DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

svlab – A Kernel Methods Package

Alexandros Karatzoglou [*] **Alex Smola** [†] **Achim Zeileis** [‡]

Kurt Hornik [§]

Abstract

svlab 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 algorithms such as Support Vector Machines and Kernel Principal Component Analysis which can be extended by the user in a very modular way. Based on this infrastructure kernel-based methods can be easily be constructed and developed.

1 Introduction

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 explicitely but only implicitely using kernel functions. This is often referred to as the "kernel trick". 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

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

[*]Institut für Statistik & Wahrscheinlichkeitstheorie, Technische Universität Wien, Austria
[†]Australian National University, Department of Engineering and RSISE, Australia
[‡]Institut für Statistik & Wahrscheinlichkeitstheorie, Technische Universität Wien, Austria
[§]Institut für Statistik, Wirtschaftsuniversität Wien, Austria

which is computationally simpler than explicitely 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 Support Vector Machines (SVMs, see Vapnik, 1998) which were applied to a wide class of problems such as classification and regression (Bernhard Schoelkopf, 2002). Diffferent kernels represent different transformations but since the 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, Support Vector Machines being the most popular paradigm.

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

Basic kernel functionality: `svlab` 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 2). 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 `svlab` implements the interior point and sequential minimization optimizers.

High-level kernel methods: These are algorithms such as Support Vector Machines and Kernel Principal Component Analysis (PCA), which use the underlying kernel and optimization routines for their computations. In particular, kernel functions can easily passed as arguments and hence users can either use an out-of-the-box kernel already implemented in `svlab` or plug in their own kernel.

2 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 explicitely. 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).

Therefore, the idea of kernels in `svlab` is that they are functions 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> rbf <- rbfdot(sigma = 0.05)
R> rbf

Kernel function of class "rbfkernel" with parameters:
        sigma        = 0.05
```

`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> rbf(x, y)

[1] 0.9801987
```

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, `svlab` 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 C++ code. To make the hyper parameters accessible for these methods objects of class `"rbfkernel"` (and also all other kernel classes implemented in `svlab`) have an addtional attribute `"par"` with a list of parameters:

```
R> attr(rbf, "par")

$sigma
[1] 0.05
```

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 `kenerlPol` with a (probably inefficient) default method and methods for, e.g., `"rbfkernel"` objects calling more efficient C++ code performing block-wise computations based on the ATLAS high performance linear algebra package.

The high level algorithms, like SVMs, implemented in `svlab` 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. `svlab` provides fast implementations of the following dot products.

- `vanilladot`, this function implements the simplest of all kernel functions namely
$$k(x, y) = \langle x, y \rangle \tag{2}$$
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
$$k(x, y) = (\text{scale} \cdot \langle x, x' \rangle + \text{offset})^{\text{degree}} \tag{3}$$
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
$$k(x, y) = exp(-\sigma \cdot \|x - y\|^2) \tag{4}$$
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
$$k(x, x') = \tanh\left(\text{scale} \cdot \langle x, x' \rangle + \text{offset}\right) \tag{5}$$
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.

3 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 Support Vector Machines of the type :

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n]. \end{array} \tag{6}$$

f and c_i are convex functions and $n \in I\!N$.
`svlab` provides an optimizer of the interior point family (LOQO) (Vanderbei, 1999) which solves the quadratic programming problem :

$$\begin{array}{ll} \text{minimize} & c^\top x + \frac{1}{2} x^\top H x \\ \text{subject to} & b \leq Ax \leq b + r \\ & l \leq x \leq u \end{array} \tag{7}$$

This optimizer can be used in both regression and classification and novelty detection in Support Vector Machines and is know to perform fast on Support Vector Machines optimization problems.

4 Support Vector Machines and other kernel methods

Based on the basic kernel functionality from Section 2 and the optimizers from Section 3 `svlab` implements Support Vector Machines for regression and classification in the function `ksvm`. The types of SVMs currently supported are:

- C-Support Vector Classification
- ν-Support Vector Classification
- ϵ-Support Vector Regression
- ν-Support Vector Regression
- One-Class Support Vector Classification

One argument which can be specified for these are the kernels, either by

```
R> ksvm(y ~ x, kernel = rbfdot, kpar = list(sigma = 0.1), ...)
```

or by

```
R> ksvm(y ~ x, kernel = rbf1, ...)
```

Similarly, kernels can be plugged into Principal Component Analysis which is implemented in the function `kpca`.
Both implementations are still work in progress, but the distinctive features will be the modular way in which kernels can easily plugged in. This is also different from the other implementation of SVMs in R—the function `svm` in the package `e1071` (Meyer, 2001; Meyer et al., 2003), which is an interface to the C++ library `libsvm` (Chang and Lin, 2001).

5 Conclusion

`svlab` provides a framework and basic infrastructure for the construction of kernel-based methods in R. Its main purpose is to a rapid development toolkit which enables the user to test existing algorithms and implement new ones in a very short time.

References

Bernhard Schoelkopf, A. S. (2002). *Learning Kernel Methods*. MIT Press.

Chang, C.-C. and Lin, C.-J. (2001). LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Meyer, D. (2001). Support vector machines. *R News*, 1(3):23–26. http://CRAN.R-project.org/doc/Rnews/.

Meyer, D., Leisch, F., and Hornik, K. (2003). The support vector machine under test. *Neurocomputing*. Forthcoming.

Vanderbei, R. (1999). Loqo : An interior point code for quadratic programming. *Optimization methods and Software*, 12:251–484.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley, New York.