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

Support Vector machines in general.

1.2. SVM packages in R

There are only a few SVM libraries readily available from within R. This includes the kernlab, the e1071 and the lasvmR packages. All of these link existing C++ sources directly from R and need more or less extensive wrappers to allow the user direct access to the options of the underlying SVM solver.

Other SVM solver, 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

2.1. Wrapper

A wrapper consists of several routines: 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.

Note that the search for binaries might have a huge toll on HPC clusters, so it is adviced to specify the paths directly instead of relying on the automatism.

3. Examples

To use the SVMBridge, one usually perform the following steps: Add the external wrapper to the SVMBridge, load the data, train a model, use the model for prediction.

Notice that the data must be written by the SVMBridge before calling the corresponding SVM Solver, as there is no way to pass data via memory to command line tools. Therefore

3.1. Adding the Wrapper to the SVMBridge

3.2. Reading Sparse Data

Most SVM solver work with the sparse data format. This format consists of: Each line (delimited by a CR/LF) is given by a label and the non-zero components of the data point, e.g. to encode the vector six dimensional vector 0000.201.4 belonging to class 3, the sparse data format would contain the line 3 4:0.2 6:1.4.

Although reading these files into R is possible via either the kernlab or the e1071 routines, both only provide 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 are nearly two orders of magnitudes faster than the corresponding e1071 routines. Notice, that currently only dense matrices are supported, TODO: work around this with the Sparse Matrix Package.

3.3. Training a 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.

3.4. Predicions

Predicting from a trained model is rather easy, as there are usually not many options. As with training, the predict routine will accept a model either in memory or from a file.

3.5. Optimization Values

At times, some of the training values are of interest, e.g. when comparing different solvers, the primal value of the SVM problem as well as the dual value might be of interest. These can be computed via the `optimizationValues()` routine.

3.6. Helper Routines

...

3.7. Performance Considerations

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. Conclusion

We provided a simple framework to attach many SVM solvers to the R Subsystem. This allows for a systematic call of them. By providing a wrapper, linking any SVM solver to R become much easier.

Acknowledgments

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

Affiliation:

Aydın Demircioğlu
Institut für Neuroinformatik
Ruhr-Universität Bochum
44790 Bochum E-mail: aydin.demircioglu@ini.rub.de
URL: http://www.ini.rub.de/PEOPLE/aydind/

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