SHOGUN - User manual

June 16, 2006

1 About this document

This is the user manual for the SHOGUN toolbox.

2 Introduction to SHOGUN

The SHOGUN machine learning toolbox's focus is on kernel methods and especially on Support Vector Machines (SVM). It provides a generic SVM object interfacing to several different SVM implementations, among them the state of the art LibSVM[1] and SVMlight[2]. Each of the SVMs can be combined with a variety of kernels. The toolbox not only provides efficient implementations of the most common kernels, like the Linear, Polynomial, Gaussian and Sigmoid Kernel but also comes with a number of recent string kernels as e.g. the Locality Improved[3], Fischer[4], TOP[5], Spectrum[6], Weighted Degree Kernel (with shifts)[7]. For the latter the efficient LINADD[8] optimizations are implemented. Also SHOGUN offers the freedom of working with custom pre-computed kernels. One of its key features is the "combined kernel" which can be constructed by a weighted linear combination of a number of sub-kernels, each of which not necessarily working on the same domain. An optimal sub-kernel weighting can be learned using Multiple Kernel Learning[9]. Currently SVM 2-class classification and regression problems can be dealt with. However SHOGUN also implements a number of linear methods like Linear Discriminant Analysis (LDA), Linear Programming Machine (LPM), (Kernel) Perceptrons and features algorithms to train hidden markov models. The input feature-objects can be dense, sparse or strings and of type int/short/double/char and can be converted into different feature types. Chains of "preprocessors" (e.g. substracting the mean) can be attached to each feature object allowing for on-the-fly pre-processing. License: GPL version 2 or newer URL: http://www.fml.tuebingen.mpg.de/raetsch/projects/shogun

2.1 Notes

As mentioned above SHOGUN interfaces to several programming languages and toolkits such as Matlab(tm), R, Python, Octave. The following sections shall give you an overview over the commands of SHOGUN. We tried to preserve the syntax of the commands in a consistent manner through all the different languages. However where it was not possible we give there will be special infos on the subtle differences of syntax and semantic in the respective toolkit

2.2 Commands

To fire up SHOGUN in R make sure that you have SHOGUN correctly installed in R. You can check this by typing (let ">" be the R prompt):

```
> library()
```

inside of R, this command should list all R packages that have been installed on your system. You should have an entry like:

```
sg                      The shogun package provides Hidden markov model
                        and Support Vector Machine algorithms for gene
                        finding and other tasks.
```

After you made sure that SHOGUN is installed correctly you can start it via:

```
> library("sg")
```

you will see some informations of the SHOGUN core. After this command R and SHOGUN are ready to receive your commands.

In general all commands in SHOGUN are issued using the function sg(...). To invoke the SHOGUN command help one types:

```
> sg("send_command","help")
```

and then a help text appears giving a short description of all commands.

2.2.1 send_command

Currently send_command supports the following options:

1. help: Gives you a help text.
2. loglevel: has the additional options ALL, WARN, ERROR.
   i. ALL: verbose logging output (useful for debugging).
   ii. WARN: less logging output (useful for error search).
   iii. ERROR: only logging output on critical errors.

   For example:

   ```
   sg("send_command","loglevel ALL")
   ```

   gives you a very verbose description of all the things happening in the SHGOGUN core.
3. clean_features: Deletes the features which we assigned before in the actual SHOGUN session.
4. clean_kernels: Deletes the kernels which we assigned before in the actual SHOGUN session.
5. new_svm: Creates a new SVM instance.
6. init_kernel: Initializes the kernel
7. svm_train: Starts the training of the SVM on the assigned features and kernels.

2.2.2 set_features and set_labels

The set_features commands is responsible for assigning features to the SHOGUN core. Suppose you have a matlab matrix or R vector "traindat" which contains your training data and you want to register this data, you simply type:

```
sg("set_features", "TRAIN", traindat)
```

telling SHOGUN that this is the data you want to train your classifier on. To register test data on issues:

```
sg("set_features", "TEST", testdat)
```

where testdat is a datastructure with your test data. T One proceeds similar when assigning labels to the training data. The command:

```
sg("set_labels", "TRAIN", trainlab)
```

tells SHOGUN that the labels of the assigned training data reside in trainlab.

2.2.3 get_kernel_matrix

The purpose of the get_kernel_matrix commands is to return a symmetric matrix representing the kernel matrix for the actual classification problem.

After typing:

```
km=sg("get_kernel_matrix")
```

km refers to a matrix object.

2.2.4 get_svm

The get_svm command returns some properties of an SVM such as the Langrange multipliers alpha, the bias b and the index of the support vectors SV (zero based).

For several reasons this commands differs a bit in the different target languages.

In R:

```
**svmAsList=sg("get_svm")**
```

returns a vector with the named entities alphas,b and SV. In Matlab:

```
**[b, alphas]=sg('get_svm');**
```

return the b and the alphas only.

2.2.5 svm_classify

The result of the classification of the test samples is obtained via:

```
out=sg("classify")
```

where out is a vector containing the classification result for each datapoint.

2.2.6 set_kernel

In general one sets a kernel by typing:

```
sg("send_command", "set_kernel NAME ..." )
```

where NAME is the name of the kernel one wishes to use and ... are additional options for the respective kernel.

Please refer to section 1.3 for detailed information on kernels.

2.3 Kernels

The following kernels are implemented in SHOGUN:

1. Linear
2. Polynomial
3. Gaussian
4. Sigmoid

2.3.1 Linear Kernel

A linear kernel is created via:

```
sg("send_command", "add_kernel LINEAR TYPE CACHESIZE");
```

For example:

```
sg('send_command', 'add_kernel LINEAR REAL 50')
```

creates a linear kernel of cache size 50 for real datavalues.

Available types for the linear kernel: BYTE, WORD CHAR, REAL, SPARSEREAL.

2.3.2 Polynomial Kernel

A polynomial kernel is created via:

```
('send_command', 'add_kernel POLY TYPE CACHESIZE DEGREE INHOMOGENE NORMALIZE');
```

For example:

```
('send_command', 'add_kernel POLY REAL 50 3 0')
```

creates a polynomial kernel. Available types for the polynomial kernel: REAL, CHAR, SPARSEREAL.

2.3.3 Gaussian Kernel

To work with a gaussian kernel on real values one issues:

```
sg("send_command", "set_kernel GAUSSIAN TYPE CACHESIZE SIGMA")
```

For example:

```
sg("send_command", "set_kernel GAUSSIAN REAL 40 1")
```

creates a gaussian kernel on real values with a cache size of 40MB and a sigma value of one. Available types for the gaussian kernel: REAL, SPARSEREAL.

2.3.4 Sigmoid Kernel

To work with a sigmoid kernel on real values one issues:

```
sg("send_command", "set_kernel SIGMOID TYPE CACHESIZE GAMMA COEFF")
```

For example:

```
sg("send_command", "set_kernel SIGMOID REAL 40 0.1 0.1")
```

creates a sigmoid kernel on real values with a cache size of 40MB, a gamma value of 0.1 and a coefficient of 0.1.

Available types for the gaussian kernel: REAL.

2.4 Let's get started

Equipped with the above information on the basic SHOGUN commands you are now able to create your own SHOGUN applications.

Let us discuss an example:

```
require(graphics)
require(lattice)
library("sg")

meshgrid <- function(a,b) {
   list(
         x=outer(b*0,a,FUN="+"),
         y=outer(b,a*0,FUN="+")
      )
}

dims=2;
num=100;

traindat <- matrix(c(rnorm(dims*num)-0.5,rnorm(dims*num)+0.5),dims,2*num)
trainlab <- c(vector(mode="numeric", num)-1, vector(mode="numeric", num)+1)
```

```
sg("send_command","loglevel ALL")
sg("set_features", "TRAIN", traindat)
sg("set_labels", "TRAIN", trainlab)
sg("send_command", "set_kernel GAUSSIAN REAL 40 1")
sg("send_command", "init_kernel TRAIN")
sg("send_command", "new_svm LIGHT")
sg("send_command", "c 10.0")
sg("send_command", "svm_train")

x1=(-49:+50)/10
x2=(-49:+50)/10
testdat=meshgrid(x1,x2)
testdat=t(matrix(c(testdat$x, testdat$y),10000,2))

sg("set_features", "TEST", testdat)
sg("send_command", "init_kernel TEST")
out=sg("svm_classify")

z=matrix(out, 100,100)

image(x1,x2,z,col=topo.colors(1000))
contour(x1,x2,z,add=T)
i=which(trainlab==+1);
matplot(traindat[1,i],traindat[2,i],cex=2.0,pch="o", type = "p", col="red",add=T)
i=which(trainlab==-1);
matplot(traindat[1,i],traindat[2,i],cex=2.0,pch="x", type = "p", col="black",add=T)

wireframe(z, shade = TRUE, aspect = c(61/87, 0.4), light.source = c(10,0,10))
```

This small example has a few SHOGUN commands which will be discussed now.

1. **sg("send_command","loglevel ALL")** sets an extra verbose loglevel.

2. **sg("set_features", "TRAIN", traindat)** registers the training samples which reside in traindat.

3. **sg("set_labels", "TRAIN", trainlab)** registers the training labels.

4. sg("send_command", "set_kernel GAUSSIAN REAL 40 1") creates a new gaussian kernel for reals with cache size 40Mb and width = 2**sigma^2.

5. **sg("send_command", "init_kernel TRAIN")** attaches the data to the kernel and does some initialization.

6. **sg("send_command", "new_svm LIGHT")** creates a new SVM object inside the SHOGUN core.

7. **sg("send_command", "c 10.0")** sets the C value of the new SVM to 10.0.

8. **sg("send_command", "svm_train")** starts the training on the samples.

9. **sg("set_features", "TEST", testdat)** registers the test samples

10. **sg("send_command", "init_kernel TEST")** attaches the data to the kernel.

11. **out=sg("svm_classify")** gives you the classification result as a vector.