# kernlab – An S4 Package for Kernel Methods in R

**Alexandros Karatzoglou**
Technische Universität Wien

**Alex Smola**
Australian National University, NICTA

**Kurt Hornik**
Wirtschaftsuniversität Wien

**Abstract**

**kernlab** is an extensible package for kernel-based machine learning methods in R. 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, online kernel methods and a spectral clustering algorithm. Moreover it provides a general purpose quadratic programming solver, and an incomplete Cholesky decomposition method.

*Keywords*: kernel methods, support vector machines, quadratic programming, ranking, clustering, S4, R.

## 1. Introduction

Machine learning is all about extracting structure from data, but it is often difficult to solve problems like classification, regression and clustering in the space in which the underlying observations have been made.

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 $\langle \Phi(x), \Phi(y) \rangle$ between the images of two data points $x, y$ in the feature space. The learning then takes place in the feature space, provided the learning algorithm can be entirely rewritten 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 $\Phi : X \rightarrow H$ is used, the dot product $\langle \Phi(x), \Phi(y) \rangle$ can be represented by a kernel function $k$

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle, \tag{1}$$

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

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.

### 1.1. Software review

The most prominent kernel based learning algorithm is without doubt the support vector machine

(SVM), so the existence of many support vector machine packages comes as little surprise. Most of the existing SVM software is written in C or C++, e.g. the award winning **libsvm**[1] (Chang and Lin 2001), **SVMlight**[2] (Joachims 1999), **SVMTorch**[3], Royal Holloway Support Vector Machines[4], **mySVM**[5], and **M-SVM**[6] with many packages providing interfaces to MATLAB (such as **libsvm**), and even some native MATLAB toolboxes[7] [8] [9].

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**[10] software which provides a MATLAB interface to various C/C++ SVM libraries and MATLAB implementations of various kernel-based algorithms, **Torch** [11] 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.

### 1.2. R software

The R package **e1071** 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, Leisch, and Hornik 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, Raabe, Luebke, and Ligges 2004) 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. 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.

### 2.1. 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

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[2] http://svmlight.joachims.org
[3] http://www.torch.ch
[4] http://svm.dcs.rhbnc.ac.uk
[5] http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.eng.html
[6] http://www.loria.fr/~guermeur/
[7] http://www.isis.ecs.soton.ac.uk/resources/svminfo/
[8] http://asi.insa-rouen.fr/~arakotom/toolbox/index
[9] http://www.cis.tugraz.at/igi/aschwaig/software.html
[10] http://www.kyb.tuebingen.mpg.de/bs/people/spider/
[11] http://www.torch.ch

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 than 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",  # the vector containing the cluster
                        centers="matrix",     # the cluster centers
                        size="vector",        # size of each cluster
                        kernelf="function",   # kernel function used
                        withinss = "vector"), # within cluster sum of squares
         prototype = structure(.Data = vector(),
                               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. Namespace

Namespaces were introduced in R 1.7.0 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 from the **methods** package.

## 2.3. Data

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

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 (van der Putten, de Ruiter, and van Someren 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` 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.

## 2.4. 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,] 0.5156599
```

The package includes implementations of the following kernels:

- the linear `vanilladot` kernel implements the simplest of all kernel functions

$$k(x, x') = \langle x, x' \rangle \tag{2}$$

  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`

$$k(x, x') = \exp(-\sigma \|x - x'\|^2) \tag{3}$$

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

- the polynomial kernel `polydot`

$$k(x, x') = (\text{scale} \cdot \langle x, x' \rangle + \text{offset})^{\text{degree}}. \tag{4}$$

  which is used in classification of images.

- the hyperbolic tangent kernel `tanhdot`

$$k(x, x') = \tanh\left(\text{scale} \cdot \langle x, x' \rangle + \text{offset}\right) \tag{5}$$

  which is mainly used as a proxy for neural networks.

- the Bessel function of the first kind kernel `besseldot`

$$k(x, x') = \frac{\text{Bessel}^n_{(\nu+1)}(\sigma\|x - x'\|)}{(\|x - x'\|)^{-n(\nu+1)}}. \tag{6}$$

  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`

$$k(x, x') = \exp(-\sigma\|x - x'\|) \tag{7}$$

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

- the ANOVA radial basis kernel `anovadot` performs well in multidimensional regression problems

$$k(x, x') = \left(\sum_{k=1}^{n} \exp(-\sigma(x^k - {x'}^k)^2)\right)^d \tag{8}$$

  where $x^k$ is the $k$th component of $x$.

## 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 $f = (f(x_1), \ldots, f(x_m))$ where

$$f(x_i) = \sum_{j=1}^{m} k(x_i, x_j)\alpha_j, \text{ hence } f = K\alpha. \tag{9}$$

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)
```

# 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 paper 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.

## 3.1. Support vector machine

Support vector machines (Vapnik 1998) have gained prominence in the field of machine learning and pattern classification and regression. The solutions to classification and regression problems sought by kernel-based algorithms such as the SVM are linear functions in the feature space:

$$f(x) = w^\top \Phi(x) \tag{10}$$

for some weight vector $w \in F$. The kernel trick can be exploited in this whenever the weight vector $w$ can be expressed as a linear combination of the training points, $w = \sum_{i=1}^{n} \alpha_i \Phi(x_i)$, implying that $f$ can be written as

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \tag{11}$$

A very important issue that arises is that of choosing a kernel $k$ for a given learning task. Intuitively, we wish to choose a kernel that induces the "right" metric in the space. Support Vector Machines choose a function $f$ that is linear in the feature space by optimizing some criterion over the sample. In the case of the 2-norm Soft Margin classification the optimization problem takes the form:

$$\begin{aligned} \text{minimize} \quad & t(w, \xi) = \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\langle x_i, w\rangle + b) \geq 1 - \xi_i \qquad (i = 1, \ldots, m) \\ & \xi_i \geq 0 \qquad (i = 1, \ldots, m) \end{aligned} \tag{12}$$

Based on similar methodology, SVMs deal with the problem of novelty detection (or one class classification) and regression.

**kernlab**'s implementation of support vector machines, `ksvm`, is based on the optimizers found in **bsvm**[12] (Hsu and Lin 2002b) and **libsvm** (Chang and Lin 2001) which includes a very efficient version of the Sequential Minimization Optimization (SMO). SMO decomposes 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 $\alpha_i$ because they must obey one linear equality constraint. At every step, SMO chooses two $\alpha_i$ to jointly optimize and finds the optimal values for these $\alpha_i$ 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 $\nu$-SVM classification formulation which is equivalent to the former but has a more natural ($\nu$) 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 than two classes (multi-class) a one-against-one or pairwise classification method (Knerr, Personnaz, and Dreyfus 1990; Kreßel 1999) is used. This method constructs $\binom{k}{2}$ classifiers 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 predicted more often

[12] http://www.csie.ntu.edu.tw/~cjlin/bsvm

wins ("Max Wins"). This method has been shown to produce robust results when used with SVMs (Hsu and Lin 2002a). Furthermore the `ksvm` implementation provides the ability to produce class probabilities as output instead of class labels. This is done by an improved implementation (Lin, Lin, and Weng 2001) of Platt's posteriori probabilities (Platt 2000) where a sigmoid function

$$P(y = 1 \mid f) = \frac{1}{1 + e^{Af+B}} \tag{13}$$

is fitted on the decision values $f$ of the binary SVM classifiers, $A$ and $B$ are estimated by minimizing the negative log-likelihood function. To extend the class probabilities to the multi-class case, each binary classifiers class probability output is combined by the `couple` method which implements methods for combing class probabilities proposed in (Wu, Lin, and Weng 2003).

Another approach for multIn order to create a similar probability output for regression, following Lin and Weng (2004), we suppose that the SVM is trained on data from the model

$$y_i = f(x_i) + \delta_i \tag{14}$$

where $f(x_i)$ is the underlying function and $\delta_i$ is independent and identical distributed random noise. Given a test data $x$ the distribution of $y$ given $x$ and allows one to draw probabilistic inferences about $y$ e.g. one can construct a predictive interval $\Phi = \Phi(x)$ such that $y \in \Phi$ with a certain probability. If $\hat{f}$ is the estimated (predicted) function of the SVM on new data then $\eta = \eta(x) = y - \hat{f}(x)$ is the prediction error and $y \in \Phi$ is equivalent to $\eta \in \Phi$. Empirical observation shows that the distribution of the residuals $\eta$ can be modeled both by a Gaussian and a Laplacian distribution with zero mean. In this implementation the Laplacian with zero mean is used :

$$p(z) = \frac{1}{2\sigma} e^{-\frac{|z|}{\sigma}} \tag{15}$$

Assuming that $\eta$ are independent the scale parameter $\sigma$ is estimated by maximizing the likelihood. The data for the estimation is produced by a three-fold cross-validation. For the Laplace distribution the maximum likelihood estimate is :

$$\sigma = \frac{\sum_{i=1}^{m} |\eta_i|}{m} \tag{16}$$

i-class classification supported by the `ksvm` function is the one proposed in Crammer and Singer (2000). This algorithm works by solving a single optimization problem including the data from all classes:

$$\begin{aligned}
\text{minimize} \quad & t(w_n, \xi) = \frac{1}{2} \sum_{n=1}^{k} \|w_n\|^2 + \frac{C}{m} \sum_{i=1}^{m} \xi_i \\
\text{subject to} \quad & \langle x_i, w_{y_i} \rangle - \langle x_i, w_n \rangle \geq b_i^n - \xi_i \qquad (i = 1, \ldots, m) \qquad (17) \\
\text{where} \quad & b_i^n = 1 - \delta_{y_i, n} \qquad (18)
\end{aligned}$$

where the decision function is

$$\text{argmax}_{m=1,\ldots,k} \langle x_i, w_n \rangle \tag{19}$$

This optimization problem is solved by a decomposition method proposed in Hsu and Lin (2002b) where optimal working sets are found (that is, sets of $\alpha_i$ values which have a high probability of being non-zero). The QP sub-problems are then solved by a modified version of the **TRON**[13] (Lin and More 1999) optimization software.

One-class classification or novelty detection (Schölkopf, Platt, Shawe-Taylor, Smola, and Williamson 1999; Tax and Duin 1999), where essentially an SVM detects outliers in a data set, is another algorithm supported by `ksvm`. SVM novelty detection works by creating a spherical decision boundary

[13] http://www-unix.mcs.anl.gov/~more/tron/

around a set of data points by a set of support vectors describing the spheres boundary. The $\nu$ parameter is used to control the volume of the sphere and consequently the number of outliers found. Again, the value of $\nu$ represents the fraction of outliers found. Furthermore, $\epsilon$-SVM (Vapnik 1995) and $\nu$-SVM (Schölkopf, Smola, Williamson, and Bartlett 2000) regression are also available.

The problem of model selection is partially addressed by an empirical observation for the popular Gaussian RBF kernel (Caputo, Sim, Furesjo, and Smola 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 =  0.0158141833456882

Number of Support Vectors : 89

Objective Function Value : -51.0358
Training error : 0
Cross validation error : 0.188354
Probability model included.

> predict(gene, genetest)

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

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

              +          -
[1,] 0.10718131 0.89281869
[2,] 0.47544472 0.52455528
[3,] 0.98851339 0.01148661
[4,] 0.03890831 0.96109169
[5,] 0.97270667 0.02729333
```

```
> set.seed(123)
> 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, data = x)
```

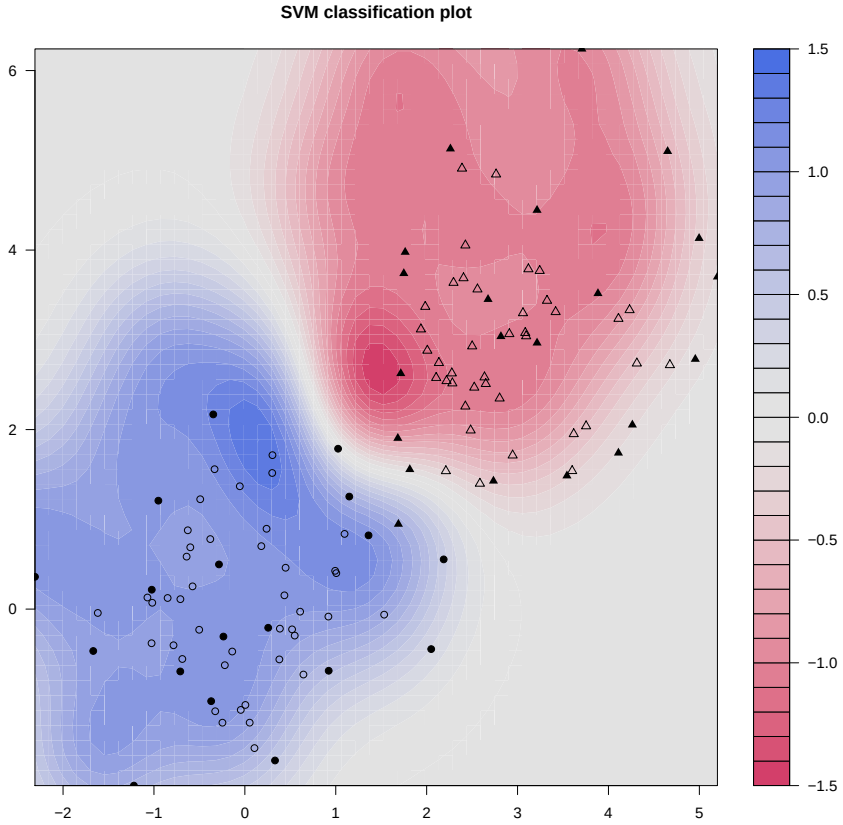


Figure 1: A contour plot of the SVM decision values for a toy binary classification problem using the `plot` function

## 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

$$y(x) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}, \mathbf{x}_n) + a_0 \tag{20}$$

where $\alpha_n$ 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 $\alpha$

$$p(\alpha, \beta) = \prod_{i=1}^{m} N(\beta_i \mid 0, a_i^{-1}) \text{Gamma}(\beta_i \mid \beta_\beta, \alpha_\beta) \tag{21}$$

governed by a set of hyper-parameters $\beta$, 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 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, kernel = "rbfdot", kpar = list(sigma = 0.1))
> rvmm

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 :  0.000952435
Training error : 0.00079788

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

## 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 than observing $t(x_i)$ we observe $y_i = t(x_i) + \xi_i$ where $\xi_i$ is assumed to be independed Gaussian distributed noise with zero mean. The posterior distribution is given by

$$p(\mathbf{y} \mid \mathbf{t}) = \left[\prod_i p(y_i - t(x_i))\right] \frac{1}{\sqrt{(2\pi)^m \det(K)}} \exp\left(\frac{1}{2}\mathbf{t}^T K^{-1} \mathbf{t}\right) \tag{22}$$

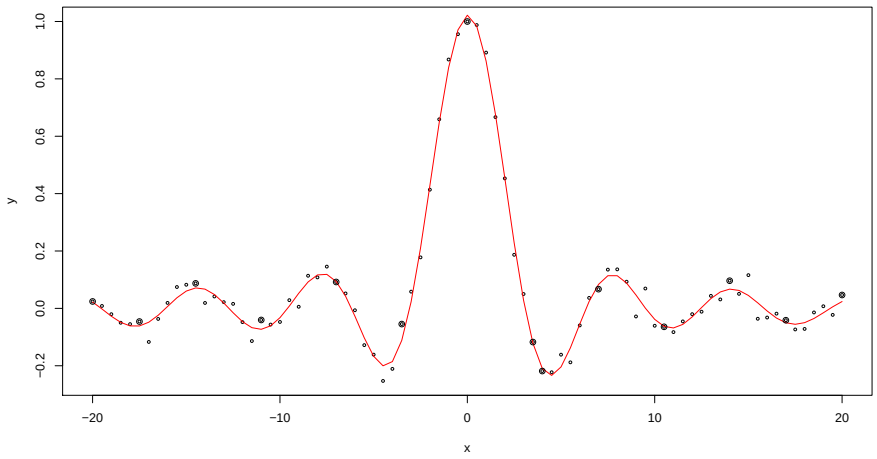


Figure 2: Relevance vector regression on data points created by the $sinc(x)$ function, relevance vectors are shown circled.

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

$$\ln p(\alpha \mid \mathbf{y}) = -\frac{1}{2\sigma^2}\|\mathbf{y} - K\alpha\|^2 - \frac{1}{2}\alpha^T K\alpha + c \tag{23}$$

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

$$\alpha = (K + \sigma^2\mathbf{1})^{-1}\mathbf{y} \tag{24}$$

Knowing $\alpha$ 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.

## 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, Weston, Gretton, Bousquet, and Schölkopf 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 points 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 $\mathbf{R}^n$ 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

$$f(t+1) = \alpha Lf(t) + (1-\alpha)y \tag{25}$$

```
> 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)
```

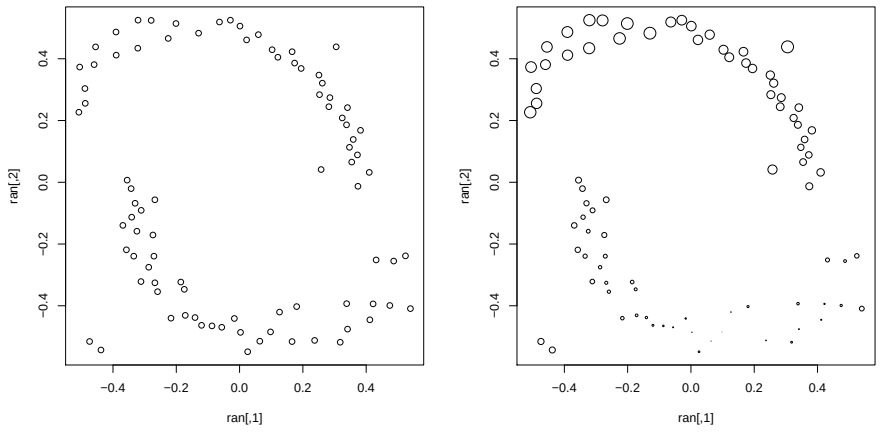


Figure 3: The points on the left are ranked according to their similarity to the upper most left point. Points with a higher rank appear bigger. Instead of ranking the points on simple Euclidean distance the structure of the data is recognized and all points on the upper structure are given a higher rank although further away in distance than points in the lower structure.

is iterated until convergence, where $\alpha$ 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.

## 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, Smola, and Williamson 2004). 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

$$R[f, P] := E_{(x,y)\ P}[l(f(x), y)] \tag{26}$$

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

$$R_{emp}[f, P] := \frac{1}{m} \sum_{t=1}^{m} l(f(x_t), y_t) \tag{27}$$

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.

$$R_{reg}[f, S] := R_{reg,\lambda}[f, S] := R_{emp}[f] = \frac{\lambda}{2}\|f\|_H^2 \tag{28}$$

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 $\lambda$ 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

$$R_inst[f, x, y] := R_{inst,\lambda}[f, x, y] := R_{reg,\lambda}[f, ((x, y))] \tag{29}$$

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

$$f_{t+1} = f_t - \eta\partial_f R_{inst,\lambda}[f, x_t, y_t]|_{f=f_t} \tag{30}$$

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

$$\partial_f l(f(x_t)), y_t) = l'(f(x_t), y_t)k(x_t, \cdot\,) \tag{31}$$

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

$$f_{t+1} := (1 - \eta\lambda)f_t - \eta_t\lambda'(f_t(x_t), y_t)k(x_t, \cdot\,) \tag{32}$$

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 = 0.1)
> sign(predict(on, x))
```

```
 [1] -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
[23]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1  1
[45]  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[67] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[89] -1 -1
```

## 3.6. Spectral clustering

Spectral clustering (Ng, Jordan, and Weiss 2001) is a recently emerged promising alternative to common clustering algorithms. In this method one uses the top eigenvectors of a matrix created

```
> data(spirals)
> sc <- specc(spirals, centers = 2)
> plot(spirals, pch = (23 - 2 * sc))
```

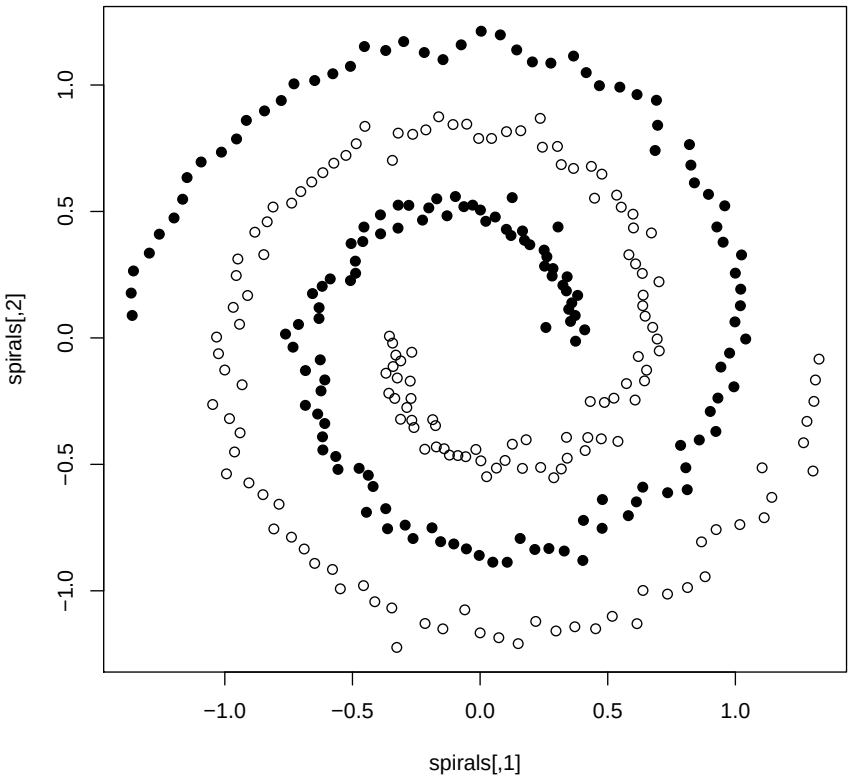


Figure 4: Clustering the two spirals data set with `specc`

by some similarity measure to cluster the data. Similarly to the ranking algorithm, an affinity matrix is created out from the data as

$$K_{ij} = \exp(-\sigma \|x_i - x_j\|^2) \tag{33}$$

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 $\sigma$ hyper-parameter. For a good value of $\sigma$ 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 $\sigma$.

## 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, Smola, and Müller 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 $x_k$, $k = 1, \ldots, M$, $x_k \in \mathbf{R}^N$, 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 $\Phi : \mathbf{R}^N \to F$, $x \mapsto \mathbf{X}$. Assuming that we deal with centered data and use the covariance matrix in $F$,

$$\hat{C} = \frac{1}{C}\sum_{j=1}^{N} \Phi(x_j)\Phi(x_j)^T \tag{34}$$

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

`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.

## 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, Mangasarian, and Schölkopf 2000) is computationaly simple and scales approximately one order of magnitute better on large data sets than standard KPCA. Choosing $\Omega[f] = \sum_{i=1}^{m} |\alpha_i|$ this yields

$$F_{LP} = \{\mathbf{w} | \mathbf{w} = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \text{with} \sum_{i=1}^{m} |\alpha_i| \leq 1\} \tag{35}$$

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

$$\nu^1 = \text{argmax}_{\nu \in F_{LP}} \frac{1}{m}\sum_{i=1}^{m} \langle \nu, \mathbf{\Phi}(x_i) - \frac{1}{m}\sum_{j=1}^{m} \mathbf{\Phi}(x_i) \rangle^2 \tag{36}$$

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 $\alpha_i$.

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.

## 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 classical CCA attempts to find linear

```
> 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")
```

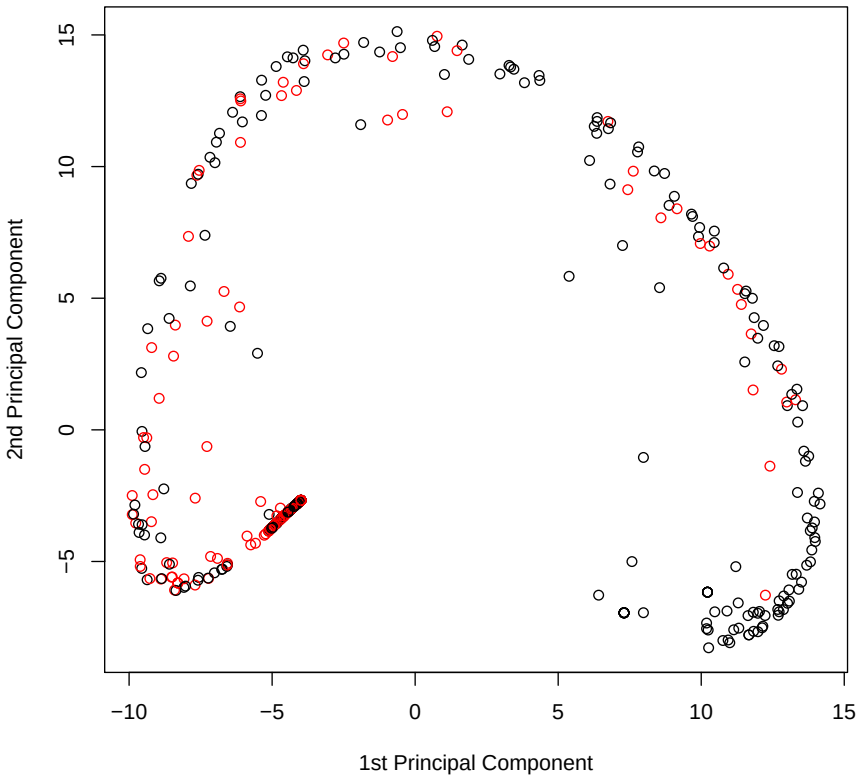


Figure 5: Projection of the spam data on two kernel principal components using an RBF kernel

```
> 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")
```

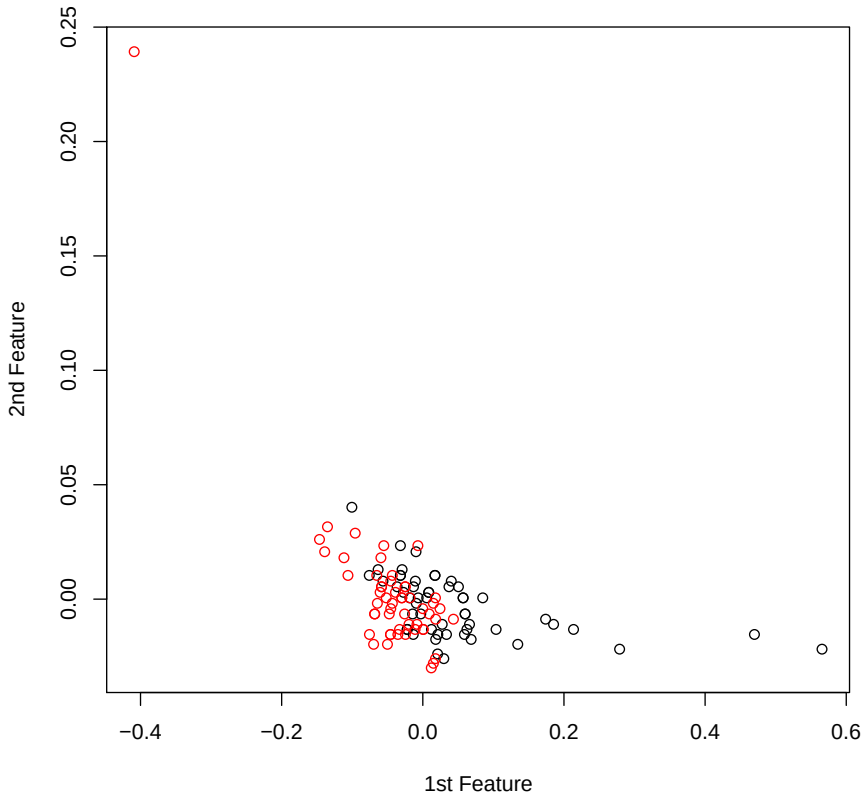


Figure 6: Projection of the spam data on two features using an RBF kernel

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 $\Phi : \mathbf{R}^N \to F$, $x \mapsto \mathbf{X}$ 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.

## 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

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n]. \end{array} \tag{37}$$

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

$$\begin{array}{ll} \text{minimize} & c^\top x + \frac{1}{2} x^\top H x \\ \text{subject to} & b \leq Ax \leq b + r \\ & l \leq x \leq u \end{array} \tag{38}$$

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

## 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 $\theta$. 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.

# 4. Conclusions

In this paper 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[14] under the GNU Public License.

A shorter version of this introduction to the R package **kernlab** is published as Karatzoglou, Smola, Hornik, and Zeileis (2004) in the *Journal of Statistical Software*.

# References

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.

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.

Hastie T, Tibshirani R, Friedman JH (2001). *The Elements of Statistical Learning*. Springer.

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 Simple Decomposition Method for Support Vector Machines." *Machine Learning*, **46**, 291–314. URL http://www.csie.ntu.edu.tw/~cjlin/papers/decomp.ps.gz.

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.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "kernlab – An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(9), 1–20. URL http://www.jstatsoft.org/v11/i09/.

Kivinen J, Smola A, Williamson R (2004). "Online Learning with Kernels." *IEEE Transactions on Signal Processing*, **52**. URL http://mlg.anu.edu.au/~smola/papers/KivSmoWil03.pdf.

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.

[14] http://CRAN.R-project.org

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.

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.

Lin CJ, More JJ (1999). "Newton's Method for Large-scale Bound Constrained Problems." *SIAM Journal on Optimization*, **9**, 1100–1127.

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.

Meyer D, Leisch F, Hornik K (2003). "The Support Vector Machine under Test." *Neurocomputing*, **55**, 169–186.

Ng AY, Jordan MI, Weiss Y (2001). "On Spectral Clustering: Analysis and an Algorithm." *Neural Information Processing Symposium 2001*. URL http://www.nips.cc/NIPS2001/papers/psgz/AA35.ps.gz.

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.

Roever C, Raabe N, Luebke K, Ligges U (2004). "**klaR** – Classification and Visualization." R package, Version 0.3-3. Available from http://cran.R-project.org.

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.

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.

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.

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.

van der Putten P, de Ruiter M, van Someren M (2000). "CoIL Challenge 2000 Tasks and Results: Predicting and Explaining Caravan Policy Ownership." *Coil Challenge 2000*. URL http://www.liacs.nl/~putten/library/cc2000/.

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.

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.

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.