Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

SVMBridge: Using SVM Solvers from R

Aydın Demircioğlu
Ruhr-Universität Bochum

Hanna Houphouet
Ruhr-Universität Bochum

Daniel Horn
TU Dortmund

Tobias Glasmachers
Ruhr-Universität Bochum

Bernd Bischl
TU München

Claus Weihs
TU Dortmund

Abstract

Most SVM Solver do not come with a wrapper in R. In case the SVM Solver is written in C++, it can be linked to a package via Rcpp. Although possible, this entails a lot of work to do, and is not possible, if the SVM Solver is written in other languages. Alternatively, one can call the SVM Solver from within R by a system command. SVMBridge eases this calls by providing a framework and ready wrappers for several SVM Solvers like LASVM, SVMperf, LLSVM and BVM/CVM.

Keywords: support vector machines, command line.

1. Introduction

1.1. Support Vector Machines

Kernelized Support Vector Machines (SVM) **?** are binary classifiers that use a kernel k to allow for non-linear classification decisions $x \mapsto \text{sign}(\langle w, \phi(x)\rangle + b)$, where ϕ is the (non-linear) feature map corresponding to the kernel $k(x, x') = \langle \phi(x), \phi(x')\rangle$ in the reproducing kernel Hilbert space $\mathcal{H}$. We use the RBF kernel $k(x, x') = e^{-\gamma||x-x'||^2}$, since it yields excellent performance and enjoys a universal approximation property. The SVM training problem is given by

$$\min_{w\in\mathcal{H}, b\in\mathbb{R}} \quad \frac{1}{2}||w||^2 + C \cdot \sum_{i=1}^{n} \max\Big(0, 1 - y_i\big(\langle w, \varphi(x_i)\rangle_{\mathcal{H}} + b\big)\Big), \tag{1}$$

where $\big\{(x_1, y_1), \ldots, (x_n, y_n)\big\}$ are the labeled training points and $C > 0$ is a regularization parameter that controls the complexity of the predictive model.

The problem, although convex, can be solved by different means, e.g. LIBSVM employs a dual decomposition method, while BSGD is a primal stochastic gradient descent solver.

1.2. SVM packages in R

In general, nearly all SVM solvers come as binary packages, often written in C. Therefore it is not possible to use them directly from within R. Only a few SVM libraries are readily available, includ LIBSVM inside the well-known packages like **kernlab** (via `ksvm()`) (**?**), and **e1071** (via `svm()`) (**?**) as well as other less known ones, e.g. **SwarmSVM** (**?**) and **lasvmR** (**?**). All of these link existing C++ sources directly from within R and need more or less extensive wrappers to allow the user direct access to the options of the underlying SVM solver.

Other SVM solvers, like the BudgetedSVM package (**?**), which contains BSGD and LLSVM, or SVMperf (**?**), cannot be called directly from R. Instead, a system command has to be issued. To ease this calls, the **SVMBridge** package has been developed.

2. The SVMBridge Package

The **SVMBridge** provides a framework to incorporate SVM solvers by means of direct calls (via `system2()` commands). This is different to **e1071** and **kernlab**, which link (and thus copy) existing C++ code directly inside the package. **SVMBridge** works thus differently: Each SVM solver must provide its own socalled *Wrapper*. This is a piece of R code that contains code e.g. to allow the proper calling of the system binary of a SVM solver as well as handling the corresponding models. Such wrapper can be registered in the SVMBridge and henceforth can be used opaquely.

3. General worflow

To use the SVMBridge, one usually perform the following steps: Make the external wrapper incl. software binaries known to the SVMBridge, load the data, train a model, use the model for prediction.

3.1. Wrappers

A wrapper is a S3 object: It must provide routines for assembling the command line call for training as well as testing, reading and writing the model and a few general routines like searching the binaries and printing. We provided default wrappers for LIBSVM, LASVM, CVM/BVM, SVMperf, BSGD and LLSVM.

Make the SVMBridge aware of a new wrapper is simply done by calling `addSVMPackage()`. The SVMBridge comes with an easy mechanism to search the corresponding binaries (specified in the wrapper) to ease the usage. To use this, one can instead call `findSVMSoftware`.

3.2. Data

Most SVM solver work with the LIBSVM sparse data format. This format saves the labels together with all non-zero indicies of every data point, e.g. $(0, 0, 0, 0.2, 0, 1.4)^T$ belonging to class 3, will be encoded as `'3 4:0.2 6:1.4'`.

Although reading these files into R is possible via e.g. the **e1071** package, most only provide pure R solutions, and therefore suffer from suboptimal performance. The SVMBridge package provides a simple sparse data format reading and writing, which are implemented in C++ and therefore can be nearly two orders of magnitudes faster than the corresponding e1071 routines.

3.3. Models

Internally models contain support vectors and their coefficients.

3.4. Training an SVM model

Multiclass is supported, where possible. Note that several SVM packages only support binary problems. In these cases, training a one-vs-all machine is rather easy, if the data is loaded into R first.

Reading a model is possible via the `readModelFromFile ()` function. This will try to detect the format of the file by calling the `isModelFile` routine of each known wrapper. Unluckily, several SVM solver use similar model formats, so that a direct detection is not possible, e.g. CVM/BVM follow the LIBSVM format, but add a comment (with '#')about the running time to the bottom of the model file. Apart from this change, there is no difference. From a practical viewpoint we extended the LIBSVM model reader to cope with this extra line, so that there is no need for an extra CVM/BVM reader. This means that reading a CVM model will make the SVMBridge to detect a LIBSVM model (if the LIBSVM wrapper is loaded). In these cases, if multiple models claims ownership, a "default" model can be provided, which will take precedence over other models. Without a default, the model will be random.

```
R> model = trainSVM (method = "LIBSVM", cost = 1.0, gamma = 2.0,
        trainDataFile = "./data.sparse", epsilon = 0.042, modelFile = modelFile)
```

The training data can be passed on via a variable in memory or by specifying the path of the data set file. Note that there are two sets of variables: Those who belong to the SVMBridge, like the trainDataFile variable or verbose flag, and those that are passed further to the underlying SVM wrapper. Not all options that are provided by the underlying SVM solver are supported by the wrappers. As our focus lies on binary classification, because of time constraints we opted to support only these options, e.g. we dropped the one-class SVDD in LIBSVM. In case there is need for other options, it is easy to enhance the wrappers. There is furthermore the subsampling option, which can be used for larger data sets.

3.5. Testing a model

Testing (or doing predictions) needs a trained SVM model as well as test data to work on. As with training, testSVM will accept data and models either from memory or from a file.

```
R> testObj = testSVM (model, testDataFile = './australian_scale',
        readPredictions = TRUE)
```

The returned object will contain meta information like testing times as well as the predictions.

3.6. Other Considerations

Although the SVMBridge was meant to be cross-platform, this goal is hard to achieve. From a users perspective, it can be used on all three major platforms (Linux, MacOS, Windows). The tests should be executed to make sure the package works as intended.

We sum up the points to keep in mind when performance is of high priority: Do not use the automatism to find the binaries, specify the paths by hand. Do not load the data into memory, specify the path of the data when training instead. Do not re-read the model into memory, let it on disk.

4. Creating a Wrapper

Adding your own SVM solver to the SVMBridge boils down to writing a S3 class with several routines. In the inst folder you will find a template that you can fill with your own code. Here we will go through the details of adding the software package ...

The flow is as follows: The SVMBridge will call createTrainingArguments method of any wrapper. The wrapper will return a string that contains all parameters that need to be passed to the binary of the underlying SVM solver. Note that this includes the model, prediction and training files.

5. Conclusion

We provided a simple framework, **SVMBridge**, to attach binary SVM solvers easily to the R landscape. The **SVMBridge** does this by an easily extensible wrappers that manage the exchange of data and models via command line.

Acknowledgments

We acknowledge support by the Mercator R.search Center R.hr, under grant Pr-2013-0015 *Support-Vektor-Maschinen für extrem große Datenmengen* and partial support by the German R.search Foundation (DFG) within the Collaborative R.search Centers SFB 823 *Statistical modelling of nonlinear dynamic processes*, Project C2.

Affiliation:

Aydın Demircioğlu, Hanna Houphouet, Tobias Glasmachers
Institut für Neuroinformatik
Ruhr-Universität Bochum
44790 Bochum
Germany
E-mail: {aydin.demircioglu, hanna.houphouet, tobias.glasmachers}@ini.rub.de
URL: http://www.ini.rub.de

Daniel Horn, Claus Weihs
Fakultät Statistik
Technische Universität Dortmund
44221 Dortmund
Germany
E-mail: {dhorn, bischl, weihs}@statistik.tu-dortmund.de
URL: https://www.statistik.tu-dortmund.de/computationalstats.html

Bernd Bischl
Institut für Statistik
Ludwig-Maximilians-Universität München
80539 München
Germany
E-mail: bernd.bischl@stat.uni-muenchen.de
URL: http://www.statistik.lmu.de/~bischl

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd