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 (SVM) are linear classifiers that try to maxime the margin of a binary problem (Cortes and Vapnik 1995). As SVMs are only linear classifiers, a kernelized version is used for more complex data. Basically, they work by solving the following convex problem:

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

Here $\mathcal{D} = \{(x_1,y_1),\ldots,(x_n,y_n)\} \in (X\times\{\pm 1\})^n$ is the labeled data, and $\varphi : X \to \mathcal{H}$ is a feature map into a reproducing kernel Hilbert space $\mathcal{H}$, corresponding to a positive definite (Mercer) kernel function $k : X \times X \to \mathbb{R}$, fulfilling $\langle\varphi(x),\varphi(x')\rangle_{\mathcal{H}} = k(x,x')$ for all $x, x' \in X$.

Furthermore, $C > 0$ is a regularization parameter. It controls the complexity of the SVM model.

Often, the problem is not directly solved, but dualized via Lagrangian theory. Then the problem can be stated as

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{s.t.} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C \;\; \forall\, i \in \{1, \ldots, n\}. \end{aligned} \tag{2}$$

The solution then takes the form $w = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i)$ and the offset b can be computed from the Karush-Kuhn-Tucker (KKT) complementarity conditions.

SVMs classify by using the model given by $h_{w,b}(x) = \text{sign}(\langle w, \phi(x) \rangle_{\mathcal{H}} + b)$

Different kernels can be used, e.g. the polynomial kernel or the RBF kernel $k(x, x') = e^{-\gamma ||x - x'||^2}$, where γ and d are parameters of the kernel, that must be tuned to the given data.

SMO.

On the practical side, LIBSVM can be regarded as the reference implementation of the SMO and the 'golden standard' of SVM solvers (Chang and Lin 2011).

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 (Djuric *et al.* 2014), which contains BSGD and LLSVM, or SVMperf (Joachims and Yu 2009), 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.

We provided default wrappers for LIBSVM, LASVM, CVM/BVM, SVMperf, BSGD and LLSVM.

3. Using a Wrapper

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

The first step is to make the SVMBridge aware of a new wrapper. This is simply done by calling `addSVMPackage()`. There are several options. 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. 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 $(0, 0, 0, 0.2, 0, 1.4)$ 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.

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

```
                    useBias = TRUE,
                    verbose = verbose
                )
```

...

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.

3.4. 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. The same is true for the data to predict. Again notice that data and models in memory must be written to disk prior to calling the prediction binary.

```
R> predictions = testSVM (model, data = '')
R> head(predictions)
```

...

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.

Specifically, this routine compute the following values: `weight` is given by $\frac{1}{2}w^T w$, with $w = \sum_i \alpha_i s_i$. Furthermore, we have that $C \sum_i max(0, 1-..)$, and `primal` is then given by $primal = w + CHinge$. `dual` on the other hand is simply $w - \sum_i |\alpha_i|$. In general it must hold that $primal \geq dual$. WHAT IF NOT? what does it mean?

3.6. Helper Routines

A few routines have been placed into the package that were helpful with testing. Namely, there are several generators for synthetical data sets, collected from different publications. WHAT ELSE? can routines be used outside? detectTilde e.g.?

3.7. Platform 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). Additional tests should be started to make sure the package works as intended.

3.8. 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. 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.

4.1. Training Call

A simple S3 method has to be created that needs to be called `createTrainingArguments.LIBSVM`. Apart from the very first argument (taking the object itself), all other paramters can be chosen freely. After training, from the output several training information will be extracted by the `extractTrainingInfo()` function.

4.2. Testing Call

Similar to the training call, `createTestArguments()` needs to writen. Again, it should accept three variables called testDataFile and modelFile as well as predictionFile. There is also a `extractTestInfo()` that will extract the important informations from the output of the test binary.

4.3. Handling Models

As only LIBSVM models are being used, one can readily use the inbuild functions of the SVMBridge. If the solver works with its own model, one needs to rewrite the readModel method of the wrapper. This will only obtain the modelFile and will return the model in the model format (S3 Object with XY, describe it somewhere else) SVMBridge needs also a writeModel functions, as models passed in memory needs to be written to disk. . Finally it is necessary to access the predictions itself. For this one needs to write the `readPredictions()` function. For LIBSVM this will read the prediction files that is being written by LIBSVM to disk.

4.4. Other functions

Every wrapper should also be able to find tiself in a given directory tree. For this `findSoftware` is provided.

At last, for querying reasons, there is also a print routine so that wrapper can print out any

further information about itself.

5. 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.

References

Chang CC, Lin CJ (2011). "LIBSVM: A library for support vector machines." *ACM Transactions on Intelligent Systems and Technology*, **2**, 27:1–27:27.

Cortes C, Vapnik V (1995). "Support vector machine." *Machine learning*, **20**(3), 273–297.

Djuric N, Lan L, Vucetic S, Wang Z (2014). "BudgetedSVM: A Toolbox for Scalable SVM Approximations." *Journal of Machine Learning Research*, **14**, 3813–3817.

Joachims T, Yu CNJ (2009). "Sparse kernel SVMs via cutting-plane training." *Machine Learning*, **76**(2-3), 179–193.

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