Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)
March 20–22, Vienna, Austria ISSN 1609-395X
Kurt Hornik, Friedrich Leisch & Achim Zeileis (eds.)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

`kernlab` – A Kernel Methods Package

Alexandros Karatzoglou
Technische Universität Wien

Alex Smola
Australian National University

Achim Zeileis
Technische Universität Wien

Kurt Hornik
Wirtschaftsuniversität Wien

Abstract

Designing software for Support Vector Machines (SVM) and kernel methods in general poses an interesting design problem. Our aim is to provide one possible solution using R object oriented features. `kernlab` is an extensible, object oriented, package for kernel-based learning in R. Its main objective is to provide a tool kit consisting of basic kernel functionality, optimizers and high level kernel algorithms which can be extended by the user in a very modular way and tries to solve some of the design problems posed by this kind of software. Based on this infrastructure kernel-based methods can easily be constructed and developed.

1 Introduction

Support Vector Machines and other kernel methods elevate the notion of data preprocessing to another level. It is often difficult to solve problems like classification, regression and clustering—or more generally: supervised and unsupervised learning—in the space in which the underlying observations have been made. One way out is to project the observations into a higher-dimensional feature space where these problems are easier to solve, e.g., by using simple linear methods. If the methods applied in the feature space are only based on dot or inner products the projection does not have to be carried out explicitly but only implicitly using kernel functions. This is often referred to as the "kernel trick" (Schoelkopf and Smola, 2002).

More precisely, if a projection $\Phi : X \to H$ 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.

Although the kernel theory is old, in the pattern recognition and statistical learning domain it was used for the first time in SVMs (see Vapnik, 1998) which were applied to a wide class of problems such as classification and regression (see e.g. Schoelkopf and Smola, 2002, for a recent survey of applications).

2 Support Vector Machines

The most prominent kernel-based learning algorithm is without doubt the SVM. Like most kernel-based learning algorithms, SVMs work by embedding the data into a high dimensional feature space and then searching for linear relations among the embedded data points. For learning this linear relationship, SVMs use an optimization algorithm implementing a learning bias derived from statistical learning theory (Vapnik, 1998). Different kernels represent different transformations but since all kernel-based algorithms are using the dot products to do computations one can easily change the kernel in a modular fashion in any kernel-based algorithm. Using this trick we can get non-linear variants of any algorithm that can be expressed in terms of dot products. SVMs are becoming increasingly popular in classification (with excellent results in Optical Character Recognition (OCR), text classification and biological sequencing), in regression and novelty detection. They offer stable results which compare well with most state of the art classification and regression methods (Meyer, Leisch, and Hornik, 2003).

2.1 Formulation

The solutions to classification and regression problems sought by kernel-based algorithms such as the SVM are linear functions in the feature space:

$$f(x) = w^\top \Phi(x) \tag{2}$$

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, $w = \sum_{i=1}^{n} \alpha_i \Phi(x)$, implying that f can be written as

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \tag{3}$$

The choice of an appropriate kernel k is very important for a given learning task. We wish to choose an intuitive kernel that induces the "right" metric in the space.

SVMs choose a function that is linear in the feature space by optimizing some criterion over the sample. In the case of the 1-norm Soft Margin classification the optimization problem takes the form:

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

It is obvious that after calculating the kernel matrix the SVM problem simplifies to a standard quadratic optimization problem. Since the calculation of the kernel matrix $K_{ij} = k(x_i, x_j)$ depends only on the choice of the kernel function and the data we can separate this computation step from the rest of the support vector computations easily. The formulation of the SVM optimization problem uses the data only in the form of inner products which are defined by the kernel allowing us to use any kernel function without changing the rest of the algorithm (kernel trick). Since the kernel represent the projection of the data into a high dimensional space this becomes particularly important when one wants to be able to use different projections in order optimize the performance of the SVM. The choice of a kernel and its fine-tuning has a massive effect on the result produced by a SVM and any other kernel method. It is therefore obvious that a modular design of a SVM Software would be desirable if one would like to be able to implement and use the ever increasing number of different kernels and algorithms developed in the field.

The SVM quadratic optimization problem can be solved with different optimization techniques like Sequential Minimal Optimization (SMO) or interior point codes. Since there is only one optimal solution for this sort of optimization problems the result of the SVM does not depend on the choice of the optimization technique. Usually an optimization technique is chosen which is known to perform well, in terms of computation time, in this sort of problem.

2.2 `libsvm` in `e1071`

The `e1071` R package offers an interface to the award winning `libsvm` a very efficient SVM implementation. `libsvm` (Chang and Lin, 2001) 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 function `svm` in `e1071` uses all standard R functionality like object orientation and and formula interface providing the usual interface for model fitting functions in R. It provides the user with a powerful tool for standard regression and classification problems. However, most of the `libsvm` code is in C++ and therefore if one would like to extend the code with e.g. new kernels or different optimizers one would have to program the core C++ code. It is therefore obvious that although this monolithic design provides very fast results it does not allow for the flexibility one would desire for using and testing new kernels and optimization techniques.

3 Kernels

As pointed out in the previous section the main advantage of kernel functions is that they allow us to easily use (dot product based) methods in some high-dimensional feature space, possibly of infinite dimensionality, without having to carry out the projection into that space explicitly. However, the main disadvantage is that especially if the number of observations is large this can still be computationally expensive and burdensome, in particular when the parameters of the kernel function (so-called hyper-parameters) still have to be optimized (tuned).

The idea of kernels in `kernlab` is that they `S4` objects, which extend the class `"function"`, implementing the kernel function $k(x, y)$ and returning a scalar, but possibly having additional information attached which can be used by generic functions performing typical kernel tasks like computing the kernel matrix or the kernel expansion. Additionally, kernel-generating functions for some frequently used classes of kernels are implemented. For example, a Radial Basis Function (RBF) kernel with parameter $\sigma = 0.05$ could be initialized by:

```
R> rbf1 <- rbfdot(sigma = 0.01)
R> rbf1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.01
```

`rbf` is then the kernel function of class `"rbfkernel"` which inherits from `"kernel"`. Every object of class `"kernel"` has to take (at least) two arguments and return the scalar value of their dot product $k(x, y)$:

```
R> x <- 1:3
R> y <- c(1, 1.64, 3.52)
R> rbf1(x, y)

         [,1]
[1,] 0.998002
```

This is all what is necessary if a user wants to plug in his own kernel: a function of class `"kernel"` taking two arguments and returning a scalar. With this function the kernel matrix $K_{ij} = k(x_i, x_j)$ can already be computed, but doing so with a `for` loop in R might be computationally inefficient. Therefore, `kernlab` has a generic function `kernelMatrix` with a default method for `"kernel"` objects doing the `for` loop but also methods for `"rbfkernel"` objects calling faster and memory efficient vectorized code. To make the hyper parameters accessible for these methods, objects of class `"rbfkernel"` (and also all other kernel classes implemented in `kernlab`) have a slot `"kpar"` with a list of parameters which can be accessed via

```
R> gkpar(rbf1)

$sigma
[1] 0.01
```

Often, it is neither advisable nor necessary to compute the full kernel matrix K when what is really of interest is not K itself but the Kernel expansion $K\alpha$ or the quadratic kernel expression matrix with elements $y_i y_j k(x_i, x_j)$. With the same idea as for `kernelMatrix` these tasks can be carried out by generic functions `kernelMult` and `kernelPol` with a (probably inefficient) default method and methods for, e.g., `"rbfkernel"` objects calling more efficient code performing block-wise computations.

The high level algorithms, like SVMs, implemented in `kernlab` rely on the `kernelMult`, `kernelPol` and `kernelMatrix` functions. Therefore, if a user wants to use SVMs with his own kernel all he has to write is a `"kernel"` function, which is usually very simple and should be enough to get a first impression of the performance of the kernel. If he wants to make the computations more efficient he should provide methods for the generic functions mentioned above which can do the computations faster than the default method. `kernlab` provides fast implementations of the following dot products.

- `vanilladot`, this function implements the simplest of all kernel functions namely
  $$k(x, y) = \langle x, y \rangle \tag{5}$$
  Still it may be useful, in particular when dealing with large sparse data vectors x, as is usually the case in text categorization.

- `polydot` implements both homogeneous and inhomogeneous kernels via the following function
  $$k(x, y) = (\text{scale} \cdot \langle x, x' \rangle + \text{offset})^{\text{degree}} \tag{6}$$
  It is mainly used for classification on images such as handwritten digits and pictures of three dimensional objects.

- `rbfdot` implements Gaussian radial basis function via the following function
  $$k(x, y) = \exp(-\sigma \cdot \|x - y\|^2) \tag{7}$$
  It is a general purpose kernel and is typically used when no further prior knowledge is available.

- `tanhdot` this function implements both hyperbolic tangent kernels via the following function
  $$k(x, x') = \tanh(\text{scale} \cdot \langle x, x' \rangle + \text{offset}) \tag{8}$$
  It is mainly used as a proxy for Neural Networks but there is no guarantee that a kernel matrix computed by this kernel will be positive definite.

4 Optimizer

In many kernel-based algorithms, learning (or equivalently statistical estimation) implies the minimization of some risk function. Typically we have to deal with quadratic or general convex problems for SVMs of the type

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n]. \end{array} \tag{9}$$

f and c_i are convex functions and $n \in I\!N$.

`kernlab` provides an implementation of an optimizer of the interior point family (Vanderbei, 1999) which solves the quadratic programming problem :

$$\begin{array}{ll} \text{minimize} & c^\top x + \frac{1}{2} x^\top H x \\ \text{subject to} & b \leq Ax \leq b + r \\ & l \leq x \leq u \end{array} \tag{10}$$

This optimizer can be used in both regression and classification and novelty detection in SVMs and is known to perform fast on SVM optimization problems.

5 Example using SVM

We will briefly demonstrate the use of the SVM algorithms in `kernlab` on a binary classification problem. The data set beeing used (`spam`) is collected at Hewlett-Packard Labs, and 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. We split our data set into a two parts using two thirds for training the SVM and one third for testing.

```
R> data(spam)
R> n <- nrow(spam)
R> testIndex <- sample(1:n, floor(n/3))
R> spamtrain <- spam[-testIndex, ]
R> spamtest <- spam[testIndex, ]
```

Then, we train a C-SVM using the `rbf1` kernel function

```
R> svmmod <- ksvm(type ~ ., data = spamtrain, kernel = rbf1, C = 5)
```

Finally, we test the model on a testset

```
R> type.pred <- predict(svmmod, spamtest[, -58])
```

Comparing the true and predicted type of the e-mails gives the following table

```
R> table(predicted = type.pred, true = spamtest[, 58])
```

```
         true
predicted nonspam spam
  nonspam 902      61
  spam     37     533
```

with a missclassification rate of 0.064.

6 Conclusion

`kernlab` aims at providing a modular tool kit for kernel methods with functionality at three different levels:

- **Basic kernel functionality**: `kernlab` contains a rather general class concept for the dot product (kernel) functions with some generic functionality. Frequently used kernel functions are already implemented along with some methods for typical kernel tasks (see Section 3). But it is very easy for users to add their own kernel function, possibly along with further functionality.
- **Optimization**: Many kernel methods require optimization of the kernel function parameters, there `kernlab` already implements the interior point and will also include sequential minimization optimizers.
- **High-level kernel methods**: These are algorithms such as SVMs Kernel Principal Component Analysis (PCA), and Kernel Canonical Correlation Analysis (KCCA), which use the underlying kernel and optimization routines for their computations. In particular, kernel functions can be easily passed as arguments and hence users can either use an out-of-the-box kernel already implemented in `kernlab` or plug in their own kernel.

Since the package is still in its early development phase many of the higher level algorithms are still not implemented. Eventually, in addition to tools for developing kernel-based methods, `kernlab` will also include many high-level methods ready for out-of-the-box use.

References

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 2003. Forthcoming.

Bernhard Schoelkopf and Alex Smola. *Learning Kernel Methods.* MIT Press, 2002.

Robert Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization methods and Software*, 12:251–484, 1999.

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

Corresponding author

Alexandros Karatzoglou
Institut für Statistik & Wahrscheinlichkeitstheorie
Technische Universität Wien
Wiedner Hauptstraße 8-10/1071
A-1040 Wien
Austria
Tel.: +43/1/58801-10772
Fax: +43/1/58801-10798
E-mail: `Alexandros.Karatzoglou@ci.tuwien.ac.at`