly covers (Karatzoglou *et al.*, 2005b)

Chapter 4 describes an application of kernel methods on text mining. Spectral clustering and kernel k-means is used to cluster a set of text document. The performance of this methods compared to more traditional approaches is encouraging. This chapter covers results presented at the GFKL 2006 conference.

The appendix contains a detailed description of some SVM formulations and the **kernlab** user manual.

Chapter 2

kernlab – An S4 package for kernel methods in R

2.1 Introduction

In this chapter we present a software package for kernel-based learning. **kernlab** is an extensible package for kernel-based machine learning methods in R (R Development Core Team, 2005). It takes advantage of R's new S4 object model and provides a framework for creating and using kernel-based algorithms. The package contains dot product primitives (kernels), implementations of support vector machines and the relevance vector machine, Gaussian processes, a ranking algorithm, kernel PCA, kernel CCA, kernel feature analysis, on-line kernel methods and a spectral clustering algorithm. Moreover it provides a general purpose quadratic programming solver, and an incomplete Cholesky decomposition method. With this package we hope to facilitate the use and development of kernel-based methods in R.

2.1.1 Software Review

Support vector machines are currently used in a wide range of fields, from bioinformatics to astrophysics. Thus, the existence of many SVM software packages comes as little surprise. Most existing software is written in C or C++, such as the award winning `libsvm` (Chang and Lin, 2001), which provides a robust and fast SVM implementation and produces state of the art results on most classification and regression problems (Meyer *et al.*, 2003), `SVMlight` (Joachims, 1999), `SVMTorch`, `Royal Holloway Support Vector Machines`, (Gammerman *et al.*, 2001), `mySVM` (Rüping, 2004), and `M-SVM` (Guermeur, 2004). Many packages provide interfaces to MATLAB (Math-

Works, 2005) (such as `libsvm`), and there are some native MATLAB toolboxes as well such as the `SVM and Kernel Methods Matlab Toolbox` (Canu *et al.*, 2003) or the `MATLAB Support Vector Machine Toolbox` (Gunn, 1998) and the `SVM toolbox for Matlab` (Schwaighofer, 2005) Putting SVM specific software aside and considering the abundance of other kernel-based algorithms published nowadays, there is little software available implementing a wider range of kernel methods with some exceptions like the `Spider`(Weston *et al.*, 2005) software which provides a MATLAB interface to various C/C++ SVM libraries and MATLAB implementations of various kernel-based algorithms, `Torch` (Collobert *et al.*, 2002) which also includes more traditional machine learning algorithms, and the occasional MATLAB or C program found on a personal web page where an author includes code from a published paper.

2.1.2 R Software

The **e1071** R package offers an interface to the award winning `libsvm` (Chang and Lin, 2001), a very efficient SVM implementation. `libsvm` provides a robust and fast SVM implementation and produces state of the art results on most classification and regression problems (Meyer *et al.*, 2003). The R interface provided in **e1071** adds all standard R functionality like object orientation and formula interfaces to `libsvm`. Another SVM related R package which was made recently available is **klaR** (Roever *et al.*, 2005) which includes an interface to `SVMlight`, a popular SVM implementation along with other classification tools like Regularized Discriminant Analysis.

However, most of the `libsvm` and **klaR** SVM code is in C++. Therefore, if one would like to extend or enhance the code with e.g. new kernels or different optimizers, one would have to modify the core C++ code.

2.2 kernlab

kernlab aims to provide the R user with basic kernel functionality (e.g., like computing a kernel matrix using a particular kernel), along with some utility functions commonly used in kernel-based methods like a quadratic programming solver, and modern kernel-based algorithms based on the functionality that the package provides. Taking advantage of the inherent modularity of kernel-based methods, **kernlab** aims to allow the user to switch between kernels on an existing algorithm and even create and use own kernel functions for the kernel methods provided in the package.

2.2.1 S4 objects

kernlab uses R's new object model described in "Programming with Data" (Chambers, 1998) which is known as the S4 class system and is implemented in the **methods** package.

In contrast with the older S3 model for objects in R, classes, slots, and methods relationships must be declared explicitly when using the S4 system. The number and types of slots in an instance of a class have to be established at the time the class is defined. The objects from the class are validated against this definition and have to comply to it at any time. S4 also requires formal declarations of methods, unlike the informal system of using function names to identify a certain method in S3.

An S4 method is declared by a call to `setMethod` along with the name and a "signature" of the arguments. The signature is used to identify the classes of one or more arguments of the method. Generic functions can be declared using the `setGeneric` function. Although such formal declarations require package authors to be more disciplined then when using the informal S3 classes, they provide assurance that each object in a class has the required slots and that the names and classes of data in the slots are consistent.

An example of a class used in **kernlab** is shown below. Typically, in a return object we want to include information on the result of the method along with additional information and parameters. Usually **kernlab**'s classes include slots for the kernel function used and the results and additional useful information.

```
setClass("specc",
         representation("vector",           # cluster indexes vector
                        centers="matrix",     # the cluster centers
                        size="vector",        # size of each cluster
                        kernelf="function",   # kernel function used
                        withinss = "vector"), # within cluster sum
         prototype = structure(.Data = vector(), # of squares
                              centers = matrix(),
                              size = matrix(),
                              kernelf = ls,
                              withinss = vector()))
```

Accessor and assignment function are defined and used to access the content of each slot which can be also accessed with the `@` operator.

2.2.2 Namespace

Namespaces were introduced in `R` 1.7 and provide a means for packages to control the way global variables and methods are being made available. Due to the number of assignment and accessor function involved, a namespace is used to control the methods which are being made visible outside the package. Since `S4` methods are being used, the **kernlab** namespace also imports methods and variables form the **methods** package.

2.2.3 Data

The **kernlab** package also includes data sets which will be used to illustrate the methods included in the package. The `spam` data set (Hastie *et al.*, 2001) set collected at Hewlett-Packard Labs classifies 4601 e-mails as spam or non-spam. The 57 variables of each data vector indicate the frequency of certain words and characters in the e-mail. The data set contains 2788 and 1813 e-mails classified as non-spam and spam, respectively.

Another data set included in **kernlab** the `income` data set (Hastie *et al.*, 2001) is taken by a marketing survey in the San Francisco Bay concerning the income of shopping mall customers. It consists of 14 demographic attributes (nominal and ordinal variables) including the income and 8993 observations.

The `ticdata` data set (Putten *et al.*, 2000) was used in the 2000 Coil Challenge and contains information on customers of an insurance company. The data consists of 86 variables and includes product usage data and socio-demographic data derived from zip area codes. The data was collected to answer the following question: Can you predict who would be interested in buying a caravan insurance policy and give an explanation why?

The `promotergene` data set is a data set of E. Coli promoter gene sequences (DNA) with 106 observations and 58 variables available at the UCI Machine Learning repository. Promoters have a region where a protein (RNA polymerase) must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align. The data contains DNA sequences of promoters and non-promoters.

The `spirals` data set was created by the `mlbench.spirals` function in the **mlbench** package (Leisch and Dimitriadou, 2001). This two-dimensional data set with 300 data points consists of two spirals where Gaussian noise is added to each data point.

2.2.4 Kernels

A kernel function k calculates the inner product of two vectors x, x' in a given feature mapping $\Phi : X \to H$. The notion of a kernel is obviously central in the making of any kernel-based algorithm and consequently also in any software package containing kernel-based methods.

Kernels in **kernlab** are S4 objects of class kernel extending the function class with one additional slot containing a list with the kernel hyper-parameters. Package **kernlab** includes 7 different kernel classes which all contain the class kernel and are used to implement the existing kernels. These classes are used in the function dispatch mechanism of the kernel utility functions described below. Existing kernel functions are initialized by "creator" functions. All kernel functions take two feature vectors as parameters and return the scalar dot product of the vectors. An example of the functionality of a kernel in **kernlab**:

```
> rbf <- rbfdot(sigma = 0.05)
> rbf

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.05

> x <- rnorm(10)
> y <- rnorm(10)
> rbf(x, y)

          [,1]
[1,] 0.3129730
```

The package includes implementations of the following kernels:

- the linear "vanilladot" kernel implements the simplest of all kernel functions
$$k(x, x') = \langle x, x' \rangle \tag{2.1}$$
which is useful specially when dealing with large sparse data vectors x as is usually the case in text categorization.

- the Gaussian radial basis function "rbfdot"
$$k(x, x') = \exp(-\sigma \|x - x'\|^2) \tag{2.2}$$
which is a general purpose kernel and is typically used when no further prior knowledge is available about the data.

- the polynomial kernel “polydot”

$$k(x, x') = \left(\text{scale} \cdot \langle x, x' \rangle + \text{offset}\right)^{\text{degree}}. \tag{2.3}$$

  which is used in classification of images.

- the hyperbolic tangent kernel “tanhdot”

$$k(x, x') = \tanh\left(\text{scale} \cdot \langle x, x' \rangle + \text{offset}\right) \tag{2.4}$$

  which is mainly used as a proxy for neural networks.

- the Bessel function of the first kind kernel “besseldot”

$$k(x, x') = \frac{\text{Bessel}^{n}_{(\nu+1)}(\sigma\|x - x'\|)}{(\|x - x'\|)^{-n(\nu+1)}}. \tag{2.5}$$

  is a general purpose kernel and is typically used when no further prior knowledge is available and mainly popular in the Gaussian Process community.

- the Laplace radial basis kernel “laplacedot”

$$k(x, x') = \exp(-\sigma\|x - x'\|) \tag{2.6}$$

  which is a general purpose kernel and is typically used when no further prior knowledge is available.

- the ANOVA radial basis kernel “anovadot”

$$k(x, x') = \left(\sum_{k=1}^{n} \exp(-\sigma(x^k - y^k)^2)\right)^d \tag{2.7}$$

  which performs well in multidimensional regression problems.

- the linear splines kernel in one dimension “splinedot”

$$k(x, x') = 1 + xx' \min(x, x') - \frac{x + x'}{2}(\min(x, x')^2 + \frac{(\min(x, x')^3)}{3} \tag{2.8}$$

  and for the multidimensional case $k(\mathbf{x}, \mathbf{x}') = \prod_{k=1}^{n} k(x^k, x'^k)$. which also performs well in multidimensional regression problems.

2.2.5 Kernel Utility Methods

The package also includes methods for computing commonly used kernel expressions (e.g., the Gram matrix). These methods are written in such a way that they take functions (i.e., kernels) and matrices (i.e., vectors of patterns) as arguments. These can be either the kernel functions already included in **kernlab** or any other function implementing a valid dot product (taking two vector arguments and returning a scalar). In case one of the already implemented kernels is used, the function calls a vectorized implementation of the corresponding function. Moreover, in the case of symmetric matrices (e.g., the dot product matrix of a Support Vector Machine) they only require one argument rather than having to pass the same matrix twice (for rows and columns).

The computations for the kernels already available in the package are vectorized whenever possible which guarantees good performance and acceptable memory requirements. Users can define their own kernel by creating a function which takes two vectors as arguments (the data points) and returns a scalar (the dot product). This function can then be based as an argument to the kernel utility methods. For a user defined kernel the dispatch mechanism calls a generic method implementation which calculates the expression by passing the kernel function through a pair of `for` loops. The kernel methods included are:

`kernelMatrix` This is the most commonly used function. It computes $k(x, x')$, i.e., it computes the matrix K where $K_{ij} = k(x_i, x_j)$ and x is a *row* vector. In particular,

```
K <- kernelMatrix(kernel, x)
```

computes the matrix $K_{ij} = k(x_i, x_j)$ where the x_i are the columns of X and

```
K <- kernelMatrix(kernel, x1, x2)
```

computes the matrix $K_{ij} = k(x1_i, x2_j)$.

`kernelFast` This method is different to `kernelMatrix` for `rbfdot`, `besseldot`, and the `laplacedot` kernel, which are all RBF kernels. It is identical to `kernelMatrix`, except that it also requires the squared norm of the first argument as additional input. It is mainly used in kernel algorithms, where columns of the kernel matrix are computed per invocation. In these cases, evaluating the norm of each column-entry as it is done on a `kernelMatrix` invocation on an RBF kernel, over and over again would cause significant computational overhead. Its invocation is via

```
K = kernelFast(kernel, x1, x2, a)
```

Here a is a vector containing the squared norms of $x1$.

`kernelMult` is a convenient way of computing kernel expansions. It returns the vector $f = (f(x_1), \dots, f(x_m))$ where

$$f(x_i) = \sum_{j=1}^{m} k(x_i, x_j)\alpha_j, \text{ hence } f = K\alpha. \tag{2.9}$$

The need for such a function arises from the fact that K may sometimes be larger than the memory available. Therefore, it is convenient to compute K only in stripes and discard the latter after the corresponding part of $K\alpha$ has been computed. The parameter `blocksize` determines the number of rows in the stripes. In particular,

```
f <- kernelMult(kernel, x, alpha)
```

computes $f_i = \sum_{j=1}^{m} k(x_i, x_j)\alpha_j$ and

```
f <- kernelMult(kernel, x1, x2, alpha)
```

computes $f_i = \sum_{j=1}^{m} k(x1_i, x2_j)\alpha_j$.

`kernelPol` is a method very similar to `kernelMatrix` with the only difference that rather than computing $K_{ij} = k(x_i, x_j)$ it computes $K_{ij} = y_i y_j k(x_i, x_j)$. This means that

```
K <- kernelPol(kernel, x, y)
```

computes the matrix $K_{ij} = y_i y_j k(x_i, x_j)$ where the x_i are the columns of x and y_i are elements of the vector y. Moreover,

```
K <- kernelPol(kernel, x1, x2, y1, y2)
```

computes the matrix $K_{ij} = y1_i y2_j k(x1_i, x2_j)$. Both `x1` and `x2` may be matrices and `y1` and `y2` vectors.

An example using these functions :

```
> poly <- polydot(degree = 2)
> x <- matrix(rnorm(60), 6, 10)
> y <- matrix(rnorm(40), 4, 10)
> kx <- kernelMatrix(poly, x)
> kxy <- kernelMatrix(poly, x, y)
```

2.3 Kernel Methods

Providing a solid base for creating kernel-based methods is part of what we are trying to achieve with this package, the other being to provide a wider range of kernel-based methods in R. In the rest of the chapter we present the kernel-based methods available in **kernlab**. All the methods in **kernlab** can be used with any of the kernels included in the package as well as with any valid user-defined kernel. User defined kernel functions can be passed to existing kernel-methods in the `kernel` argument.

2.3.1 Support Vector Machine

Since the Support Vector Machine implementation in **kernlab** is covered extensivly in chapter 3 we will only include a sort description in this section.

Support vector machines (Vapnik, 1998) have gained prominence in the field of machine learning and pattern classification and regression. **kernlab**'s implementation of support vector machines, `ksvm`, is based on the optimizers found in `bsvm` (Hsu and Lin, 2002c) and `libsvm` (Chang and Lin, 2001) which includes an very efficient version of the Sequential Minimization Optimization (SMO).

The SVM implementation in `ksvm` includes SVM various formulation for classification including the C-SVM classification algorithm along with the ν-SVM classification formulation which is equivalent to the former but has a more natural (ν) model parameter taking values in $[0, 1]$ and is proportional to the fraction of support vectors found in the data set and the training error.

For classification problems which include more then two classes (multi-class) a one-against-one or pairwise classification method (Knerr *et al.*, 1990; Kreßel, 1999) is used. Furthermore the `ksvm` implementation provides the ability to produce class probabilities as output instead of class labels.

Another approach for multi-class classification supported by the `ksvm` function is the one proposed in (Crammer and Singer, 2000). One-class classification or novelty detection (Schölkopf *et al.*, 1999; Tax and Duin, 1999), where essentially an SVM detects outliers in a data set, is another algorithm supported by `ksvm`. Furthermore, ϵ-SVM (Vapnik, 1995) and ν-SVM (Schölkopf *et al.*, 2000) regression are also available.

The problem of model selection is partially addressed by an empirical observation for the popular Gaussian RBF kernel (Caputo *et al.*, 2002), where the optimal values of the hyper-parameter of sigma are shown to lie in between the 0.1 and 0.9 quantile of the $\|x - x'\|$ statistics. The `sigest` function uses a sample of the training set to estimate the quantiles and returns a vector containing the values of the quantiles. Pretty much any value within this interval leads to good performance.

An example for the `ksvm` function is shown below.

```
> data(promotergene)
> tindex <- sample(1:dim(promotergene)[1],
+     5)
> genetrain <- promotergene[-tindex, ]
> genetest <- promotergene[tindex, ]
> gene <- ksvm(Class ~ ., data = genetrain,
+     kernel = "rbfdot", kpar = "automatic",
+     C = 60, cross = 3, prob.model = TRUE)

Using automatic sigma estimation (sigest) for RBF or laplace kernel

> gene

Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 60

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0158476658476658

Number of Support Vectors : 89
Training error : 0
Cross validation error : 0.108437
Probability model included.

> predict(gene, genetest)
```

```
[1] + + + + +
Levels: + -

> predict(gene, genetest, type = "probabilities")

             +           -
[1,] 0.9955643 0.004435729
[2,] 0.9082671 0.091732929
[3,] 0.7248942 0.275105826
[4,] 0.5031184 0.496881647
[5,] 0.5157837 0.484216310
```

2.3.2 Relevance Vector Machine

The relevance vector machine (Tipping, 2001) is a probabilistic sparse kernel model identical in functional form to the SVM making predictions based on a function of the form

$$y(x) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}, \mathbf{x}_n) + a_0 \tag{2.10}$$

where α_n are the model "weights" and $K(\cdot\,,\cdot\,)$ is a kernel function. It adopts a Bayesian approach to learning, by introducing a prior over the weights α

$$p(\alpha, \beta) = \prod_{i=1}^{m} N(\beta_i \mid 0, a_i^{-1}) \mathrm{Gamma}(\beta_i \mid \beta_\beta, \alpha_\beta) \tag{2.11}$$

governed by a set of hyper-parameters β, one associated with each weight, whose most probable values are iteratively estimated for the data. Sparsity is achieved because in practice the posterior distribution in many of the weights is sharply peaked around zero. Furthermore, unlike the SVM classifier, the non-zero weights in the RVM are not associated with examples close to the decision boundary, but rather appear to represent "prototypical" examples. These examples are termed *relevance vectors*.

kernlab currently has an implementation of the RVM based on a type II maximum likelihood method for which can be used for regression. The functions returns an S4 object containing the model parameters along with indexes for the relevance vectors and the kernel function and hyper-parameters used.

```
> rvmm <- rvm(x, y)
> rvmm
> x <- rbind(matrix(rnorm(120), , 2), matrix(rnorm(120,
+     mean = 3), , 2))
> y <- matrix(c(rep(1, 60), rep(-1, 60)))
> svp <- ksvm(x, y, type = "C-svc")

Using automatic sigma estimation (sigest) for RBF or laplace kernel

> plot(svp)
```

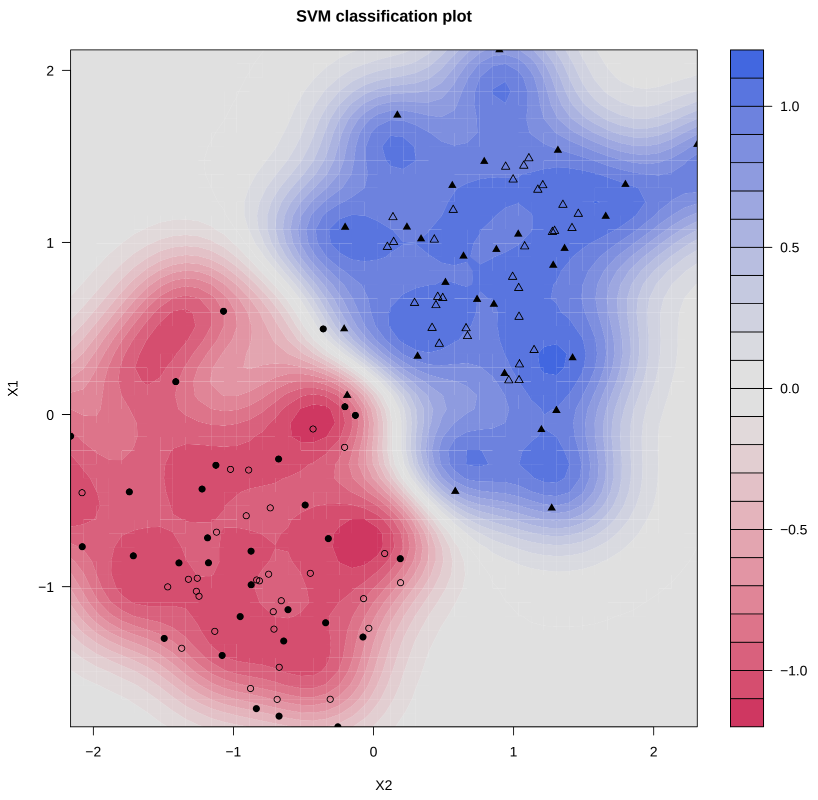

Figure 2.1: A contour plot of the SVM decision values for a toy binary classification problem using the plot function

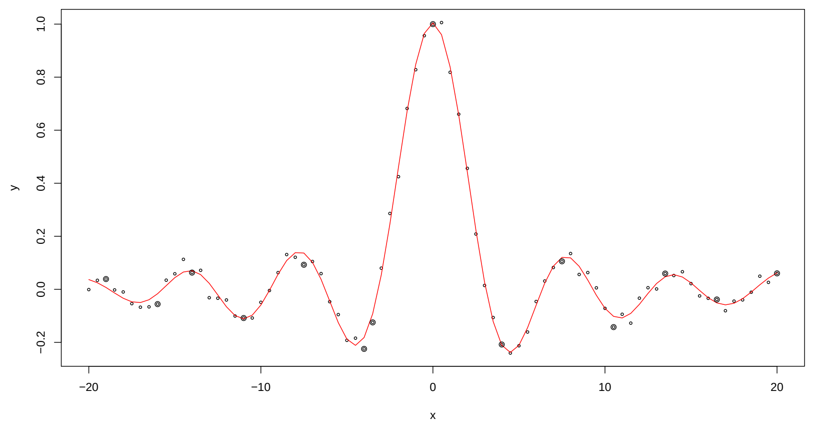

Figure 2.2: Relevance vector regression on data points created by the *sinc*(x) function, relevance vectors are shown circled.

```
Relevance Vector Machine object of class "rvm"
Problem type: regression

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.1

Number of Relevance Vectors : 14
Variance :  0.000581305
Training error : 0.000487676
Cross validation error : -1

> ytest <- predict(rvmm, x)
```

2.3.3 Gaussian Processes

Gaussian Processes (Williams and Rasmussen, 1995) are based on the “prior” assumption that adjacent observations should convey information about each other. In particular, it is assumed that the observed variables are normal, and that the coupling between them takes place by means of the covariance matrix of a normal distribution. Using the kernel matrix as the covariance matrix is a convenient way of extending Bayesian modeling of linear estimators to nonlinear situations. Furthermore it represents the counterpart of the “kernel trick” in methods minimizing the regularized risk.

For regression estimation we assume that rather then observing $t(x_i)$ we observe $y_i = t(x_i) + \xi_i$ where ξ_i is assumed to be independed Gaussian distributed noise with zero mean. The posterior distribution is given by

$$p(\mathbf{y} \mid \mathbf{t}) = \left[\prod_i p(y_i - t(x_i))\right] \frac{1}{\sqrt{(2\pi)^m \det(K)}} \exp\left(\frac{1}{2}\mathbf{t}^T K^{-1}\mathbf{t}\right) \tag{2.12}$$

and after substituting $\mathbf{t} = K\alpha$ and taking logarithms

$$\ln p(\alpha \mid \mathbf{y}) = -\frac{1}{2\sigma^2}\|\mathbf{y} - K\alpha\|^2 - \frac{1}{2}\alpha^T K\alpha + c \tag{2.13}$$

and maximizing $\ln p(\alpha \mid \mathbf{y})$ for α to obtain the maximum a posteriori approximation yields

$$\alpha = (K + \sigma^2\mathbf{1})^{-1}\mathbf{y} \tag{2.14}$$

Knowing α allows for prediction of y at a new location x through $y = K(x, x_i)\alpha$. In similar fashion Gaussian Processes can be used for classification.

`gausspr` is the function in **kernlab** implementing Gaussian processes for classification and regression.

2.3.4 Ranking

The success of Google has vividly demonstrated the value of a good ranking algorithm in real world problems. **kernlab** includes a ranking algorithm based on work published in (Zhou *et al.*, 2003). This algorithm exploits the geometric structure of the data in contrast to the more naive approach which uses the Euclidean distances or inner products of the data. Since real world data are usually highly structured, this algorithm should perform better than a simpler approach based on a Euclidean distance measure.

First, a weighted network is defined on the data and an authoritative score is assigned to every point. The query points act as source nodes that continually pump their scores to the remaining points via the weighted network, and the remaining points further spread the score to their neighbors. The spreading process is repeated until convergence and the point are ranked according to the scores they received.

Suppose we are given a set of data points $X = x_1, \ldots, x_s, x_{s+1}, \ldots, x_m$ in $\mathbb{R}^n$ where the first s points are the query points and the rest are the points to be ranked. The algorithm works by connecting the two nearest points iteratively until a connected graph $G = (X, E)$ is obtained where E is the set of edges. The affinity matrix K defined e.g. by $K_{ij} = \exp(-\sigma\|x_i - x_j\|^2)$

if there is an edge $e(i,j) \in E$ and 0 for the rest and diagonal elements. The matrix is normalized as $L = D^{-1/2}KD^{-1/2}$ where $D_{ii} = \sum_{j=1}^{m} K_{ij}$, and

$$f(t+1) = \alpha L f(t) + (1-\alpha)y \tag{2.15}$$

is iterated until convergence, where α is a parameter in $[0,1)$. The points are then ranked according to their final scores $f_i(t_f)$.

kernlab includes an `S4` method implementing the ranking algorithm. The algorithm can be used both with an edge-graph where the structure of the data is taken into account, and without which is equivalent to ranking the data by their distance in the projected space.

2.3.5 Online Learning with Kernels

The `onlearn` function in **kernlab** implements the online kernel algorithms for classification, novelty detection and regression descibed in (Kivinen *et al.*, 2004a). In batch learning, it is typically assumed that all the examples are immediately available and are drawn independently from some distribution P. One natural measure of quality for some f in that case is the expected risk

$$R[f,P] := E_{(x,y)\ P}[l(f(x),y)] \tag{2.16}$$

Since usually P is unknown a standard approach is to instead minimize the empirical risk

$$R_{emp}[f,P] := \frac{1}{m}\sum_{t=1}^{m} l(f(x_t),y_t) \tag{2.17}$$

Minimizing $R_{emp}[f]$ may lead to overfitting (complex functions that fit well on the training data but do not generalize to unseen data). One way to avoid this is to penalize complex functions by instead minimizing the regularized risk.

$$R_{reg}[f,S] := R_{reg,\lambda}[f,S] := R_{emp}[f] = \frac{\lambda}{2}\|f\|_H^2 \tag{2.18}$$

where $\lambda > 0$ and $\|f\|_H = \langle f,f\rangle_H^{\frac{1}{2}}$ does indeed measure the complexity of f in a sensible way. The constant λ needs to be chosen appropriately for each problem. Since in online learning one is interested in dealing with one example at the time the definition of an instantaneous regularized risk on a single example is needed

$$R_inst[f,x,y] := R_{inst,\lambda}[f,x,y] := R_{reg,\lambda}[f,((x,y))] \tag{2.19}$$

```
> data(spirals)
> ran <- spirals[rowSums(abs(spirals) <
+     0.55) == 2, ]
> ranked <- ranking(ran, 54, kernel = "rbfdot",
+     kpar = list(sigma = 100), edgegraph = TRUE)
> ranked[54, 2] <- max(ranked[-54, 2])
> c <- 1:86
> op <- par(mfrow = c(1, 2), pty = "s")
> plot(ran)
> plot(ran, cex = c[ranked[, 3]]/40)
```

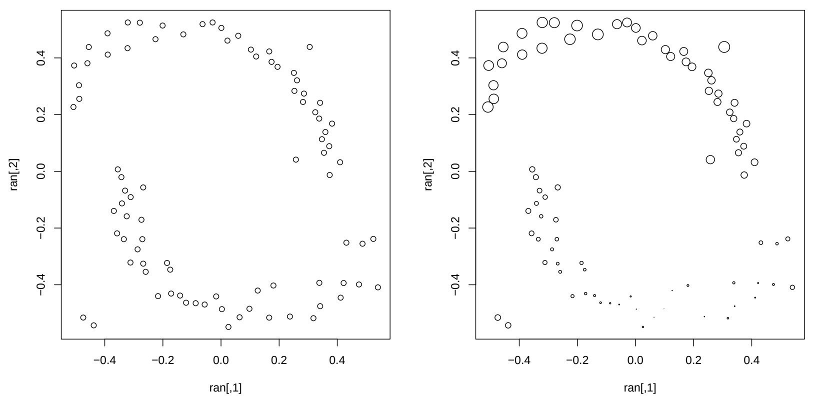

Figure 2.3: The points on the left are ranked according to their similarity to the upper most left point. Points with a higher rank appear bigger. Instead of ranking the points on simple Euclidean distance the structure of the data is recognized and all points on the upper structure are given a higher rank although further away in distance then points in the lower structure.

The implemented algorithms are classical stochastic gradient descent algorithms performing gradient descent on the instantaneous risk. The general form of the update rule is :

$$f_{t+1} = f_t - \eta \partial_f R_{inst,\lambda}[f, x_t, y_t]|_{f=f_t} \tag{2.20}$$

where $f_i \in H$ and $\partial_f <$ is short hand for $\partial\ \partial f$ (the gradient with respect to f) and $\eta_t > 0$ is the learning rate. Due to the learning taking place in a *reproducing kernel Hilbert space* H the kernel k used has the property $\langle f, k(x, \cdot\,)\rangle_H = f(x)$ and therefore

$$\partial_f l(f(x_t)), y_t) = l'(f(x_t), y_t)k(x_t, \cdot\,) \tag{2.21}$$

where $l'(z, y) := \partial_z l(z, y)$. Since $\partial_f \|f\|_H^2 = 2f$ the update becomes

$$f_{t+1} := (1 - \eta\lambda)f_t - \eta_t \lambda'(f_t(x_t), y_t)k(x_t, \cdot\,) \tag{2.22}$$

The `onlearn` function implements the online learning algorithm for regression, classification and novelty detection. The online nature of the algorithm requires a different approach to the use of the function. An object is used to store the state of the algorithm at each iteration t this object is passed to the function as an argument and is returned at each iteration $t + 1$ containing the model parameter state at this step. An empty object of class `onlearn` is initialized using the `inlearn` function.

```
> x <- rbind(matrix(rnorm(90), , 2), matrix(rnorm(90) +
+     3, , 2))
> y <- matrix(c(rep(1, 45), rep(-1, 45)),
+     , 1)
> on <- inlearn(2, kernel = "rbfdot", kpar = list(sigma = 0.2),
+     type = "classification")
> ind <- sample(1:90, 90)
> for (i in ind) on <- onlearn(on, x[i,
+     ], y[i], nu = 0.03, lambda = 0.01)
> sign(predict(on, x[c(1:10, 81:90), ]))

 [1]  1  1  1  1  1  1  1  1  1  1 -1  1 -1 -1 -1
[16] -1 -1 -1 -1 -1
```

2.3.6 Spectral Clustering

Spectral clustering (Ng *et al.*, 2001b) is a recently emerged promising alternative to common clustering algorithms. In this method one uses the top eigenvectors of a matrix created by some similarity measure to cluster the data. Similarly to the ranking algorithm, an affinity matrix is created out from the data as

$$K_{ij} = \exp(-\sigma \|x_i - x_j\|^2) \tag{2.23}$$

and normalized as $L = D^{-1/2} K D^{-1/2}$ where $D_{ii} = \sum_{j=1}^{m} K_{ij}$. Then the top k eigenvectors (where k is the number of clusters to be found) of the affinity matrix are used to form an $n \times k$ matrix Y where each column is normalized again to unit length. Treating each row of this matrix as a data point, `kmeans` is finally used to cluster the points.

kernlab includes an S4 method called `specc` implementing this algorithm which can be used through an formula interface or a matrix interface. The S4 object returned by the method extends the class "vector" and contains the assigned cluster for each point along with information on the centers size and within-cluster sum of squares for each cluster. In case a Gaussian RBF kernel is being used a model selection process can be used to determine the optimal value of the σ hyper-parameter. For a good value of σ the values of Y tend to cluster tightly and it turns out that the within cluster sum of squares is a good indicator for the "quality" of the sigma parameter found. We then iterate through the sigma values to find an optimal value for σ.

2.3.7 Kernel Principal Components Analysis

Principal Component Analysis (PCA) is a powerful technique for extracting structure from possibly high-dimensional datasets. PCA is an orthogonal transformation of the coordinate system in which we describe the data. The new coordinates by which we represent the data are called principal components. Kernel PCA (Schölkopf *et al.*, 1998) performs a nonlinear transformation of the coordinate system by finding principal components which are nonlinearly related to the input variables. Given a set of centered observations x_k, $k = 1, \dots, M$, $x_k \in \mathbb{R}^N$, PCA diagonalizes the covariance matrix $C = \frac{1}{M}\sum_{j=1}^{M} x_j x_j^T$ by solving the eigenvalue problem $\lambda \mathbf{v} = C\mathbf{v}$. The same computation can be done in a dot product space F which is related to the input space by a possibly nonlinear map $\Phi : \mathbb{R}^N \to F$, $x \mapsto \mathbf{X}$. Assuming that we deal with centered data and use the covariance matrix in F,

$$\hat{C} = \frac{1}{C} \sum_{j=1}^{N} \Phi(x_j)\Phi(x_j)^T \tag{2.24}$$

the kernel principal components are then computed by taking the eigenvectors of the centered kernel matrix $K_{ij} = \langle \Phi(x_j), \Phi(x_j) \rangle$.

```
> data(spirals)
> sc <- specc(spirals, centers = 2)
> plot(spirals, col = sc)
```

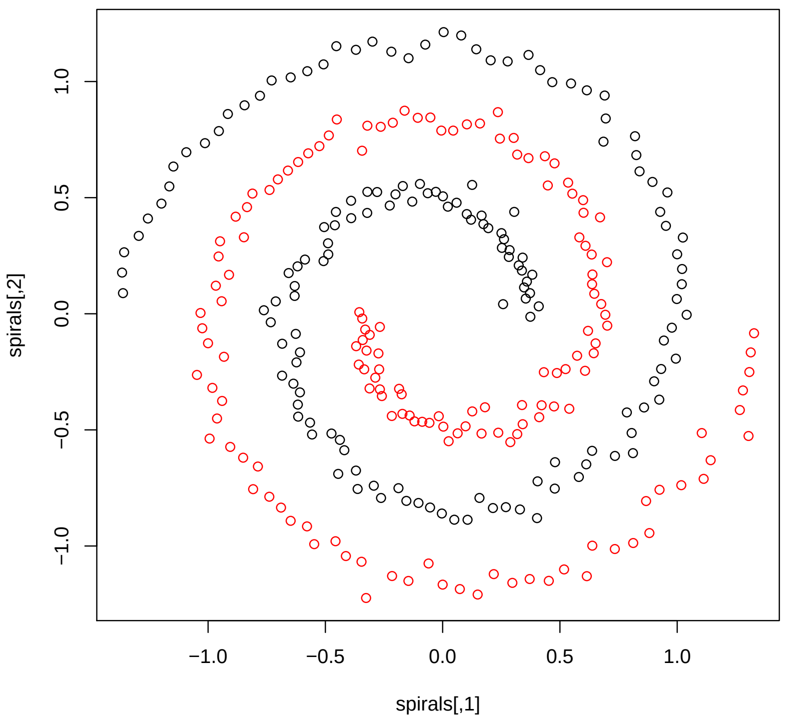

Figure 2.4: Clustering the two spirals data set with `specc`

`kpca`, the the function implementing KPCA in **kernlab**, can be used both with a formula and a matrix interface, and returns an S4 object of class `kpca` containing the principal components the corresponding eigenvalues along with the projection of the training data on the new coordinate system. Furthermore, the `predict` function can be used to embed new data points into the new coordinate system.

2.3.8 Kernel Feature Analysis

Whilst KPCA leads to very good results there are nevertheless some issues to be addressed. First the computational complexity of the standard version of KPCA, the algorithm scales $O(m^3)$ and secondly the resulting feature extractors are given as a dense expansion in terms of the of the training patterns. Sparse solutions are often achieven in supervised learning settings by using an l_1 penalty on the expansion coefficients. An algorithm can be derived using the same approach in feature extraction requiring only n basis functions to compute the first n feature. Kernel feature analysis (Smola *et al.*, 2000) is computationaly simple and scales approximately one order of magnitute better on large data sets then standard KPCA. Choosing $\Omega[f] = \sum_{i=1}^{m} |\alpha_i|$ this yields

$$F_{LP} = \{\mathbf{w}|\mathbf{w} = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \text{with} \sum_{i=1}^{m} |\alpha_i| \leq 1\} \tag{2.25}$$

This setting leads to the first "principal vector" in the l_1 context

$$\nu^1 = \text{argmax}_{\nu \in F_{LP}} \frac{1}{m} \sum_{i=1}^{m} \langle \nu, \mathbf{\Phi}(x_i) - \frac{1}{m} \sum_{j=1}^{m} \mathbf{\Phi}(x_i) \rangle^2 \tag{2.26}$$

Subsequent "principal vectors" can be defined by enforcing optimality with respect to the remaining orthogonal subspaces. Due to the l_1 constrain the solution has the favorable property of beeing sparse in terms of the coefficients α_i.

The function `kfa` in **kernlab** implements Kernel Feature Analysis by using a projection pursuit technique on a sample of the data. Results are then returned in an S4 object.

2.3.9 Kernel Canonical Correlation Analysis

Canonical Correlation Analysis (CCA) is concerned with describing the linear relations between variables. If we have two data sets x_1 and x_2, then the

```
> data(spam)
> train <- sample(1:dim(spam)[1], 400)
> kpc <- kpca(~., data = spam[train, -58],
+     kernel = "rbfdot", kpar = list(sigma = 0.001),
+     features = 2)
> kpcv <- pcv(kpc)
> plot(rotated(kpc), col = as.integer(spam[train,
+     58]), xlab = "1st Principal Component",
+     ylab = "2nd Principal Component")
```

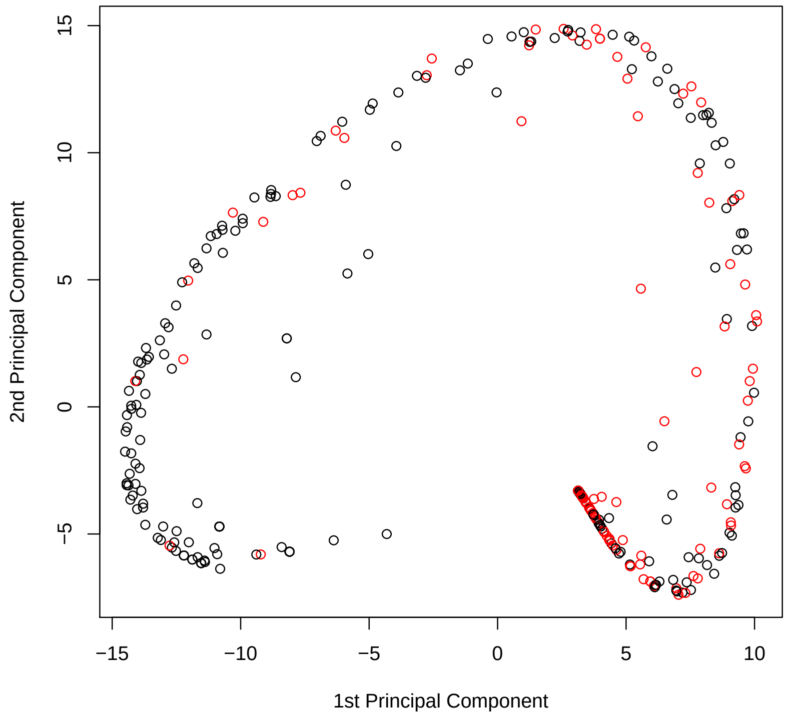

Figure 2.5: Projection of the spam data on two kernel principal components using an RBF kernel

```
> data(promotergene)
> f <- kfa(~., data = promotergene, features = 2,
+     kernel = "rbfdot", kpar = list(sigma = 0.013))
> plot(predict(f, promotergene), col = as.numeric(promotergene[,
+     1]), xlab = "1st Feature", ylab = "2nd Feature")
```

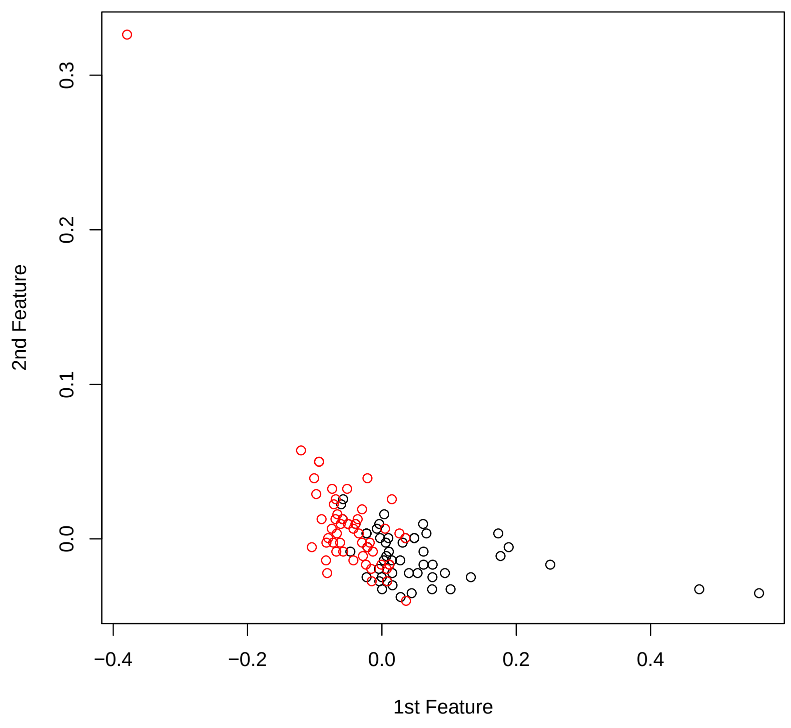

Figure 2.6: Projection of the spam data on two features using an RBF kernel

classical CCA attempts to find linear combination of the variables which give the maximum correlation between the combinations. I.e., if

$$y_1 = \mathbf{w_1 x_1} = \sum_j w_1 x_{1j}$$
$$y_2 = \mathbf{w_2 x_2} = \sum_j w_2 x_{2j}$$

one wishes to find those values of $\mathbf{w_1}$ and $\mathbf{w_2}$ which maximize the correlation between y_1 and y_2. Similar to the KPCA algorithm, CCA can be extended and used in a dot product space F which is related to the input space by a possibly nonlinear map $\Phi : \mathbb{R}^N \to F$, $x \mapsto \mathbf{X}$ as

$$y_1 = \mathbf{w_1 \Phi(x_1)} = \sum_j w_1 \Phi(x_{1j})$$
$$y_2 = \mathbf{w_2 \Phi(x_2)} = \sum_j w_2 \Phi(x_{2j})$$

Following (Kuss and Graepel, 2003), the **kernlab** implementation of a KCCA projects the data vectors on a new coordinate system using KPCA and uses linear CCA to retrieve the correlation coefficients. The `kcca` method in **kernlab** returns an S4 object containing the correlation coefficients for each data set and the corresponding correlation along with the kernel used.

2.3.10 Interior Point Code Quadratic Optimizer

In many kernel based algorithms, learning implies the minimization of some risk function. Typically we have to deal with quadratic or general convex problems for Support Vector Machines of the type

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n]. \end{array} \tag{2.27}$$

f and c_i are convex functions and $n \in \mathbb{N}$. **kernlab** provides the S4 method `ipop` implementing an optimizer of the interior point family (Vanderbei, 1999) which solves the quadratic programming problem

$$\begin{array}{ll} \text{minimize} & c^\top x + \frac{1}{2} x^\top H x \\ \text{subject to} & b \leq Ax \leq b + r \\ & l \leq x \leq u \end{array} \tag{2.28}$$

This optimizer can be used in regression, classification, and novelty detection in SVMs.

2.3.11 Incomplete Cholesky Decomposition

When dealing with kernel based algorithms, calculating a full kernel matrix should be avoided since it is already a $O(N^2)$ operation. Fortunately, the fact that kernel matrices are positive semidefinite is a strong constraint and good approximations can be found with small computational cost. The Cholesky decomposition factorizes a positive semidefinite $N \times N$ matrix K as $K = ZZ^T$, where Z is an upper triangular $N \times N$ matrix. Exploiting the fact that kernel matrices are usually of low rank, an *incomplete Cholesky decomposition* (Wright, 1999) finds a matrix $\tilde{Z}$ of size $N \times M$ where $M \ll N$ such that the norm of $K - \tilde{Z}\tilde{Z}^T$ is smaller than a given tolerance θ. The main difference of incomplete Cholesky decomposition to the standard Cholesky decomposition is that pivots which are below a certain threshold are simply skipped. If L is the number of skipped pivots, we obtain a $\tilde{Z}$ with only $M = N - L$ columns. The algorithm works by picking a column from K to be added by maximizing a lower bound on the reduction of the error of the approximation. **kernlab** has an implementation of an incomplete Cholesky factorization called `inc.chol` which computes the decomposed matrix $\tilde{Z}$ from the original data for any given kernel without the need to compute a full kernel matrix beforehand. This has the advantage that no full kernel matrix has to be stored in memory.

2.4 Conclusions

In this chapter we described **kernlab**, a flexible and extensible kernel methods package for R with existing modern kernel algorithms along with tools for constructing new kernel based algorithms. It provides a unified framework for using and creating kernel-based algorithms in R while using all of R's modern facilities, like S4 classes and namespaces. Our aim for the future is to extend the package and add more kernel-based methods as well as kernel relevant tools. Sources and binaries for the latest version of **kernlab** are available at CRAN[1] under the GNU Public License.

[1]http://cran.r-project.org

Chapter 3

Support Vector Machines in R

Being among the most popular and efficient classification and regression methods currently available, implementations of support vector machines exist in almost every popular programming language. Currently four R packages contain SVM related software. The purpose of this chapter is to present and compare these implementations.

3.1 Introduction

Support Vector learning is based on simple ideas which originated in statistical learning theory (Vapnik, 1998). The simplicity comes from the fact that Support Vector Machines (SVMs) apply a simple linear method to the data but in a high-dimensional feature space non-linearly related to the input space. Moreover, even though we can think of SVMs as a linear algorithm in a high-dimensional space, in practice, it does not involve any computations in that high-dimensional space. This simplicity combined with state of the art performance on many learning problems (classification, regression, and novelty detection) has contributed to the popularity of the SVM. In the rest of the chapter we provide a short introduction into Support Vector Machines, an overview of the SVM related software available in R and other programming languages, and section on the data sets we will be using. We then describe the four available SVM implementations in R and present the results of a timing benchmark.

3.2 Support Vector Machines

SVMs use an implicit mapping Φ of the input data into a high-dimensional feature space defined by a kernel function, i.e., a function returning the inner product $\langle \Phi(x), \Phi(x') \rangle$ between the images of two data points x, x' in the feature space. The learning then takes place in the feature space, and the data points only appear inside dot products with other points. This is often referred to as the "kernel trick" (Schölkopf and Smola, 2002). More precisely, if a projection $\Phi : X \to H$ is used, the dot product $\langle \Phi(x), \Phi(x') \rangle$ can be represented by a kernel function k

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle, \tag{3.1}$$

which is computationally simpler than explicitly projecting x and x' into the feature space H.

Training a SVM for classification regression or novelty detection involves solving a quadratic problem. Using a standard quadratic problem solver for training an SVM would involve solving a big QP problem even for a moderate sized data set and computing an $m \times m$ matrix in memory where m the number of training points and would thous limit the size of problems an SVM could be applied to. To handle this issue, methods like SMO (Platt, 1998), chunking (Osuna *et al.*, 1997), and simple SVM (Vishwanathan *et al.*, 2003) where concieved. These algorithms iteratevely compute the solution of the SVM problem and scale $O(N^k)$ where k is between 1 and 2.5 and have a linear space complexity.

3.2.1 Classification

In classification, support vector machines separate the different classes of data by a hyper-plane

$$\langle \mathbf{w}, \Phi(x) \rangle + b = 0 \tag{3.2}$$

corresponding to the decision function

$$f(x) = \operatorname{sign}(\langle \mathbf{w}, \Phi(x) \rangle + b) \tag{3.3}$$

It can be shown that the optimal, in terms of classification performance, hyper-plane (Vapnik, 1998) is the one with the maximal margin of separation between the two classes. It can be constructed by solving a constrained quadratic optimization problem whose solution $\mathbf{w}$ has an expansion $\mathbf{w} = \sum_i \alpha_i \Phi(x_i)$ in terms of a subset of training patterns that lie on the margin. These training patterns, called *support vectors*, carry all relevant information about the classification problem. Omitting the details of the calculation, there is just one crucial property of the algorithm that we need to emphasize: both the quadratic programming problem and the final decision function depend only on dot products between patterns. This allows the use of the "kernel trick" and the generalization of this linear algorithm to the nonlinear case.

In the case of the L_2-norm soft margin classification the primal optimization problem takes the form:

$$\begin{aligned} \text{minimize} \quad & t(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\langle \Phi(x_i), \mathbf{w}\rangle + b) \geq 1 - \xi_i \qquad (i = 1, \dots, m) \\ & \xi_i \geq 0 \qquad (i = 1, \dots, m) \end{aligned} \tag{3.4}$$

where m is the number of training patterns, and $y_i = \pm 1$. As in most kernel methods, the SVM solution $\mathbf{w}$ can be shown to have an expansion

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i) \tag{3.5}$$

where non-zero coefficients (support vectors) occur when a point (x_i, y_i) meets the constraint. The coefficients α_i are found by solving the following (dual) quadratic programming problem:

$$\begin{aligned} \text{maximize} \quad & W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{C}{m} \qquad (i = 1, \dots, m) \\ & \sum_{i=1}^{m}\alpha_i y_i = 0. \end{aligned} \tag{3.6}$$

This is a typical quadratic problem of the form:

$$\begin{aligned} \text{minimize} \quad & c^\top x + \tfrac{1}{2}x^\top H x \\ \text{subject to} \quad & b \leq Ax \leq b + r \\ & l \leq x \leq u \end{aligned} \tag{3.7}$$

where $H \in \mathbb{R}^{m\times m}$ with entries $H_{ij} = y_i y_j k(x_i, x_j)$, $c = (1, \dots, 1) \in \mathbb{R}^m$, $u = (C, \dots, C) \in \mathbb{R}^m$, $l = (0, \dots, 0) \in \mathbb{R}^m$, $A = (y_1, \dots, y_m) \in \mathbb{R}^m$, $b = 0$,

$r = 0$. The problem can easily be solved in a standard QP solver such as `quadprog()` in package **quadprog** (Weingessel, 2004) or `ipop()` in package **kernlab** (Karatzoglou *et al.*, 2005a), both available in R (R Development Core Team, 2005). Techniques taking advantage of the special structure of the SVM QP problem like SMO and chunking (Osuna *et al.*, 1997) though offer much better performance in terms of speed, scalability and memory usage.

The cost parameter C of the SVM formulation in Equation 3.7 controls the penalty paid by the SVM for missclassifying a training point and thus the complexity of the prediction function. A high cost value C will force the SVM to create a complex enough prediction function to missclassify as few training points as possible, while a lower cost parameter will lead to a simpler prediction function. Therefore, this type of SVM is usually called C-SVM.

Another formulation of the classification with a more intuitive hyperparameter than C is the ν-SVM (Schölkopf *et al.*, 2000). The ν parameter has the interesting property of being an upper bound on the training error and a lower bound on the fraction of support vectors found in the data set, thus controlling the complexity of the classification function build by the SVM (see Appendix for details).

For multi-class classification, mostly voting schemes such as one-against-one and one-against-all are used. In the one-against-all method k binary SVM classifiers are trained, where k is the number of classes, each trained to separate one class from the rest. The classifiers are then combined by comparing their decision values on a test data instance and labeling it according to the classifier with the highest decision value.

In the one-against-one classification method (also called pairwise classification; see Knerr *et al.*, 1990; Kreßel, 1999), $\binom{k}{2}$ classifiers are constructed where each one is trained on data from two classes. Prediction is done by voting where each classifier gives a prediction and the class which is most frequently predicted wins ("Max Wins"). This method has been shown to produce robust results when used with SVMs (Hsu and Lin, 2002a). Although this suggests a higher number of support vector machines to train the overall CPU time used is less compared to the one-against-all method since the problems are smaller and the SVM optimization problem scales super-linearly.

Furthermore, SVMs can also produce class probabilities as output instead of class labels. This is can done by an improved implementation (Lin *et al.*, 2001) of Platt's a posteriori probabilities (Platt, 2000) where a sigmoid function

$$P(y = 1 \mid f) = \frac{1}{1 + e^{Af+B}} \tag{3.8}$$

is fitted to the decision values f of the binary SVM classifiers, A and B being estimated by minimizing the negative log-likelihood function. This is equivalent to fitting a logistic regression model to the estimated decision values. To extend the class probabilities to the multi-class case, all binary classifiers class probability output can be combined as proposed in Wu *et al.* (2003).

In addition to these heuristics for extending a binary SVM to the multi-class problem, there have been reformulations of the support vector quadratic problem that deal with more than two classes. One of the many approaches for native support vector multi-class classification is the one proposed in Crammer and Singer (2000), which we will refer to as 'spoc-svc'. This algorithm works by solving a single optimization problem including the data from all classes. The primal formulation is:

$$\begin{aligned} \text{minimize} \quad & t(\{\mathbf{w_n}\}, \xi) = \frac{1}{2}\sum_{n=1}^{k} \|\mathbf{w_n}\|^2 + \frac{C}{m}\sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & \langle \Phi(x_i), \mathbf{w}_{y_i} \rangle - \langle \Phi(x_i), \mathbf{w}_n \rangle \geq b_i^n - \xi_i \qquad (i = 1, \dots, m) \\ \text{where} \quad & b_i^n = 1 - \delta_{y_i,n} \end{aligned} \tag{3.9}$$

where the decision function is

$$\text{argmax}_{n=1,\dots,k} \langle \Phi(x_i), \mathbf{w}_n \rangle \tag{3.10}$$

Details on performance and benchmarks on various approaches for multi-class classification can be found in Hsu and Lin (2002b).

3.2.2 Novelty detection

SVMs have also been extended to deal with the problem of *novelty detection* (or one-class classification; see Schölkopf *et al.*, 1999; Tax and Duin, 1999), where essentially an SVM detects outliers in a data set. SVM novelty detection works by creating a spherical decision boundary around a set of data points by a set of support vectors describing the sphere's boundary. The primal optimization problem for support vector novelty detection is the following:

$$\begin{aligned}
\text{minimize} \quad & t(\mathbf{w}, \xi, \rho) = \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \frac{1}{m\nu}\sum_{i=1}^{m}\xi_i \\
\text{subject to} \quad & \langle \Phi(x_i), \mathbf{w}\rangle + b \geq \rho - \xi_i \qquad (i = 1, \ldots, m) \\
& \xi_i \geq 0 \qquad (i = 1, \ldots, m).
\end{aligned} \tag{3.11}$$

The ν parameter is used to control the volume of the sphere and consequently the number of outliers found. The value of ν sets an upper bound on the fraction of outliers found in the data.

3.2.3 Regression

By using a different loss function called the ϵ-insensitive loss function $\|y - f(x)\|_\epsilon = \max\{0, \|y - f(x)\| - \epsilon\}$, SVMs can also perform regression. This loss function ignores errors that are smaller than a certain threshold $\epsilon > 0$ thus creating a tube around the true output. The primal becomes:

$$\begin{aligned}
\text{minimize} \quad & t(\mathbf{w}, \xi) = \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\
\text{subject to} \quad & (\langle \Phi(x_i), \mathbf{w}\rangle + b) - y_i \leq \epsilon - \xi_i \\
& y_i - (\langle \Phi(x_i), \mathbf{w}\rangle + b) \leq \epsilon - \xi_i^* \\
& \xi_i^* \geq 0 \qquad (i = 1, \ldots, m)
\end{aligned} \tag{3.12}$$

We can estimate the accuracy of SVM regression by computing the scale parameter of a Laplacian distribution on the residuals $\zeta = y - f(x)$, where $f(x)$ is the estimated decision function (Lin and Weng, 2004).

The dual problems of the various classification, regression and novelty detection SVM formulations can be found in the Appendix.

3.2.4 R software overview

The first implementation of SVM in R (R Development Core Team, 2005) was introduced in the **e1071** (Dimitriadou *et al.*, 2005) package. The `svm()` function in **e1071** provides a rigid interface to `libsvm` along with visualization and parameter tuning methods.

Package **kernlab** features a variety of kernel-based methods and includes a SVM method based on the optimizers used in `libsvm` and `bsvm` (Hsu and Lin, 2002c). It aims to provide a flexible and extensible SVM implementation.

Package **klaR** (Roever *et al.*, 2005) includes an interface to `SVMlight`, a popular SVM implementation that additionally offers classification tools such as Regularized Discriminant Analysis.

Finally, package **svmpath** (Hastie, 2004) provides an algorithm that fits the entire path of the SVM solution (i.e., for any value of the cost parameter).

In the remainder of the chapterwe will extensively review and compare these four SVM implementations.

3.3 Data

Throughout this chapter, we will use the following data sets accessible through R, most of them originating from the UCI machine learning database (Blake and Merz, 1998):

iris This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*. The data set is provided by base R.

spam A data set collected at Hewlett-Packard Labs which classifies 4601 e-mails as spam or non-spam. In addition to this class label there are 57 variables indicating the frequency of certain words and characters in the e-mail. The data set is provided by the **kernlab** package.

musk This dataset in package **kernlab** describes a set of 476 molecules of which 207 are judged by human experts to be musks and the remaining 269 molecules are judged to be non-musks. The data has 167 variables which describe the geometry of the molecules.

promotergene Promoters have a region where a protein (RNA polymerase) must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align. The dataset in package **kernlab** contains DNA sequences of promoters and non-promoters in a data frame with 106 observations and 58 variables. The DNA bases are coded as follows: 'a' adenine, 'c' cytosine, 'g' guanine, and 't' thymine.

Vowel Speaker independent recognition of the eleven steady state vowels of British English using a specified training set of LPC derived log area ratios. The vowels are indexed by integers 0 to 10. This dataset in package **mlbench** (Leisch and Dimitriadou, 2001) has 990 observations on 10 independent variables.

DNA in package **mlbench** consists of 3,186 data points (splice junctions). The data points are described by 180 indicator binary variables and the problem is to recognize the 3 classes ('ei', 'ie', neither), i.e., the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out).

BreastCancer in package **mlbench** is a data frame with 699 observations on 11 variables, one being a character variable, 9 being ordered or nominal, and 1 target class. The objective is to identify each of a number of benign or malignant classes.

BostonHousing Housing data in package **mlbench** for 506 census tracts of Boston from the 1970 census. There are 506 observations on 14 variables.

B3 German Bussiness Cycles from 1955 to 1994 in package **klaR**. A data frame with 157 observations on the following 14 variables.

Dataset	#Examples	#Attributes				Class Distribution (%)
		b	c	m	tot.	
iris	150			5	5	33.3/33.3/33.3
spam	4601			57	57	39.40/60.59
musk	476			166	166	42.99 / 57.00
promotergene	106			57	57	50.00 / 50.00
Vowel	990		1	9	10	10.0/10.0/...
DNA	3186	180			180	24.07/24.07/51.91
BreastCancer	699		9		9	34.48 / 65.52
BostonHousing	506	1		12	14	
B3	506			13	13	37.57/15.28/29.93/17.19

Table 3.1: The data sets used throughout the chapter. Legend: b=binary, c=categorical, m=metric.

3.4 `ksvm` in kernlab

Package **kernlab** (Karatzoglou *et al.*, 2004) aims to provide the R user with basic kernel functionality (e.g., like computing a kernel matrix using a particular kernel), along with some utility functions commonly used in kernel-based methods like a quadratic programming solver, and modern kernel-based algorithms based on the functionality that the package provides. It also takes advantage of the inherent modularity of kernel-based methods, aiming to allow the user to switch between kernels on an existing algorithm and even create and use own kernel functions for the various kernel methods provided in the package.

kernlab uses R's new object model described in "Programming with Data" (Chambers, 1998) which is known as the S4 class system and is implemented in package **methods**. In contrast to the older S3 model for objects in R, classes, slots, and methods relationships must be declared explicitly when using the S4 system. The number and types of slots in an instance of a class have to be established at the time the class is defined. The objects from the class are validated against this definition and have to comply to it at any time. S4 also requires formal declarations of methods, unlike the informal system of using function names to identify a certain method in S3. Package **kernlab** is available from CRAN (`http://cran.r-project.org`) under the GPL license.

The `ksvm()` function, **kernlab**'s implementation of SVMs, provides a standard formula interface along with a matrix interface. `ksvm()` is mostly programmed in R but uses, through the `.Call` interface, the optimizers found in `bsvm` and `libsvm` (Chang and Lin, 2001) which provide a very efficient C++ version of the Sequential Minimization Optimization (SMO). The SMO algorithm solves the SVM quadratic problem (QP) without using any numerical QP optimization steps. Instead, it chooses to solve the smallest possible optimization problem involving two elements of α_i because the must obey one linear equality constraint. At every step, SMO chooses two α_i to jointly optimize and finds the optimal values for these α_i analytically, thus avoiding numerical QP optimization, and updates the SVM to reflect the new optimal values.

The SVM implementations available in `ksvm()` include the C-SVM classification algorithm along with the ν-SVM classification. Also included is a bound constraint version of C classification (C-BSVM) which solves a slightly different QP problem (Mangasarian and Musicant, 1999, including the offset β in the objective function) using a modified version of the `TRON` (Lin and More, 1999) optimization software. For regression, `ksvm()` includes the ϵ-SVM regression algorithm along with the ν-SVM regression formulation. In addition, a bound constraint version (ϵ-BSVM) is provided, and novelty detection (one-class classification) is supported.

For classification problems which include more then two classes (multi-class case) two options are available: a one-against-one (pairwise) classification method or the native multi-class formulation of the SVM (spoc-svc) described in Section 2. The optimization problem of the native multi-class SVM implementation is solved by a decomposition method proposed in Hsu and Lin (2002c) where optimal working sets are found (that is, sets of α_i values which have a high probability of being non-zero). The QP sub-problems are then solved by a modified version of the TRON optimization software.

The `ksvm()` implementation can also compute class-probability output by using Platt's probability methods (Equation 3.8) along with the multi-class extension of the method in Wu *et al.* (2003). The prediction method can also return the raw decision values of the support vector model:

```
> library("kernlab")
> data("iris")
> irismodel <- ksvm(Species ~ ., data = iris,
+     type = "C-bsvc", kernel = "rbfdot",
+     kpar = list(sigma = 0.1), C = 10,
+     prob.model = TRUE)
> irismodel

Support Vector Machine object of class "ksvm"

SV type: C-bsvc  (classification)
 parameter : cost C = 10

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.1

Number of Support Vectors : 32
Training error : 0.02
Probability model included.

> predict(irismodel, iris[c(3, 10, 56, 68,
+     107, 120), -5], type = "probabilities")

          setosa  versicolor   virginica
[1,] 0.986432820 0.007359407 0.006207773
[2,] 0.983323813 0.010118992 0.006557195
[3,] 0.004852528 0.967555126 0.027592346
[4,] 0.009546823 0.988496724 0.001956452
[5,] 0.012767340 0.069496029 0.917736631
[6,] 0.011548176 0.150035384 0.838416441
```

```
> predict(irismodel, iris[c(3, 10, 56, 68,
+     107, 120), -5], type = "decision")
```

```
          [,1]       [,2]       [,3]
[1,] -1.460398 -1.1910251 -3.8868836
[2,] -1.357355 -1.1749491 -4.2107843
[3,]  1.647272  0.7655001 -1.3205306
[4,]  1.412721  0.4736201 -2.7521640
[5,]  1.844763  1.0000000  1.0000019
[6,]  1.848985  1.0069010  0.6742889
```

`ksvm` allows for the use of any valid user defined kernel function by just defining a function which takes two vector arguments and returns its Hilbert Space dot product in scalar form.

```
> k <- function(x, y) {
+     (sum(x * y) + 1) * exp(0.001 * sum((x -
+         y)^2))
+ }
> class(k) <- "kernel"
> data("promotergene")
> gene <- ksvm(Class ~ ., data = promotergene,
+     kernel = k, C = 10, cross = 5)
> gene
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 10


Number of Support Vectors : 66
Training error : 0
Cross validation error : 0.141558
```

The implementation also includes the following computationally efficiently implemented kernels: Gaussian RBF, polynomial, linear, sigmoid, Laplace, Bessel RBF, spline, and ANOVA RBF.

N-fold cross-validation of an SVM model is also supported by `ksvm`, and the training error is reported by default.

The problem of model selection is partially addressed by an empirical observation for the popular Gaussian RBF kernel (Caputo *et al.*, 2002), where the optimal values of the width hyper-parameter σ are shown to lie in between the 0.1 and 0.9 quantile of the $\|x - x'\|^2$ statistics. The `sigest()` function uses a sample of the training set to estimate the quantiles and returns a vector containing the values of the quantiles. Pretty much any value within this interval leads to good performance.

The object returned by the `ksvm()` function is an S4 object of class ksvm with slots containing the coefficients of the model (support vectors), the parameters used (C, ν, etc.), test and cross-validation error, the kernel function, information on the problem type, the data scaling parameters, etc. There are accessor functions for the information contained in the slots of the ksvm object.

The decision values of binary classification problems can also be visualized via a contour plot with the `plot()` method for the ksvm objects. This function is mainly for simple problems. An example is shown in Figure 3.1.

```
> x <- rbind(matrix(rnorm(120), , 2), matrix(rnorm(120,
+     mean = 3), , 2))
> y <- matrix(c(rep(1, 60), rep(-1, 60)))
> svp <- ksvm(x, y, type = "C-svc", kernel = "rbfdot",
+     kpar = list(sigma = 2))
> plot(svp)
```

3.5 svm in e1071

Package **e1071** provides an interface to `libsvm` (Chang and Lin, 2001, current version: 2.8), complemented by visualization and tuning functions. `libsvm` is a fast and easy-to-use implementation of the most popular SVM formulations (C and ν classification, ϵ and ν regression, and novelty detection). It includes the most common kernels (linear, polynomial, RBF, and sigmoid), only extensible by changing the C++ source code of `libsvm`. Multi-class classification is provided using the one-against-one voting scheme. Other features include the computation of decision and probability values for predictions (for both classification and regression), shrinking heuristics during the fitting process, class weighting in the classification mode, handling of sparse data, and the computation of the training error using cross-validation. `libsvm` is distributed under a very permissive, BSD-like licence.

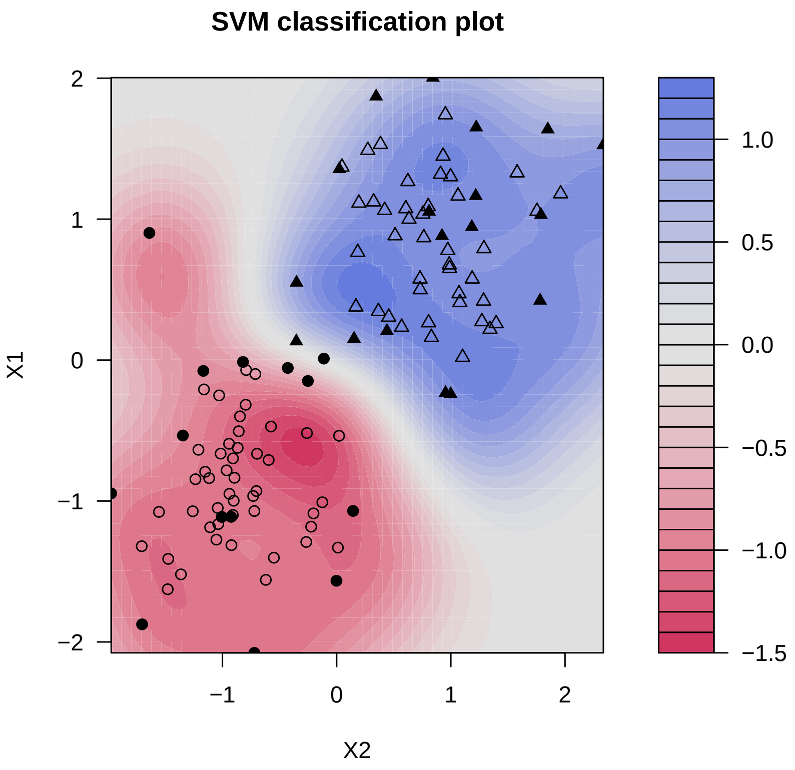

Figure 3.1: A contour plot of the fitted decision values for a simple binary classification problem.

The R implementation is based on the S3 class mechanisms. It basically provides a training function with standard and formula interfaces, and a `predict()` method. In addition, a `plot()` method visualizing data, support vectors, and decision boundaries if provided. Hyper-parameter tuning is done using the `tune()` framework in **e1071** performing a grid search over specified parameter ranges.

The sample session starts with a C classification task on the iris data, using the radial basis function kernel with fixed hyper-parameters C and γ:

```
> library("e1071")
> model <- svm(Species ~ ., data = iris_train,
+     method = "C-classification", kernel = "radial",
+     cost = 10, gamma = 0.1)
> summary(model)


Call:
svm(formula = Species ~ ., data = iris_train,
    method = "C-classification", kernel = "radial",
    cost = 10, gamma = 0.1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.1

Number of Support Vectors:  27

 ( 12 12 3 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

We can visualize a 2-dimensional projection of the data with highlighting classes and support vectors (see Figure 3.2):

```
> plot(model, iris_train, Petal.Width ~
+      Petal.Length, slice = list(Sepal.Width = 3,
+      Sepal.Length = 4))
```

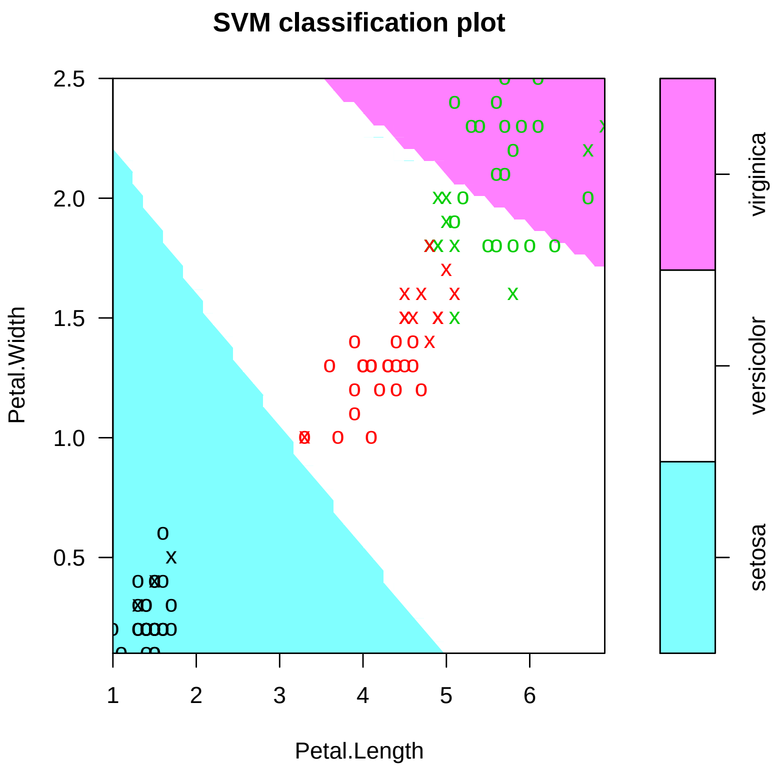

Figure 3.2: SVM plot visualizing the iris data. Support vectors are shown as 'X', true classes are highlighted through symbol color, predicted class regions are visualized using colored background.

Predictions from the model, as well as decision values from the binary classifiers, are obtained using the `predict()` method:

```
> (pred <- predict(model, head(iris), decision.values = TRUE))

[1] setosa setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica

> attr(pred, "decision.values")
```

```
  virginica/versicolor virginica/setosa
1            -3.833133        -1.156482
2            -3.751235        -1.121963
3            -3.540173        -1.177779
4            -3.491439        -1.153052
5            -3.657509        -1.172285
6            -3.702492        -1.069637
  versicolor/setosa
1         -1.393419
2         -1.279886
3         -1.456532
4         -1.364424
5         -1.423417
6         -1.158232
```

Probability values can be obtained in a similar way.

In the next example, we again train a classification model on the spam data. This time, however, we will tune the hyper-parameters on a subsample using the `tune` framework of **e1071**:

```
> tobj <- tune.svm(type ~ ., data = spam_train[1:300,
+     ], gamma = 10^(-6:-3), cost = 10^(1:2))
> summary(tobj)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
 0.001   10

- best performance: 0.1233333

- Detailed performance results:
  gamma cost     error
1 1e-06   10 0.4133333
2 1e-05   10 0.4133333
3 1e-04   10 0.1900000
4 1e-03   10 0.1233333
5 1e-06  100 0.4133333
```

```
6 1e-05  100 0.1933333
7 1e-04  100 0.1233333
8 1e-03  100 0.1266667
```

`tune.svm()` is a convenience wrapper to the `tune()` function that carries out a grid search over the specified parameters. The `summary()` method on the returned object indicates the misclassification rate for each parameter combination and the best model. By default, the error measure is computed using a 10-fold cross validation on the given data, but `tune()` offers several alternatives (e.g., separate training and test sets, leave-one-out-error, etc.). In this example, the best model in the parameter range is obtained using $C = 10$ and $\gamma = 0.001$, yielding a misclassification error of 12.33%. A graphical overview on the tuning results (that is, the error landscape) can be obtained by drawing a contour plot (see Figure 3.3):

```
> plot(tobj, transform.x = log10, xlab = expression(log[10](gamma)),
+     ylab = "C")
```

Using the best parameters, we now train our final model. We estimate the accuracy in two ways: by 10-fold cross validation on the training data, and by computing the predictive accuracy on the test set:

```
> bestGamma <- tobj$best.parameters[[1]]
> bestC <- tobj$best.parameters[[2]]
> model <- svm(type ~ ., data = spam_train,
+     cost = bestC, gamma = bestGamma, cross = 10)
> summary(model)

Call:
svm(formula = type ~ ., data = spam_train,
    cost = bestC, gamma = bestGamma, cross = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.001

Number of Support Vectors:  313
```

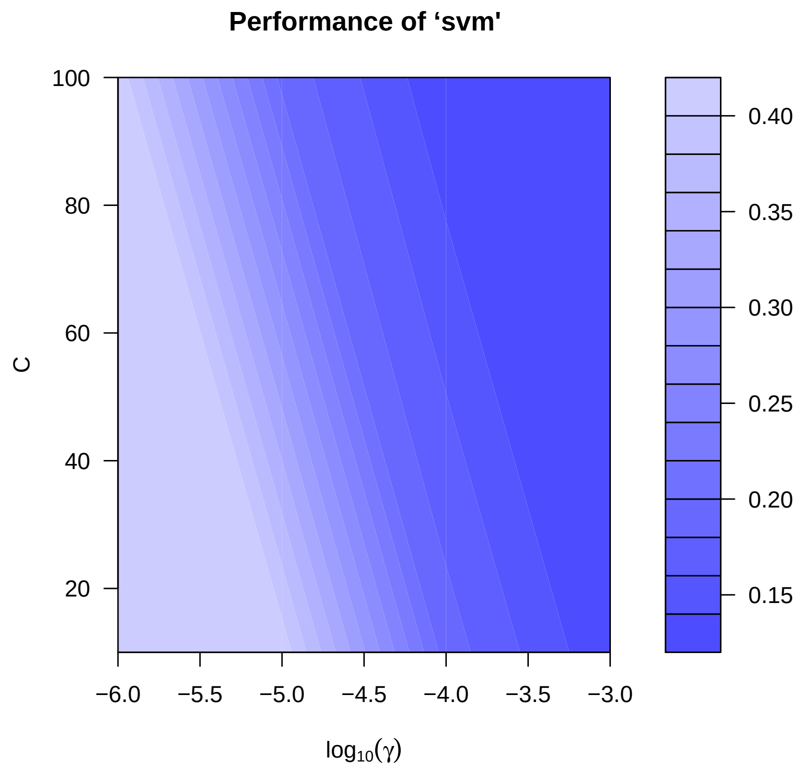

Figure 3.3: Contour plot of the error landscape resulting from a grid search on a hyper-parameter range.

```
 ( 162 151 )


Number of Classes:  2

Levels:
 nonspam spam

10-fold cross-validation on training data:

Total Accuracy: 91.7
Single Accuracies:
 94 91 92 90 91 91 92 90 92 94
```

```
> pred <- predict(model, spam_test)
> (acc <- table(pred, spam_test$type))

pred      nonspam spam
  nonspam    2075  196
  spam        115 1215

> classAgreement(acc)

$diag
[1] 0.9136351

$kappa
[1] 0.8169207

$rand
[1] 0.8421442

$crand
[1] 0.6832857
```

3.6 svmlight in klar

Package **klaR** (Roever *et al.*, 2005) includes utility functions for classification and visualization, and provides the `svmlight()` function which is a fairly simple interface to the `SVMlight` package. The `svmlight()` function in **klaR** is written in the `S3` object system and provides a formula interface along with standard matrix, data frame, and formula interfaces. The `SVMlight` package is available only for non-commercial use, and the installation of the package involves placing the `SVMlight` binaries in the path of the operating system. The interface works by using temporary text files where the data and parameters are stored before being passed to the `SVMlight` binaries.

`SVMlight` utilizes a special active set method (Joachims, 1999) for solving the SVM QP problem where q variables (the active set) are selected per iteration for optimization. The selection of the active set is done in a way which maximizes the progress towards the minimum of the objective function. At each iteration a QP subproblem is solved using only the active set until the final solution is reached.

The **klaR** interface function `svmlight()` supports the C-SVM formulation for classification and the ϵ-SVM formulation for regression. `SVMlight` uses the one-against-all method for multi-class classification where k classifiers are trained. Compared to the one-against-one method, this requires usually less binary classifiers to be built but the problems each classifier has to deal with are bigger.

The `SVMlight` implementation provides the Gaussian, polynomial, linear, and sigmoid kernels. The `svmlight()` interface employs a character string argument to pass parameters to the `SVMlight` binaries. This allows direct access to the feature-rich `SVMlight` and allows, e.g., control of the SVM parameters (cost, ϵ), the choice of the kernel function and the hyper-parameters, the computation of the leave-one-out error, and the control of the verbosity level.

The S3 object returned by the `svmlight()` function in **klaR** is of class svmlight and is a list containing the model coefficients along with information on the learning task, like the type of problem, and the parameters and arguments passed to the function. The svmlight object has no `print()` or `summary()` methods. The `predict()` method returns the class labels in case of classification along with a class membership value (class probabilities) or the decision values of the classifier.

```
> library("klaR")
> data("B3")
> Bmod <- svmlight(PHASEN ~ ., data = B3,
+     svm.options = "-c 10 -t 2 -g 0.1 -v 0")
> predict(Bmod, B3[c(4, 9, 30, 60, 80, 120),
+     -1])
```

```
$class
[1] 3 3 4 3 4 1
Levels: 1 2 3 4

$posterior
              1          2          3          4
[1,] 0.09633177 0.09627103 0.71112031 0.09627689
[2,] 0.09628235 0.09632512 0.71119794 0.09619460
[3,] 0.09631525 0.09624314 0.09624798 0.71119362
[4,] 0.09632530 0.09629393 0.71115614 0.09622463
[5,] 0.09628295 0.09628679 0.09625447 0.71117579
[6,] 0.71123818 0.09627858 0.09620351 0.09627973
```

3.7 svmpath

The performance of the SVM is highly dependent on the value of the regularization parameter C, but apart from grid search, which is often computationally expensive, there is little else a user can do to find a value yielding good performance. Although the ν-SVM algorithm partially addresses this problem by reformulating the SVM problem and introducing the ν parameter, finding a correct value for ν relies on at least some knowledge of the expected result (test error, number of support vectors, etc.).

Package **svmpath** (Hastie, 2004) contains a function `svmpath()` implementing an algorithm which solves the C-SVM classification problem for all the values of the regularization cost parameter $\lambda = 1/C$ (Hastie *et al.*, 2004). The algorithm exploits the fact that the loss function is piecewise linear and thus the parameters (coefficients) $\alpha(\lambda)$ of the SVM model are also piecewise linear as functions of the regularization parameter λ. The algorithm solves the SVM problem for all values of the regularization parameter with essentially a small multiple ($\approx$ 3) of the computational cost of fitting a single model.

The algorithm works by starting with a high value of λ (high regularization) and tracking the changes to the model coefficients α as the value of λ is decreased. When λ decreases, $||\alpha||$ and hence the width of the margin decrease, and points move from being inside to outside the margin. Their corresponding coefficients α_i change from $\alpha_i = 1$ when they are inside the margin to $\alpha_i = 0$ when outside. The trajectories of the α_i are piecewise linear in λ and by tracking the break points all values in between can be found by simple linear interpolation.

The `svmpath()` implementation in R currently supports only binary C classification. The function must be used through a S3 matrix interface where the y label must be $+1$ or -1. Similarly to `ksvm()`, `svmpath()` allows the use of any user defined kernel function, but in its current implementation requires the direct computation of full kernel matrices, thus limiting the size of problems `svmpath()` can be used on since the full $m \times m$ kernel matrix has to be computed in memory. The implementation comes with the Gaussian RBF and polynomial kernel as built-in kernel functions and also provides the user with the option of using a precomputed kernel matrix K.

The function call returns an object of class `svmpath` which is a list containing the model coefficients (α_i) for the break points along with the offsets and the value of the regularization parameter $\lambda = 1/C$ at the points. Also included is information on the kernel function and its hyper-parameter. The `predict()` method for `svmpath` objects returns the decision values, or the binary labels $(+1, -1)$ for a specified value of the $\lambda = 1/C$ regularization parameter. The `predict()` method can also return the model coefficients α for any value of the λ parameter.

```
> library("svmpath")
> data("svmpath")
> attach(balanced.overlap)
> svmpm <- svmpath(x, y, kernel.function = radial.kernel,
+     param.kernel = 0.1)
> predict(svmpm, x, lambda = 0.1)
```

```
            [,1]
 [1,] -0.8399810
 [2,] -1.0000000
 [3,] -1.0000000
 [4,] -1.0000000
 [5,]  0.1882592
 [6,] -2.2363430
 [7,]  1.0000000
 [8,] -0.2977907
 [9,]  0.3468992
[10,]  0.1933259
[11,]  1.0580215
[12,]  0.9309218
```

```
> predict(svmpm, lambda = 0.2, type = "alpha")
```

```
$alpha0
[1] -0.3809953

$alpha
 [1] 1.0000000 1.0000000 0.9253461 1.0000000
 [5] 1.0000000 0.0000000 1.0000000 1.0000000
 [9] 1.0000000 1.0000000 0.0000000 0.9253461

$lambda
[1] 0.2
```

3.8 Benchmarking

In the following we compare the four SVM implementations in terms of training time. In this comparison we only focus on the actual training time of the

	`ksvm()` (**kernlab**)	`svm()` (**e1071**)	`svmlight()` (**klaR**)	`svmpath()` (**svmpath**)
spam	18.50	17.90	34.80	34.00
musk	1.40	1.30	4.65	13.80
Vowel	1.30	0.30	21.46	NA
DNA	22.40	23.30	116.30	NA
BreastCancer	0.47	0.36	1.32	11.55
BostonHousing	0.72	0.41	92.30	NA

Table 3.2: The training times for the SVM implementations on different datasets in seconds. Timings where done on an AMD Athlon 1400 Mhz computer running Linux.

SVM excluding the time needed for estimating the training error or the cross-validation error. In implementations which scale the data (`ksvm()`, `svm()`) we include the time needed to scale the data. We include both binary and multi-class classification problems as well as a few regression problems. The training is done using a Gaussian kernel where the hyper-parameter was estimated using the `sigest()` function in **kernlab**, which estimates the 0.1 and 0.9 quantiles of $\|x - x'\|^2$. The data was scaled to unit variance and the features for estimating the training error and the fitted values were turned off and the whole data set was used for the training. The mean value of 10 runs is given in table 3.2; we do not report the variance since it was practically 0 in all runs. The runs were done with version 0.6-2 of **kernlab**, version 1.5-11 of **e1071**, version 0.9 of **svmpath**, and version 0.4-1 of **klaR**.

Table 3.2 contains the training times for the SVM implementations on the various datasets. `ksvm()` and `svm()` seem to perform on a similar level in terms of training time with the `svmlight()` function being significantly slower. When comparing `svmpath()` with the other implementations, one has to keep in mind that it practically estimates the SVM model coefficients for the whole range of the cost parameter C. The `svmlight()` function seems to suffer from the fact that the interface is based on reading and writing temporary text files as well as from the optimization method (chunking) used from the `SVMlight` software which in these experiments does not seem to perform as well as the SMO implementation in `libsvm`. The `svm()` in **e1071** and the `ksvm()` function in **kernlab** seem to be on par in terms of training time performance with the `svm()` function being slightly faster on multi-class problems.

3.9 Conclusions

Table 3.3 provides a quick overview of the four SVM implementations. `ksvm()` in **kernlab** is a flexible SVM implementation which includes the most SVM formulations and kernels and allows for user defined kernels as well. It provides many useful options and features like a method for plotting, class probabilities output, cross validation error estimation, automatic hyper-parameter estimation for the Gaussian RBF kernel, but lacks a proper model selection tool. The `svm()` function in **e1071** is a robust interface to the award winning `libsvm` SVM library and includes a model selection tool, the `tune()` function, and a sparse matrix interface along with a `plot()` method and features like accuracy estimation and class-probabilities output, but does not give the user the flexibility of choosing a custom kernel. `svmlight()` in package **klaR** provides a very basic interface to `SVMlight` and has many drawbacks. It does not exploit the full potential of `SVMlight` and seems to be quite slow. The `SVMlight` license is also quite restrictive and in particular only allows non-commercial usage. `svmpath()` does not provide many features but can nevertheless be used as an exploratory tool, in particular for locating a proper value for the regularization parameter $\lambda = 1/C$.

The existing implementations provide a relatively wide range of features and options but the implementations can be extended by incorporating new features which arise in the ongoing research in SVM. One obvious extension would be to allow for weights on the data points (Lin and Wang, 1999) which is currently not supported by any of the implementations, the return of the original predictor coefficients in the case of the linear kernel or an interface and kernel for doing computation directly on structured data like strings trees.

	`ksvm()` (**kernlab**)	`svm()` (**e1071**)	`svmlight()` (**klaR**)	`svmpath()` (**svmpath**)
Formulations	C-SVC, ν-SVC, C-BSVC, spoc-SVC, one-SVC, ϵ-SVR, ν-SVR, ϵ-BSVR	C-SVC, ν-SVC, one-SVC, ϵ-SVR, ν-SVR	C-SVC, ϵ-SVR	binary C-SVC
Kernels	Gaussian, polynomial, linear, sigmoid, Laplace, Bessel, Anova, Spline	Gaussian, polynomial, linear, sigmoid	Gaussian, polynomial, linear, sigmoid	Gaussian, polynomial
Optimizer	SMO, TRON	SMO	chunking	NA
Model Selection	hyper-parameter estimation for Gaussian kernels	grid-search function	NA	NA
Data	formula, matrix	formula, matrix, sparse matrix	formula, matrix	matrix
Interfaces	`.Call`	`.C`	temporary files	`.C`
Class System	`S4`	`S3`	none	`S3`
Extensibility	custom kernel functions	NA	NA	custom kernel functions
Add-ons	plot function	plot functions, accuracy	NA	plot function
License	GPL	GPL	non-commercial	GPL

Table 3.3: A quick overview of the SVM implementations.

Chapter 4

Step Size-Adapted Online Support Vector Learning

4.1 Introduction

Stochastic ("online") gradient methods incrementally update their hypothesis by descending a stochastic approximation of the gradient computed from just the current observation. Although they require more iterations to converge than traditional deterministic ("batch") techniques, each iteration is faster as there is no need to go through the entire training set to measure the current gradient. For large, redundant data sets, or continuing (potentially non-stationary) streams of data, stochastic gradient thus outperforms classical optimization methods. Much work in this area centers on the key issue of choosing an appropriate time-dependent gradient step size η_t.

Recent years have seen a growing interest in stochastic gradient algorithms applied to kernel methods (Kivinen *et al.*, 2004b). These algorithms typically let η_t simply decay in $O(t^{-\frac{1}{2}})$. Here we adopt the more sophisticated approach of *stochastic meta-descent*: performing a simultaneous stochastic gradient descent on the step size itself. Translating this technique into the kernel framework yields a fast online optimization method for SVMs.

In this chapter we present an online Support Vector Machine (SVM) that uses Stochastic Meta-Descent (SMD) to adapt its step size automatically. We formulate the online learning problem as a stochastic gradient descent in Reproducing Kernel Hilbert Space (RKHS) and translate SMD to the nonparametric setting, where its gradient trace parameter is no longer a coefficient vector but an element of the RKHS. We derive efficient updates that allow us to perform the step size adaptation in linear time. We apply the online SVM framework to a variety of loss functions and in particular show how to achieve efficient online multiclass classification. Experimental evidence suggests that our algorithm outperforms existing methods.

Outline. We begin by providing an overview of SMD in Section 4.2. We then briefly describe the optimization problems arising from SVMs and Gaussian Processes in Section 4.3. The application of SMD to these problems is discussed in Section 4.4. Experiments are presented in Section 4.5, followed by a discussion.

4.2 Stochastic Meta-Descent

The SMD online algorithm (Schraudolph, 1999, 2002) for gradient step size adaptation can greatly accelerate the convergence of stochastic gradient descent; successful applications to date include independent component analysis (Schraudolph and Giannakopoulos, 2000), turbulent flow modeling (Milano, 2002), and visual hand tracking (Bray *et al.*, 2005). SMD updates a system's parameters $\boldsymbol{\alpha}$ by the simple gradient descent

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \boldsymbol{\eta}_t \cdot \boldsymbol{g}_t , \tag{4.1}$$

where $\boldsymbol{g}$ denotes the stochastic gradient, and $\cdot$ the element-wise (Hadamard) product. The vector $\boldsymbol{\eta}$ of individual step sizes is adapted multiplicatively

$$\boldsymbol{\eta}_t = \boldsymbol{\eta}_{t-1} \cdot \max(\tfrac{1}{2}, 1 + \mu\, \boldsymbol{v}_t \cdot \boldsymbol{g}_t) \tag{4.2}$$

using a scalar meta-learning rate μ. Finally, the auxiliary vector $\boldsymbol{v}$ used in (4.2) is itself updated iteratively via

$$\boldsymbol{v}_{t+1} = \varrho \boldsymbol{v}_t + \boldsymbol{\eta}_t \cdot (\boldsymbol{g}_t - \varrho \boldsymbol{G}_t \boldsymbol{v}_t) , \tag{4.3}$$

where $\boldsymbol{G}_t \succeq 0$ is an extended Gauss-Newton approximation (Schraudolph, 2002) of the Hessian at time t, and $0 \le \varrho \le 1$ a decay factor. SMD is derived as a dual gradient descent procedure, minimizing the objective with respect to both $\boldsymbol{\alpha}$ and $\boldsymbol{\eta}$ simultaneously. The $\boldsymbol{G}\boldsymbol{v}_t$ term in (4.3) is typically computed implicitly (Schraudolph, 2002) using efficient procedures from algorithmic differentiation (Griewank, 2000) that do not require explicit — and likely to be computationally expensive — computation of $\boldsymbol{G}$.

4.3 Online Kernel Methods

We now present various kernel methods from a loss function and regularization point of view. Our notation closely follows (Kivinen *et al.*, 2004b) with minor modifications and extensions.

4.3.1 Optimization Problem

Denote by $\mathcal{X}$ the space of observations and $\mathcal{Y}$ be the space of labels (wherever appropriate). We use $|\mathcal{Y}|$ to denote the size of $\mathcal{Y}$. Given a sequence $\{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$ of examples and a loss function $l : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \to \mathbb{R}$ the goal is to minimize the regularized risk

$$J(f) = \frac{1}{m} \sum_{i=1}^{m} l(\boldsymbol{x}_i, y_i, f) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2, \tag{4.4}$$

where $\mathcal{H}$ is a Reproducing Kernel Hilbert Space (RKHS) of functions on $\mathcal{X} \times \mathcal{Y}$. Its defining kernel is denoted by $k : (\mathcal{X} \times \mathcal{Y})^2 \to \mathbb{R}$ and it satisfies $\langle f, k((\boldsymbol{x}, y), \cdot) \rangle_{\mathcal{H}} = f(\boldsymbol{x}, y)$. In a departure from tradition we let our kernel depend on the labels as well as the observations. Finally, we make the assumption that l only depends on f via its evaluations at $f(\boldsymbol{x}_i, y)$ and that l is piecewise differentiable.

By the reproducing property of $\mathcal{H}$ we can compute derivatives of the evaluation functional. That is,

$$\partial_f f(\boldsymbol{x}, y) = \partial_f \langle f, k((\boldsymbol{x}, y), \cdot) \rangle_{\mathcal{H}} = k((\boldsymbol{x}, y), \cdot). \tag{4.5}$$

Since l depends on f only via its evaluations we can see that $\partial_f l(\boldsymbol{x}, y, f) \in \mathcal{H}$, and more specifically

$$\partial_f l(\boldsymbol{x}, y, f) \in \operatorname{span}\{k((\boldsymbol{x}, \tilde{y}), \cdot) \text{ where } \tilde{y} \in \mathcal{Y}\}. \tag{4.6}$$

Using the stochastic approximation of $J(f)$:

$$J_t(f) := l(\boldsymbol{x}_t, y_t, f) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 \tag{4.7}$$

and setting $g_t := \partial_f J_t(f_t)$, we can write the following online learning algorithm:

Algorithm 1 Online learning (adaptive step size)

```
1. Initialize $f_0 = 0$
2. **Repeat**
    (a) Draw data sample $(\boldsymbol{x}_t, y_t)$
    (b) Adapt step size $\eta_t$
    (c) Update $f_{t+1} \leftarrow f_t - \eta_t g_t$
```

Practical considerations are how to implement steps 2.b and 2.c efficiently. We will discuss 2.c below. Step 2.b, which distinguishes the present algorithm from the update rules of previous algorithms, is discussed in Section 4.4.

4.3.2 Loss Functions

We now give specific details for two loss functions used in kernel methods. Similar derivations can be found for binary classification, logistic regression, ε-insensitive regression, Huber's robust regression, LMS problems, and graph-structured output domains.

Multiclass Classification Here we employ a definition of the margin arising from log-likelihood ratios. This leads to

$$l(\boldsymbol{x}, y, f) = \max(0, 1 + \max_{\tilde{y} \neq y} f(\boldsymbol{x}, \tilde{y}) - f(\boldsymbol{x}, y))$$

$$\partial_f l(\boldsymbol{x}, y, f) = \begin{cases} 0 \text{ if } f(\boldsymbol{x}, y) \geq 1 + f(\boldsymbol{x}, y^*) \\ k((\boldsymbol{x}, y^*), \cdot) - k((\boldsymbol{x}, y), \cdot) \text{ otherwise} \end{cases} \tag{4.8}$$

Here y^* is the maximizer of the $\max_{\tilde{y} \neq y}$ operation. If several y^* exist we pick one arbitrarily, *e.g.* by dictionary order. Note that when the number of classes is exactly two (binary classification) and $k((x, y), (x', y')) = \frac{yy'}{2} k(x, x')$ the loss function reduces to the well-known hinge loss.

Novelty Detection uses a trimmed version of the log-likelihood as a loss function. This means that labels are ignored and the one-class margin needs to exceed 1, leading to

$$l(\boldsymbol{x}, y, f) = \max(0, 1 - f(\boldsymbol{x}))$$

$$\partial_f l(\boldsymbol{x}, y, f) = \begin{cases} 0 \text{ if } f(\boldsymbol{x}) \geq 1 \\ -k(\boldsymbol{x}, \cdot) \text{ otherwise} \end{cases} \tag{4.9}$$

Table 4.1 summarizes the expansion coefficient(s) $\boldsymbol{\xi}_t$ arising from the derivative of the loss at time t.

Table 4.1: Gradient expansion coefficients.

task	expansion coefficient
Multiclass Classification	$\boldsymbol{\xi}_t = \mathbf{0}$ if $f_t(\boldsymbol{x}_t, y_t) \geq 1 + f_t(\boldsymbol{x}_t, y^*)$ $\xi_{t,y_t} = -1, \xi_{t,y^*} = 1$ otherwise
Novelty Detection	$\xi_t = \begin{cases} 0 & \text{if } f_t(\boldsymbol{x}_t) \geq 1 \\ -1 & \text{otherwise} \end{cases}$

4.3.3 Coefficient Updates

Since the update step 2.c in Algorithm 1 is not particularly useful in Hilbert space, we now rephrase it in terms of kernel function expansions. From (4.7) it follows that $g_t = \partial_f l(\boldsymbol{x}_t, y_t, f_t) + \lambda f_t$ and consequently

$$\begin{aligned} f_{t+1} &= f_t - \eta_t \left[\partial_f l(\boldsymbol{x}_t, y_t, f_t) + \lambda f_t\right] \\ &= (1 - \lambda\eta_t) f_t - \eta_t \partial_f l(\boldsymbol{x}_t, y_t, f_t). \end{aligned} \tag{4.10}$$

Using the initialization $f_1 = 0$ this implies that

$$f_{t+1}(\cdot) = \sum_{i=1}^{t} \sum_{y} \alpha_{tiy} k((\boldsymbol{x}_i, y), \cdot). \tag{4.11}$$

With some abuse of notation we will use the same expression for the cases where $\mathcal{H}$ is defined on $\mathcal{X}$ rather than $\mathcal{X} \times \mathcal{Y}$. In this setting we replace (4.11) by the sum over i only (with corresponding coefficients α_{ti}). Whenever necessary we will use $\boldsymbol{\alpha}_t$ to refer to the entire coefficient vector (or matrix) and α_{ti} (or α_{tiy}) will refer to the specific coefficients. Observe that we can write

$$g_t(\cdot) = \sum_{i=1}^{t} \sum_{y} \gamma_{tiy} k((\boldsymbol{x}_i, y), \cdot), \tag{4.12}$$

$$\text{where} \quad \boldsymbol{\gamma}_t := \begin{bmatrix} \lambda \boldsymbol{\alpha}_{t-1} \\ \boldsymbol{\xi}_t^\top \end{bmatrix}. \tag{4.13}$$

We can now rewrite (4.10) using the expansion coefficients as

$$\boldsymbol{\alpha}_t = \begin{bmatrix} (1 - \lambda\eta_t)\boldsymbol{\alpha}_{t-1} \\ -\eta_t \boldsymbol{\xi}_t^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{bmatrix} - \eta_t \boldsymbol{\gamma}_t. \tag{4.14}$$

Note that conceptually $\boldsymbol{\alpha}$ grows indefinitely as it acquires an additional row with each new data sample. Practical implementations will of course retain only a buffer of past examples with nonzero coefficients. If the loss function has a bounded gradient (as in all cases of Table 4.1) then the quality of the approximation increases exponentially with the number of terms retained (Kivinen *et al.*, 2004b), so good solutions can be obtained with limited buffer size.

4.3.4 Handling Offsets

In many situations, for instance in binary classification, it is advantageous to predict with $f(\cdot, y) + b_y$ where $f \in \mathcal{H}$ and $\mathbf{b} \in \mathbb{R}^{|\mathcal{Y}|}$ is an offset parameter. While the update equations described above remain unchanged, the offset $\mathbf{b}$ is now adapted as well:

$$\mathbf{b}_{t+1} = \mathbf{b}_t - \eta_t \partial_{\mathbf{b}} J_t(f_t + \mathbf{b}_t) = \mathbf{b}_t - \eta_t \boldsymbol{\xi}_t. \tag{4.15}$$

4.4 Online SVMD

We now show how the SMD framework described in Section 4.2 can be used to adapt the step size for online SVMs. The updates given in Section 4.3 remain as before, the only difference being that the step size η_t is adapted before its value is used to update $\boldsymbol{\alpha}$.

4.4.1 Scalar Representation

When η is a scalar, (4.2) becomes

$$\eta_{t+1} = \eta_t \max(\tfrac{1}{2}, 1 - \mu \langle g_{t+1}, v_{t+1} \rangle), \tag{4.16}$$

where μ is the meta-step size described in Section 4.2. The update for v is now given by

$$v_{t+1} = \varrho v_t - \eta_t (g_t + \varrho \boldsymbol{H}_t v_t), \tag{4.17}$$

where $\boldsymbol{H}_t$ is the Hessian of the objective function. Note that now $\boldsymbol{H}_t$ is an operator in Hilbert space. For piecewise linear loss functions, such as (4.8), and (4.9), we have $\boldsymbol{H}_t = \lambda \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity operator, and obtain the simple update

$$v_{t+1} = (1 - \eta_t \lambda) \varrho v_t - \eta_t g_t. \tag{4.18}$$

4.4.2 Expansion in Hilbert Space

The above discussion implies that v can be expressed as a linear combination of kernel functions, and consequently is also a member of the RKHS defined by $k(\cdot, \cdot)$. Thus v cannot be updated explicitly, as is done in the normal SMD algorithm (Section 4.2). Instead we write

$$v_{t+1}(\cdot) = \sum_{i=1}^{t} \sum_{y} \beta_{tiy} k((\boldsymbol{x}_i, y), \cdot) \tag{4.19}$$

and update the coefficients $\boldsymbol{\beta}$. This is sufficient for our purpose because we only need to be able to compute the inner products $\langle g, v \rangle_{\mathcal{H}}$ in order to update η.

Analogous to the update on $\boldsymbol{\alpha}$ we can determine the updates on $\boldsymbol{\beta}$ via

$$\boldsymbol{\beta}_t = \begin{bmatrix} (1 - \eta_t \lambda) \varrho \boldsymbol{\beta}_{t-1} \\ 0 \end{bmatrix} - \eta_t \boldsymbol{\gamma}_t. \tag{4.20}$$

Although (4.20) suffices in principle to implement the overall algorithm, a naive implementation of the inner product $\langle g_t, v_t \rangle$ takes $O(t^2)$ time. In the next section we show how these updates can be performed in linear time.

4.4.3 Linear-Time Incremental Updates

We now turn to computing $\langle g_{t+1}, v_{t+1}\rangle$ in linear time by bringing it into an incremental form. We use the notation $f(\boldsymbol{x}_t, \cdot)$ to denote the vector of $f(\boldsymbol{x}_t, \tilde{y})$ for $\tilde{y} \in \mathcal{Y}$. Expanding g_{t+1} into $\lambda f_{t+1} + \boldsymbol{\xi}_{t+1}$ we can write

$$\langle g_{t+1}, v_{t+1}\rangle = \lambda \pi_{t+1} + \boldsymbol{\xi}_{t+1}^\top v_{t+1}(\boldsymbol{x}_{t+1}, \cdot), \tag{4.21}$$

where $\pi_t := \langle f_t, v_t\rangle$. The function update (4.10) yields

$$\pi_{t+1} = (1 - \lambda\eta_t) \langle f_t, v_{t+1}\rangle - \eta_t \boldsymbol{\xi}_t^\top v_{t+1}(\boldsymbol{x}_t, \cdot). \tag{4.22}$$

The v update (4.17) then gives us

$$\langle f_t, v_{t+1}\rangle = \varrho(1 - \lambda\eta_t)\pi_t - \eta_t \langle f_t, g_t\rangle, \tag{4.23}$$

and using $g_t = \lambda f_t + \boldsymbol{\xi}_t$ again we have

$$\langle f_t, g_t\rangle = \lambda \|f_t\|^2 + \boldsymbol{\xi}_t^\top f_t(\boldsymbol{x}_t, \cdot). \tag{4.24}$$

Finally, the squared norm of f can be maintained via:

$$\begin{aligned} \|f_{t+1}\|^2 = {} & (1 - \lambda\eta_t)^2 \|f_t\|^2 \\ & - 2\eta_t(1 - \lambda\eta_t)\boldsymbol{\xi}_t^\top f_t(\boldsymbol{x}_t, \cdot) \\ & + \eta_t^2 \boldsymbol{\xi}_t^\top k((\boldsymbol{x}_t, \cdot), (\boldsymbol{x}_t, \cdot))\boldsymbol{\xi}_t. \end{aligned} \tag{4.25}$$

The above sequence (4.21)–(4.25) of equations, including the evaluation of the associated functionals, can be performed in $O(t)$ time.

4.5 Experiments

To show the utility of our approach we performed experiments on the the USPS data set following (Kivinen *et al.*, 2004b). The USPS dataset have been used extensively in the past, especially in the SVM community. It consists of 7291 training and 2007 test examples represented as 256 dimensional vectors(16x16 matrices) with entries between 0 and 255 (grey level). The vectors are images of handwritten digits from 0 to 9 and the task is to learn to asign a handwritten digit to its corresponding class. Since our approach is particularly useful when the data is non-stationary we create a data set where we use 4 letters from the data set (0-3) and split them in two classes. We create a data set of 1000 data points where the task at the first 500 is to classify between the 0 and 1 classes and at the second part of the data

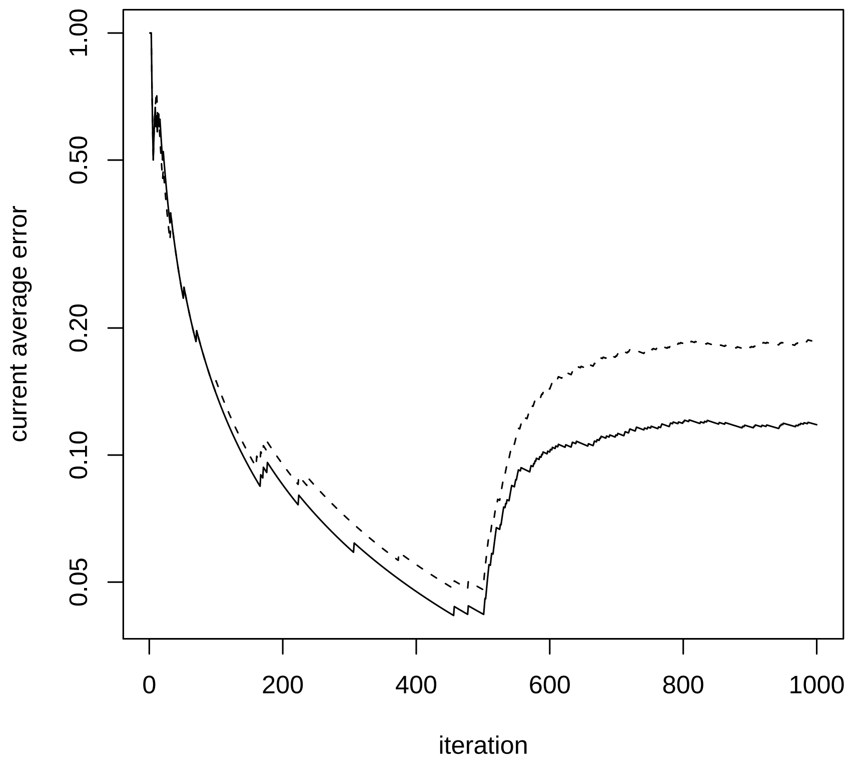

Figure 4.1: Average error rate (on a log scale) incurred over a single run through the digits 0 and 1 until iteration 500 and 2, 3 from 500 onwards of the USPS data set, for SVMD (solid) *vs.* online SVM with scheduled step size decay (dashed). SVMD performs better throughout the data.

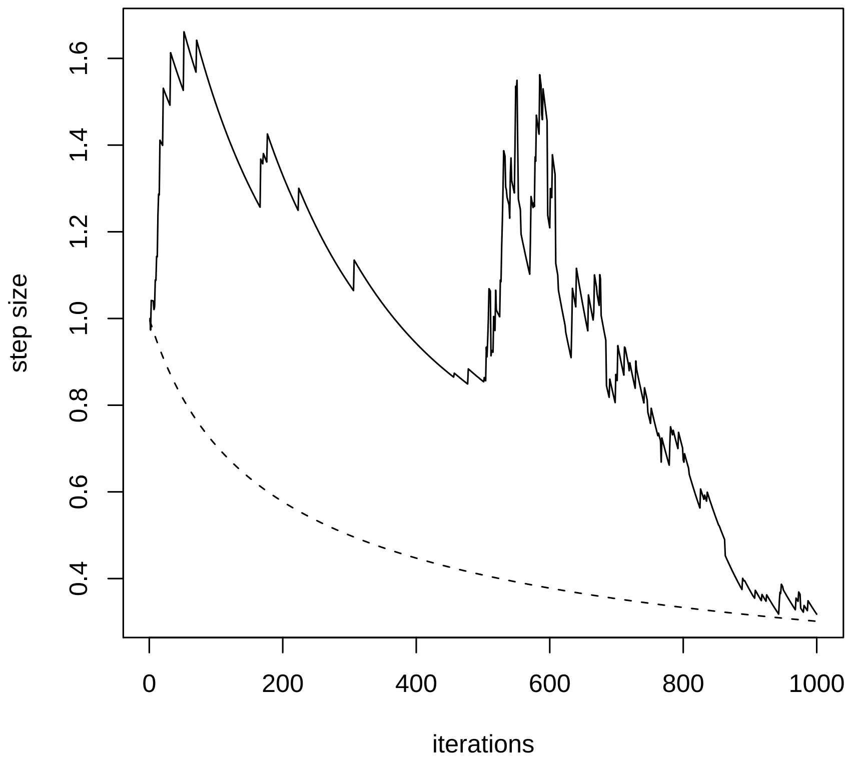

Figure 4.2: Step size for classification over a single run of the USPS data set where the first half of the data set are zeros and ones and the second half twos and threes. The dased line is the sheduled step size of the online SVM. Observe how the SVMD values (solid) of η_t respond to the change in the dataset.

between the 2 and 3 classes. This sudden change of the distribution in the data although not usual illustrates the usefulness of the SVMD algorithm.

We compare SVMD to the conventional online SVM algorithm on binary classification, using a standard decay for the learning rate $\eta_t = \sqrt{\tau/(\tau+t)}$, where τ was tuned appropriately to obtain good performance. A Gaussian kernel width $\sigma = 10$ was used and the SVMD parameters where set to $\mu = 0.1$ and $\rho = 1$ for this experiment.

We plot the value of the average error rate during the iterations in Figure 4.1 as well as the value of η_t in Figure 4.2. In Figure 4.1 we can see that SVMD outperforms the online SVM algorithm during the initial 500 stationary iterations and adapts well to the sudden change in the distribution of the data outperforming the online SVM.

SVMD adapts the values of the learning rate η_t quite well to the changes in the data set. Although both algorithms start with an initial value of $\eta_t = 1$ in the case of SVMD η is raised at the initial iterations providing slightly better performance and then decays until halfway through the data where in response to the change of the distribution η increases again. This shows that our method is well suited for learning non-stationary distributions.

For our next experiment we use the full USPS dataset and performe multi-class classification. Figure 4.3 shows our results for 10-way multiclass classification using soft margin loss with $\lambda = 2.15 \times 10^{-7}$, $\eta_0 = 0.1$, and $\mu = 0.1$. We plot current average error rate ,that is, the total number of errors divided by the iteration number, and the step size in Figure 4.4. Online SVMD (solid) makes significantly fewer classification errors compared to other methods, adapting η from its conservative initial value to achieve good performance. In our experiments we find the performance of online SVMD fairly independent of the initial step size.

4.6 Conclusions

We presented online SVMD, an extension of the SMD step size adaptation method to the kernel framework. Using an incremental approach to reduce a naive $O(t^2)$ computation to $O(t)$, we showed how the SMD parameters can be updated efficiently even though they now reside in an RKHS. In experiments online SVMD outperformed the online SVM with scheduled step size decay.

Adaptive step size control is clearly most useful when faced with a data stream exhibiting irregular nonstationarities that thwart less reactive approaches, such as fixed decay schedules. We expect SVMD to become the first viable online kernel algorithm for such data.

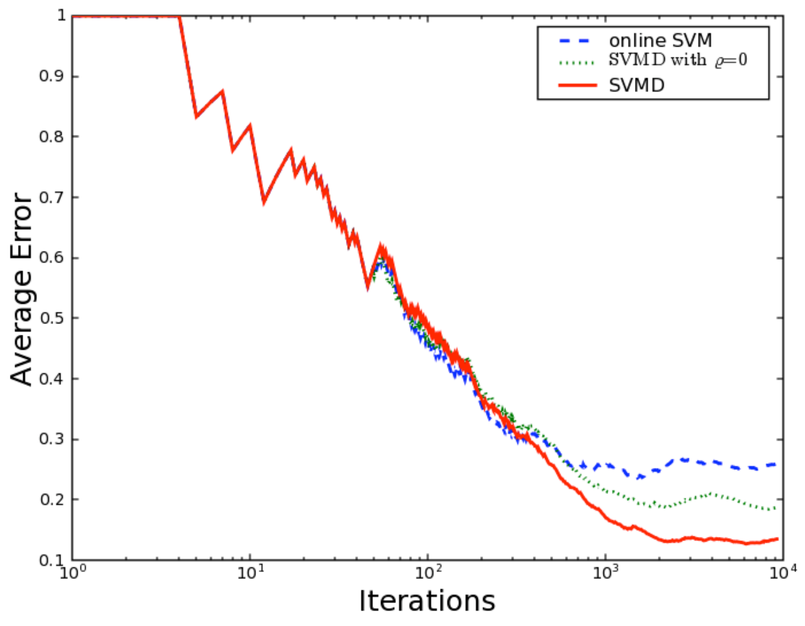

Figure 4.3: Online 10-way multiclass classification over a single run through the USPS dataset. Current average error (left) for SVMD with $\varrho = 0.99$ (solid), $\varrho = 0$ (dotted), and online SVM with step size decay using $\tau = 100$ (dashed).

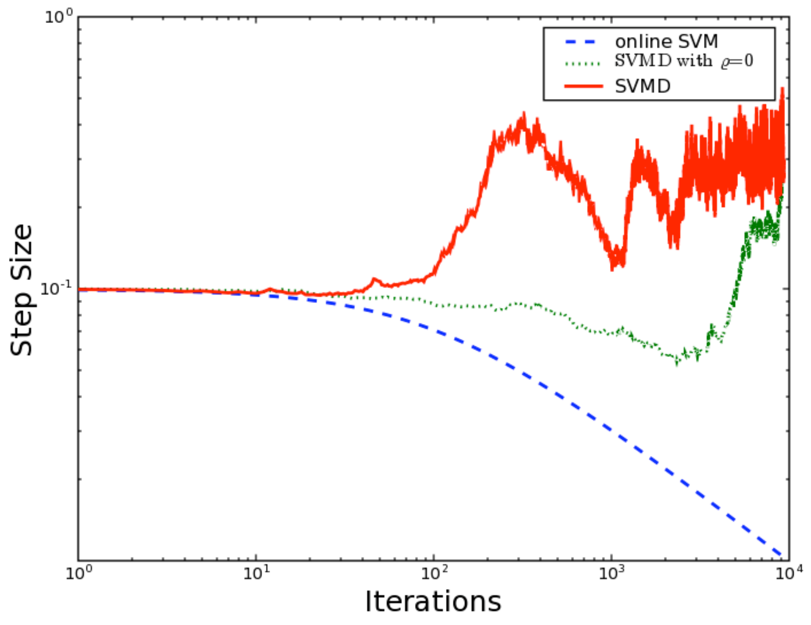

Figure 4.4: Online 10-way multiclass classification over a single run through the USPS dataset. Step size for SVMD with $\varrho = 0.99$ (solid), $\varrho = 0$ (dotted), and online SVM with step size decay using $\tau = 100$ (dashed).

Future research will focus on the use of online learning techniques to prove worst case loss bounds for SVMD.

Chapter 5

Text clustering with string kernels in R

5.1 Introduction

The application of machine learning techniques to large collections of text documents is a major research area with many application such as document filtering and ranking. Kernel-based methods have been shown to perform rather well in this area, particularly in text classification with SVM using either a simple "bag of words" representation (i.e. term frequencies with various normalizations) (Joachims, 1999), or more sophisticated approaches like string kernels (Lodhi *et al.*, 2002), or word-sequence kernels (Cancedda *et al.*, 2003). Despite the good performance of kernel methods in classification of text documents, little has been done on the field of clustering text documents with kernel-based methods.

In this chapter we present a package which provides a general framework, including tools and algorithms, for text mining in R (R Development Core Team, 2005) using the S4 class system. Using this package and the `kernlab` R package we explore the use of kernel methods for clustering (e.g., kernel k-means and spectral clustering) on a set of text documents, using string kernels. We compare these methods to a more traditional clustering technique like k-means on a bag of word representation of the text and evaluate the viability of kernel-based methods as a text clustering technique.

5.2 Software

R is a natural choice for a text mining environment. Besides the basic string and character processing functions it includes an abundance of statistical analysis functions and packages and provides a Machine Learning task view with a wide range of software.

5.2.1 The `textmin` R Package

The `textmin` package provides a framework for text mining applications within R. It fully supports the new S4 class system and integrates seamlessly into the R architecture.

The basic framework classes for handling text documents are:

`textdocument`: Encapsulates a text document, irrelevant from its origin, in one class. Several slots are available for additional meta data, like an unique identification number or a description.

`textdoccol`: Represents a collection of text documents. The constructor provides import facilities for common data formats in text mining applications, like the Reuters21578 news format or the Reuters Corpus Volume 1 format.

`termdocmatrix`: Stands for a term-document matrix with documents (in fact their id numbers) as rows and terms as columns. Such a term-document matrix can be easily built from a text document collection. A bunch of weighting schemes are available, like binary, term frequency or term frequency inverse document frequency. This class can be used as a fast representation for all kinds of bag-of-words text mining algorithms.

Further, this package ships with scripts to extract all possible splits from the Reuters21578 (Lewis, 1997) XML data.

5.2.2 kernlab

The kernel methods used in this chapter are already present in the **kernlab** package. For performance reasons we implemented two versions of string kernels in C linked through the `.Call` interface to R. For convenience we also implemented a kernel matrix interface for the spectral clustering `specc()` and the kernel k-means `kkmeans()` methods.

5.3 Methods

The k-means clustering algorithm is one of the most commonly used clustering methods providing solid results but also having some drawbacks. Denoting clusters by π_j and a partitioning of points as ${\pi_j}_{j=1}^k$ the k-means objective function using Euclidean distances becomes :

$$D({\pi_j}_{j=1}^k) = \sum_{j=1}^{k} \sum_{a \in \pi_j} \|a - m_j\|^2$$

$$\text{where} \quad m_j = \tfrac{1}{\|\pi_j\|} \textstyle\sum_{a \in \pi_j} a \tag{5.1}$$

A major drawback of k-means is that it cannot separate clusters that are not linearly separable in input space.

5.3.1 Kernel k-means

One technique for dealing with this problem is mapping the data into a high-dimensional non-linear feature space with the use of a kernel. Kernel k-means uses a kernel function to compute the inner product of the data in the feature space. All computations are then expressed in terms of inner products thus allowing the implicit mapping of the data into this feature space. If Φ is the mapping function then the k-means objective function using Euclidean distances becomes :

$$\mathcal{D}({\pi_j}_{j=1}^k) = \sum_{j=1}^{k} \sum_{a \in \pi_j} \|\Phi(a) - m_j\|^2$$

$$\text{where} \quad m_j = \tfrac{1}{\|\pi_j\|} \textstyle\sum_{a \in \pi_j} \Phi(a) \tag{5.2}$$

in the expansion of the square norm only inner products of the form $\langle \Phi(a), \Phi(b) \rangle$ appear which are computed by the kernel function $k(a, b)$.

The implementation of kernel k-means included in **kernlab** makes use of the triangle inequality (Elkan, 2003) in order to avoid unnecessary and computational expensive distance calculations. This leads to significant speedup particularly on large data sets with a high number of clusters.

5.3.2 Spectral Clustering

Spectral clustering (Ng *et al.*, 2001b), (Shi and Malik, 2000) works by embedding the data points of the partitioning problem into the subspace of the k largest eigenvectors of a normalized affinity matrix. The use of an affinity matrix also brings one of the advantages of kernel methods to spectral clustering, since one can define a suitable affinity for a given application. For example if the feature vectors represent color histograms simple k-means clustering is inappropriate since an L_2 distance between histograms isn't meaningful. In such a case one can employ a suitable affinity function such as the χ^2-distance. In our case we use a string kernel to define the affinities between two documents and construct the kernel matrix. The data is then embedded into the subspace of the largest eigenvectors of the normalized kernel matrix. This embedding usually leads to more straightforward clustering problems since points tend to form tight clusters in the eigenvector subspace. Using a simple clustering method like k-means on the embedded points usually leads to good performance. It can be shown that most spectral clustering methods boil down to a a graph partitioning problem (Dhillon *et al.*, 2004) that can be solved by a weighted kernel k-means algorithm.

5.3.3 String kernels

String kernels (Watkins, 2000), (Herbrich, 2002) are defined as a similarity measure between two sets of characters x and x'. The generic form of string kernels is given by the equation :

$$k(x,x') = \sum_{s \sqsubseteq x, s' \sqsubseteq x'} \lambda_s \delta_{s,s'} = \sum_{s \in A^*} \text{num}_s(x)\text{num}_s(x')\lambda_s \tag{5.3}$$

where A^* represents the set of all non empty strings and λ_s is a weight or decay factor which can be chosen to be fixed for all substrings or can be set to a different value for each substring. This generic representation includes a large number of special cases, e.g. setting $\lambda_s \neq 0$ only for substrings that start and end with a white space character gives the "bag of words" kernel (Joachims, 2002). In this chapter we will focus on the case where $\lambda_s = 0$ for all $|s| > n$ that is comparing all substrings of length less that n, this kernel will be referred to in the rest of the chapter as full string kernel. We also consider the case where $\lambda_s = 0$ for all $|s| \neq n$ which we referred to as the string kernel. The computational complexity of the string kernels we consider is $O(\text{n}, |x|, |x'|)$.

5.4 Experiments

We will now compare the performance of the various clustering techniques on text data by running a series of experiments on the well known Reuters text data set.

5.4.1 Data

The Reuters-21578 dataset (Lewis, 1997) contains stories for the Reuters news agency. It was compiled by David Lewis in 1987, is publicly available and is currently one of the most widely used datasets for text categorization research. A Reuters category can contain as few as 1 or as many as 2877 documents. In our experiments we used a subset of the Reuters dataset so that the computation of a full kernel matrix in memory was not a concern. We used the "crude" which contains about 580 documents, the "corn" category which includes 280 documents and a sample of 1100 documents from the "acq" category. Our dataset thus consist of 1720 documents.

We removed the stop words that occur in a stop list and any empty documents and convert all characters to lower case. We also removed punctuation and white space and performed stemming on the documents using the **Rstem** (Lang, 2005) `omegahat` R package.

5.4.2 Experimental Setup

We perform clustering on the dataset using the kernel k-means and spectral clustering methods in the **kernlab** package and the k-means method in R. For the kernel k-means and spectral methods we also use a the string kernels implementations provided in **kernlab**. In order to learn more about the effect of the string kernels hyper-parameters on the clustering results we run the clustering algorithms over a range of the length parameter n which controls the length of the strings compared in the two character sets and the decay factor λ. We study the effects of the parameters by keeping the value of the decay parameter λ fixed and varying the length parameter. Note that for each parameter set a new kernel matrix containing different information has to be computed.

We use values from $n = 3$ to $n = 14$ for the length parameter and $\lambda = 0.2$, $\lambda = 0.5$ and $\lambda = 0.8$ for the decay factor. We also use both the string (or spectral) and the full string kernel and normalize in order to remove any bias introduced by document length. We thus use a new embedding $\hat{\phi} = \frac{\phi(s)}{\|\phi(s)\|}$

which gives rise to the kernel :

$$\hat{K}(s,s') = \langle \hat{\phi}(s), \hat{\phi}(s') \rangle = \left\langle \frac{\phi(s)}{\|\phi(s)\|} \frac{\phi(s')}{\|\phi(s')\|} \right\rangle =$$

$$\frac{\langle \phi(s), \phi(s') \rangle}{\|\phi(s)\|\|\phi(s')\|} = \frac{K(s,s')}{\sqrt{K(s,s)K(s',s')}} \tag{5.4}$$

For the classical k-means method we create a term document matrix of the term frequencies and also an inverse term frequencies matrix.

5.4.3 Performance measure

We evaluate the performance of the various clustering techniques using the recall rate which is a typical measure for evaluating the performance of a clustering algorithm when the actual labels of the clustered data are known. Given a discovered cluster γ and the associated reference cluster Γ, recall R is defined as in :

$$R = \frac{\sum_{\Gamma=1}^{k} n_{\gamma\Gamma}}{\sum_{\Gamma=1}^{k} N_{\Gamma}} \tag{5.5}$$

where $n_{\gamma\Gamma}$ is the number of documents from reference cluster Γ assigned to cluster γ, N_{Γ} is the total number of documents in cluster γ and N_{Γ} is the total number of documents in reference cluster Γ.

5.4.4 Results

The main goal of these experiments is to establish if kernel methods along with string kernels are a valiable solution for grouping a set of text documents.

From the experiments we run it became obvious that the λ parameter influences the performance only minimally and thus we chose to look at the results in relation to the string length kernel parameter which seems to have a more profound influence on the performance of the kernel-based clustering methods. The performance of the k-means clustering method is also very similar with both the simple document matrix or the inverse frequency document matrix.

Figure 5.1 shows the average recall rate over 10 runs for the spectral clustering methods, and the kernel k-means method with the full string kernel compared to the reference recall rate of the inverse term document matrix clustered with a simple k-means algorithm. The plot shows that both the spectral method and kernel k-means fail to improve over the performance of the standard k-means clustering technique. We also note that the spectral clustering technique provides very stable results thous almost zero variance. This can be attributed to the fact that the projection of the data into the eigenspace groups the data into tight clusters which are easy to separate with a standard clustering technique.

Figure 5.1 displays the average recall rate of the kernel k-means with a string kernel along with the standard k-means clustering results. It is clear that for a range of values of the string length parameter the kernel k-kmeans functions outperforms k-means clustering and the full string kernel methods with a full string kernel. The method does not provide stable performance and the variance of the recall rate over the 10 runs seems quite high compared to the other methods.

Figure 5.3 shows the recall rate of the spectral clustering method with a string kernel averaged over 10 runs compared to the standard k-means clustering results. This is clearly the best performing clustering method for this set of text documents and also exhibits some interesting behavior. For rather small lengths of substrings considered $(3, 4, 5)$ the performance seems to increase monotonically and at the the value of 6 hits a threshold. For the range of values between 6 and 10 the performance increase is much smaller and for the value of 10 the highest recall rate of 0.927 is reached. For higher values of the length parameter the performance drops sharply only to increase again for a string length value of 14. Again this method is very stable and exhibits minimal variance.

5.4.5 Timing

We have also evaluated the methods in terms of running time. The experiments where run on a Linux machine with a 2.6 GHz Pentium 4 CPU. Table 5.1 provides the running time for the calculation of a full kernel matrix and the running time for the clustering methods. Note that the running time for the kernel-based clustering methods is the time needed to cluster data with a precomputed kernel matrix. From the results it is clear that most of the computing time is spend on the calculation of the kernel matrix.

5.5 Conclusions

From the results it is clear that the spectral clustering technique combined with a string kernel outperforms all other methods and provides very strong performance even comparable to the classification performance of an SVM

kernel matrix calculations	≈ 2 h.
spectral clustering	≈ 20 sec.
kernel k-means	≈ 30 sec.
term matrix k-means	≈ 40 sec.

Table 5.1: Timings for the clustering methods and the computation of the kernel matrix.

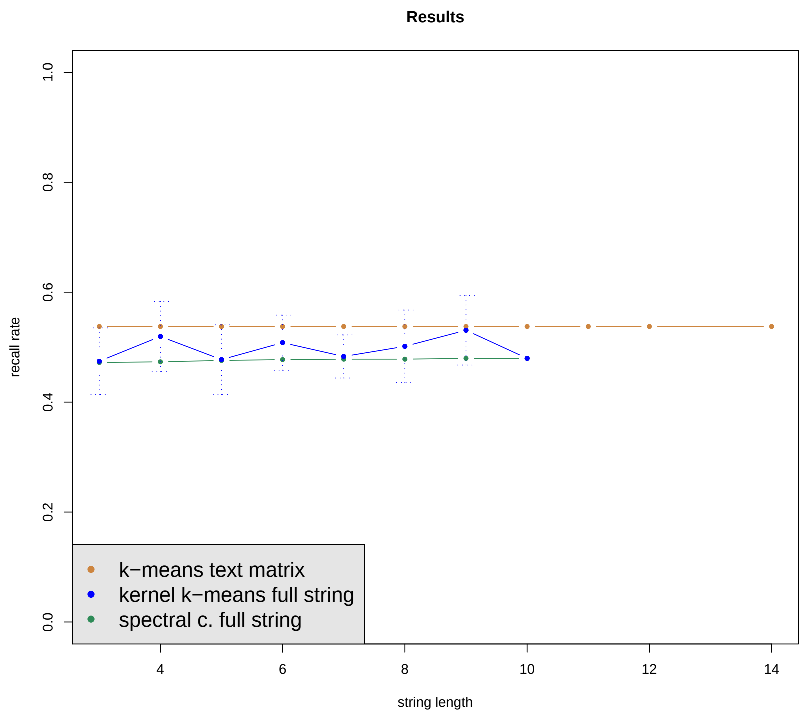

Figure 5.1: Average recall rate over 10 runs for the spectral clustering , kernel k-means, with full string kernels and k-means on a inverse frequencies term matrix methods. On the y axis is the recall rate and the x axis the string length hyper-parameter of the string kernel.

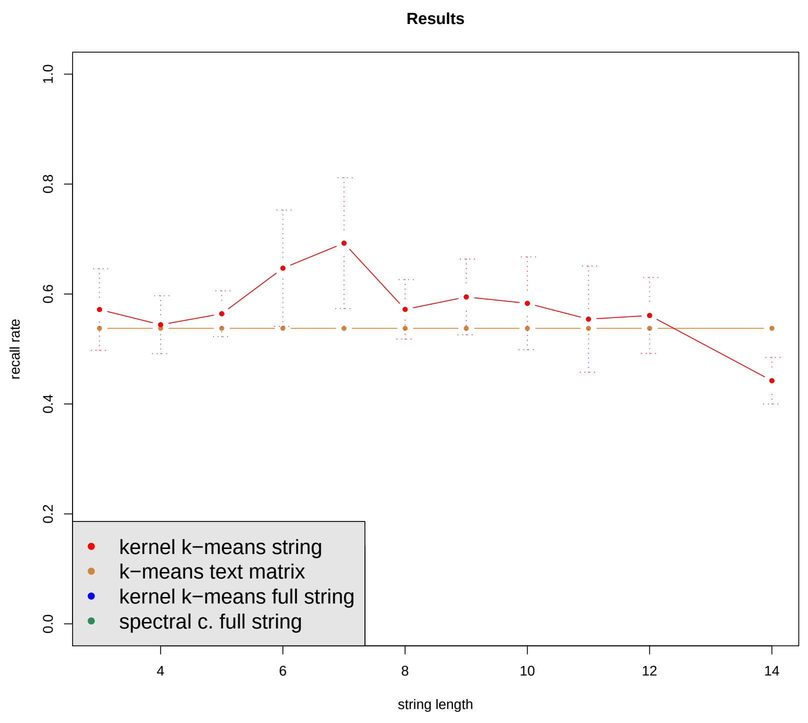

Figure 5.2: Average recall rate over 10 runs for the kernel k-means, with string/spectral kernels and k-means on a inverse frequencies term matrix methods. On the y axis is the recall rate and the x axis the string length hyper-parameter of the string kernel.

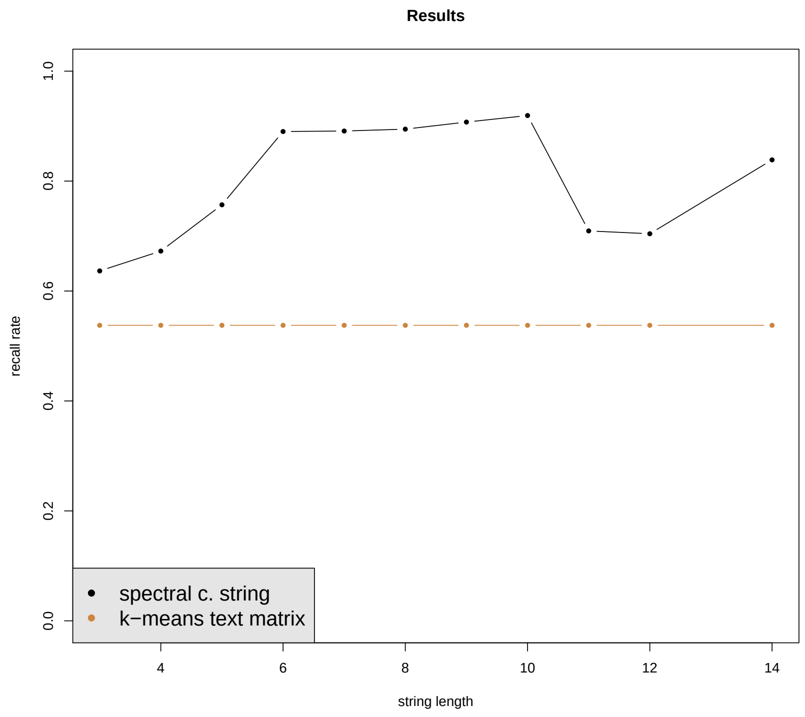

Figure 5.3: Average recall rate over 10 runs for the spectral clustering with a string/spectral kernel and the k-means on a inverse frequencies term matrix methods. The x axis represents the string length hyper-parameter of the string kernel.

with a string kernel on similar dataset (Lodhi *et al.*, 2002). This is very encouraging and shows that kernel-based clustering methods can be considered as a viable text grouping method. The behavior of the kernel-based algorithms, particularly of the spectral clustering method, seem to strongly depend on the value of the string length parameter. It is an open question if the range of good values of this parameter $(6-10)$ on this dataset can be also used on other text datasets in the same or other languages to provide good performance. It is interesting to note that a string length of 6 to 10 characters corresponds to the size of one or two words in the English language. It would also be interesting to study the behavior of the method for string lengths higher than 14. The good performance of the spectral clustering technique could be an indication that graph partitioning methods combined with string kernels could provide good results on text clustering.

One drawback of the kernel based methods is the amount of time spend on the computation of the kernel matrix and, particularly for the spectral methods, the necessity to store a full $m \times m$ where m the number of text documents, in memory. A suffix tree based implementation of the string kernels as in (Vishwanathan and Smola, 2004) combined with the Nystrom method (Williams and Seeger, 2001) for computing the eigenvectors of the kernel matrix as in (Fowlkes *et al.*, 2004) by using only a sample of the data points could provide a solution to this issues. It would also be interesting to explore the application of some other types of string kernels on text clustering. Of particular interest would be the mismatch (Leslie *et al.*, 2004) kernels especially on raw (i.e. non pre-processed) text data.

Appendix A

SVM formulations

A.0.1 ν-SVM formulation for classification

The primal quadratic programming problem for the ν-SVM is the following:

$$\begin{aligned} \text{minimize} \quad & t(\mathbf{w}, \xi, \rho) = \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\langle \Phi(x_i), \mathbf{w}\rangle + b) \geq \rho - \xi_i \qquad (i = 1, \ldots, m) \\ & \xi_i \geq 0 \qquad (i = 1, \ldots, m), \qquad \rho \geq 0. \end{aligned} \tag{A.1}$$

The dual is of the form:

$$\begin{aligned} \text{maximize} \quad & W(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{1}{m} \qquad (i = 1, \ldots, m) \\ & \sum_{i=1}^{m}\alpha_i y_i = 0 \\ & \sum_{i=1}^{m}\alpha_i \geq \nu \end{aligned} \tag{A.2}$$

A.0.2 spoc-svm for classification

The dual of the Crammer and Singer multi-class SVM problem is of the form:

$$
\begin{aligned}
\text{maximize} \quad & W(\alpha) = \sum_{i=1}^{l} \alpha_i \epsilon_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\
\text{subject to} \quad & 0 \leq \alpha_i \leq C \qquad (i = 1, \dots, m) \\
& \sum_{m=1}^{k} \alpha_i^m = 0, \qquad (i = 1, \dots, l) \\
& \sum_{i=1}^{m} \alpha_i \geq \nu
\end{aligned} \tag{A.3}
$$

A.0.3 Bound constraint C-SVM for classification

The primal form of the bound constraint C-SVM formulation is:

$$
\begin{aligned}
\text{minimize} \quad & t(\mathbf{w}, \xi) = \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{2} \beta^2 + \frac{C}{m} \sum_{i=1}^{m} \xi_i \\
\text{subject to} \quad & y_i(\langle \Phi(x_i), \mathbf{w} \rangle + b) \geq 1 - \xi_i \qquad (i = 1, \dots, m) \\
& \xi_i \geq 0 \qquad (i = 1, \dots, m)
\end{aligned} \tag{A.4}
$$

The dual form of the bound constraint C-SVM formulation is:

$$
\begin{aligned}
\text{maximize} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j (y_i y_j + k(x_i, x_j)) \\
\text{subject to} \quad & 0 \leq \alpha_i \leq \frac{C}{m} \qquad (i = 1, \dots, m) \\
& \sum_{i=1}^{m} \alpha_i y_i = 0.
\end{aligned} \tag{A.5}
$$

A.0.4 SVM for regression

The dual form of the ϵ-SVM regression is:

$$\text{maximize}\alpha \in \mathbf{R}^m = \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_i^* - \alpha_i)k(x_i, x_j) \\ \\ -\epsilon\sum_{i=1}^{m}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} y_i(\alpha_i^* - \alpha_i) \end{cases} \tag{A.6}$$

$$\text{subject to } \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \text{ and } a_i, a_i^* \in [0, C/m]$$

The primal form of the ν-SVM formulation is:

$$\begin{aligned} \text{minimize} \quad & t(\mathbf{w}, \xi^*, \epsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{\nu\epsilon} + \frac{1}{m}\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & (\langle\Phi(x_i), \mathbf{w}\rangle + b) - y_i \geq \epsilon - \xi_i \qquad (i = 1, \ldots, m) \quad \text{(A.7)} \\ & y_i - (\langle\Phi(x_i), \mathbf{w}\rangle + b) \geq \epsilon - \xi_i^* \qquad (i = 1, \ldots, m) \quad \text{(A.8)} \\ & \xi_i^* \geq 0, \qquad \epsilon \geq 0, \qquad (i = 1, \ldots, m) \end{aligned}$$

The dual form of the ν-SVM formulation is:

$$\begin{aligned} \text{maximize} \quad & W(\alpha^*) = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)y_i - \frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j) \\ \text{subject to} \quad & \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) \\ & \alpha_i^* \in \left[0, \frac{C}{m}\right], \\ & \sum_{i=1}^{m}(\alpha_i + \alpha_i^*) \leq C\nu \end{aligned} \tag{A.9}$$

A.0.5 SVM novelty detection

The dual form of the SVM QP for novelty detection is:

$$\begin{aligned} \text{minimize} \quad & W(\alpha) = \sum_{i,j}\alpha_i\alpha_j k(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu m} \qquad (i = 1, \ldots, m) \\ & \sum_i \alpha_i = 1 \end{aligned} \tag{A.10}$$

Appendix B

kernlab Reference Manual

`as.kernelMatrix` *Assing kernelMatrix class to matrix objects*

Description

`as.kernelMatrix` in package **in Package 'kernlab'** can be used to assing the kernelMatrix class to matrix objects representing a kernel matrix. This matrixes can then be used with the kernelMatrix interfaces which most of the functions in **'kernlab'** support.

Usage

```
## S4 method for signature 'matrix':
as.kernelMatrix(x)
```

Arguments

`x`	matrix to be assinged the `kernelMatrix` class

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`kernelMatrix`, `dots`

Examples

`couple` *Probabilities Coupling function*

Description

`couple` is used to link class-probability estimates produced by pairwise coupling in multi-class classification problems.

Usage

```
couple(probin, coupler = "minpair")
```

Arguments

`probin`	The pairwise coupled class-probability estimates
`coupler`	The type of coupler to use. Currently `minpar` and `pkpd` and `vote` are supported (see reference for more details). If `vote` is selected the returned value is a primitive estimate passed on given votes.

Details

As binary classification problems are much easier to solve many techniques exist to decompose multi-class classification problems into many binary classification problems (voting, error codes, etc.). Pairwise coupling (one against one) constructs a rule for discriminating between every pair of classes and then selecting the class with the most winning two-class decisions. By using Platt's probabilities output for SVM one can get a class probability for each of the $k(k-1)/2$ models created in the pairwise classification. The couple method implements various techniques to combine these probabilities.

Value

A matrix with the resulting probability estimates.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Ting-Fan Wu, Chih-Jen Lin, ruby C. Weng
Probability Estimates for Multi-class Classification by Pairwise Coupling
Neural Information Processing Symposium 2003
http://books.nips.cc/papers/files/nips16/NIPS2003_0538.pdf

See Also

`predict.ksvm`, `ksvm`

Examples

```
## create artificial pairwise probabilities
pairs <- matrix(c(0.82,0.12,0.76,0.1,0.9,0.05),2)

couple(pairs)

couple(pairs, coupler="pkpd")

couple(pairs, coupler ="vote")
```

`csi-class` *Class "csi"*

Description

The reduced Cholesky decomposition object

Objects from the Class

Objects can be created by calls of the form `new("csi", ...)`. or by calling the `csi` function.

Slots

`.Data`: Object of class `"matrix"` contains the decomposed matrix

`pivots`: Object of class `"vector"` contains the pivots performed

`diagresidues`: Object of class `"vector"` contains the diagonial residues

`maxresiduals`: Object of class `"vector"` contains the maximum residues

predgain Object of class `"vector"` contains the predicted gain before adding each column

truegain Object of class `"vector"` contains the actual gain after adding each column

Q Object of class `"matrix"` contains Q from the QR decomposition of the kernel matrix

R Object of class `"matrix"` contains R from the QR decomposition of the kernel matrix

Extends

Class `"matrix"`, directly.

Methods

diagresidues `signature(object = "csi")`: returns the diagonial residues

maxresiduals `signature(object = "csi")`: returns the maximum residues

pivots `signature(object = "csi")`: returns the pivots performed

predgain `signature(object = "csi")`: returns the predicted gain before adding each column

truegain `signature(object = "csi")`: returns the actual gain after adding each column

Q `signature(object = "csi")`: returns Q from the QR decomposition of the kernel matrix

R `signature(object = "csi")`: returns R from the QR decomposition of the kernel matrix

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

csi, inchol-class

Examples

```
data(iris)

## create multidimensional y matrix
yind <- t(matrix(1:3,3,150))
ymat <- matrix(0, 150, 3)
ymat[yind==as.integer(iris[,5])] <- 1

datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- csi(datamatrix,ymat, kernel=rbf, rank = 30)
dim(Z)
pivots(Z)
# calculate kernel matrix
K <- crossprod(t(Z))
# difference between approximated and real kernel matrix
(K - kernelMatrix(kernel=rbf, datamatrix))[6,]
```

---

csi *Cholesky decomposition with Side Information*

Description

The `csi` function in **in Package 'kernlab'** is an implementation of an incomplete Cholesky decomposition algorithm which exploits side information (e.g. classification labels, regression responses) to compute a low rank decomposition of a kernel matrix from the data.

Usage

```
## S4 method for signature 'matrix':
csi(x, y, kernel="rbfdot", kpar=list(sigma=0.1), rank,
centering = TRUE, kappa = 0.99 ,delta = 40 ,tol = 1e-5)
```

Arguments

`x`	The data matrix indexed by row
`y`	the classification labels or regression repsonses. In classification y is a $m \times n$ matrix where m the number of data and n the number of classes y and y_i is 1 if the corresponting x belongs to class i.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class `kernel`, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel • `stringdot` String kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. For valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`rank`	maximal rank
`centering`	if `TRUE` centering is performed (default: TRUE)

`kappa`	trade-off between approximation of K and prediction of Y (default: 0.99)
`delta`	number of columns of cholesky performed in advance (default: 40)
`tol`	minimum gain at each iteration (default: 1e-4)

Details

An incomplete cholesky decomposition calculates Z where $K = ZZ'$ K being the kernel matrix. Since the rank of a kernel matrix is usually low, Z tends to be smaller then the complete kernel matrix. The decomposed matrix can be used to create memory efficient kernel-based algorithms without the need to compute and store a complete kernel matrix in memory.
`csi` uses the class labels, or regression responses to compute a more apropriate aproximation for the problem at hand considering the aditional information from the response variable.

Value

An S4 object of class ”inchol” which is an extension of the class ”matrix”. The object is the decomposed kernel matrix along with the slots :

`pivots`	Indices on which pivots where done
`diagresidues`	Residuals left on the diagonal
`maxresiduals`	Residuals picked for pivoting
`predgain`	predicted gain before adding each column
`truegain`	actual gain after adding each column
`Q`	QR decomposition of the kernel matrix
`R`	QR decomposition of the kernel matrix

slots can be accessed either by `object@slot` or by accessor functions with the same name (e.g. `pivots(object)`)

Author(s)

Alexandros Karatzoglou (based on Matlab code by Francis Bach)
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Francis R. Bach, Michael I. Jordan
Predictive low-rank decomposition for kernel methods.
Proceedings of the Twenty-second International Conference on Machine Learning (ICML) 2005
http://cmm.ensmp.fr/~bach/bach_jordan_csi.pdf

See Also

inchol, chol

Examples

```
data(iris)

## create multidimensional y matrix
yind <- t(matrix(1:3,3,150))
ymat <- matrix(0, 150, 3)
ymat[yind==as.integer(iris[,5])] <- 1

datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- csi(datamatrix,ymat, kernel=rbf, rank = 30)
dim(Z)
pivots(Z)
# calculate kernel matrix
K <- crossprod(t(Z))
# difference between approximated and real kernel matrix
(K - kernelMatrix(kernel=rbf, datamatrix))[6,]
```

`dots` *Kernel Functions*

Description

The kernel generating functions provided in kernlab.
The Gaussian RBF kernel $k(x, x') = \exp(-\sigma \|x - x'\|^2)$
the Polynomial kernel $k(x, x') = (scale < x, x' > +offset)^{degree}$.
the Linear kernel $k(x, x') =< x, x' >$
the Hyperbolic tangent kernel $k(x, x') = \tanh(scale < x, x' > +offset)$ the Laplacian kernel $k(x, x') = \exp(-\sigma \|x - x'\|)$
the Bessel kernel $k(x, x') = (-Bessel_{(\nu+1)}^{n} \sigma \|x - x'\|^2)$
the ANOVA RBF kernel $k(x, x') = \sum_{1 \leq i_1 \ldots < i_D \leq N} \prod_{d=1}^{D} k(x_{id}, {x'}_{id})$ where k(x,x) is an Gaussian RBF kernel. the Spline kernel $\prod_{d=1}^{D} 1 + x_i x_j + x_i x_j min(x_i, x_j) - \frac{x_i + x_j}{2} min(x_i, x_j)^2 + \frac{min(x_i, x_j)^3}{3}$

Usage

```
rbfdot(sigma = 1)

polydot(degree = 1, scale = 1, offset = 1)

tanhdot(scale = 1, offset = 1)

vanilladot()

laplacedot(sigma = 1)

besseldot(sigma = 1, order = 1, degree = 1)

anovadot(sigma = 1, degree = 1)

splinedot()
```

Arguments

`sigma`	The inverse kernel width used by the Gaussian the Laplacian, the Bessel and the ANOVA kernel
`degree`	The degree of the polynomial, bessel or ANOVA kernel function. This has to be an positive integer.
`scale`	The scaling parameter of the polynomial and tangent kernel is a convenient way of normalizing patterns without the need to modify the data itself
`offset`	The offset used in a polynomial or hyperbolic tangent kernel
`order`	The order of the Bessel function to be used as a kernel

Details

The kernel generating function are used to initialize a kernel function which calculates the dot (inner) product between two feature vectors in a Hilbert Space. These functions can be passed as a `kernel` argument on almost all functions in kernlab (eg. `ksvm`, `kpca` etc).

Although using one of the existing kernel functions as a `kernel` argument in various functions in kernlab has the advantage that optimized code is used to calculate various kernel expressions, any other function implementing a dot product of class `kernel` can also be used as a kernel argument. This allows the user to use, test and develop special kernels for a given data set or algorithm.

Value

Return an S4 object of class `kernel` which extents the `function` class. The resulting function implements the given kernel calculating the inner (dot) product between two vectors.

`kpar`	a list containing the kernel parameters (hyperparameters) used.

The kernel parameters can be accessed by the `kpar` function.

Note

If the offset in the Polynomial kernel is set to 0, we obtain homogeneous polynomial kernels, for positive values, we have inhomogeneous kernels. Note that for negative values the kernel does not satisfy Mercer's condition and thus the optimizers may fail.

In the Hyperbolic tangent kernel if the offset is negative the likelihood of obtaining a kernel matrix that is not positive definite is much higher (since then even some diagonal elements may be negative), hence if this kernel has to be used, the offset should always be positive. Note, however, that this is no guarantee that the kernel will be positive.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`kernelMatrix` , `kernelMult`, `kernelPol`

Examples

```
rbfkernel <- rbfdot(sigma = 0.1)
rbfkernel

kpar(rbfkernel)

## create two vectors
x <- rnorm(10)
y <- rnorm(10)

## calculate dot product
rbfkernel(x,y)
```

`gausspr-class` *Class "gausspr"*

Description

The Gaussian Processes object

Objects from the Class

Objects can be created by calls of the form `new("gausspr", ...)`. or by calling the `gausspr` function

Slots

`tol`: Object of class `"numeric"` contains tolerance of termination criteria

`kernelf`: Object of class `"function"` contains the kernel function used

`kpar`: Object of class `"list"` contains the kernel parameter used

`kcall`: Object of class `"ANY"` contains the used function call

`type`: Object of class `"character"` contains type of problem

`terms`: Object of class `"ANY"` contains the terms representation of the symbolic model used (when using a formula)

`xmatrix`: Object of class `"matrix"` containing the data matrix used

`ymatrix`: Object of class `"ANY"` containing the response matrix

`fitted`: Object of class `"ANY"` containing the fitted values `lev`: Object of class `"vector"` containing the levels of the response (in case of classification)

`nclass`: Object of class `"numeric"` containing the number of classes (in case of classification)

`alpha`: Object of class `"ANY"` containing the computes alpha values

`alphaindex` Object of class `"list"` containing the indexes for the alphas in various classes (in multi-class problems).

`nvar`: Object of class `"numeric"` containing the computed variance

`error`: Object of class `"numeric"` containing the training error

`cross`: Object of class `"numeric"` containing the cross validation error

`n.action`: Object of class `"ANY"` containing the action performed in NA

Methods

alpha `signature(object = "gausspr")`: returns the alpha vector

cross `signature(object = "gausspr")`: returns the cross validation error

error `signature(object = "gausspr")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "gausspr")`: returns the call performed

kernelf `signature(object = "gausspr")`: returns the kernel function used

kpar `signature(object = "gausspr")`: returns the kernel parameter used

lev `signature(object = "gausspr")`: returns the response levels (in classification)

type `signature(object = "gausspr")`: returns the type of problem

xmatrix `signature(object = "gausspr")`: returns the data matrix used

ymatrix `signature(object = "gausspr")`: returns the response matrix used

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`gausspr`, `ksvm-class`

Examples

```
# train model
data(iris)
test <- gausspr(Species~.,data=iris,var=2)
test
alpha(test)
error(test)
lev(test)
```

`gausspr` *Gaussian processes for regression and classification*

Description

`gausspr` is an implementation of Gaussian processes for classification and regression.

Usage

```
## S4 method for signature 'formula':
gausspr(x, data=NULL, ..., subset, na.action = na.omit)

## S4 method for signature 'vector':
gausspr(x,...)

## S4 method for signature 'matrix':
gausspr(x, y, type="classification", kernel="rbfdot",
        kpar=list(sigma = 0.1),var=1, tol=0.001, cross=0,
        fit=TRUE, ... , subset, na.action = na.omit)
```

Arguments

`x`	a symbolic description of the model to be fit or a matrix or vector when a formula interface is not used. When not using a formula x is a matrix or vector containg the variables in the model

`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'gausspr' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`type`	Type of problem. Either "classification" or "regression"
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel. The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`var`	the initial noise variance
`tol`	tolerance of termination criterion (default: 0.001)

`fit`	indicates whether the fitted values should be computed and included in the model or not (default: 'TRUE')
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Mean Squared Error for regression
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

A Gaussian process is specified by a mean and a covariance function. The mean is a function of x (which is often the zero function), and the covariance is a function $C(x, x')$ which expresses the expected covariance between the value of the function y at the points x and x'. The actual function $y(x)$ in any data modelling problem is assumed to be a single sample from this Gaussian distribution. Model parameter estimation in classification is done by a gradient descent algorithm.

Value

An S4 object of class "gausspr" containing the fitted model along with information. Accessor functions can be used to access the slots of the object which include :

`alpha`	The resulting model parameters
`error`	Training error (if fit == TRUE)

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Christopher K.I. Williams, Carl Edward Rasmussen
Gaussian Processes for Regression

Advances in Neural Information Processing Systems, NIPS
http://books.nips.cc/papers/files/nips08/0514.pdf

See Also

rvm, ksvm

Examples

```
# train model
data(iris)
test <- gausspr(Species~.,data=iris,var=2)
test
alpha(test)

# predict on the training set
predict(test,iris[,-5])

# create regression data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.03)

# regression with gaussian processes
foo <- gausspr(x, y)
foo

# predict and plot
ytest <- predict(foo, x)
plot(x, y, type ="l")
lines(x, ytest, col="red")
```

---

inchol-class   *Class "inchol"*

Description

The reduced Cholesky decomposition object

Objects from the Class

Objects can be created by calls of the form `new("inchol", ...)`. or by calling the `inchol` function.

Slots

`.Data`: Object of class `"matrix"` contains the decomposed matrix

`pivots`: Object of class `"vector"` contains the pivots performed

`diagresidues`: Object of class `"vector"` contains the diagonial residues

`maxresiduals`: Object of class `"vector"` contains the maximum residues

Extends

Class `"matrix"`, directly.

Methods

diagresidues `signature(object = "inchol")`: returns the diagonial residues

maxresiduals `signature(object = "inchol")`: returns the maximum residues

pivots `signature(object = "inchol")`: returns the pivots performed

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`inchol`, `csi-class`

Examples

```
data(iris)
datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- inchol(datamatrix,kernel=rbf)
dim(Z)
pivots(Z)
diagresidues(Z)
maxresiduals(Z)
```

`inchol` *Incomplete Cholesky decomposition*

Description

`inchol` computes the incomplete Cholesky decomposition of the kernel matrix from a data matrix.

Usage

```
inchol(x, kernel="rbfdot", kpar=list(sigma=0.1), tol = 0.001,
          maxiter = dim(x)[1], blocksize = 50, verbose = 0)
```

Arguments

`x`	The data matrix indexed by row
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class `kernel`, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot"

- `scale, offset` for the Hyperbolic tangent kernel function ”tanhdot”
- `sigma, order, degree` for the Bessel kernel ”besseldot”.
- `sigma, degree` for the ANOVA kernel ”anovadot”.

Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.

`tol`	algorithm stops when remaining pivots bring less accuracy then `tol` (default: 0.001)
`maxiter`	maximum number of iterations and colums in Z
`blocksize`	add this many columns to matrix per iteration
`verbose`	print info on algorithm convergence

Details

An incomplete cholesky decomposition calculates Z where $K = ZZ'$ K being the kernel matrix. Since the rank of a kernel matrix is usually low, Z tends to be smaller then the complete kernel matrix. The decomposed matrix can be used to create memory efficient kernel-based algorithms without the need to compute and store a complete kernel matrix in memory.

Value

An S4 object of class ”inchol” which is an extension of the class ”matrix”. The object is the decomposed kernel matrix along with the slots :

`pivots`	Indices on which pivots where done
`diagresidues`	Residuals left on the diagonal
`maxresiduals`	Residuals picked for pivoting

slots can be accessed either by `object@slot` or by accessor functions with the same name (e.g. `pivots(object))`

Author(s)

Alexandros Karatzoglou (based on Matlab code by S.V.N. (Vishy) Vishwanathan and Alex Smola)
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Francis R. Bach, Michael I. Jordan
Kernel Independent Component Analysis
Journal of Machine Learning Research 3, 1-48
http://www.jmlr.org/papers/volume3/bach02a/bach02a.pdf

See Also

chol

Examples

```
data(iris)
datamatrix <- as.matrix(iris[,-5])
# initialize kernel function
rbf <- rbfdot(sigma=0.1)
rbf
Z <- inchol(datamatrix,kernel=rbf)
dim(Z)
pivots(Z)
# calculate kernel matrix
K <- crossprod(t(Z))
# difference between approximated and real kernel matrix
(K - kernelMatrix(kernel=rbf, datamatrix))[6,]
```

`income` *Income Data*

Description

Customer Income Data from a marketing survey.

Usage

```
data(income)
```

Format

A data frame with 14 categorical variables (8993 observations).

Explanation of the variable names:

1	`INCOME`	annual income of household (Personal income if single)	ordinal
2	`SEX`	sex	nominal
3	`MARITAL.STATUS`	marital status	nominal
4	`AGE`	age	ordinal
5	`EDUCATION`	educational grade	ordinal
6	`OCCUPATION`	type of work	nominal
7	`AREA`	how long the interviewed person has lived in the San Francisco/Oakland/San Jose area	ordinal
8	`DUAL.INCOMES`	dual incomes (if married)	nominal
9	`HOUSEHOLD.SIZE`	persons living in the household	ordinal
10	`UNDER18`	persons in household under 18	ordinal
11	`HOUSEHOLDER`	householder status	nominal
12	`HOME.TYPE`	type of home	nominal
13	`ETHNIC.CLASS`	ethnic classification	nominal
14	`LANGUAGE`	language most often spoken at home	nominal

Details

A total of N=9409 questionnaires containg 502 questions were filled out by shopping mall customers in the San Francisco Bay area. The dataset is an extract from this survey. It consists of 14 demographic attributes. The dataset is a mixture of nominal and ordinal variables with a lot of missing data. The goal is to predict the Anual Income of Household from the other 13 demographics attributes.

Source

Impact Resources, Inc., Columbus, OH (1987).

`inlearn` *Onlearn object initialization*

Description

Online Kernel Algorithm object `onlearn` initialization function.

Usage

```
## S4 method for signature 'numeric':
inlearn(d, kernel = "rbfdot", kpar = list(sigma = 0.1),
        type = "novelty", buffersize = 1000)
```

Arguments

`d`	the dimensionality of the data to be learned
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function. The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the `kpar` parameter as well.
`type`	the type of problem to be learned by the online algorithm : `classification`, `regression`, `novelty`
`buffersize`	the size of the buffer to be used

Details

The `inlearn` is used to initialize a blank `onlearn` object.

Value

The function returns an `S4` object of class `onlearn` that can be used by the `onlearn` function.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`onlearn`, `onlearn-class`

Examples

```
## create toy data set
x <- rbind(matrix(rnorm(100),,2),matrix(rnorm(100)+3,,2))
y <- matrix(c(rep(1,50),rep(-1,50)),,1)

## initialize onlearn object
on <- inlearn(2,kernel="rbfdot",kpar=list(sigma=0.2),type="classification")

## learn one data point at the time
for(i in sample(1:100,100))
on <- onlearn(on,x[i,],y[i],nu=0.03,lambda=0.1)

sign(predict(on,x))
```

`ipop-class` *Class "ipop"*

Description

The quadratic problem solver class

Objects from the Class

Objects can be created by calls of the form `new("ipop", ...)`. or by calling the `ipop` function.

Slots

`primal`: Object of class `"vector"` the primal solution of the problem

`dual`: Object of class `"numeric"` the dual of the problem

`how`: Object of class `"character"` convergence information

Methods

`primal` Return the primal of the problem

`dual` Return the dual of the problem

`how` Return information on convergence

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`ipop`

Examples

```
## solve the Support Vector Machine optimization problem
data(spam)

## sample a scaled part (300 points) of the spam data set
m <- 300
set <- sample(1:dim(spam)[1],m)
x <- scale(as.matrix(spam[,-58]))[set,]
y <- as.integer(spam[set,58])
y[y==2] <- -1

##set C parameter and kernel
C <- 5
rbf <- rbfdot(sigma = 0.1)
```

```
## create H matrix etc.
H <- kernelPol(rbf,x,,y)
c <- matrix(rep(-1,m))
A <- t(y)
b <- 0
l <- matrix(rep(0,m))
u <- matrix(rep(C,m))
r <- 0

sv <- ipop(c,H,A,b,l,u,r)
primal(sv)
dual(sv)
how(sv)
```

`ipop` *Quadratic Programming Solver*

Description

ipop solves the quadratic programming problem :
$\min(c' * x + 1/2 * x' * H * x)$
subject to:
$b <= A * x <= b + r$
$l <= x <= u$

Usage

```
ipop(c, H, A, b, l, u, r, sigf = 7, maxiter = 40, margin = 0.05,
     bound = 10, verb = 0)
```

Arguments

`c`	Vector or one column matrix appearing in the quadratic function
`H`	Matrix appearing in the quadratic function
`A`	Matrix defining the constrains under which we minimize the quadratic function
`b`	Vector or one column matrix defining the constrains
`l`	Lower bound vector or one column matrix

`u`	Upper bound vector or one column matrix
`r`	Vector or one column matrix defining constrains
`sigf`	Precision (default: 7 significant figures)
`maxiter`	Maximum number of iterations
`margin`	how close we get to the constrains
`bound`	Clipping bound for the variables
`verb`	Display convergence information during runtime

Details

ipop uses an interior point method to solve the quadratic programming problem.

Value

An S4 object with the following slots

`primal`	Vector containing the primal solution of the quadratic problem
`dual`	The dual solution of the problem
`how`	Character string describing the type of convergence

all slots can be accessed through accessor functions (see example)

Author(s)

Alexandros Karatzoglou (based on Matlab code by Alex Smola)
alexandros.karatzoglou@ci.tuwien.ac.at

References

R. J. Vanderbei
LOQO: An interior point code for quadratic programming
Optimization Methods and Software 11, 451-484, 1999
`http://www.sor.princeton.edu/~rvdb/ps/loqo3.ps.gz`

See Also

`solve.QP`

Examples

```
## solve the Support Vector Machine optimization problem
data(spam)

## sample a scaled part (500 points) of the spam data set
m <- 500
set <- sample(1:dim(spam)[1],m)
x <- scale(as.matrix(spam[,-58]))[set,]
y <- as.integer(spam[set,58])
y[y==2] <- -1

##set C parameter and kernel
C <- 5
rbf <- rbfdot(sigma = 0.1)

## create H matrix etc.
H <- kernelPol(rbf,x,,y)
c <- matrix(rep(-1,m))
A <- t(y)
b <- 0
l <- matrix(rep(0,m))
u <- matrix(rep(C,m))
r <- 0

sv <- ipop(c,H,A,b,l,u,r)
sv
dual(sv)
```

`kcca-class` *Class "kcca"*

Description

The "kcca" class

Objects from the Class

Objects can be created by calls of the form `new("kcca", ...)`. or by the calling the `kcca` function.

Slots

`kcor`: Object of class `"vector"` describing the correlations

`xcoef`: Object of class `"matrix"` estimated coefficients for the `x` variables

`ycoef`: Object of class `"matrix"` estimated coefficients for the `y` variables

`xvar`: Object of class `"matrix"` holds the canonical variates for `x`

`yvar`: Object of class `"matrix"` holds the canonical variates for `y`

Methods

kcor `signature(object = "kcca")`: returns the correlations

xcoef `signature(object = "kcca")`: returns the estimated coefficients for the `x` variables

ycoef `signature(object = "kcca")`: returns the estimated coefficients for the `y` variables

xvar `signature(object = "kcca")`: returns the canonical variates for `x`

yvar `signature(object = "kcca")`: returns the canonical variates for `y`

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`kcca`, `kpca-class`

Examples

`kcca` *Kernel Canonical Correlation Analysis*

Description

Computes the canonical correlation analysis in a feature space.

Usage

```
## S4 method for signature 'matrix':
kcca(x, y, kernel="rbfdot", kpar=list(sigma=0.1), ...)
```

Arguments

`x`	a matrix containing data index by row
`y`	a matrix containing data index by row
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`...`	adittional parameters for the `kpca` function

Details

The kernel version of canonical correlation analysis.

Value

An S4 object containg the following slots:

`kcor`	Correlation coefficients in feature space
`xcoef`	estimated coefficients for the `x` variables in the feature space
`ycoef`	estimated coefficients for the `y` variables in the feature space
`xvar`	The canonical variates for `x`
`yvar`	The canonical variates for `y`

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Malte Kuss, Thore Graepel
The Geometry Of Kernel Canonical Correlation Analysis
`http://www.kyb.tuebingen.mpg.de/publications/pdfs/pdf2233.pdf`

See Also

`cancor` `kpca`

Examples

`kernel-class` *Class "kernel" "rbfkernel" "polykernel", "tanhkernel", "vanillakernel"*

Description

The build in `kernel` classes in `kernelab`

Objects from the Class

Objects can be created by calls of the form `new("rbfkernel", ...)`, `new{"polykernel"}`, `new{"tanhkernel"}`, `new{"vanillakernel"}` or by calling the `rbfdot`, `polydot`, `tanhdot`, `vanilladot` functions.

Slots

`.Data`: Object of class `"function"` containing the kernel function

`kpar`: Object of class `"list"` containing the kernel parameters

Extends

Class `"kernel"`, directly. Class `"function"`, by class `"kernel"`.

Methods

kernelMatrix `signature(kernel = "rbfkernel", x = "matrix")`: computes the kernel matrix

kernelMult `signature(kernel = "rbfkernel", x = "matrix")`: computes the quadratic kernel expression

kernelPol `signature(kernel = "rbfkernel", x = "matrix")`: computes the kernel expansion

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`dots`

Examples

```
rbfkernel <- rbfdot(sigma = 0.1)
rbfkernel
is(rbfkernel)
kpar(rbfkernel)
```

`kernelMatrix`	*Kernel Matrix functions*

Description

`kernelMatrix` calculates the kernel matrix $K_{ij} = k(x_i, x_j)$ or $K_{ij} = k(x_i, y_j)$.
`kernelPol` computes the quadratic kernel expression $H = z_i z_j k(x_i, x_j)$, $H = z_i k_j k(x_i, y_j)$.
`kernelMult` calculates the kernel expansion $f(x_i) = \sum_{i=1}^{m} zk(x_i, x_j)$
`kernelFast` computes the kernel matrix this function identical to `kernelMatrix`, except that it also requires the squared norm of the first argument as additional input.

Usage

```
## S4 method for signature 'kernel':
kernelMatrix(kernel, x, y = NULL)

## S4 method for signature 'kernel':
kernelPol(kernel, x, y = NULL, z, k = NULL)

## S4 method for signature 'kernel':
kernelMult(kernel, x, y = NULL, z, blocksize = 256)

## S4 method for signature 'kernel':
kernelFast(kernel, x, y, a)
```

Arguments

`kernel`	the kernel function to be used to calculate the kernel matrix. This has to be a function of class `kernel`, i.e. either one of the build in kernel functions or a fnction taking two vector arguments and returning a scalar.
`x`	a data matrix to be used to calculate the kernel matrix
`y`	second data matrix to calculate the kernel matrix
`z`	a suitable vector or matrix
`k`	a suitable vector or matrix
`a`	the squared norm of `x` e.g. `rowSums(x^2)`

`blocksize`	the kernel expansion computations are done block wise to avoid storing the kernel matrix into memory. `blocksize` defines the size of the computational blocks.

Details

Common functions used during kernel based computations.
The `kernel` parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be initialized by using the following functions:

- `rbfdot` Radial Basis kernel function
- `polydot` Polynomial kernel function
- `vanilladot` Linear kernel function
- `tanhdot` Hyperbolic tangent kernel function
- `laplacedot` Laplacian kernel function
- `besseldot` Bessel kernel function
- `anovadot` ANOVA RBF kernel function
- `splinedot` the Spline kernel

(see example.)

`kernelFast` is mainly used in situations where colums of the kernel matrix are computed per invocation. In these cases, evaluating the norm of each row-entry over and over again would cause significant computational overhead.

Value

`kernelMatrix` returns a symmetric diagonal semi-definite matrix.
`kernelPol` returns a matrix.
`kernelMult` usually returns a one-column matrix.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`rbfdot`, `polydot`, `tanhdot`, `vanilladot`

Examples

```
## use the spam data
data(spam)
dt <- as.matrix(spam[c(10:20,3000:3010),-58])

## initialize kernel function
rbf <- rbfdot(sigma = 0.05)
rbf

## calculate kernel matrix
kernelMatrix(rbf, dt)

yt <- as.matrix(as.integer(spam[c(10:20,3000:3010),58]))
yt[yt==2] <- -1

## calculate the quadratic kernel expression
kernelPol(rbf, dt, ,yt)

## calculate the kernel expansion
kernelMult(rbf, dt, ,yt)
```

`kfa-class` *Class "kfa"*

Description

The class of the object returned by the Kernel Feature Analysis `kfa` function

Objects from the Class

Objects can be created by calls of the form `new("kfa", ...)` or by calling the `kfa` method. The objects contain the features along with the alpha values.

Slots

`alpha`: Object of class `"matrix"` containing the alpha values

`alphaindex`: Object of class `"vector"` containing the indexes of the selected feature

`kernelf`: Object of class `"function"` containing the kernel function used

`xmatrix`: Object of class `"matrix"` containing the selected features

`kcall`: Object of class `"ANY"` containig the `kfa` function call

`terms`: Object of class `"ANY"` containing the formula terms

Methods

alpha `signature(object = "kfa")`: returns the alpha values

alphaindex `signature(object = "kfa")`: returns the index of the selected features

kcall `signature(object = "kfa")`: returns the function call

kernelf `signature(object = "kfa")`: returns the kernel function used

predict `signature(object = "kfa")`: used to embed more data points to the feature base

xmatrix `signature(object = "kfa")`: returns the selected features.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`kfa`, `kpca-class`

Examples

```
data(promotergene)
f <- kfa(~.,data=promotergene)
```

`kfa` *Kernel Feature Analysis*

Description

The Kernel Feature Analysis algorithm is an algorithm for extracting structure from possibly high-dimensional data sets. Similar to `kpca` a new basis for the data is found. The data can then be projected on the new basis.

Usage

```
## S4 method for signature 'formula':
kfa(x, data = NULL, na.action = na.omit, ...)

## S4 method for signature 'matrix':
kfa(x, kernel = "rbfdot", kpar = list(sigma = 0.1),
   features = 0, subset = 59, normalize = TRUE, na.action = na.omit)
```

Arguments

`x`	The data matrix indexed by row or a formula describing the model. Note, that an intercept is always included, whether given in the formula or not.
`data`	an optional data frame containing the variables in the model (when using a formula).
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`features`	Number of features (principal components) to return. (default: 0 , all)
`subset`	the number of features sampled (used) from the data set

`normalize`	normalize the feature selected (default: TRUE)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

Kernel Feature analysis is similar to Kernel PCA, but instead of extracting eigenvectors of the training dataset in feature space, it approximates the eigenvectors by selecting training patterns which are good basis vectors for the training set. It works by choosing a fixed size subset of the data set and scaling it to unit length (under the kernel). It then chooses the features that maximize the value of the inner product (kernel function) with the rest of the patterns.

Value

`kfa` returns an object of class `kfa` containing the features selected by the algorithm.

`xmatrix`	contains the features selected
`alpha`	contains the sparse alpha vector

The `predict` function can be used to embed new data points into to the selected feature base.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Alex J. Smola, Olvi L. Mangasarian and Bernhard Schoelkopf
Sparse Kernel Feature Analysis
Data Mining Institute Technical Report 99-04, October 1999
`ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-04.ps`

See Also

`kpca`, `kfa-class`

Examples

```
data(promotergene)
f <- kfa(~.,data=promotergene,features=2,kernel="rbfdot",kpar=list(sigma=0.01))
plot(predict(f,promotergene),col=as.numeric(promotergene[,1]))
```

`kha-class` *Class "kha"*

Description

The Kernel Hebbian Algorithm class

Objects objects of class "kha"

Objects can be created by calls of the form `new("kha", ...)`. or by calling the `kha` function.

Slots

`pcv`: Object of class `"matrix"` containing the principal component vectors

`eig`: Object of class `"vector"` containing the coresponding normalization values

`eskm`: Object of class `"vector"` containing the kernel sum

`kernelf`: Object of class `"function"` containing the kernel function used

`kpar`: Object of class `"list"` containing the kernel parameters used

`xmatrix`: Object of class `"matrix"` conatining the data matrix used

`kcall`: Object of class `"ANY"` containing the function call

`n.action`: Object of class `"ANY"` containg the action performed on NA

Methods

eig `signature(object = "kha")`: returns the normalization values

kcall `signature(object = "kha")`: returns the performed call

kernelf `signature(object = "kha")`: returns the used kernel function

pcv `signature(object = "kha")`: returns the principal component vectors

eskm `signature(object = "kha")`: returns the kernel sum

predict `signature(object = "kha")`: embeeds new data

xmatrix `signature(object = "kha")`: returns the used data matrix

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

ksvm-class, kcca-class

Examples

```
# another example using the iris
data(iris)
test <- sample(1:50,20)

kpc <- kha(~.,data=iris[-test,-5],kernel="rbfdot",kpar=list(sigma=0.2),
           features=2)

#print the principal component vectors
pcv(kpc)
kernelf(kpc)
eig(kpc)
```

---

kha *Kernel Principal Components Analysis*

Description

Kernel Hebbian Algorithm is a nonlinear iterative algorithm for principal component analysis.

Usage

```
## S4 method for signature 'formula':
kha(x, data = NULL, na.action, ...)

## S4 method for signature 'matrix':
kha(x, kernel = "rbfdot", kpar = list(sigma = 0.1), features = 5,
         eta = 0.005, th = 1e-4, maxiter = 10000, verbose = FALSE,
        na.action = na.omit...)
```

Arguments

`x`	The data matrix indexed by row or a formula descibing the model. Note, that an intercept is always included, whether given in the formula or not.
`data`	an optional data frame containing the variables in the model (when using a formula).
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`features`	Number of features (principal components) to return. (default: 5)
`eta`	The hebbian learning rate (default : 0.005)

`th`	the smallest value of the convergence step (default : 0.0001)
`maxiter`	the maximum number of iterations.
`verbose`	print convergence every 100 iterations. (default : FALSE)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

The original form of KPCA can only be used on small data sets since it requieres the estimation of the eigenvectors of a full kernel matrix. The Kernel Hebbian Algorithm iteratively estimates the Kernel Principal Components with only linear order memory complexity. (see ref. for more details)

Value

An S4 object containing the principal component vectors along with the corresponding normalization values.

`pcv`	a matrix containing the principal component vectors (column wise)
`eig`	The normalization values
`xmatrix`	The original data matrix

all the slots of the object can be accessed by accessor functions.

Note

The predict function can be used to embed new data on the new space

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Kwang In Kim, M.O. Franz and B. Schölkopf
Kernel Hebbian Algorithm for Iterative Kernel Principal Component Analysis
Max-Planck-Institut für biologische Kybernetik, Tübingen (109)
http://www.kyb.tuebingen.mpg.de/publications/pdfs/pdf2302.pdf

See Also

kpca, kfa, kcca, pca

Examples

```
# another example using the iris
data(iris)
test <- sample(1:150,20)

kpc <- kha(~.,data=iris[-test,-5],kernel="rbfdot",kpar=list(sigma=0.2),
           features=2)

#print the principal component vectors
pcv(kpc)

#plot the data projection on the components
plot(predict(kpc,iris[,-5]),col=as.integer(iris[,5]),
     xlab="1st Principal Component",ylab="2nd Principal Component")
```

---

kkmeans *Kernel k-means*

Description

A weigthed kernel version of the famous k-means algorithm.

Usage

```
## S4 method for signature 'formula':
kkmeans(x, data = NULL, na.action = na.omit, ...)

## S4 method for signature 'matrix':
```

```
kkmeans(x, centers, kernel = "rbfdot", kpar = list(sigma = 0.1),
        alg="kkmeans", p=1, na.action = na.omit, ...)

## S4 method for signature 'kernelMatrix':
kkmeans(x, centers, ...)

## S4 method for signature 'list':
kkmeans(x, centers, kernel = "stringdot",
        kpar = list(length=4, lambda=0.5),
        alg ="kkmeans", p = 1, na.action = na.omit, ...)
```

Arguments

`x`	the matrix of data to be clustered, a symbolic description of the model to be fit, a kernel Matrix of class `kernelMatrix`, or a list of character vectors.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'kkmeans' is called from.
`centers`	Either the number of clusters or a set of initial cluster centers. If the first, a random set of rows in the eigenvectors matrix are chosen as the initial centers.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel "Gaussian" • `polydot` Polynomial kernel • `vanilladot` Linear kernel • `tanhdot` Hyperbolic tangent kernel • `laplacedot` Laplacian kernel • `besseldot` Bessel kernel • `anovadot` ANOVA RBF kernel • `splinedot` Spline kernel • `stringdot` String kernel Setting the kernel parameter to "matrix" treats `x` as a kernel matrix calling the `kernelMatrix` interface. The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.

`kpar`	a character string or the list of hyper-parameters (kernel parameters). The default character string `"automatic"` uses a heuristic the determine a suitable value for the width parameter of the RBF kernel. A list can also be used containing the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`alg`	the algorithm to use. Options currently include `kkmeans` and `kerninghan`.
`p`	a parameter used to keep the affinity matrix positive semidefinite
`na.action`	The action to perform on NA
`...`	additional parameters

Details

The algorithm is implemented using the triangle inequality to avoid unnecessary and computational expensive distance calculations. This leads to significant speedup particularly on large data sets with a high number of clusters. With a particular choice of weights this algorithm becomes equivalent to Kernighan-Lin, and the norm-cut graph partitioning algorithms.
The function also support input in the form of a kernel matrix or a list of characters for text clustering.

Value

An S4 object of class `specc` wich extends the class `vector` containing integers indicating the cluster to which each point is allocated. The following slots contain useful information

`centers`	A matrix of cluster centers.
`size`	The number of point in each cluster
`withinss`	The within-cluster sum of squares for each cluster
`kernelf`	The kernel function used

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Inderjit Dhillon, Yuqiang Guan, Brian Kulis
A Unified view of Kernel k-means, Spectral Clustering and Graph Partitioning
UTCS Technical Report
`http://www.cs.utexas.edu/users/kulis/pubs/spectral_techreport.pdf`

See Also

`specc`, `kpca`, `kcca`

Examples

```
## Cluster the iris data set.
data(iris)

sc <- kkmeans(as.matrix(iris[,-5]), centers=3)

sc
centers(sc)
size(sc)
withinss(sc)
```

`kpca-class` *Class "kpca"*

Description

The Kernel Principal Components Analysis class

Objects of class "kpca"

Objects can be created by calls of the form `new("kpca", ...)`. or by calling the `kpca` function.

Slots

`pcv`: Object of class `"matrix"` containing the principal component vectors

`eig`: Object of class `"vector"` containing the coresponding eigenvalues

`rotated`: Object of class `"matrix"` containing the projection of the data on the principal components

`kernelf`: Object of class `"function"` containing the kernel function used

`kpar`: Object of class `"list"` containing the kernel parameters used

`xmatrix`: Object of class `"matrix"` conatining the data matrix used

`kcall`: Object of class `"ANY"` containing the function call

`n.action`: Object of class `"ANY"` containg the action performed on NA

Methods

eig `signature(object = "kpca")`: returns the eigenvalues

kcall `signature(object = "kpca")`: returns the performed call

kernelf `signature(object = "kpca")`: returns the used kernel function

pcv `signature(object = "kpca")`: returns the principal component vectors

predict `signature(object = "kpca")`: embeeds new data

rotated `signature(object = "kpca")`: returns the projected data

xmatrix `signature(object = "kpca")`: returns the used data matrix

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`ksvm-class`, `kcca-class`

Examples

```
# another example using the iris
data(iris)
test <- sample(1:50,20)

kpc <- kpca(~.,data=iris[-test,-5],kernel="rbfdot",kpar=list(sigma=0.2),
            features=2)

#print the principal component vectors
pcv(kpc)
rotated(kpc)
kernelf(kpc)
eig(kpc)
```

`kpca` *Kernel Principal Components Analysis*

Description

Kernel Principal Components Analysis is a nonlinear form of principal component analysis.

Usage

```
## S4 method for signature 'formula':
kpca(x, data = NULL, na.action, ...)

## S4 method for signature 'matrix':
kpca(x, kernel = "rbfdot", kpar = list(sigma = 0.1), features = 0,
    th = 1e-4, ...)
```

Arguments

`x`	The data matrix indexed by row or a formula descibing the model. Note, that an intercept is always included, whether given in the formula or not.
`data`	an optional data frame containing the variables in the model (when using a formula).
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`features`	Number of features (principal components) to return. (default: 0 , all)
`th`	the value of the eigenvalue under which principal components are ignored (only valid when features = 0). (default : 0.0001)

`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

Using kernel functions one can efficiently compute principal components in high-dimensional feature spaces, related to input space by some non-linear map.

Value

An S4 object containing the principal component vectors along with the corresponding eigenvalues.

`pcv`	a matrix containing the principal component vectors (column wise)
`eig`	The corresponding eigenvalues
`rotated`	The original data projected (rotated) on the principal components
`xmatrix`	The original data matrix

all the slots of the object can be accessed by accessor functions.

Note

The predict function can be used to embed new data on the new space

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Schoelkopf B., A. Smola, K.-R. Mueller :
Nonlinear component analysis as a kernel eigenvalue problem
Neural Computation 10, 1299-1319
`http://mlg.anu.edu.au/~smola/papers/SchSmoMul98.pdf`

See Also

kcca, pca

Examples

```
# another example using the iris
data(iris)
test <- sample(1:150,20)

kpc <- kpca(~.,data=iris[-test,-5],kernel="rbfdot",kpar=list(sigma=0.2),
             features=2)

#print the principal component vectors
pcv(kpc)

#plot the data projection on the components
plot(rotated(kpc),col=as.integer(iris[-test,5]),xlab="1st Principal Component",
       ylab="2nd Principal Component")

#embed remaining points
emb <- predict(kpc,iris[test,-5])
points(emb,col=iris[test,5])
```

`ksvm-class` *Class "ksvm"*

Description

An S4 class containing the output (model) of the `ksvm` Support Vector Machines function

Objects from the Class

Objects can be created by calls of the form `new("ksvm", ...)` or by calls to the `ksvm` function.

Slots

`type`: Object of class `"character"` containing the support vector machine type ("C-svc", "nu-svc", "C-bsvc", "spoc-svc", "one-svc", "eps-svr", "nu-svr", "eps-bsvr") `param`: Object of class `"list"` containing the Support Vector Machine parameters (C, nu, epsilon)

`kernelf`: Object of class `"function"` containing the kernel function

`kpar`: Object of class `"list"` containing the kernel function parameters (hyperparameters)

`kcall`: Object of class `"ANY"` containing the `ksvm` function call

`scaling`: Object of class `"ANY"` containing the scaling information performed on the data

`terms`: Object of class `"ANY"` containing the terms representation of the symbolic model used (when using a formula)

`xmatrix`: Object of class `"matrix"` the data matrix used during computations (possibly scaled and whithout NA)

`ymatrix`: Object of class `"ANY"` the response matrix/vector

`fitted`: Object of class `"ANY"` with the fitted values, predictions using the training set.

`lev`: Object of class `"vector"` with the levels of the response (in the case of classifiaction)

`prob.model`: Object of class `"list"` with the class prob. model

`prior`: Object of class `"list"` with the prior of the training set

`nclass`: Object of class `"numeric"` containing the number of classes (in the case of classification)

`alpha`: Object of class `"ANY"` containing the resulting alpha vector (list or matrix in case of multiclass classification) (support vectors)

`coef`: Object of class `"ANY"` containing the resulting coefficients

`alphaindex`: Object of class `"list"` containing

`b`: Object of class `"numeric"` containing the resulting offset

`SVindex`: Object of class `"vector"` containing the indexes of the support vectors

`nSV`: Object of class `"numeric"` containing the number of suppport vector machines

`error`: Object of class `"numeric"` containing the training error

`cross`: Object of class `"numeric"` containing the cross-validation error

`n.action`: Object of class `"ANY"` containing the action performed for NA

Methods

SVindex `signature(object = "ksvm")`: return the indexes of support vectors

alpha `signature(object = "ksvm")`: returns the complete alpha vector (wit zero values)

alphaindex `signature(object = "ksvm")`: returns the indexes of non-zero alphas (support vectors)

cross `signature(object = "ksvm")`: returns the cross-validation error

error `signature(object = "ksvm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values (predict on training set)

kernelf `signature(object = "ksvm")`: returns the kernel function

kpar `signature(object = "ksvm")`: returns the kernel parameters (hyperparameters)

lev `signature(object = "ksvm")`: returns the levels in case of classification

prob.model `signature(object="ksvm")`: returns class prob. model values

prior `signature(object="ksvm")`: returns the prior of the training set

kcall `signature(object="ksvm")`: returns the `ksvm` function call

scaling `signature(object = "ksvm")`: returns the scaling values

show `signature(object = "ksvm")`: prints the object information

type `signature(object = "ksvm")`: returns the problem type

xmatrix `signature(object = "ksvm")`: returns the data matrix used

ymatrix `signature(object = "ksvm")`: returns the response vector

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzolgou@ci.tuwien.ac.at⟩

See Also

`ksvm`, `rvm-class`, `gausspr-class`

Examples

```
## simple example using the promotergene data set
data(promotergene)

## train a support vector machine
gene <- ksvm(Class~.,data=promotergene,kernel="rbfdot",kpar=list(sigma=0.015),
             C=50,cross=4)
gene

# the kernel  function
kernelf(gene)
# the alpha values
alpha(gene)
# the coefficients
coef(gene)
# the fitted values
fitted(gene)
# the cross validation error
cross(gene)
```

`ksvm` *Support Vector Machines*

Description

Support Vector Machines are an excellent tool for classification novelty detection as well as regression. `ksvm` supports the well known C-svc, nu-svc, (classification) one-class-svc (novelty) eps-svr, nu-svr (regression) formulations along with the Crammer-Singer for multi-class classification formulation spoc-svc and bound-constraint SVM C-bsvc, eps-bsvr.
The implementation also supports class-probabilities output and confidence intervals for regression.

Usage

```
## S4 method for signature 'formula':
ksvm(x, data = NULL, ..., subset, na.action = na.omit, scaled = TRUE)

## S4 method for signature 'vector':
ksvm(x, ...)
```

```
## S4 method for signature 'matrix':
ksvm(x, y = NULL, scaled = TRUE, type = NULL, kernel ="rbfdot",
     kpar = list(sigma = 0.1), C = 1, nu = 0.2, epsilon = 0.1,
     prob.model = FALSE, class.weights = NULL, cachesize = 40,
     tol = 0.001, shrinking = TRUE, cross = 0, fit = TRUE, ...,
     subset, na.action = na.omit)
```

Arguments

`x`	a symbolic description of the model to be fit. Note, that the intercept is always excluded, whether given in the formula or not. When not using a formula x is a matrix or vector containg the variables in the model
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'ksvm' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.
`type`	`ksvm` can be used for classification , for regression, or for novelty detection. Depending on whether `y` is a factor or not, the default setting for `type` is `C-svc` or `eps-svr`, respectively, but can be overwritten by setting an explicit value. Valid options are: • `C-svc` (classification) • `nu-svc` (classification) • `C-bsvc` bound-constraint svm (classification) • `spoc-svc` (Crammer Singer multi-class) • `one-svc` (novelty detection) • `eps-svr` (regression) • `nu-svr` (regression) • `eps-svr` bound-constraint svm (regression)
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments.

	kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel "Gaussian" • `polydot` Polynomial kernel • `vanilladot` Linear kernel • `tanhdot` Hyperbolic tangent kernel • `laplacedot` Laplacian kernel • `besseldot` Bessel kernel • `anovadot` ANOVA RBF kernel • `splinedot` Spline kernel • `stringdot` String kernel Setting the kernel parameter to "matrix" treats `x` as a kernel matrix calling the `kernelMatrix` interface. The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the kpar parameter as well. In the case of a Radial Basis kernel function (Gaussian) kpar can also be set to the string "automatic" which uses the heuristics in `sigest` to calculate a good `sigma` value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").
`C`	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.

`nu`	parameter needed for `nu-svc`, `one-svc`, and `nu-svr`. The `nu` parameter sets the upper bound on the training error and the lower bound on the fraction of data points to become Support Vectors (default: 0.2).
`epsilon`	epsilon in the insensitive-loss function used for `eps-svr`, `nu-svr` and `eps-bsvm` (default: 0.1)
`prob.model`	if set to `TRUE` builds a model for calculating class probabilities or in case of regression, calculates the scaling parameter of the Laplacian distribution fitted on the residuals. Fitting is done on output data created by performing a 3-fold cross-validation on the training data. For details see references. (default: `FALSE`)
`class.weights`	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.
`cachesize`	cache memory in MB (default 40)
`tol`	tolerance of termination criterion (default: 0.001)
`shrinking`	option whether to use the shrinking-heuristics (default: `TRUE`)
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression
`fit`	indicates whether the fitted values should be computed and included in the model or not (default: `TRUE`)
`...`	additional parameters for the low level fitting function
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

Details

For multiclass-classification with k levels, k>2, `ksvm` uses the 'one-against-one'-approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme.
If the predictor variables include factors, the formula interface must be used

to get a correct model matrix. In classification when `prob.model` is `TRUE` a 3-fold cross validation is performed on the data and a sigmoid function is fitted on the resulting decision values f. The `plot` function for binary classification `ksvm` objects displays a contour plot of the decision values with the corresponding support vectors highlighted. The predict function can return probabilistic output (probability matrix) in the case of classification by setting the `type` parameter to "probabilities".

Value

An S4 object of class `"ksvm"` containing the fitted model, Accessor functions can be used to access the slots of the object (see examples) which include:

`alpha`	The resulting support vectors, (alpha vector) (possibly scaled).
`alphaindex`	The index of the resulting support vectors in the data matrix. Note that this index refers to the pre-processed data (after the possible effect of `na.omit` and `subset`)
`coef`	The corresponding coefficients times the training labels.
`b`	The negative intercept.
`nSV`	The number of Support Vectors
`error`	Training error
`cross`	Cross validation error, (when cross > 0)
`prob.model`	Contains the width of the Laplacian fitted on the residuals in case of regression, or the parameters of the sigmoid fitted on the decision values in case of classification.

Note

Data is scaled internally, usually yielding better results.

Author(s)

Alexandros Karatzoglou (SMO optimizers in C/C++ by Chih-Chung Chang & Chih-Jen Lin)
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

- Chang, Chih-Chung and Lin, Chih-Jen:
  LIBSVM: a library for Support Vector Machines
  `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

- Exact formulations of models, algorithms, etc. can be found in the document:
  Chang, Chih-Chung and Lin, Chih-Jen:
  LIBSVM: a library for Support Vector Machines
  http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz
- J. Platt
  Probabilistic outputs for support vector machines and comparison to regularized likelihood methods
  Advances in Large Margin Classifiers, A. Smola, P. Bartlett, B. Schoelkopf and D. Schuurmans, Eds. Cambridge, MA: MIT Press, 2000.
  http://citeseer.nj.nec.com/platt99probabilistic.html
- H.-T. Lin, C.-J. Lin and R. C. Weng
  A note on Platt's probabilistic outputs for support vector machines
  http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.ps
- C.-W. Hsu and C.-J. Lin
  A comparison on methods for multi-class support vector machines
  IEEE Transactions on Neural Networks, 13(2002) 415-425.
  http://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.ps.gz
- C.-W. Hsu and C.-J. Lin.
  A simple decomposition method for support vector machines
  Machine Learning 46(2002), 291-314.
  http://www.csie.ntu.edu.tw/~cjlin/papers/decomp.ps.gz
- K. Crammer, Y. Singer
  On the learnability and design of output codes for multiclass prolems
  Computational Learning Theory, 35-46, 2000.
  http://www.cs.huji.ac.il/~kobics/publications/mlj01.ps.gz

See Also

predict.ksvm, couple

Examples

```
## simple example using the spam data set
data(spam)

## create test and training set
index <- sample(1:dim(spam)[1])
spamtrain <- spam[index[1:floor(2 * dim(spam)[1]/3)], ]
spamtest <- spam[index[((2 * ceiling(dim(spam)[1]/3)) + 1):dim(spam)[1]], ]

## train a support vector machine
```

```
filter <- ksvm(type~.,data=spamtrain,kernel="rbfdot",
               kpar=list(sigma=0.05), C=5,cross=3)
filter

## predict mail type on the test set
mailtype <- predict(filter,spamtest[,-58])

## Check results
table(mailtype,spamtest[,58])

## Another example with the famous iris data
data(iris)

## Create a kernel function using the build in rbfdot function
rbf <- rbfdot(sigma=0.1)
rbf

## train a bound constraint support vector machine
irismodel <- ksvm(Species~.,data=iris,type="C-bsvc",kernel=rbf,
                  C=10,prob.model=TRUE)

irismodel

## get fitted values
fitted(irismodel)

## Test on the training set with probabilities as output
predict(irismodel, iris[,-5], type="probabilities")

## Demo of the plot function
x <- rbind(matrix(rnorm(120),,2),matrix(rnorm(120,mean=3),,2))
y <- matrix(c(rep(1,60),rep(-1,60)))

svp <- ksvm(x,y,type="C-svc")
plot(svp)


#### Use custom kernel

k <- function(x,y) {(sum(x*y) +1)*exp(-0.001*sum((x-y)^2))}
class(k) <- "kernel"

data(promotergene)

## train svm using custom kernel
gene <- ksvm(Class~.,data=promotergene,kernel=k,C=10,cross=5)

gene
```

```
## regression
# create data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.03)

# train support vector machine
regm <- ksvm(x,y,epsilon=0.01,kpar=list(sigma=16),cross=3)
plot(x,y,type="l")
lines(x,predict(regm,x),col="red")
```

`lssvm-class` *Class "lssvm"*

Description

The Gaussian Processes object

Objects from the Class

Objects can be created by calls of the form `new("lssvm", ...)`. or by calling the `lssvm` function

Slots

`tol`: Object of class `"numeric"` contains tolerance of termination criteria

`kernelf`: Object of class `"function"` contains the kernel function used

`kpar`: Object of class `"list"` contains the kernel parameter used

`kcall`: Object of class `"ANY"` contains the used function call

`type`: Object of class `"character"` contains type of problem

`terms`: Object of class `"ANY"` contains the terms representation of the symbolic model used (when using a formula)

`xmatrix`: Object of class `"matrix"` containing the data matrix used

`ymatrix`: Object of class `"ANY"` containing the response matrix

`fitted`: Object of class `"ANY"` containing the fitted values

`lev`: Object of class `"vector"` containing the levels of the response (in case of classification)

`nclass`: Object of class `"numeric"` containing the number of classes (in case of classification) `alpha`: Object of class `"ANY"` containing the computes alpha values

`alphaindex` Object of class `"list"` containing the indexes for the alphas in various classes (in multi-class problems).

`nvar`: Object of class `"numeric"` containing the computed variance

`error`: Object of class `"numeric"` containing the training error

`cross`: Object of class `"numeric"` containing the cross validation error

`n.action`: Object of class `"ANY"` containing the action performed in NA

Methods

alpha `signature(object = "lssvm")`: returns the alpha vector

cross `signature(object = "lssvm")`: returns the cross validation error

error `signature(object = "lssvm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "lssvm")`: returns the call performed

kernelf `signature(object = "lssvm")`: returns the kernel function used

kpar `signature(object = "lssvm")`: returns the kernel parameter used

lev `signature(object = "lssvm")`: returns the response levels (in classification)

type `signature(object = "lssvm")`: returns the type of problem

xmatrix `signature(object = "lssvm")`: returns the data matrix used

ymatrix `signature(object = "lssvm")`: returns the response matrix used

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`lssvm`, `ksvm-class`

Examples

```
# train model
data(iris)
test <- lssvm(Species~.,data=iris,var=2)
```

```
test
alpha(test)
error(test)
lev(test)
```

---

lssvm *Least Squares Support Vector Machine*

Description

The `lssvm` function in package **in Package ‘kernlab’** is an implementation of a reduced version of the Least Squares SVM. By utilizing the `csi` function a decomposition of the kernel matrix is computed and all subsequend calculations are done using this decomposed matrix.

Usage

```
## S4 method for signature 'formula':
lssvm(x, data=NULL, ..., subset, na.action = na.omit, scaled = TRUE)

## S4 method for signature 'vector':
lssvm(x, ...)

## S4 method for signature 'matrix':
lssvm(x, y = NULL, scaled = TRUE, kernel = "rbfdot",
kpar = "automatic", type = NULL, tau = 0.01, tol = 0.0001, rank =
floor(dim(x)[1]/4), delta = 40, cross = 0, fit = TRUE, ..., subset,
na.action = na.omit)

## S4 method for signature 'kernelMatrix':
lssvm(x, y, type = NULL, tau = 0.01, tol  =
      0.0001, rank = floor(dim(x)[1]/3), delta = 40,
       cross = 0, fit = TRUE, ...)

## S4 method for signature 'list':
lssvm(x, y, scaled = TRUE, kernel = "stringdot",
       kpar = list(length=4, lambda = 0.5), type = NULL, tau = 0.01,
       reduced = TRUE, tol  = 0.0001, rank = floor(dim(x)[1]/3),
       delta = 40, cross = 0, fit = TRUE, ..., subset)
```

Arguments

`x`	a symbolic description of the model to be fit, a matrix or vector containing the training data when a formula interface is not used or a `kernelMatrix` or a list of character vectors.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'lssvm' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for classification or regression - currently nor suported -).
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally to zero mean and unit variance. The center and scale values are returned and used for later predictions.
`type`	Type of problem. Either "classification" or "regression". Depending on whether `y` is a factor or not, the default setting for `type` is "classification" or "regression" respectively, but can be overwritten by setting an explicit value. (regression is currently not supported)
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel "Gaussian" • `polydot` Polynomial kernel • `vanilladot` Linear kernel • `tanhdot` Hyperbolic tangent kernel • `laplacedot` Laplacian kernel • `besseldot` Bessel kernel • `anovadot` ANOVA RBF kernel • `splinedot` Spline kernel • `stringdot` String kernel Setting the kernel parameter to "matrix" treats `x` as a kernel matrix calling the `kernelMatrix` interface.

	The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". • `length, lambda, normalized` for the "stringdot" kernel where length is the length of the strings considered, lambda the decay factor and normalized a logical parameter determining if the kernel evaluations should be normalized. Hyper-parameters for user defined kernels can be passed through the kpar parameter as well. `kpar` can also be set to the string "automatic" which uses the heuristics in `sigest` to calculate a good `sigma` value for the Gaussian RBF or Laplace kernel, from the data. (default = "automatic").
`tau`	the regularization parameter (default 0.01)
`reduced`	if set to `FALSE` the full linear problem of the lssvm is solved, when `TRUE` a reduced method using `csi` is used.
`rank`	the maximal rank of the decomposed kernel matrix, see `csi`
`delta`	number of columns of cholesky performed in advance, see `csi` (default 40)
`tol`	tolerance of termination criterion for the `csi` function, lower tolerance leads to more preciese approximation but may increase the training time and the decomposed matrix size (default: 0.0001)
`fit`	indicates whether the fitted values should be computed and included in the model or not (default: 'TRUE')
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Mean Squared Error for regression

`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

Least Squares Support Vector Machines are reformulation to the standard SVMs that lead to solving linear KKT systems. The algorithm is based on the minimization of a classical penalized least-squares cost function. The current implementation approximates the kernel matrix by an incomplete Cholesky factorization optained by the `csi` function, thus the solution is an approximation to the exact solution of the lssvm optimization problem. The quality of the solution depends on the approximation and can be influenced by the "rank" , "delta", and "tol" parameters.

Value

An S4 object of class `"lssvm"` containing the fitted model, Accessor functions can be used to access the slots of the object (see examples) which include:

`alpha`	the parameters of the `"lssvm"`
`coef`	the model coefficients (identical to alpha)
`b`	the model offset.
`xmatrix`	the training data used by the model

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

J. A. K. Suykens and J. Vandewalle
Least Squares Support Vector Machine Classifiers
Neural Processing Letters vol. 9, issue 3, June 1999

See Also

ksvm, gausspr, csi

Examples

```
## simple example
data(iris)

lir <- lssvm(Species~.,data=iris)

lir

lirr <- lssvm(Species~.,data= iris, reduced = FALSE)

lirr

## Using the kernelMatrix interface

iris <- unique(iris)

rbf <- rbfdot(0.5)

k <- kernelMatrix(rbf, as.matrix(iris[,-5]))

klir <- lssvm(k, iris[, 5])

klir

pre <- predict(klir, k)
```

---

musk *Musk data set*

Description

This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks.

Usage

```
data(musk)
```

Format

A data frame with 476 observations on the following 167 variables.

Variables 1-162 are ”distance features” along rays. The distances are measured in hundredths of Angstroms. The distances may be negative or positive, since they are actually measured relative to an origin placed along each ray. The origin was defined by a ”consensus musk” surface that is no longer used. Hence, any experiments with the data should treat these feature values as lying on an arbitrary continuous scale. In particular, the algorithm should not make any use of the zero point or the sign of each feature value.

Variable 163 is the distance of the oxygen atom in the molecule to a designated point in 3-space. This is also called OXY-DIS.

Variable 164 is the X-displacement from the designated point.

Variable 165 is the Y-displacement from the designated point.

Variable 166 is the Z-displacement from the designated point.

Class: 0 for non-musk, and 1 for musk

Source

UCI Machine Learning data repository

Examples

```
data(musk)

muskm <- ksvm(Class~.,data=musk,kernel="rbfdot",C=1000)

muskm
```

`onlearn-class` *Class ”onlearn”*

Description

The class of objects used by the Kernel-based Online learning algorithms

Objects from the Class

Objects can be created by calls of the form `new("onlearn", ...)`. or by calls to the function `inlearn`.

Slots

`kernelf`: Object of class `"function"` containing the used kernel function

`buffer`: Object of class `"numeric"` containing the size of the buffer

`kpar`: Object of class `"list"` containing the hyperparameters of the kernel function.

`xmatrix`: Object of class `"matrix"` containing the data points (similar to support vectors)

`fit`: Object of class `"numeric"` containing the decision function value of the last data point

`onstart`: Object of class `"numeric"` used for indexing

`onstop`: Object of class `"numeric"` used for indexing

`alpha`: Object of class `"ANY"` containing the model parameters

`rho`: Object of class `"numeric"` containing model parameter

`b`: Object of class `"numeric"` containing the offset

`pattern`: Object of class `"factor"` used for dealing with factors

`type`: Object of class `"character"` containing the problem type (classification, regression, or novelty

Methods

alpha `signature(object = "onlearn")`: returns the model parameters

b `signature(object = "onlearn")`: returns the offset

buffer `signature(object = "onlearn")`: returns the buffer size

fit `signature(object = "onlearn")`: returns the last decision function value

kernelf `signature(object = "onlearn")`: return the kernel function used

kpar `signature(object = "onlearn")`: returns the hyper-parameters used

onlearn `signature(obj = "onlearn")`: the learning function

predict `signature(object = "onlearn")`: the predict function

rho `signature(object = "onlearn")`: returns model parameter

show `signature(object = "onlearn")`: show function

type `signature(object = "onlearn")`: returns the type of proplem

xmatrix `signature(object = "onlearn")`: returns the stored data points

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`onlearn`, `inlearn`

Examples

```
## create toy data set
x <- rbind(matrix(rnorm(100),,2),matrix(rnorm(100)+3,,2))
y <- matrix(c(rep(1,50),rep(-1,50)),,1)

## initialize onlearn object
on <- inlearn(2,kernel="rbfdot",kpar=list(sigma=0.2),type="classification")

## learn one data point at the time
for(i in sample(1:100,100))
on <- onlearn(on,x[i,],y[i],nu=0.03,lambda=0.1)

sign(predict(on,x))
```

`onlearn` *Kernel Online Learning algorithms*

Description

Online Kernel-based Learning algorithms for classification, novelty detection, and regression.

Usage

```
## S4 method for signature 'onlearn':
onlearn(obj, x, y = NULL, nu = 0.2, lambda = 1e-04)
```

Arguments

`obj`	`obj` an object of class `onlearn` created by the initialization function `inlearn` containing the kernel to be used during learning and the parameters of the learned model
`x`	vector or matrix containing the data. Factors have to be numerically coded. If `x` is a matrix the code is run internally one sample at the time.
`y`	the class label in case of classification. Only binary classification is supported and class labels have to be -1 or +1.
`nu`	the parameter similarly to the `nu` parameter in SVM bounds the training error.
`lambda`	the learning rate

Details

The online algorithms are based on a simple stochastic gradient descent method in feature space. The state of the algorithm is stored in an object of class `onlearn` and has to be passed to the function at each iteration.

Value

The function returns an `S4` object of class `onlearn` containing the model parameters and the last fitted value which can be retrieved by the accessor method `fit`. The value returned in the classification and novelty detection problem is the decision function value phi. The accessor methods `alpha` returns the model parameters.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Kivinen J. Smola A.J. Williamson R.C.
Online Learning with Kernels
IEEE Transactions on Signal Processing vol. 52, Issue 8, 2004
`http://mlg.anu.edu.au/~smola/papers/KivSmoWil03.pdf`

See Also

`inlearn`

Examples

```
## create toy data set
x <- rbind(matrix(rnorm(100),,2),matrix(rnorm(100)+3,,2))
y <- matrix(c(rep(1,50),rep(-1,50)),,1)

## initialize onlearn object
on <- inlearn(2,kernel="rbfdot",kpar=list(sigma=0.2),type="classification")

ind <- sample(1:100,100)
## learn one data point at the time
for(i in ind)
on <- onlearn(on,x[i,],y[i],nu=0.03,lambda=0.1)

## or learn all the data
on <- onlearn(on,x[ind,],y[ind],nu=0.03,lambda=0.1)

sign(predict(on,x))
```

`plot` *plot method for support vector object*

Description

Plot a binary classification support vector machine object. The `plot` function returns a contour plot of the decision values.

Usage

```
## S4 method for signature 'ksvm':
plot(object, data=NULL, grid = 50, slice = list())
```

Arguments

`object`	a `ksvm` classification object created by the `ksvm` function
`data`	a data frame or matrix containing new data
`grid`	granularity for the contour plot.
`slice`	a list of named numeric values for the dimensions held constant (only needed if more than two variables are used). Dimensions not specified are fixed at 0.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

ksvm

Examples

```
## Demo of the plot function
x <- rbind(matrix(rnorm(120),,2),matrix(rnorm(120,mean=3),,2))
y <- matrix(c(rep(1,60),rep(-1,60)))

svp <- ksvm(x,y,type="C-svc")
plot(svp)
```

`prc-class` *Class "prc"*

Description

Principal Components Class

Objects of class "prc"

Objects from the class cannot be created directly but only contained in other classes.

Slots

`pcv`: Object of class `"matrix"` containing the principal component vectors

`eig`: Object of class `"vector"` containing the coresponding eigenvalues

`kernelf`: Object of class `"kfunction"` containing the kernel function used

`kpar`: Object of class `"list"` containing the kernel parameters used

`xmatrix`: Object of class `"input"` containing the data matrix used

`kcall`: Object of class `"ANY"` containing the function call

`n.action`: Object of class `"ANY"` containg the action performed on NA

Methods

eig `signature(object = "prc")`: returns the eigenvalues

kcall `signature(object = "prc")`: returns the performed call

kernelf `signature(object = "prc")`: returns the used kernel function

pcv `signature(object = "prc")`: returns the principal component vectors

predict `signature(object = "prc")`: embeeds new data

xmatrix `signature(object = "prc")`: returns the used data matrix

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`kpca-class`,`kha-class`, `kfa-class`

`predict.ksvm` *predict method for support vector object*

Description

Prediction of test data using support vector machines

Usage

```
## S4 method for signature 'ksvm':
predict(object, newdata, type = "response", coupler = "minpair")
```

Arguments

`object`	an S4 object of class `ksvm` created by the `ksvm` function
`newdata`	a data frame or matrix containing new data
`type`	one of `response`, `probabilities` ,`votes` indicating the type of output: predicted values, matrix of class probabilities, or matrix of vote counts.
`coupler`	Coupling method used in the multiclass case, can be one of `minpair` or `pkpd` (see reference for more details).

Value

If `type(object)` is `C-svc`, `nu-svc`, `C-bsvm` or `spoc-svc` the vector returned depends on the argument `type`:

`response`	predicted classes (the classes with majority vote).
`probabilities`	matrix of class probabilities (one column for each class and one row for each input).
`votes`	matrix of vote counts (one column for each class and one row for each new input)

If `type(object)` is `eps-svr`, `eps-bsvr` or `nu-svr` a vector of predicted values is returned. If `type(object)` is `one-classification` a vector of logical values is returned.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

- T.F. Wu, C.J. Lin, R.C. Weng.
  Probability estimates for Multi-class Classification by Pairwise Coupling
  `http://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf`
- H.T. Lin, C.J. Lin, R.C. Weng
  A note on Platt's probabilistic outputs for support vector machines
  `http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.ps`

Examples

```
## example using the promotergene data set
data(promotergene)

## create test and training set
ind <- sample(1:dim(promotergene)[1],20)
genetrain <- promotergene[-ind, ]
genetest <- promotergene[ind, ]

## train a support vector machine
gene <- ksvm(Class~.,data=genetrain,kernel="rbfdot",kpar=list(sigma=0.015),
             C=70,cross=4,prob.model=TRUE)
```

```
gene

## predict gene type probabilities on the test set
genetype <- predict(gene,genetest,type="probabilities")
genetype
```

`promotergene` *E. coli promoter gene sequences (DNA)*

Description

Promoters have a region where a protein (RNA polymerase) must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align. The data contains DNA sequences of promoters and non-promoters.

Usage

```
data(promotergene)
```

Format

A data frame with 106 observations and 58 variables. The first variable `Class` is a factor with levels `+` for a promoter gene and `-` for a non-promoter gene. The remaining 57 variables `V2 to V58` are factors describing the sequence. The DNA bases are coded as follows: `a` adenine `c` cytosine `g` guanine `t` thymine

Source

UCI Machine Learning data repository
`ftp://ftp.ics.uci.edu/pub/machine-learning-databases/molecular-biology/promoter-gene-sequences`

References

Towell, G., Shavlik, J. and Noordewier, M.
Refinement of Approximate Domain Theories by Knowledge-Based Artificial Neural Networks.
In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)

Examples

```
data(promotergene)

## Create classification model using Gaussian Processes

prom <- gausspr(Class~.,data=promotergene,kernel="rbfdot",
               kpar=list(sigma=0.02),cross=4)
prom

## Create model using Support Vector Machines

promsv <- ksvm(Class~.,data=promotergene,kernel="laplacedot",kpar="automatic",
               C=60,cross=4)
promsv
```

`ranking-class` *Class "ranking"*

Description

Object of the class `"ranking"` are created from the `ranking` function and extend the class `matrix`

Objects from the Class

Objects can be created by calls of the form `new("ranking", ...)`.

Slots

`.Data`: Object of class `"matrix"` containing the data ranking and scores

`convergence`: Object of class `"matrix"` containing the convergence matrix

`edgegraph`: Object of class `"matrix"` containing the edgegraph

Extends

Class `"matrix"`, directly.

Methods

show `signature(object = "ranking")`: displays the ranking score matrix

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

ranking

Examples

```
data(spirals)

## create data set to be ranked
ran<-spirals[rowSums(abs(spirals)<0.55)==2,]

## rank points according to "relevance" to point 54 (up left)
ranked<-ranking(ran,54,kernel="rbfdot",kpar=list(sigma=100),
                edgegraph=TRUE)

ranked
edgegraph(ranked)[1:10,1:10]
```

`ranking` *Ranking*

Description

A universal ranking algorithm which assigns importance/ranking to data points given a query.

Usage

```
## S4 method for signature 'matrix':
ranking(x, y,  kernel ="rbfdot", kpar = list(sigma = 1),
        scale = TRUE, alpha = 0.99, iterations = 600,
        edgegraph = FALSE, convergence = FALSE ,...)
```

Arguments

`x`	a matrix containing the data to be ranked
`y`	The index of the query point in the data matrix or a vector of length equal to the rows of the data matrix having a one at the index of the query points index and zero at all the other points.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`scale`	If TRUE the data matrix columns are scaled to zero mean and unit variance.
`alpha`	The `alpha` parameter takes values between 0 and 1 and is used to control the authoritative scores received from the unlabeled

	points. For 0 no global structure is found the algorithm ranks the points similarly to the original distance metric.
`iterations`	Maximum number of iterations
`edgegraph`	Construct edgegraph (only supported with the RBF kernel)
`convergence`	Include convergence matrix in results
`...`	Additional arguments

Details

A simple universal ranking algorithm which exploits the intrinsic global geometric structure of the data. In many real world applications this should be superior to a local method in which the data are simply ranked by pairwise Euclidean distances. Firstly a weighted network is defined on the data and an authoritative score is assigned to each query. The query points act as source nodes that continually pump their authoritative scores to the remaining points via the weighted network and the remaining points further spread the scores they received to their neighbors. This spreading process is repeated until convergence and the points are ranked according to their score at the end of the iterations.

Value

An S4 object of class `ranking` which extends the `matrix` class. The first column of the returned matrix contains the original index of the points in the data matrix the second column contains the final score received by each point and the third column the ranking of the point. The object contains the following slots :

`edgegraph`	Containing the edgegraph of the data points.
`convergence`	Containing the convergence matrix

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

D. Zhou, J. Weston, A. Gretton, O. Bousquet, B. Schoelkopf
Ranking on Data Manifolds
Advances in Neural Information Processing Systems 16.
MIT Press Cambridge Mass. 2004
http://www.kyb.mpg.de/publications/pdfs/pdf2334.pdf

See Also

ranking-class, specc

Examples

```
data(spirals)

## create data from spirals
ran <- spirals[rowSums(abs(spirals) < 0.55) == 2,]

## rank points according to similarity to the most upper left point
ranked <- ranking(ran, 54, kernel = "rbfdot", kpar = list(sigma = 100), edgegraph = TR
ranked[54, 2] <- max(ranked[-54, 2])
c<-1:86
op <- par(mfrow = c(1, 2),pty="s")
plot(ran)
plot(ran, cex=c[ranked[,3]]/40)
```

`rvm-class` *Class "rvm"*

Description

Relevance Vector Machine Class

Objects from the Class

Objects can be created by calls of the form `new("rvm", ...)`. or by calling the `rvm` function.

Slots

`tol`: Object of class `"numeric"` contains tolerance of termination critiria used.

`kernelf`: Object of class `"function"` contains the kernel function used

`kpar`: Object of class `"list"` contains the hyperparameter used

`kcall`: Object of class `"ANY"` contains the function call

`type`: Object of class `"character"` contains type of problem

`terms`: Object of class `"ANY"` containing the terms representation of the symbolic model used (when using a formula interface) `xmatrix`: Object of class `"matrix"` contains the data matrix used during computation

`ymatrix`: Object of class `"ANY"` contains the response matrix

`fitted`: Object of class `"ANY"` with the fitted values, (predict on trianing set).

`lev`: Object of class `"vector"` contains the levels of the response (in classification)

`nclass`: Object of class `"numeric"` contains the number of classes (in classification)

`alpha`: Object of class `"ANY"` containing the the resulting alpha vector

`nvar`: Object of class `"numeric"` containing the calculated variance (in case of regression)

`mlike`: Object of class `"numeric"` containing the computed maximum likelihood

`RVindex`: Object of class `"vector"` containing the indexes of the resulting relevance vectors

`nRV`: Object of class `"numeric"` containing the number of relevance vectors

`cross`: Object of class `"ANY"` containing the relusting cross validation error

`error`: Object of class `"numeric"` containing the training error

`n.action`: Object of class `"ANY"` containing the action performed on NA

Methods

RVindex `signature(object = "rvm")`: returns the index of the relevance vectors

alpha `signature(object = "rvm")`: returns the resulting alpha vector

cross `signature(object = "rvm")`: returns the resulting cross validation error

error `signature(object = "rvm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values

kcall `signature(object = "rvm")`: returns the function call

kernelf `signature(object = "rvm")`: returns the used kernel function

kpar `signature(object = "rvm")`: returns the parameters of the kernel function

lev `signature(object = "rvm")`: returns the levels of the response (in classification)

mlike `signature(object = "rvm")`: returns the estimated maiximum likelihood

nvar `signature(object = "rvm")`: returns the calculated variance (in regression)

type `signature(object = "rvm")`: returns the type of problem

xmatrix `signature(object = "rvm")`: returns the data mmatrix used during computation

ymatrix `signature(object = "rvm")`: returns the used response

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`rvm`, `ksvm-class`

Examples

```
# create data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.05)

# train relevance vector machine
foo <- rvm(x, y)
foo

alpha(foo)
RVindex(foo)
fitted(foo)
kernelf(foo)
nvar(foo)

## show slots
slotNames(foo)
```

`rvm` *Relevance Vector Machine*

Description

The Relevance Vector Machine is a Bayesian model for regression and classification of identical functional form to the support vector machine. The `rvm` function currently supports only regression.

Usage

```
## S4 method for signature 'formula':
rvm(x, data=NULL, ..., subset, na.action = na.omit)

## S4 method for signature 'vector':
rvm(x, ...)

## S4 method for signature 'matrix':
rvm(x, y, type="regression", kernel="rbfdot", kpar=list(sigma=0.1),
alpha=1, var=0.1, var.fix=FALSE, iterations=100, verbosity=0, tol=
.Machine, double.eps,minmaxdiff = 1e-3, cross = 0, fit =TRUE, subset,
na.action = na.omit,...)
```

Arguments

`x`	a symbolic description of the model to be fit. Note, that an intercept is always included, whether given in the formula or not. When not using a formula x is a matrix or vector containg the variables in the model.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'rvm' is called from.
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`type`	`rvm` can only be used for regression at the moment.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab

	provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings: • `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	the list of hyper-parameters (kernel parameters). This is a list which contains the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`alpha`	The initial alpha vector. Can be either a vector of length equal to the number of data points or a single number.
`var`	the initial noise variance
`var.fix`	Keep noise variance fix during iterations (default: FALSE)
`iterations`	Number of iterations allowed (default: 100)
`tol`	tolerance of termination criterion
`minmaxdiff`	termination criteria. Stop when max difference is equall to this parameter (default:1e-3)
`verbosity`	print information on algorithm convergence (default = FALSE)
`fit`	indicates whether the fitted values should be computed and included in the model or not (default: TRUE)

`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the Mean Squared Error for regression
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)
`...`	additional parameters

Details

The Relevance Vector Machine typically leads to sparser models then the SVM. It also performs better in many cases (specially in regression).

Value

An S4 object of class "rvm" containing the fitted model. Accessor functions can be used to access the slots of the object which include :

`alpha`	The resulting relevance vectors
`alphaindex`	The index of the resulting relevance vectors in the data matrix
`nRV`	Number of relevance vectors
`RVindex`	The indexes of the relevance vectors
`error`	Training error (if fit == TRUE)

...

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Tipping, M. E.
Sparse Bayesian learning and the relevance vector machine
Journal of Machine Learning Research 1, 211-244
`http://www.jmlr.org/papers/volume1/tipping01a/tipping01a.pdf`

See Also

ksvm

Examples

```
# create data
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.05)

# train relevance vector machine
foo <- rvm(x, y)
foo
# print relevance vectors
alpha(foo)
RVindex(foo)

# predict and plot
ytest <- predict(foo, x)
plot(x, y, type ="l")
lines(x, ytest, col="red")
```

---

sigest *Hyperparameter estimation for the Gaussian Radial Basis kernel*

Description

Given a range of values for the ”sigma” inverse width parameter in the Gaussian Radial Basis kernel for use with Support Vector Machines. The estimation is based on the data to be used.

Usage

```
## S4 method for signature 'formula':
sigest(x, data=NULL, frac = 0.25, na.action = na.omit, scaled = TRUE)
## S4 method for signature 'matrix':
sigest(x, frac = 0.25, scaled = TRUE, na.action = na.omit)
```

Arguments

`x`	a symbolic description of the model upon the estimation is based. When not using a formula x is a matrix or vector containing the data
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'ksvm' is called from.
`frac`	Fraction of data to use for estimation. By default a quarter of the data is used to estimate the range of the sigma hyperparameter.
`scaled`	A logical vector indicating the variables to be scaled. If `scaled` is of length 1, the value is recycled as many times as needed and all non-binary variables are scaled. Per default, data are scaled internally to zero mean and unit variance (since this the default action in `ksvm` as well). The center and scale values are returned and used for later predictions.
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

Details

`sigest` estimates the range of values for the sigma parameter which would return good results when used with a Support Vector Machine (`ksvm`). The estimation is based upon the 0.1 and 0.9 quantile of $\|x - x'\|^2$. Basicly any value in between those two bounds will produce good results.

Value

Returns a vector of length 2 defining the range (upper bound and lower bound) of the sigma hyperparameter.

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

B. Caputo, K. Sim, F. Furesjo, A. Smola,
Appearance-based object recognition using SVMs: which kernel should I use?
Proc of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision, Whistler, 2002.

See Also

ksvm

Examples

```
## estimate good sigma values for promotergene
data(promotergene)
srange <- sigest(Class~.,data = promotergene)
srange

s <- sum(srange)/2
s
## create test and training set
ind <- sample(1:dim(promotergene)[1],20)
genetrain <- promotergene[-ind, ]
genetest <- promotergene[ind, ]

## train a support vector machine
gene <- ksvm(Class~.,data=genetrain,kernel="rbfdot",kpar=list(sigma = s),
             C=50,cross=3)
gene

## predict gene type on the test set
promoter <- predict(gene,genetest[,-1])

## Check results
table(promoter,genetest[,1])
```

---

spam *Spam E-mail Database*

Description

A data set collected at Hewlett-Packard Labs, that classifies 4601 e-mails as spam or non-spam. In addition to this class label there are 57 variables indicating the frequency of certain words and characters in the e-mail.

Usage

```
data(spam)
```

Format

A data frame with 4601 observations and 58 variables.

The first 48 variables contain the frequency of the variable name (e.g., business) in the e-mail. If the variable name starts with num (e.g., num650) the it indicates the frequency of the corresponding number (e.g., 650). The variables 49-54 indicate the frequency of the characters ';', '(', '[', '!', '$', and '#'. The variables 55-57 contain the average, longest and total run-length of captial letters. Variable 58 indicates the type of the mail and is either `"nonspam"` or `"spam"`, i.e. unsolicited commercial e-mail.

Details

The data set contains 2788 e-mails classified as `"nonspam"` and 1813 classified as `"spam"`.

The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography... This collection of spam e-mails came from the collectors' postmaster and individuals who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalized spam filter. One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter.

Source

- Creators: Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt at Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304
- Donor: George Forman (gforman at nospam hpl.hp.com) 650-857-7835

These data have been taken from the UCI Repository Of Machine Learning Databases at `http://www.ics.uci.edu/~mlearn/MLRepository.html`

References

T. Hastie, R. Tibshirani, J.H. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

`specc-class` *Class "specc"*

Description

The Spectral Clustering Class

Objects from the Class

Objects can be created by calls of the form `new("specc", ...)`. or by calling the function `specc`.

Slots

`centers`: Object of class `"matrix"` containing the cluser centers

`size`: Object of class `"vector"` containing the number of points in each cluster

`withinss`: Object of class `"vector"` containing the within-cluster sum of squares for each cluster

`kernelf` Object of class `kernel` containing the used kernel function.

Methods

centers `signature(object = "specc")`: returns the cluster centers

withinss `signature(object = "specc")`: returns the within-cluster sum of squares for each cluster

size `signature(object = "specc")`: returns the number of points in each cluster

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

See Also

`specc`, `kpca-class`

Examples

```
## Cluster the spirals data set.
data(spirals)

sc <- specc(spirals, centers=2)

centers(sc)
size(sc)
```

---

specc *Spectral Clustering*

Description

A spectral clustering algorithm. This algorithm clusters points using eigenvectors of kernel matrixes derived from the data.

Usage

```
## S4 method for signature 'formula':
specc(x, data = NULL, na.action = na.omit, ...)

## S4 method for signature 'matrix':
specc(x, centers, kernel = "rbfdot", kpar = list(sigma = 0.1),
      iterations = 200, mod.sample = 0.6, na.action = na.omit, ...)
```

Arguments

`x`	the matrix of data to be clustered or a symbolic description of the model to be fit.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'specc' is called from.
`centers`	Either the number of clusters or a set of initial cluster centers. If the first, a random set of rows in the eigenvectors matrix are chosen as the initial centers.
`kernel`	the kernel function used in training and predicting. This parameter can be set to any function, of class kernel, which computes a dot product between two vector arguments. kernlab provides the most popular kernel functions which can be used by setting the kernel parameter to the following strings:

	• `rbfdot` Radial Basis kernel function "Gaussian" • `polydot` Polynomial kernel function • `vanilladot` Linear kernel function • `tanhdot` Hyperbolic tangent kernel function • `laplacedot` Laplacian kernel function • `besseldot` Bessel kernel function • `anovadot` ANOVA RBF kernel function • `splinedot` Spline kernel The kernel parameter can also be set to a user defined function of class kernel by passing the function name as an argument.
`kpar`	a character string or the list of hyper-parameters (kernel parameters). The default character string `"automatic"` uses a heuristic the determine a suitable value for the width parameter of the RBF kernel. The second option `"local"` (local scaling) uses a more advanced heuristic and sets a width parameter for every point in the data set. This is particularly useful when the data incorporates multiple scales. A list can also be used containing the parameters to be used with the kernel function. Valid parameters for existing kernels are : • `sigma` inverse kernel width for the Radial Basis kernel function "rbfdot" and the Laplacian kernel "laplacedot". • `degree, scale, offset` for the Polynomial kernel "polydot" • `scale, offset` for the Hyperbolic tangent kernel function "tanhdot" • `sigma, order, degree` for the Bessel kernel "besseldot". • `sigma, degree` for the ANOVA kernel "anovadot". Hyper-parameters for user defined kernels can be passed through the kpar parameter as well.
`mod.sample`	Proportion of data to use when estimating sigma default 0.6
`iterations`	The maximum number of iterations allowed.
`na.action`	The action to perform on NA
`...`	additional parameters

Details

In Spectral Clustering one uses the top `k` (number of clusters) eigenvectors of a matrix derived from the distance between points. Very good results are obtained by using a standard clustering technique to cluster the resulting eigenvector matrixes.

Value

An S4 object of class `specc` wich extends the class `vector` containing integers indicating the cluster to which each point is allocated. The following slots contain useful information

`centers`	A matrix of cluster centers.
`size`	The number of point in each cluster
`withinss`	The within-cluster sum of squares for each cluster
`kernelf`	The kernel function used

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzoglou@ci.tuwien.ac.at⟩

References

Andrew Y. Ng, Michael I. Jordan, Yair Weiss
On Spectral Clustering: Analysis and an Algorithm
Neural Information Processing Symposium 2001
`http://www.nips.cc/NIPS2001/papers/psgz/AA35.ps.gz`

See Also

`kkmeans`, `kpca`, `kcca`

Examples

```
## Cluster the spirals data set.
data(spirals)

sc <- specc(spirals, centers=2)

sc
centers(sc)
size(sc)
withinss(sc)

plot(spirals, col=sc)
```

`spirals` *Spirals Dataset*

Description

A toy data set representing two spirals with Gaussian noise. The data was created with the `mlbench.spirals` function in `mlbench`.

Usage

```
data(spirals)
```

Format

A matrix with 300 observations and 2 variables.

Examples

```
data(spirals)
plot(spirals)
```

`ticdata` *The Insurance Company Data*

Description

This data set used in the CoIL 2000 Challenge contains information on customers of an insurance company. The data consists of 86 variables and includes product usage data and socio-demographic data derived from zip area codes. The data was collected to answer the following question: Can you predict who would be interested in buying a caravan insurance policy and give an explanation why ?

Usage

```
data(ticdata)
```

Format

ticdata: Dataset to train and validate prediction models and build a description (9822 customer records). Each record consists of 86 attributes, containing sociodemographic data (attribute 1-43) and product ownership (attributes 44-86). The sociodemographic data is derived from zip codes. All customers living in areas with the same zip code have the same sociodemographic attributes. Attribute 86, `CARAVAN:Number of mobile home policies`, is the target variable.

Data Format

1	`STYPE`	Customer Subtype
2	`MAANTHUI`	Number of houses 1 - 10
3	`MGEMOMV`	Avg size household 1 - 6
4	`MGEMLEEF`	Average age
5	`MOSHOOFD`	Customer main type
6	`MGODRK`	Roman catholic
7	`MGODPR`	Protestant ...
8	`MGODOV`	Other religion
9	`MGODGE`	No religion
10	`MRELGE`	Married
11	`MRELSA`	Living together
12	`MRELOV`	Other relation
13	`MFALLEEN`	Singles
14	`MFGEKIND`	Household without children
15	`MFWEKIND`	Household with children
16	`MOPLHOOG`	High level education
17	`MOPLMIDD`	Medium level education
18	`MOPLLAAG`	Lower level education
19	`MBERHOOG`	High status
20	`MBERZELF`	Entrepreneur
21	`MBERBOER`	Farmer
22	`MBERMIDD`	Middle management
23	`MBERARBG`	Skilled labourers
24	`MBERARBO`	Unskilled labourers
25	`MSKA`	Social class A
26	`MSKB1`	Social class B1
27	`MSKB2`	Social class B2
28	`MSKC`	Social class C
29	`MSKD`	Social class D
30	`MHHUUR`	Rented house
31	`MHKOOP`	Home owners
32	`MAUT1`	1 car
33	`MAUT2`	2 cars

34	MAUTO	No car
35	MZFONDS	National Health Service
36	MZPART	Private health insurance
37	MINKM30	Income >30.000
38	MINK3045	Income 30-45.000
39	MINK4575	Income 45-75.000
40	MINK7512	Income 75-122.000
41	MINK123M	Income <123.000
42	MINKGEM	Average income
43	MKOOPKLA	Purchasing power class
44	PWAPART	Contribution private third party insurance
45	PWABEDR	Contribution third party insurance (firms)
46	PWALAND	Contribution third party insurane (agriculture)
47	PPERSAUT	Contribution car policies
48	PBESAUT	Contribution delivery van policies
49	PMOTSCO	Contribution motorcycle/scooter policies
50	PVRAAUT	Contribution lorry policies
51	PAANHANG	Contribution trailer policies
52	PTRACTOR	Contribution tractor policies
53	PWERKT	Contribution agricultural machines policies
54	PBROM	Contribution moped policies
55	PLEVEN	Contribution life insurances
56	PPERSONG	Contribution private accident insurance policies
57	PGEZONG	Contribution family accidents insurance policies
58	PWAOREG	Contribution disability insurance policies
59	PBRAND	Contribution fire policies
60	PZEILPL	Contribution surfboard policies
61	PPLEZIER	Contribution boat policies
62	PFIETS	Contribution bicycle policies
63	PINBOED	Contribution property insurance policies
64	PBYSTAND	Contribution social security insurance policies
65	AWAPART	Number of private third party insurance 1 - 12
66	AWABEDR	Number of third party insurance (firms) ...
67	AWALAND	Number of third party insurane (agriculture)
68	APERSAUT	Number of car policies
69	ABESAUT	Number of delivery van policies
70	AMOTSCO	Number of motorcycle/scooter policies
71	AVRAAUT	Number of lorry policies
72	AAANHANG	Number of trailer policies
73	ATRACTOR	Number of tractor policies
74	AWERKT	Number of agricultural machines policies
75	ABROM	Number of moped policies
76	ALEVEN	Number of life insurances

77	APERSONG	Number of private accident insurance policies
78	AGEZONG	Number of family accidents insurance policies
79	AWAOREG	Number of disability insurance policies
80	ABRAND	Number of fire policies
81	AZEILPL	Number of surfboard policies
82	APLEZIER	Number of boat policies
83	AFIETS	Number of bicycle policies
84	AINBOED	Number of property insurance policies
85	ABYSTAND	Number of social security insurance policies
86	CARAVAN	Number of mobile home policies 0 - 1

Note: All the variables starting with M are zipcode variables. They give information on the distribution of that variable, e.g. Rented house, in the zipcode area of the customer.

Details

Information about the insurance company customers consists of 86 variables and includes product usage data and socio-demographic data derived from zip area codes. The data was supplied by the Dutch data mining company Sentient Machine Research and is based on a real world business problem. The training set contains over 5000 descriptions of customers, including the information of whether or not they have a caravan insurance policy. The test set contains 4000 customers. The test and data set are merged in the ticdata set. More information about the data set and the CoIL 2000 Challenge along with publications based on the data set can be found at http://www.liacs.nl/~putten/library/cc2000/.

Source

- UCI KDD Archive:http://kdd.ics.uci.edu
- Donor: Sentient Machine Research
  Peter van der Putten
  Sentient Machine Research
  Baarsjesweg 224
  1058 AA Amsterdam
  The Netherlands
  +31 20 6186927
  pvdputten@hotmail.com, putten@liacs.nl

References

Peter van der Putten, Michel de Ruiter, Maarten van Someren *CoIL Challenge 2000 Tasks and Results: Predicting and Explaining Caravan Policy Ownership*
`http://www.liacs.nl/~putten/library/cc2000/`

`vm-class` *Class "vm"*

Description

An S4 VIRTUAL class used as a base for the various vector machine classes in **kernlab**

Objects from the Class

Objects from the class cannot be created directly but only contained in other classes.

Slots

`alpha`: Object of class `"listI"` containing the resulting alpha vector (list in case of multiclass classification) (support vectors)

`type`: Object of class `"character"` containing the vector machine type e.g. ("C-svc", "nu-svc", "C-bsvc", "spoc-svc", "one-svc", "eps-svr", "nu-svr", "eps-bsvr")

`kernelf`: Object of class `"function"` containing the kernel function

`kpar`: Object of class `"list"` containing the kernel function parameters (hyperparameters)

`kcall`: Object of class `"call"` containing the function call

`terms`: Object of class `"ANY"` containing the terms representation of the symbolic model used (when using a formula)

`xmatrix`: Object of class `"input"` the data matrix used during computations (support vectors) (possibly scaled and whithout NA)

`ymatrix`: Object of class `"output"` the response matrix/vector

`fitted`: Object of class `"output"` with the fitted values, predictions using the training set.

`lev`: Object of class `"vector"` with the levels of the response (in the case of classifiaction)

`nclass`: Object of class `"numeric"` containing the number of classes (in the case of classification)

`error`: Object of class `"numeric"` containing the training error

`cross`: Object of class `"numeric"` containing the cross-validation error

`n.action`: Object of class `"ANY"` containing the action performed for NA

Methods

alpha `signature(object = "vm")`: returns the complete alpha vector (wit zero values)

cross `signature(object = "vm")`: returns the cross-validation error

error `signature(object = "vm")`: returns the training error

fitted `signature(object = "vm")`: returns the fitted values (predict on training set)

kernelf `signature(object = "vm")`: returns the kernel function

kpar `signature(object = "vm")`: returns the kernel parameters (hyperparameters)

lev `signature(object = "vm")`: returns the levels in case of classification

kcall `signature(object="vm")`: returns the function call

type `signature(object = "vm")`: returns the problem type

xmatrix `signature(object = "vm")`: returns the data matrix used(support vectors)

ymatrix `signature(object = "vm")`: returns the response vector

Author(s)

Alexandros Karatzoglou
⟨alexandros.karatzolgou@ci.tuwien.ac.at⟩

See Also

`ksvm-class`, `rvm-class`, `gausspr-class`

Bibliography

Agrawal R, Srikant R (1994). "Fast Algorithms for Mining Association Rules." In JB Bocca, M Jarke, C Zaniolo (eds.), "Proc. 20th Int. Conf. Very Large Data Bases, VLDB," pp. 487–499. Morgan Kaufmann. URL http://www.almaden.ibm.com/software/projects/hdb/Publications/papers/vldb94.pdf.

Baldi P, Brunak S (1998). *Bioinformatics The Machine Learning Approach*. MIT Press.

Blake C, Merz C (1998). "UCI Repository of Machine Learning Databases." University of California, Irvine, Dept. of Information and Computer Sciences,http://www.ics.uci.edu/~mlearn/MLRepository.html.

Boser BE, Guyon IM, Vapnik VN (1992). "A Training Algorithm for Optimal Margin Classifiers." In D Haussler (ed.), "Proceedings of the Annual Conference on Computational Learning Theory," pp. 144 – 152. ACM Press, Pittsburgh, PA.

Bray M, Koller-Meier E, Müller P, Van Gool L, Schraudolph NN (2005). "Stochastic Optimization for High-Dimensional Tracking in Dense Range Maps." *IEE Proceedings Vision, Image & Signal Processing*.

Cancedda N, Gaussier E, Goutte C, Renders JM (2003). "Word-Sequence Kernels." *Journal of Machine Learning Research*, **3**, 1059–1082. URL http://mitpress.mit.edu/journals/pdf/jmlr_3_6_1059_0.pdf.

Canu S, Grandvalet Y, Rakotomamonjy A (2003). "SVM and Kernel Methods Matlab Toolbox." Perception Systemes et Information, INSA de Rouen, Rouen, France. http://asi.insa-rouen.fr/~arakotom/toolbox/index.

Caputo B, Sim K, Furesjo F, Smola A (2002). "Appearance-based Object Recognition using SVMs: Which Kernel Should I Use?" *Proc of NIPS* *workshop on Statistical methods for computational experiments in visual processing and computer vision, Whistler, 2002.*

Chambers JM (1998). *Programming with Data*. Springer, New York. ISBN 0-387-98503-4.

Chang CC, Lin CJ (2001). "LIBSVM: A Library for Support Vector Machines." Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Collins M, Duffy N (2001). "Convolution Kernels for Natural Language." *Advances in Neural Information Processing Systems*, **14**. URL http://books.nips.cc/papers/files/nips14/AA58.pdf.

Collobert R, Bengio S, Mariethoz J (2002). "Torch: A Modular Machine Learning Software Library." http://www.torch.ch/.

Crammer K, Singer Y (2000). "On the Learnability and Design of Output Codes for Multiclass Prolems." *Computational Learning Theory*, pp. 35–46. URL http://www.cs.huji.ac.il/~kobics/publications/mlj01.ps.gz.

Dhillon I, Guan Y, Kulis B (2004). "A Unified View of Kernel k-means, Spectral Clustering and Graph Partitioning." *UTCS Technical Report*. URL http://www.cs.utexas.edu/users/kulis/pubs/spectral_techreport.pdf.

Dimitriadou E, Hornik K, Leisch F, Meyer D, Weingessel A (2005). "e1071: Misc Functions of the Department of Statistics (e1071), TU Wien, Version 1.5-11." Available from http://cran.R-project.org.

Elkan C (2003). "Using the Triangle Inequality to Accelerate k-Means." *In Proceedings of the Twentieth International Conference on Machine Learning (ICML'03)*, pp. 147–153. URL http://www-cse.ucsd.edu/~elkan/kmeansicml03.pdf.

Fowlkes C, Belongie S, Chung F, Malik J (2004). "Spectral grouping using the Nystrom method." *Transactions on Pattern Analysis and Machine Intelligence*, **26(2)**, 214–225. URL http://www.cs.berkeley.edu/~malik/papers/FBCM-nystrom.pdf.

Gammerman A, Bozanic N, Schölkopf B, Vovk V, Vapnik V, Bottou L, Smola A, Watkins C, LeCun Y, Saunders C, Stitson M, Weston J (2001). "Royal Holloway Support Vector Machines." URL http://svm.dcs.rhbnc.ac.uk/dist/index.shtml.

Griewank A (2000). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Frontiers in Applied Mathematics. SIAM, Philadelphia.

Guermeur Y (2004). "M-SVM." Lorraine Laboratory of IT Research and its Applications, URL `http://www.loria.fr/~guermeur/`.

Gunn SR (1998). "Matlab Support Vector Machines." University of Southampton, Electronics and Computer Science, URL `http://www.isis.ecs.soton.ac.uk/resources/svminfo/`.

Guyon I, Boser B, Vapnik V (1993). "Automatic Capacity Tuning of Very Large VC-Dimension Classifiers." In SJ Hanson, JD Cowan, CL Giles (eds.), "Advances in Neural Information Processing Systems 5," pp. 147 – 155. Morgan Kaufmann Publishers.

Hastie T (2004). "**svmpath**: The SVM Path Algorithm." R package, Version 0.9. Available from `http://cran.R-project.org`.

Hastie T, Rosset S, Tibshirani R, Zhu J (2004). "The Entire Regularization Path for the Support Vector Machine." *Journal of Machine Learning Research*, **5**, 1391–1415. URL `http://www.jmlr.org/papers/volume5/hastie04a/hastie04a.pdf`.

Hastie T, Tibshirani R, Friedman JH (2001). *The Elements of Statistical Learning.* Springer.

Haussler D (1999). "Convolutional Kernels on Discrete Structures." *Technical Report UCSC-CRL-99 - 10*, Computer Science Department, UC Santa Cruz.

Herbrich R (2002). *Learning Kernel Classifiers Theory and Algorithms.* Adaptive Computation and Machine Learning. The MIT Press.

Hsu CW, Lin CJ (2002a). "A Comparison of Methods for Multi-class Support Vector Machines." *IEEE Transactions on Neural Networks*, **13**, 1045–1052. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.ps.gz`.

Hsu CW, Lin CJ (2002b). "A Comparison of Methods for Multi-class Support Vector Machines." *IEEE Transactions on Neural Networks*, **13**, 415–425. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.ps.gz`.

Hsu CW, Lin CJ (2002c). "A Simple Decomposition Method for Support Vector Machines." *Machine Learning*, **46**, 291–314. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/decomp.ps.gz`.

Jaakkola T, Haussler D (1999). "Exploiting Generative Models in Discriminative Classifiers." *Advances in Neural Information Processing Systems*, **12**. URL http://books.nips.cc/papers/files/nips11/0487.pdf.

Joachims T (1998). "Text Categorization with Support Vector Machines: Learning with Many Relevant Features." In "Proceedings of the European Conference on Machine Learning," pp. 137 – 142. Springer, Berlin.

Joachims T (1999). "Making Large-scale SVM Learning Practical." *In Advances in Kernel Methods — Support Vector Learning*. URL http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_99a.ps.gz.

Joachims T (2002). *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms*. The Kluwer International Series In Engineerig And Computer Science. Kluwer Academic Publishers, Boston.

Karatzoglou A, Meyer D, Hornik K (2006). "Support Vector Machine in R (forthcoming)." *Journal of Statistical Software*.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "kernlab - An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(9). URL http://www.jstatsoft.org/counter.php?id=105&url=v11/i09/v11i09.pdf&ct=1.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2005a). "**kernlab** – Kernel Methods." R package, Version 0.6-2. Available from http://cran.R-project.org.

Karatzoglou A, Vishwanathan S, Schraudolph NN, Smola AJ (2005b). "Step Size-Adapted Online Support Vector Learning." *Proc. 8th Intl. Symp. Signal Processing & Applications*, **2**, 823–826. URL http://ieeexplore.ieee.org/xpl/RecentCon.jsp?punumber=10550.

Kivinen J, Smola A, Williamson R (2004a). "Online Learning with Kernels." *IEEE Transactions on Signal Processing*, **52**. URL http://mlg.anu.edu.au/~smola/papers/KivSmoWil03.pdf.

Kivinen J, Smola A, Williamson RC (2004b). "Online Learning with Kernels." *IEEE Transactions on Signal Processing*, **52**(8).

Knerr S, Personnaz L, Dreyfus G (1990). "Single-layer Learning Revisited: A Stepwise Procedure for Building and Training a Neural Network." *J. Fogelman, editor, Neurocomputing: Algorithms, Architectures and Applications*.

Kreßel U (1999). "Pairwise Classification and Support Vector Machines." *B. Schölkopf, C. J. C. Burges, A. J. Smola, editors, Advances in Kernel Methods — Support Vector Learning*, pp. 255–268.

Kuss M, Graepel T (2003). "The Geometry of Kernel Canonical Correlation Analysis." *MPI-Technical Reports*. URL http://www.kyb.mpg.de/publication.html?publ=2233.

Lang DT (2005). *Rstem: Interface to Snowball implementation of Porter's word stemming algorithm*. R package version 0.2-0.

Leisch F, Dimitriadou E (2001). "**mlbench**—A Collection for Artificial and Real-world Machine Learning Benchmarking Problems." R package, Version 0.5-6. Available from http://CRAN.R-project.org.

Leslie C, Eskin E, Cohen A, Weston J, Noble WS (2004). "Mismatch String Kernels for Discriminative Protein Classification." *Bioinformatics*, **20(4)**, 467–476. URL http://www1.cs.columbia.edu/compbio/mismatch/journal-mismatch-final.pdf.

Leslie C, Eskin E, Weston J, Noble WS (2002). "Mismatch String Kernels For SVM Protein Classification." In "Proceedings of Neural Information Processing Systems 2002," In press.

Lewis D (1997). "Reuters-21578 Text Categorization Test Collection." URL http://www.daviddlewis.com/resources/testcollections/reuters21578/.

Lin CF, Wang SD (1999). "Fuzzy Support Vector Machines." *IEEE Transactions on Neural Networks*, **13**, 464–471. URL ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-18.ps.

Lin CJ, More JJ (1999). "Newton's Method for Large-scale Bound Constrained Problems." *SIAM Journal on Optimization*, **9**, 1100–1127. URL http://www-unix.mcs.anl.gov/~more/tron/.

Lin CJ, Weng RC (2004). "Probabilistic Predictions for Support Vector Regression." Available at http://www.csie.ntu.edu.tw/~cjlin/papers/svrprob.pdf.

Lin HT, Lin CJ, Weng RC (2001). "A Note on Platt's Probabilistic Outputs for Support Vector Machines." Available at http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.ps.

Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002). "Text Classification using String Kernels." *Journal of Machine Learning Research*, **2**, 419–444. URL http://www.jmlr.org/papers/volume2/lodhi02a/lodhi02a.pdf.

Mangasarian O, Musicant D (1999). "Successive Overrelaxation for Support Vector Machines." *IEEE Transactions on Neural Networks*, **10**, 1032–1037. URL ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-18.ps.

MathWorks T (2005). "Matlab - The Language of Technical Computing." URL http://www.mathworks.com.

Meyer D, Leisch F, Hornik K (2003). "The Support Vector Machine under Test." *Neurocomputing*, **55**, 169–186.

Mika S, Rätsch G, Weston J, Schölkopf B, Müller KR (1999). "Fisher discriminant analysis with kernels." *Neural Networks for Signal Processing IX*, pp. 41–48. URL http://ieeexplore.ieee.org/iel5/6375/17054/00788121.pdf?tp=&arnumber=788121&isnumber=17054.

Milano M (2002). *Machine Learning Techniques for Flow Modeling and Control*. Ph.D. thesis, Eidgenössische Technische Hochschule (ETH), Zürich, Switzerland.

Mitchell T (1997). *Machine Learning*. McGraw Hill.

Ng AY, Jordan MI, Weiss Y (2001a). "On Spectral Clustering: Analysis and an Algorithm." *Neural Information Processing Symposium 2001*. URL http://www.nips.cc/NIPS2001/papers/psgz/AA35.ps.gz.

Ng AY, Jordan MI, Weiss Y (2001b). "On Spectral Clustering: Analysis and an Algorithm." *Advances in Neural Information Processing Systems*, **14**. URL http://www.nips.cc/NIPS2001/papers/psgz/AA35.ps.gz.

Osuna E, Freund R, Girosi F (1997). "Improved Training Algorithm for Support Vector Machines." *IEEE NNSP Proceedings 1997*. URL http://citeseer.ist.psu.edu/osuna97improved.html.

Platt JC (1998). "Probabilistic Outputs for Support Vector Machines and Comparison to Regularized Likelihood Methods." *B. Schölkopf, C. J. C. Burges, A. J. Smola, editors, Advances in Kernel Methods — Support Vector Learning*. URL http://research.microsoft.com/~jplatt/abstracts/smo.html.

Platt JC (2000). "Probabilistic Outputs for Support Vector Machines and Comparison to Regularized Likelihood Methods." *Advances in Large Margin Classifiers, A. Smola, P. Bartlett, B. Schölkopf and D. Schuurmans, Eds.* URL http://citeseer.nj.nec.com/platt99probabilistic.html.

Putten PVD, Ruiter MD, Someren MV (2000). "CoIL Challenge 2000 Tasks and Results: Predicting and Explaining Caravan Policy Ownership." *Coil Challenge 2000.* URL http://www.liacs.nl/~putten/library/cc2000/.

Quinlan JR (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers.

R Development Core Team (2005). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Ripley BD (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press.

Roever C, Raabe N, Luebke K, Ligges U (2005). "**klaR** – Classification and Visualization." R package, Version 0.4-1. Available from http://cran.R-project.org.

Rüping S (2004). "mySVM - A Support Vector Machine." University of Dortmund, Computer Science, URL http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html.

Russell S, Norvig P (2002). *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2 edition.

Schölkopf B, Platt J, Shawe-Taylor J, Smola AJ, Williamson RC (1999). "Estimating the Support of a High-Dimensonal Distribution." *Microsoft Research, Redmond, WA*, **TR 87**. URL http://research.microsoft.com/research/pubs/view.aspx?msr_tr_id=MSR-TR-99-87.

Schölkopf B, Smola A (2002). *Learning with Kernels.* MIT Press.

Schölkopf B, Smola A, Müller KR (1998). "Nonlinear Component Analysis as a Kernel Eigenvalue Problem." *Neural Computation*, **10**, 1299–1319. URL http://mlg.anu.edu.au/~smola/papers/SchSmoMul98.pdf.

Schölkopf B, Smola AJ, Williamson RC, Bartlett PL (2000). "New Support Vector Algorithms." *Neural Computation*, **12**, 1207–1245. URL http://caliban.ingentaselect.com/vl=3338649/cl=47/nw=1/rpsv/cgi-bin/cgi?body=linker&reqidx=0899-7667(2000)12:5L.1207.

Schraudolph NN (1999). "Local Gain Adaptation in Stochastic Gradient Descent." In "Proceedings of the International Conference on Artificial Neural Networks," pp. 569–574. IEE, London, Edinburgh, Scotland.

Schraudolph NN (2002). "Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent." *Neural Computation*, **14**(7), 1723–1738.

Schraudolph NN, Giannakopoulos X (2000). "Online Independent Component Analysis With Local Learning Rate Adaptation." In SA Solla, TK Leen, KR Müller (eds.), "Neural Information Processing Systems," volume 12, pp. 789–795. The MIT Press, Cambridge, MA.

Schwaighofer A (2005). "SVM toolbox for Matlab." Intelligent Data Analysis group (IDA), Fraunhofer FIRST, URL http://ida.first.fraunhofer.de/~anton/software.html.

Shawe-Taylor J, Christianini N (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Shi J, Malik J (2000). "Normalized Cuts and Image Segmentation." *Transactions on Pattern Analysis and Machine Intelligence*, **22(8)**, 888–905. URL http://www.cs.berkeley.edu/~malik/papers/SM-ncut.pdf.

Smola AJ, Mangasarian OL, Schölkopf B (2000). "Sparse Kernel Feature Analysis." *24th Annual Conference of Gesellschaft für Klassifikation*. URL ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-04.ps.

Stitson M, Gammerman A, Vapnik V, Vovk V, Watkins C, Weston J (1997). "Support vector regression with ANOVA decomposition kernels." *Technical Report CSD-TR-97 - 22*, Royal Holloway, University of London.

Tax DMJ, Duin RPW (1999). "Support Vector Domain Description." *Pattern Recognition Letters*, **20**, 1191–1199. URL http://www.ph.tn.tudelft.nl/People/bob/papers/prl_99_svdd.pdf.

Thomas Gärtner PAF, Wrobel S (2003). "On Graph Kernels: Hardness Results and Efficient Alternatives." *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory and Seventh Kernel Workshop (COLT-2003)*. URL http://springerlink.metapress.com/link.asp?id=vfhe64530q1x4ylj.

Tipping ME (2001). “Sparse Bayesian Learning and the Relevance Vector Machine.” *Journal of Machine Learning Research*, **1**, 211–244. URL http://www.jmlr.org/papers/volume1/tipping01a/tipping01a.pdf.

Vanderbei R (1999). “LOQO: An Interior Point Code for Quadratic Programming.” *Optimization Methods and Software*, **12**, 251–484. URL http://www.sor.princeton.edu/~rvdb/ps/loqo6.pdf.

Vapnik V (1995). *The Nature of Statistical Learning Theory*. Springer, NY.

Vapnik V (1998). *Statistical Learning Theory*. Wiley, New York.

Vishwanathan S, Smola A (2002). “Fast Kernels on Strings and Trees.” In “Proceedings of Neural Information Processing Systems 2002,” URL http://users.rsise.anu.edu.au/~vishy/papers/VisSmo02.pdf.

Vishwanathan S, Smola A (2004). “Fast Kernels for String and Tree Matching.” In K Tsuda, B Schölkopf, J Vert (eds.), “Kernels and Bioinformatics,” MIT Press, Cambridge, MA. URL http://users.rsise.anu.edu.au/~vishy/papers/VisSmo04.pdf.

Vishwanathan S, Smola A, Murty N (2003). “SimpleSVM.” *Proceedings of the 20th International Conference on Machine Learning ICML-03*. URL http://www.hpl.hp.com/conferences/icml2003/papers/352.pdf.

Watkins C (2000). “Dynamic Alignment Kernels.” In A Smola, PL Bartlett, B Schölkopf, D Schuurmans (eds.), “Advances in Large Margin Classifiers,” pp. 39 – 50. MIT Press, Cambridge, MA.

Weingessel A (2004). “**quadprog** – Functions to Solve Quadratic Programming Problems.” R package, Version 1.4-7. Available from http://cran.R-project.org.

Weston J, Elisseeff A, BakIr G, Sinz F (2005). “Spider: Object-Oriented Machine Learning Library.” Max Planc Institute for Biological Cybernetics, URL http://www.kyb.tuebingen.mpg.de/bs/people/spider/.

Williams CKI, Rasmussen CE (1995). “Gaussian Processes for Regression.” *Advances in Neural Information Processing*, **8**. URL http://books.nips.cc/papers/files/nips08/0514.pdf.

Williams CKI, Seeger M (2001). “Using the Nystrom method to Speed up Kernel Machines.” In TK Leen, TG Dietterich, V Tresp (eds.), “Advances in Neural Information Processing Systems 13,” pp. 682 – 688. MIT Press, Cambridge, MA. URL http://books.nips.cc/papers/files/nips13/WilliamsSeeger.pdf.

Witten IH, Frank E (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2 edition.

Wright S (1999). "Modified Cholesky Factorizations in Interior-point Algorithms for Linear Programming." *Journal in Optimization*, **9**, 1159–1191.

Wu TF, Lin CJ, Weng RC (2003). "Probability Estimates for Multi-class Classification by Pairwise Coupling." *Advances in Neural Information Processing*, **16**. URL http://books.nips.cc/papers/files/nips16/NIPS2003_0538.pdf.

Zhou D, Weston J, Gretton A, Bousquet O, Schölkopf B (2003). "Ranking on Data Manifolds." *Advances in Neural Information Processing Systems*, **16**. URL http://www.kyb.mpg.de/publications/pdfs/pdf2334.pdf.

List of Tables

List of Figures