SimonsSVM: A Fast and Scalable Support Vector Machine Implementation for R

Philipp Thomann & Ingo Steinwart

Institute of Stochastics and Applications
University of Stuttgart

European R Users Meeting, Posnan, October 13, 2016

Support Vector Machines

SVMs are kernel methods for non-parametric classification and regression in the widest sense. They have been studied widely both in theory and in practice.

Historical Development

Classification V. Vapnik et al. (1992 –): Hinge loss

Regression G. Wahba (1971 –): Least squares loss

Other scenarios since 1995

Books

Vapnik 1998, Schöllkopf/Smola 2002, Steinwart/Christmann 2008.

Implementations

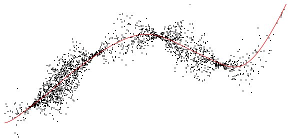
`libsvm` (R-package `e1071`), `SVMlight` (R-package `klaR`), `kernlab`, . . .

SimonsSVM in one Minute

```r
install.packages("SimonsSVM",
        repos="http://www.isa.uni-stuttgart.de/R")
library(SimonsSVM)
```

Least-Squares regression

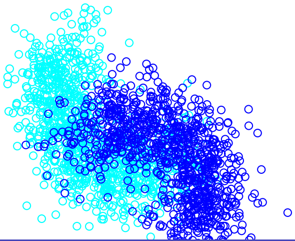
```r
# Load test and training data
reg <- smldata('reg-1d')
model <- svm(Y~., reg$train)
errors(test(model, reg$test))
#> 0.00541
```



Available regression scenarios: least-squares regression, quantile regression, expectile regression.

Classification

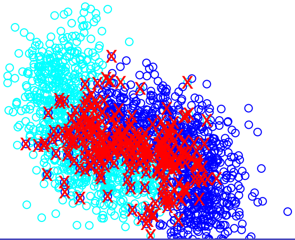
```
banana <- smldata('banana-bc')
model <- svm(Y~., banana$train, display=1)
#> Considering cell 1 out of 1 for task 1 out of 1.
#> Fold 1: training set size 1600,   validation set size 400.
#> Fold 2: training set size 1600,   validation set size 400.
#> Fold 3: training set size 1600,   validation set size 400.
#> [...]
errors(test(model, banana$test) )
#> 0.147
```



Available classification scenarios: multi-class classification (default for factors), Neyman-Pearson learning, ROC

Classification

```
banana <- smldata('banana-bc')
model <- svm(Y~., banana$train, display=1)
#> Considering cell 1 out of 1 for task 1 out of 1.
#> Fold 1: training set size 1600,   validation set size 400.
#> Fold 2: training set size 1600,   validation set size 400.
#> Fold 3: training set size 1600,   validation set size 400.
#> [...]
errors(test(model, banana$test) )
#> 0.147
```



Available classification scenarios: multi-class classification (default for factors), Neyman-Pearson learning, ROC

Summary of Features

- Optimized for speed and fully automatic learning.
- Fully integrated cross validation let's the user focus on understandable parameters.
- Meaningful default values often make tuning of parameters unnecessary.
- Partitioning strategies decrease memory usage and training and testing time for big problems. Large-scale problems become treatable.
- Many standard learning scenarios are included.
- Comprehensive documentation.
- Kernel computation is multi-threaded and can make use of GPUs.
- Provides Gauss RBF and Poisson kernel, more can be added.
- Bindings for `R`/`MATLAB`/`Java`/`Python`
- Open Source

Benchmarks

Test Errors (relative to SimonsSVM)

	Single runs with the recommended param.				5-fold Cross-valid. on 10 × 10-grid	
	SimonsSVM-cv	libsvm	SVMlight	kernlab	SimonsSVM-cv	libsvm-cv
BANK-MARKETING	1.0	1.089	1.089	1.085	1.0	0.990
COD-RNA	1.0	2.878	1.428	1.342	1.0	0.931
COVTYPE	1.0	1.351	1.353	1.213	1.0	0.999
THYROID-ANN	1.0	1.529	1.533	1.377	1.0	0.949

Always mean of 10 single-threaded classification-runs on samples of size 2000 on an Intel® Core®-i7 with 16GB RAM running Debian Linux.

Runtime (relative to SimonsSVM)

	Single runs with the recommended param.			
	SimonsSVM	libsvm	SVMlight	kernlab
BANK-MARKETING	1.0	2.8	6.1	3.4
COD-RNA	1.0	6.0	5.0	3.2
COVTYPE	1.0	8.1	12.2	5.9
THYROID-ANN	1.0	2.2	5.7	3.6

Benchmarks

Test Errors (relative to SimonsSVM)

	Single runs with the recommended param.				5-fold Cross-valid. on 10 × 10-grid	
	SimonsSVM-cv	libsvm	SVMlight	kernlab	SimonsSVM-cv	libsvm-cv
BANK-MARKETING	1.0	1.089	1.089	1.085	1.0	0.990
COD-RNA	1.0	2.878	1.428	1.342	1.0	0.931
COVTYPE	1.0	1.351	1.353	1.213	1.0	0.999
THYROID-ANN	1.0	1.529	1.533	1.377	1.0	0.949

Always mean of 10 single-threaded classification-runs on samples of size 2000 on an Intel® Core®-i7 with 16GB RAM running Debian Linux.

Runtime (relative to SimonsSVM)

	Single runs with the recommended param.			
	SimonsSVM	libsvm	SVMlight	kernlab
BANK-MARKETING	1.0	2.8	6.1	3.4
COD-RNA	1.0	6.0	5.0	3.2
COVTYPE	1.0	8.1	12.2	5.9
THYROID-ANN	1.0	2.2	5.7	3.6

Benchmarks

Test Errors (relative to SimonsSVM)

	Single runs with the recommended param.				5-fold Cross-valid. on 10 $\times$ 10-grid	
	`SimonsSVM-cv`	`libsvm`	`SVMlight`	`kernlab`	`SimonsSVM-cv`	`libsvm-cv`
BANK-MARKETING	1.0	1.089	1.089	1.085	1.0	0.990
COD-RNA	1.0	2.878	1.428	1.342	1.0	0.931
COVTYPE	1.0	1.351	1.353	1.213	1.0	0.999
THYROID-ANN	1.0	1.529	1.533	1.377	1.0	0.949

Always mean of 10 single-threaded classification-runs on samples of size 2000 on an Intel® Core®-i7 with 16GB RAM running Debian Linux.

Runtime (relative to SimonsSVM)

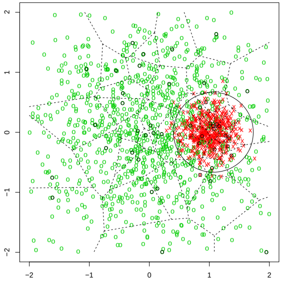
	Single runs with the recommended param.				5-fold Cross-valid. on 10 $\times$ 10-grid	
	`SimonsSVM`	`libsvm`	`SVMlight`	`kernlab`	`SimonsSVM-cv`	`libsvm-cv`
BANK-MARKETING	1.0	2.8	6.1	3.4	27.5	2310.4
COD-RNA	1.0	6.0	5.0	3.2	24.0	1789.3
COVTYPE	1.0	8.1	12.2	5.9	29.8	3197.9
THYROID-ANN	1.0	2.2	5.7	3.6	27.8	1623.9

Split Feature space into Cells

SVMs usually are at least $O(n^2)$ in both time and memory. `Simons' SVM` has a solution

```
svm(Y~., covtype_full, useCells=TRUE)
```

You can choose from 6 different methods.



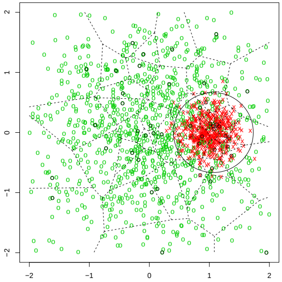
The full COVERTYPE data set (about 523.000 samples) would need 1.6TB of memory. `Simons' SVM`, by splitting it into cells of size at most 2000, trains and tests on a Core-i5 laptop with 12GB of memory in 9.8 minutes!

Split Feature space into Cells

SVMs usually are at least $O(n^2)$ in both time and memory. `Simons' SVM` has a solution

```
svm(Y~., covtype_full, useCells=TRUE)
```

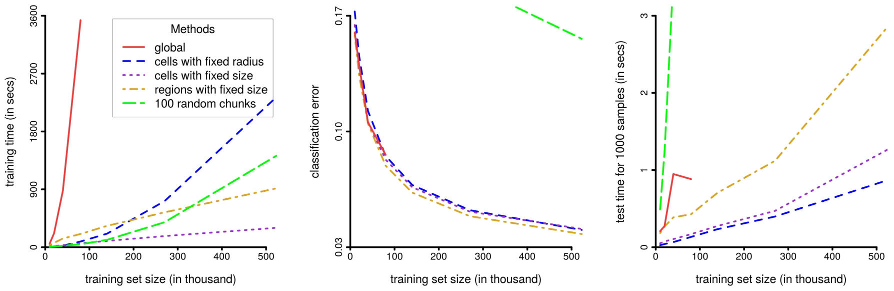
You can choose from 6 different methods.



The full COVERTYPE data set (about 523.000 samples) would need 1.6TB of memory. `Simons' SVM`, by splitting it into cells of size at most 2000, trains and tests on a Core-i5 laptop with 12GB of memory in 9.8 minutes!

Split Feature space into Cells (II)

This works without sacrificing generalization:



Also large-scale data sets are possible to treat:
E.g. we trained and tested the HIGGS data set (10 million samples) in five hours on a Xeon hexacore with 64GB memory.

Implementation Details

- Written in `C++` and its main functionality is accessible through a small number of high-level `C++` classes.
- The routines for computing the kernel matrices and for evaluating the SVM models on the test data are multi-threaded.
- Time critical inner loops are vectorized whenever this is possible. (`SSE2`, `AVX`, and `AVX2`)

The solution algorithms are of SMO-type. They do not use any offset in the decision functions: This is not needed for Gaussian RBF and Poisson kernels, and it makes the algorithm much faster and better to use in cross-validation.

Interfaces

There is a command line version of `Simons' SVM` with scripts for the learning scenarios.

The `R`-bindings are the most mature of the bindings:

- Full integration into the `R`-workflow including several helper functions
- 2 Vignettes: `demo` and `documentation`
- online help (e.g. `?svm`, . . .)

All the bindings (`R`/`MATLAB`/`Java`/`Python`):

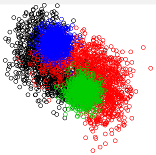
- share a common C-interface to `Simons' SVM C++` code.
- can be compiled either optimized for the current machine (`native`) or to use only `SSE2` (`generic`).
- All operations are in-memory and in-process.

The command line version and all the bindings have been tested on `Linux` and `macOS`, as well as on `Windows 8`.

Multiclass classification

Multiclass classification has to be reduced to binary tasks:

- all-vs-all (Default)
- one-vs-all (aka one-vs-rest)



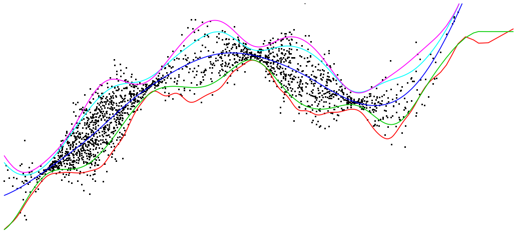
```
model <- svm(Y~., banana_mc$train)
# banana_mc$train$Y is factor with 4 levels
errors(test(model, banana_mc$test))
#> result  1vs2  1vs3  1vs4  2vs3  2vs4  3vs4
#>  0.217 0.142 0.112 0.093 0.074 0.074 0.001
model <- mcSVM(Y~., banana_mc$train, mc_type="OvA_ls")
errors(test(model, banana_mc$test))
#>    result 1vsOthers 2vsOthers 3vsOthers 4vsOthers
#>    0.2147    0.1545    0.1227    0.0777    0.0737
```

The first element in the errors is the overall test error.

Quantile regression

This uses the quantile solver with pinball loss and performs selection for every quantile provided.

```
quantiles_list <- c(0.05, 0.1, 0.5, 0.9, 0.95)
model <- qtSVM(Y ~ ., reg$train, weights=quantiles_list)
```

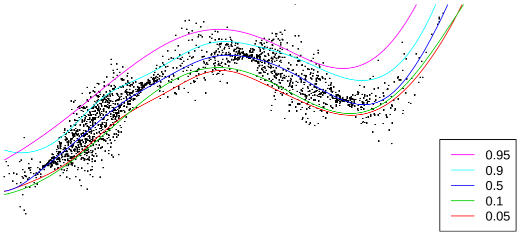


In this plot you see estimations for two lower and upper quantiles as well as the median of the distribution of the label y given x.

Expectile regression (solver implemented by M. Farooq)

This uses the expectile solver with weighted least squares loss and performs selection for every weight. The 0.5-expectile in fact is just the ordinary least squares regression and hence estimates the mean of y given x. And in the same way as quantiles generalize the median, expectiles generalize the mean.

```
expectiles_list <- c(0.05, 0.1, 0.5, 0.9, 0.95)
model <- exSVM(Y ~ ., reg$train, weights=expectiles_list)
```



Neyman-Pearson-Learning

Neyman-Pearson-Learning attempts classification under the constraint that the probability of false positives (Type-I error) is bound by a significance level alpha, which we call the NPL-constraint.

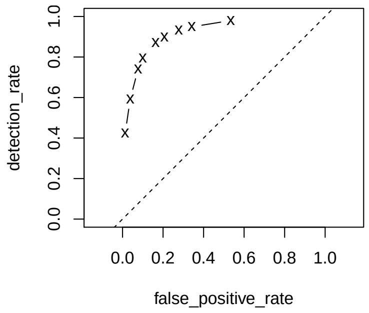
```
model <- nplSVM(Y ~ ., banana, class=-1)
# [...]
#>                       1      2     3     4     5
#> npl_constraints   0.025 0.0333 0.050 0.075 0.100
#> false_alarm_rate 0.037 0.0390 0.053 0.058 0.086
#> detection_rate   0.563 0.5630 0.678 0.692 0.770
```

You can see that the false alarm rate in the test set meet the NPL-constraints quite nicely, and on the other hand the the detection rate is increasing.

ROC curve

Receiver Operating Characteristic curve (ROC curve) plots trade-off between the false alarm rate and the detection rate for different weights.

```r
model <- rocSVM(Y ~ ., banana)
```



This shows nice learning, since the ROC curve is near the north-west corner.

Questions? Bug-reports?

Please visit our Homepage

`http://www.isa.uni-stuttgart.de/software/`

or write me an E-Mail

`philipp.thomann@mathematik.uni-stuttgart.de`

Questions? Bug-reports?

Please visit our Homepage

`http://www.isa.uni-stuttgart.de/software/`

or write me an E-Mail

`philipp.thomann@mathematik.uni-stuttgart.de`

Dziękuję