# User Manual of the Potential Support Vector Machine

Tilman Knebel and Sepp Hochreiter
Department of Electrical Engineering and Computer Science
Technische Universität Berlin
10587 Berlin, Germany
{tk,hochreit}@cs.tu-berlin.de

## Contents

**Abstract**

This is the user manual for the PSVM (see [ncpsvm]) software library which is designed for MICROSOFT Windows as well as for UNIX systems. Compiling the software results in a program which can be used with command line options (e.g. kernel type, learning/testing, etc.) which does not depend on other software or on a particular software-environment. The PSVM software package also includes a MATLAB interface for convenient working with the PSVM package. In addition lots of sample data and scripts are included. The PSVM software contains a classification, a regression, and a feature selection mode and is based on an efficient SMO optimization technique. The software can directly be applied to dyadic (matrix) data sets or it can be used with kernels like standard SVM software. In contrast to standard SVMs the kernel function does not have to be positive definite, e.g. the software already implements the indefinite sin-kernel. An important feature of the software is that is allows for $n$-fold cross validation and for hyperparameter selection. For classification tasks it offers the determination of the significance level and ROC data. In summary the basic features of the software are

- WINDOWS and UNIX compatible
- no dependencies to other software
- command line interface
- MATLAB interface
- $n$-fold cross validation
- hyperparameter selection
- relational data
- non-Mercer kernels
- significance testing
- computation of Receiver-Oprator-Characteristic (ROC) curves

If you use the software please cite [ncpsvm] in your publications.

# 1 Installation for Windows

## 1.1 Download and Deflating the Zipfile

Download the file `psvm.zip` and unzip the file with your compression software. A free compression utility FreeZip is on the web at
`http://www.pcworld.com/downloads/file_description/0,fid,6383,00.asp`
and can be used for unzipping the P-SVM software by the command:
`unzip psvm.zip`

The archive contains source files of the command line application and additional folders with MATLAB functions and samples. Windows executables are in the subfolder `windows`. The PSVM can be started as follows

- open a command window
- change into the subfolder `windows`
- type `psvm`

or within MATLAB:

- start MATLAB
- change into the subfolder `matlabdemo`
- type `startup`
- available commands: `psvmtrain, psvmpredict,` and `psvmgetnp, psvmputnp, psvmkernel, psvmsmo`

## 1.2 Requirements

To produce the executables from the sources, a C++-compiler with standard libraries is required. The software produces output for visualizing graphs of hyperparameter selection, ROC, and significance testing. If you want to use them, 'gnuplot' is required. The gnuplot webpage is at `http://www.gnuplot.info/download.html`. Select and install the latest release file appended with "W32". Connect the filename extension ".plt" with the gnuplot application to view the graphs by clicking at the corresponding script files.

## 1.3 Compilation of the Command Line Application

The needed source files for the command line application are:

```
psvm.cpp
psvm.h
psvmparse.cpp
psvmparse.h
```

The precompiler option `-D float` can be used to reduce the internal precision from double to float. This saves memory costs, but may increase computational time. Another flag is for switching the file handling: If `C_TXT` is defined by the precompiler option `-D C_TXT`, all input and output files were extended with '.txt'.

### 1.4 Compilation of the MATLAB Functions

The sources for MATLAB functions are in the subfolder 'matlabsrc'. MATLAB offers its own compiler interface called 'mex', which refers to a MATLAB internal compiler or to an external compiler. The internal compiler can not be used for psvm sourcecodes because the Standard Template Library (STL) is missing. Type `mex -setup`, select your external compiler and compile the sourcefiles by:

```
mex psvmtrain.cpp psvm.cpp
mex psvmpredict.cpp psvm.cpp
mex psvmkernel.cpp psvm.cpp
mex psvmgetnp.cpp psvm.cpp
mex psvmputnp.cpp psvm.cpp
mex psvmsmo.cpp psvm.cpp
```

The successfully compiled files are dynamic link libraries (DLL) and can be used in the same way like normal `.m` matlab script files. The help functionality within MATLAB for the compiled functions is offered by corresponding `.m` files, which are also included in the folder with the sources.

For Microsoft Visual C++, use following compiler options:

```
/I "<MATLAB-Dir>/extern/include"
/D "MATLAB_MEX_FILE"
/D "_USRDLL"
```

Linker options for MS Visual C++:

```
/DLL /LIBPATH: "<MATLAB-Dir>/extern/lib/win32/microsoft/msvc60"
libmatlb.lib libmat.lib libmex.lib libmx.lib
```

Occurrences of `<MATLAB-dir>` have to be replaced by your matlab directory.

## 2 Installation for Unix based systems

### 2.1 Download and Deflating the Tarfile

Download the file `psvm.tar.gz` and decompress the file by entering the following commands:

```
gunzip psvm.tar
tar -xf psvm.tar
```

The archive contains source files of the command line application and additional folders with MATLAB functions and samples. After deflating, change into the main psvm folder by `cd psvm`. The PSVM can be started after compilation as follows:

- type `psvm`

or within MATLAB:

- start MATLAB
- change into the subfolder `matlabdemo`
- type `startup`
- available commands: `psvmtrain, psvmpredict,` and `psvmgetnp, psvmputnp, psvmkernel, psvmsmo`

## 2.2 Requirements

To install the basic software only a C++-compiler with standard libraries is required. The binaries use shared libraries, so the system searches at runtime for the shared objects files libg++ and libstdg++. If an error occurs after starting the application, search the folder containing the two libraries (maybe /usr/gnu/lib) and set up an environment variable as follows:

```
LD_LIBRARY_PATH=/usr/gnu/lib:LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

The software produces output for visualizing graphs of hyperparameter selection, ROC, and significance testing. If you want to use them, 'gnuplot' is required. The gnuplot webpage offers sources and binaries for many platforms at: `http://www.gnuplot.info/download.html`

## 2.3 Compilation of the Command Line Application

Next, enter
`make`, which generates the main program `psvm`. The internal precision is of type double, that means the mantissa stores 15 decimal digits.

`make float`: generates the main program `psvm`, but the internal precision is of type float (7 decimal digits). The advantage is a reduction of needed memory but the computational time may increase.

`make psvm`: like `make` but does not compile already compiled object files. This is useful if the code of only one source file has been changed.

For experts:
The compilation command within the makefile is

```
g++ -O3 psvm.cpp psvmparse.cpp -o psvm
```

The precompiler option `-D smofloat` can be used to reduce the internal precision from double to float which is done within `make float`. Another flag is for switching the file handling: If `C_TXT` is defined by the precompiler option `-D C_TXT`, all input and output files were extended with '.txt'.

## 2.4 Compilation of the MATLAB Functions

The sources for MATLAB functions are in the subfolder 'matlabsrc'. Change into it by
`cd matlabsrc`

MATLAB offers its own compiler interface called 'mex', which is used by the makefile. Run the makefile by
`make`
Because the default mex configuration uses gcc which does not work for psvm, the package contains modified configuration files, where 'gcc' is simply replaced with 'g++'. So if 'make' does not work, please try
`make all12`,
`make all13`, or
`make all14`
depending on your MATLAB version. For other versions, search for the MATLAB internal file `/matlab/bin/gccopts.sh`, replace all occurrences of 'gcc' with 'g++' and save it in the psvm folder as `/psvm/matlabsrc/g++opts65.sh` and `/psvm/libsvm/g++opts65.sh`.

The successfully compiled files have a machine dependent extension like `.mexsol` (UNIX), `.mexglx` (LINUX) or `.dll` (WINDOWS) and can be used like normal `.m` matlab script files. The help functionality within MATLAB for the compiled functions is offered by corresponding `.m` files, which are also included in the folder with the sources.

For experts:
The compilation of the matlab sources using g++ without mex is achieved with following compilation parameters, where the phrase 'sol2' is machine dependent:

```
g++ psvmtrain.cpp psvm.cpp <MATLAB-dir>/extern/src/mexversion.c
 -o psvmtrain.mexsol
 -I<MATLAB-dir>/extern/include -DMATLAB_MEX_FILE -fPIC -shared
 -Wl,-L<MATLAB-dir>/bin/sol2,-L<MATLAB-dir>/extern/lib/sol2,
 -lmex,-lmx,-lmat,-M<MATLAB-dir>/extern/lib/sol2/mexFunction.map,
```

This is a sample for psvmtrain, the other sources can be compiled in the same way. Replace `<MATLAB-dir>` with your matlab directory. Watch that no spaces occur behind option Wl.

# 3 License


If you use the software please cite [ncpsvm] in your publications.

# 4 Manual

## 4.1 Command Line Application

Change to the folder where the compiled 'psvm' is and type `psvm`. Without any option, the program gives a list of possible commands. The software uses and expects numerical data in plain text ASCII format for input and output files. The first dimension is delimited by *newline*, the second by spaces. For vectorial data this means that the delimiter for vectors is *newline* and its components are separated by spaces. For dyadic (matrix) data, this means that the delimiter for data objects is *newline* and the descriptions within the data objects are separated by spaces. All filenames were extended as follows:

| | |
|---|---|
| .data | - input: relation (kernel) matrix $\boldsymbol{K}$ or vectorial data $\boldsymbol{X}$ |
| .y | - input: vector $\boldsymbol{y}$; real valued (regression) or $+1/-1$ (classification) |
| .yout | - output: predicted (output) vector $\boldsymbol{o}$ |
| .zdata | - input: complex feature vector matrix $\boldsymbol{Z}$ (optional) |
| .model | - output,input: model data in plain text |
| .fi | - output: feature vector indices |
| .fv | - output: feature vectors |
| .fr | - output: feature ranking |
| .msdata | - output: error distribution over parameter variation |
| .msclass.plt | - output: gnuplot script for hyperparameter selection visualization |
| .msregress.plt | - output: gnuplot script for hyperparameter selection visualization |
| .sigdata | - output: errors for training with randomly permutated labels (for -sig) |
| .sigclass.plt | - output: gnuplot script for significance visualization |
| .sigclass1.plt | - output: gnuplot script for another significance visualization |
| .sigregress.plt | - output: gnuplot script for significance visualization |
| .sigregress.plt | - output: gnuplot script for another significance visualization |
| .rocdata | - output: roc points |
| .roctest.plt | - output gnuplot script for roc visualization |

The extensions are to be extended itself by "`.txt`" if the application is compiled with the preprocessor definition "`C_TXT`".

Program options:

-train [name]

**training dataset [name]**
The program searches for a file [name].data and loads it. The program expects *newline* delimited vectors, which components are delimited by spaces. If the loaded data is a dyadic kernel matrix, use the option *-isrelation* additionally. Otherwise, the program assumes vectorial data. For vectorial data, the program searches for an optional file [name].zdata and uses it if found as complex feature vectors. The $\boldsymbol{y}$-vector is expected in [name].y as space separated numerical data. For classification, the class labels are $+1$ or $-1$ and for regression real values are expected. The program constructs a model which can be applied to the training data or a new data set.

-trainr [name]

**regression training dataset [name]**
This command works in the same way as *-train* but supresses messages concerning classification results. If hyperparameter selection is used, it chooses the best hyperparameter with respect to regression performance.

-trainc [name]

**classification training dataset [name]**
This command works in the same way as *-train* but supresses messages concerning regression results. If hyperparameter selection is used, it chooses the best hyperparameter with respect to classification performance.

-predictc [name]

**saves predicted class labels for dataset [name]**
The program searches for a file [name].data and loads it. Expected are files with *newline* delimited numerical vectors, which elements are space delimited. With a model generated by *-train* or loaded by *-model*, the program predicts binary labels and saves them into [name].yout

-predict [name]

**saves real valued predictions for dataset [name]**
This option equals *-predictc*, but the predicted targets are real valued.

-test [name]

**gets model error for dataset dataset [name]**
The program searches for files [name].data and [name].y and loads them. Expected is a complete dataset as described in *-train* option. With a model generated by *-train* or loaded by *-model*, the program predicts real values or class labels, compares them with the loaded $\boldsymbol{y}$-vector and gives a classification error rate and a mean squared error.

-model [name]

**load/save model**
If *-train* is used, the generated model will be saved in [name].model, otherwise a model is loaded from [name].model.

-fi [name]

**saves detected support feature indices**
This option extracts features from a trained or loaded model into a file [name].fi. The features are numbered in the same sequence as in the data file used for training beginning with "1". The features can also be retrieved from the model file generated by -model.

-fv [name]

**saves detected support feature vectors**
This option extracts support feature vectors from a trained or loaded model row by row into a file [name].fv. The feature vectors can also be retrieved from the model file generated by -model.

-frank [name]

**saves detected feature ranking**
This option extracts a feature ranking from a trained or loaded model into a file [name].fr. The features are numbered in the same sequence as in the data file used for training beginning with "1". The software sorts the features by their numerical influence to the predictor and prints a reference to the corresponding column of the datafile followed by the value of the corresponding Lagrangian parameter, which is the argument for sorting. The feature ranking can also be retrieved from the model file generated by -model.

-stats [name]

**shows statistics for dataset [name]**
The program tries to load the files [name].data and [name].y and optional [name].zdata. Then it computes all scalar products and euclidean distances for all pairs of input vectors and shows minima, maxima, means and variances. The results can help determining the optimal kernel parameters. A second calculation determines the upper limit for epsilon, which leads to one support vector at least. If epsilon is above the limit, no support vectors or features can be found. Note that the value depends on the kernel parameters.

-crossval [n]

**$n$-fold cross validation mode, only with -train**
The dataset given in -train will be split into $n$ parts, and a model is generated for all possible selections of $(n-1)$ parts. The remaining parts will be used for testing. Result is a classification error rate and a mean squared error for real valued regression data. If $n$ is much smaller than the number of data objects, a randomly shuffling of the data may be useful (option *-shuffle*). For $n = -1$ the program uses leave-one-out crosscvalidation.

-shuffle [s]

**randomly shuffeling the data objects**
If $s = 0$ no shuffling is done, if $s = 1$ all input data is randomly shuffled in a static way and with $s = 2$ the shuffling is initialized with timeseed. Shuffling is useful in combination with crossvalidation to assert a training based on inhomogenious subsets of the whole training dataset.

-c [c1][c2 dc]

**set the parameter $C$ (default 1000)**
If $C$ is high, the model is more exact, if $C$ is low, the constraints are weakened. A high value is recommended for feature extraction. $C$ is an upper bound for $\alpha$ generated by the SMO. This implies that $C$ should remain below a limit, which depends on the training data, epsilon, and the (un)used kernel. Amounts above this limit behave all equally. The optional parameters $c2, dc$ are for doing a hyperparameter selection with range *c1* to *c2* with constant stepsize *dc*.

-e [epsilon][e2 de]

**set the epsilon in loss function (default 0.01)**
Epsilon corresponds to the expected variance of noise in the prediction error. If it is low, the model is more precise. The upper limit of values which influence the predictor can be found by *-stats*. Epsilon is set to this value with the option *-e -1*. The optional parameters are for doing a hyperparameter selection with range *epsilon* to *e2* with constant stepsize *de*.

-g [epsitol]

**set tolerance of epsilon (default 0.05)**
This controls the termination of the SMO, the computational cost increases if epsitol decreases. If epsitol equals zero, the training may never end. The internal optimization of $\alpha$ is stopped, if its corresponding primal constraint is fulfilled for epsilon $\pm$ epsilon·epsitol.

-anneal [a b c d]

**Heuristic1: $\epsilon$-annealing**
If $a = 0$ annealing is not used. If $a = 1$ the calculation starts with a high epsilon which can be relatively controlled with $b$, and decreases with factor $c$. A new decrease step starts, if the termination criterion is fulfilled with factor $d$. So high values in $b$ and low values in $c$ relative to one lead to more decrease steps. The precision within the decrease steps is weakened with a high value of $d$ relative to one.

-block [a b c d]

**Heuristic2: blockoptimization**
If $a = 0$ block optimization is not used. If $a = 1$ the algorithm starts to watch oszillations. A blockoptimization occurs if more than $b$ Lagrangian parameters are collected, or if more than $c$ bounded Lagrangian parameters are collected, or if more than $d$ multiplied with the number of collected Lagrangian parameters iterations are done since the last block optimization. It might be useful for reduction of computational time to increase particularly $c$ if the problem is very hard and takes a lot of time.

-wss [a b]

**Heuristic3: working set selection**
If $a = 0$ the software searches for KKT-violations randomly. If $a = 1$ the software searches the largest KKT-violation. If $a = 2$ the software does a random search of KKT violations, but it ignores violations below the value $b$ multiplied with the largest KKT violation.

-reduce [a b c]

**Heuristic4: problem reduction**
If $a = 0$ no problem reduction occurs. If $a = 1$ the algorithm breaks the dataset into pieces of size $b$ and calculates the psvm using an epsilon modified with factor $c$. All dimensions which are not choosen as feature or support vector will be recombined and a final psvm run occurs. This heuristic may lead to wrong models if the parameters are not individually adapted. So it is turned off by default.

-kernel [type] [p1 [p2 [p3]]] **apply kernel of type 0 LINEAR (default), 1 POLY, 2 RBF, 3 SIGMOID, 4 PLUMMER, 5 SIN**
The kernel matrix will be generated from input data using the specified kernel function.
Standard case:
0: $k_{l,p} =< \boldsymbol{x}_l, \boldsymbol{x}_p >$
1: $k_{l,p} = (p1 \cdot < \boldsymbol{x}_l, \boldsymbol{x}_p > +p2)^{p3}$
2: $k_{l,p} = \exp(-p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\|^2)$
3: $k_{l,p} = \tanh(p1 \cdot < \boldsymbol{x}_l, \boldsymbol{x}_p > +p2)$
4: $k_{l,p} = (p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\|^2 + p2)^{-p3}$
5: $k_{l,p} = \sin(p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\| + p2) \cdot \exp(-p3 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\|)$

For the use of Z-data:
0: $k_{l,p} =< \boldsymbol{x}_l, \boldsymbol{z}_p >$
1: $k_{l,p} = (p1 \cdot < \boldsymbol{x}_l, \boldsymbol{z}_p > +p2)^{p3}$
2: $k_{l,p} = \exp(-p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\|^2)$
3: $k_{l,p} = \tanh(p1 \cdot < \boldsymbol{x}_l, \boldsymbol{z}_p > +p2)$
4: $k_{l,p} = (p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\|^2 + p2)^{-p3}$
5: $k_{l,p} = \sin(p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\| + p2) \cdot \exp(-p3 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\|)$

For experimental reasons, it is even possible to calculate a function for relational data:
0: $k_{l,p} = x_{l,p}$
1: $k_{l,p} = (p1 \cdot x_{l,p} + p2)^{p3}$
2: $k_{l,p} = \exp(-p1 \cdot x_{l,p})$
3: $k_{l,p} = \tanh(p1 \cdot x_{l,p} + p2)$
4: $k_{l,p} = (p1 \cdot x_{l,p} + p2)^{-p3}$
5: $k_{l,p} = \sin(p1 \cdot x_{l,p} + p2) \cdot \exp(-p3 \cdot x_{l,p})$

All kernel parameters $p1$,$p2$, and $p3$ are optional, so '-kernel 2 p1' is recommended in case of an RBF kernel, and '-kernel 3 p1 p2' would be used for a sigmoid kernel. Missing parameters are set to default values $p1 = 0.1$, $p2 = 0.1$, $p3 = 2$. For determining convenient values for $p1$, see option *-stats*.

-isrelation

**input data is dyadic (matrix) data**
If *-isrelation* is set, the input data is interpreted as kernel matrix $\boldsymbol{K}$. The classification and regression is based on the relations. If *-isrelation* is not set, the input data is interpreted as vectorial data $\boldsymbol{X}$ and the kernel matrix $\boldsymbol{K}$ is computed from the input vectors $\boldsymbol{X}$, or, if a file [name].zdata was found, from $\boldsymbol{X}$ and $\boldsymbol{Z}$.

-flimit [f][g]

**limit maximum number of support vectors for testing (default -1)**
The options *-train*, *-predict*, *-predictc*, and *-test* use normally all support vectors of the model. *-flimit* reduces the number of used support vectors by selecting only those with the highest alpha, so that $f$ support vectors remain. If $f$ equals -1, no limitation occurs. The limitation works also for crossvalidation and hyperparameter selection for experimental reasons. The optional parameter $g$ is for setting the behavier if more than $f$ features or support vectors are bounded caused by low $C$. $g = 0$ deletes all features or support vectors in this case, while $g = 1$ deletes all but $f$ features or support vectors. The last case leads to nondeterministic models, because bounded features are not sortable.

-sig [n]

**significance testing**
Compares the training error with errors for data with randomly label permutations. After n permutations, the program stops and calculates the significance and its trust interval. Additionally the program generates two gnuplot scripts for viewing the error, the bitdistance of the random labels and the rank of its error.

-timeout [t]

**stops SMO after $t$ seconds(default -1)**
If the time in seconds for SMO-calculation reaches $t$, the SMO is stopped. Note, that for performance reasons the precision of the time measure is +-60 seconds. The difference between pressing `^C` and using the timeout is, that the result calculated so far is available.

-verbose [0 | 1]

**1: give progress info (default); 0: give results only**
This option is useful for hyperparameter selection to avoid the progress messages for each training phase.

## 4.2 Model Files Description

The software is able to save a generated model into a file by the command `-model`. The content of this file depends on whether the psvm was used in the dyadic mode, vectorial mode, or vectorial mode with complex feature vectors. The content is organized as follows:

- *all modes:* bias $b$ of the linear model
- *dyadic mode:* number of descripting "row" objects of the training and test data
  *vectorial mode:* number of data vectors used in training
  *vectorial mode with complex feature vectors:* number of complex feature vectors
- *all modes:* number of features obtained by the PSVM,
  *vectorial modes:* number of support vectors obtained by the PSVM
- *dyadic mode:* "1"
  *vectorial modes:* "0"
- *all modes:* used kernel function and kernel parameters
- model table: the columns are
  - *dyadic mode:* feature index - nonzero dimension of $\alpha$
  - *vectorial modes:* support vector index - nonzero dimension of $\alpha$, index corresponds to a data vector (*complex feature vector mode:* complex feature vector) after shuffeling
  - *dyadic mode:* feature expression - amount of $\alpha_{featureindex}$
  - *vectorial mode:* support vector expression - amount of $\alpha_{featureindex}$
  - normalization minimum
  - normalization maximum
  - normalization mean
  - normalization scaling
- *vectorial modes:* support vector table:
  - number of dimensions of the support vectors
  - one row for each support vector

The prediction of a target value when a test vector is given can be obtained by

- *vectorial modes only:* evaluation of the kernel function of the test vector and all support vectors

- *dyadic mode only:* selecting the components of the test object with the indices described in the "feature index" column of the model table
- limiting the data range for all results using the corresponding minimum and maximum values in the model table
- subtraction of the corresponding mean values and multiplication with the corresponding scale values of the model table
- multiplication with the "expression" values of the model table
- adding the bias $b$

## 4.3 Data File Descriptions

The software produces datafiles after hyperparameter selection, significance testing, and ROC-testing. They are ready for further processings but the columns are unlabeled.

### 4.3.1 Hyperparameter Selection Output

If hyperparameter selection is used, the software calculates model errors for a number of hyperparameters. The hyperparameter which is normally prefered has the lowest generalization error (approximated by crossvalidation).
The file with extension ".msdata" contains four columns:

1. hyperparameter $\epsilon$
2. hyperparameter $C$
3. error rate (classification)
4. mean squared error (regression)

The software creates corresponding files with extensions ".msclass.plt" and ".msregress.plt" which are to be openend with Gnuplot.

### 4.3.2 Significance Testing Output

Significance testing gives information about the probability that a possibly good classification error rate is caused by chance. The idea is to calculate classificators for dataset where the elements of the label vector has been randomly permutated. The better the results with wrong labels are, the larger is the probability, that a good result from the real labels does not rely on the real labels.
The file with extension ".sigdata" contains one row per random permutation and seven columns which are:

1. number of changed labels in a certain training phase after randomly permutating the original labels
2. error rate in case of classification
3. mean squared error in case of regression
4. selected hyperparameter $C$ in case of classification
5. selected hyperparameter $C$ in case of regression
6. selected hyperparameter $\epsilon$ in case of classification
7. selected hyperparameter $\epsilon$ in case of regression

The first row corresponds to the result from the original labels. The data gives information of the used hyperparameters during permutation. If the values in the last four columns reach the user given bounds of the hyperparameter selection, the user has to enlarge the search region in order to get proper results. A visualization of the effect of the label permutation can be drawn by opening the corresponding "*.sigclass.plt" and "*.sigregress.plt" with Gnuplot. Shown is the error vs. number of flipped labels during the permutation.
More important is a second visualization which relies on a file with extension ".sigdata1", the columns are:

1. ranking position
2. error rate in case of classification
3. mean squared error in case of regression

All entries are sorted in ascending order and the corresponding visualization script in file ".sigclass1.plt" and ".sigregress1.plt" shows a distribution function of the error. The error which corresponds to the original label vector is marked with a single line, it is not written in the plain datafile.
The corresponding significance levels and trust intervals are printed only to the screen and to the logfile.

### 4.3.3 Calculation of ROC Curves

Using the option "-test" includes the calculation of the Receiver Operating Characteristic (ROC) curve for given predictions and the real labels. The P-SVM internally calculates a score for each example which is to be classified. The score is compared to a threshold (normally zero) to obtain a decision. Changing the threshold influences the relation of positive to negative predictions. The larger the distance of the score from the threshold, the clearer is the descision of the P-SVM. For more information about ROC curves, see the literature [tfroc].

The ROC curve maps the true-positive rate on the false-positive rate while the threshold moves from $-\infty$ to $+\infty$. A true positive is a data object of real class "true" which is correctly classified. A false positive is a data object of real class "false" which is classified as "true". The columns of the corresponding datafile are:

- number of false positives divided by the number of real negatives
- number of true positives divided by the number of real positives

## 4.4 Command Line Sample

The package contains a few sample datasets in the folder 'dataset'.

'cluster1' is a 2-dimensional data set with 200 vectors building 4 overlapping clusters. Two clusters are of class '+1' the others of class '-1'. A good training is achieved by:
`psvm -trainc cluster1 -c 10 -e 0.4 -kernel 2 0.05 -model cluster1`
The training parameters are C=10, epsilon=0.4 and the model will be saved in 'cluster1.model'. The generalization performance can be determined by
`psvm -trainc cluster1 -c 10 -e 0.4 -kernel 2 0.05 -crossval 5`
Use complex feature vectors with another prepared dataset:
`psvm -trainc cluster1z -c 1 -e 1 -kernel 4 0.001 -crossval 5`
A visualization of this sample is included in the MATLAB samples.

'cluster2' is a 10-dimensional set with 200 vectors building three clusters. The labels are build from Euclidean distances to the cluster means with additional noise. A good training based on regression is achieved by:
`psvm -trainr cluster2 -kernel 2 0.0001 -crossval 5 -c 8 -e 0.06`

'arcene' is a data set, that was provided at the NIPS 2003 feature selection challenge `http://clopinet.com/isabelle/Projects/NIPS2003` and can be downloaded via
`http://www.nipsfsc.ecs.soton.ac.uk/datasets/ARCENE.zip`. It contains 100 training vectors with distances to 10000 features. The vectors are labeled as belonging to one of two classes by '+1' and '-1'. The label file is extended with '.labels', for using it with psvm, it must be renamed to '.y'. A good training is achieved by
`psvm -trainc arcene_train -isrelation -c 5 -e 1 -model arcene -crossval 3.`
An included test set 'arcene_test' can be classified with the saved model:

`psvm -model arcene -predictc arcene_test` The label predictions will be written into 'arcene_test.yout'.

## 4.5 Constructing your own kernel

The kernel type no. 6 (see option -kernel) is prepared for a user defined function. Open the source file 'psvm.cpp' and search two functions kf_USER at the end of the file. They look like this:

```
inline ftype PSVMmodel::kf_USER(ftype* x,ftype* z,int n)
{//enter here the formula to get the argument for your kernel function
    double d=0;
    for(int i=0;i<n;i++) d+=(double)x[i]*(double)z[i];
    return (ftype)kf_USER(d);
}
inline double PSVMmodel::kf_USER(double x)
//enter here your kernel function
//use par.k1 par.k2 par.k3
{return x;}
```

The first function quantifies a scalar relation of two vectors, this is normally a scalar product or the Euclidean distance. The result will be mapped using the second function which can be e.g. a polynomial, exponential, trigonometric or other function.

## 4.6 MATLAB Functions

The package contains compilable MATLAB functions for accessing the `PSVM` from MATLAB scripts. The functions psvmtrain and psvmpredict perform all four steps of the training, where the functions psvmkernel, psvmgetnp, psvmputnp, and psvmsmo allow direct access to the psvm internal routines. See the introducing "PSVM Documentation.pdf" for details. The PSVM routines become accessible by:

- start MATLAB
- change into the subfolder `matlabdemo`
- type `startup`
- available commands: `psvmtrain, psvmpredict,` and `psvmgetnp, psvmputnp, psvmkernel, psvmsmo`

**MATLAB function psvmtrain** :

```
[model,error]=psvmtrain(X,Y,kernelpar,psvmpar)
[model,error]=psvmtrain(X,Z,Y,kernelpar,psvmpar)
[rmodel,error]=psvmtrain(K,Y,psvmpar)
```

Where:

| | |
|---|---|
| X | [x1;x2;x3;...;xL] data matrix as row vectors |
| Y | [y1 y2 y3 ... yL] label vector |
| Z | [z1;z2;z3;...;zP] complex features as row vectors |
| K | relational data with L rows and P columns |
| kernelpar | [kerneltype,p1,p2,p3] |
| kerneltype | 0=linear (default), 1=polynomial, 2=RBF, 3=sigmoid, 4=plummer, 5=sinus<br>Standard case:<br>0: $k_{l,p} =< \boldsymbol{x}_l, \boldsymbol{x}_p >$<br>1: $k_{l,p} = (p1 \cdot < \boldsymbol{x}_l, \boldsymbol{x}_p > +p2)^{p3}$<br>2: $k_{l,p} = \exp(-p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\|^2)$<br>3: $k_{l,p} = \tanh(p1 \cdot < \boldsymbol{x}_l, \boldsymbol{x}_p > +p2)$<br>4: $k_{l,p} = (p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\|^2 + p2)^{-p3}$<br>5: $k_{l,p} = \sin(p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\| + p2) \cdot \exp(-p3 \cdot \|\boldsymbol{x}_l - \boldsymbol{x}_p\|)$<br>For the use of Z-data:<br>0: $k_{l,p} =< \boldsymbol{x}_l, \boldsymbol{z}_p >$<br>1: $k_{l,p} = (p1 \cdot < \boldsymbol{x}_l, \boldsymbol{z}_p > +p2)^{p3}$<br>2: $k_{l,p} = \exp(-p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\|^2)$<br>3: $k_{l,p} = \tanh(p1 \cdot < \boldsymbol{x}_l, \boldsymbol{z}_p > +p2)$<br>4: $k_{l,p} = (p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\|^2 + p2)^{-p3}$<br>5: $k_{l,p} = \sin(p1 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\| + p2) \cdot \exp(-p3 \cdot \|\boldsymbol{x}_l - \boldsymbol{z}_p\|)$ |
| psvmpar | [C,epsilon,epsitol,timeout, doreduction,redsize,redfact,domaxsel, selectionfact, doblock,blockNmax,blockCmax,blockOver, doannealing, estart,edec,eterm] |
| C | SMO parameter for slack minimization (default=1e9) |
| epsilon | SMO parameter for $\epsilon$-insensitivity (default=-1) |
| epsitol | SMO termination crit., tolerance for epsilon, should be near zero (default=0.05) |
| timeout | amount of seconds for SMO,<br>-1 means no timeout (default) |
| doreduction | 0 - no reduction,<br>1 - reduction turned on with redsize and redfact |
| domaxsel | 0 - random WSS, 1 - maxselection,<br>2 - mix of 0 and 1 with selectionfact |
| doblock | 0 - no block optimization,<br>1 - use blockoptimization using blockNmax, blockCmax and blockOver |
| doannealing | 0 - no annealing,<br>1 - use annealing with estart,edec and eterm |
| model | [alpha],[b],[normpar],[supportvectors],[kernelpar] |
| rmodel | [alpha],[b],[normpar] for relational data |
| error | [MSE ; classification errore rate] |

This function constructs a complete PSVM model. The difference between the three syntax forms is the generation of the kernel matrix. The first form uses only vectorial data $\boldsymbol{X}$, the second uses additional complex feature vectors to construct a kernel matrix, and the third form uses relational data. All parameter vector elements are optional but the sequence with all preceding elements must be maintained. The output is a cell matrix containing all generated model components. The third syntax uses another model type, because it uses no kernel function.

**MATLAB function psvmpredict** :
`[Yt]=psvmpredict(Kt,rmodel)`
`[Yt]=psvmpredict(Xt,model)`
This function does the prediction using the model

- rmodel: `{alpha,b,normpar}` for dyadic data
- model: `{alpha,b,normpar,supportvectors,kernelpar}` for vectorial data

The first case corresponds to the dyadic mode and does the following for each test object (for each row in Kt):

1. selection of the components of the test object where `rmodel{1}` is zero. The indices are stored in MATLAB sparse format and can be obtained by `find(rmodel{1})`.
2. bound the data range for all results using `rmodel{3}(:,1)` as maximum and `rmodel{3}(:,2)` as mimimum
3. subtraction of the mean values `rmodel{3}(:,3)` and multiplication with the scale values `rmodel{3}(:,4)`
4. scalar multiplication with the nonzero "expression" values in `rmodel{1}`
5. addition of the bias `rmodel{2}`

The second case corresponds to the vectorial mode and does the following for each test object (for each row in Xt):

1. evaluation of the kernel function described in `model{5}` between the test object and all support vectors `model{4}`.
2. limiting the data range for all results using `model{3}(:,1)` as maximum and `model{3}(:,2)` as mimimum
3. subtraction of the mean values `model{3}(:,3)` and multiplication with the scale values `model{3}(:,4)`

4. scalar multiplication with the nonzero "expression" values in `model{1}`
5. adding the bias `model{2}`

The predicted value obtained with `psvmpredict` can be manually transformed into binary class labels by the MATLAB function `sign`.

**MATLAB function psvmgetnp** :
`[np]=psvmgetnp(K)`
This function performs the first normalization step of a kernel matrix $\boldsymbol{K}$ by computing the statistics. The column vector functions are [max,min,mean,scaling].

**MATLAB function psvmputnp** :
`[KK]=psvmgetnp(K,np)`
$\boldsymbol{K}$ is normalized by a pre-calculated statistics matrix $\boldsymbol{np}$. This function corresponds to the second and third step of the function psvmpredict.

**MATLAB function psvmkernel** :
`[K]=psvmgetnp(X,Z,kernelparams)`
`kernelparams=[kerneltype,p1,p2,p3]`
If vectorial data is given, this function applies a kernel function to a given column vector matrix $\boldsymbol{X}$ and a complex feature vector matrix $\boldsymbol{Z}$ for getting a kernel matrix. In standard cases $\boldsymbol{Z}$ and $\boldsymbol{X}$ are identical. This function corresponds to the first step of the function psvmpredict in vectorial mode. For a description of the kernel parameters see 'psvmtrain'.

**MATLAB function psvmsmo** :
`[a]=psvmsmo(KK,Y,psvmparams)`
`[a,qn]=psvmsmo(KK,Y,psvmparams)`
`psvmparams=[C,epsilon,alphalow,epsitol,timeout,algo,e2,e3]`
$\boldsymbol{KK}$ is the normalized Kernel matrix, $\boldsymbol{y}$ is the label vector, and 'params' is a vector of SMO-parameters. See psvmtrain for details of the parameters. The result vector $\boldsymbol{a}$ is sparse and contains the features which are found. The SMO only computes such rows of its internal matrix $\boldsymbol{Q}$ which are needed, where $\boldsymbol{qn}$ gives the number of calculated rows.

## 4.7 MATLAB Samples

The package contains a few demo scripts in the folder `/psvm/matlabdemo/`. The demo scripts use compiled functions from the *libsvm* library, which offers standard SVM learning and prediction. For compilating libsvm change into the folder 'libsvm' and type `make`. If this is not working, please try

`make all12`,
`make all13`, or
`make all14`
depending on your MATLAB version. For other versions, search for the file `/matlab/bin/gccopts.sh`, replace all occurrences of 'gcc' with 'g++' and save it as `/psvm/matlabsrc/g++opts65.sh` and `/psvm/libsvm/g++opts65.sh`.

Please note the copyright information for *libsvm* in the next section. Windows users can use already compiled `.dll` files included in the package in folder `/psvm/windows`. For using the path in the `startup.m` script, change into the folder `/psvm/matlabdemo` and start matlab, which automatically uses the startup script for including the path to the compilations and helper functions.

The sample script 'cluster1.m' creates a normal distributed dataset with two clusters per class of $\pm 1$. A model is calculated and the prediction topology showed in a 3D-surface. The script includes calls to the SVM-functions from *libsvm*, and shows the differences between the two SVM approaches.
The sample script 'cluster2.m' generates data from three clusters and labels it with distances to the means. The PSVM does regression and shows the mean squared error. As in 'cluster1.m', the results are compared with *libsvm*. This sample demonstrates the superior regression performance compared with libsvm.
The sample script 'multikernel.m' uses the relational feature by using five kernel matrices with different kernel functions as one large relational matrix.

## 4.8 Copyright for included libsvm functions

```
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

## 4.9 Software Library

The sources of the psvm-library are in the files `psvm.cpp` and `psvm.h`. The internal precision can be downsized from double to float by defining smofloat directly in the compilation command or in the source header by preprocessor command. It controls the definition of the internal type `ftype`. All functions use this type instead of `double` or `float`. The message flow can be stopped by setting the exported variable `verbose` and `verbimp` from true to false. `verbimp` controls the occurrence of very important messages, while `verbose` gives information about the current processing step of the P-SVM. Within MATLAB-procedures, both variables has to be false, because MATLAB uses other `printf` functions. The psvm uses a logfile feature, so if you set up the corresponding global variable 'flog' with a file handle, all messages on the screen will be stored into this logfile. The default values for all parameters are set in the headerfile in structure PSVMmodel::par, which is used for contolling all psvm parameters. For implementation details see the sources, there are many comments. The main two classes are `Dataset` and `PSVMmodel`. Datasets contains all data matrices with or without Y-labels.
A dataset can be loaded with:

```
bool Dataset::load(char* fnData,char* fnClass=NULL,int trans=0);
```

The third argument is zero for vectorial data used in kernel functions. It must be one for loading relational data matrices. Building an instance of PSVMmodel initializes automatically the parameter structure `PSVMmode::par` with defaults as described in header file. If needed, the parameters can be overwritten. The PSVMmodel class contains all model components. For learning, use the following function:

```
bool PSVMmodel::getModel(Dataset* set,Trainres *res);
```

For using all hyperpameter and crossvalidation loops use:

```
bool PSVMmodel::getModelWithLoops(Dataset* set,Trainres *res);
```

These training functions refer to the following functions:

```
void PSVMmodel::kernel(Dataset* xset,Dataset* zset);
void PSVMmodel::kernel(Dataset* xset);
bool PSVMmodel::getRankFromSet(Dataset* set);
void PSVMmodel::getB(Dataset* set);
void PSVMmodel::getSVZ(Dataset* set);
void PSVMmodel::np.getFromSet(Dataset* set);
void PSVMmodel::np.putOnSet(Dataset* set);
```

For predictiong labels by a model, use the following functions. The used model is implicit determinated in the PSVMmodel class.

```
void getY(ftype* yy,Dataset* set,int flimit=-1,bool full=true);
```

Nearly all classes have destructors to free memory which was reserved within the class procedures. The function `getModelWithLoops` reserves memory for a hyperparameter selection error matrix in the given structure storing the results. The caller is responsible to delete it after using.

# References

[ncpsvm] Sepp Hochreiter, Klaus Obermayer; Support Vector Machines for Dyadic Data, *Neural Computation*, 2005.

[torch] Ronan Collobert, Samy Bengio, Johnny Mariéthoz; a machine learning library; http://www.torch.ch

[libsvm] C.-C. Chang, C.-J. Lin; LIBSVM; http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[tfroc] Tom Fawcett; ROC Graphs: Notes and Practical Considerations for Researchers; HP Laboratories, MS 1143