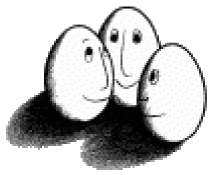

SVM^{light}

Support Vector Machine

Author: Thorsten Joachims <thorsten@joachims.org>
Cornell University
Department of Computer Science

Developed at:
University of Dortmund, Informatik, AI-Unit
Collaborative Research Center on 'Complexity Reduction in Multivariate Data' (SFB475)

Version: 6.02
Date: 14.08.2008

Overview

SVM^{light} is an implementation of Support Vector Machines (SVMs) in C. The main features of the program are the following:

- fast optimization algorithm
  - working set selection based on steepest feasible descent
  - "shrinking" heuristic
  - caching of kernel evaluations
  - use of folding in the linear case
- solves classification and regression problems. For multivariate and structured outputs use SVM^{struct}.
- solves ranking problems (e. g. learning retrieval functions in *STRIVER* search engine).
- computes XiAlpha-estimates of the error rate, the precision, and the recall
- efficiently computes Leave-One-Out estimates of the error rate, the precision, and the recall
- includes algorithm for approximately training large transductive SVMs (TSVMs) (see also Spectral Graph Transducer)
- can train SVMs with cost models and example dependent costs
- allows restarts from specified vector of dual variables
- handles many thousands of support vectors
- handles several hundred-thousands of training examples
- supports standard kernel functions and lets you define your own
- uses sparse vector representation

NEW Machine Learning Course: If you would like to learn more about Machine Learning, you can find videos, slides, and readings of the course I teach at Cornell here.

NEW SVM^{struct}: SVM learning for multivariate and structured outputs like trees, sequences, and sets (available here).

NEW SVM^{perf}: New training algorithm for linear classification SVMs that can be much faster than SVM^{light} for large datasets. It also lets you directly optimize multivariate performance measures like F1-Score, ROC-Area, and the Precision/Recall Break-Even Point. (available here).

NEW! SVM*rank*: New algorithm for training Ranking SVMs that is much faster than SVM*light* in '-z p' mode. (available here).

Description

SVM*light* is an implementation of Vapnik's Support Vector Machine [Vapnik, 1995] for the problem of pattern recognition, for the problem of regression, and for the problem of learning a ranking function. The optimization algorithms used in SVM*light* are described in [Joachims, 2002a]. [Joachims, 1999a]. The algorithm has scalable memory requirements and can handle problems with many thousands of support vectors efficiently.

The software also provides methods for assessing the generalization performance efficiently. It includes two efficient estimation methods for both error rate and precision/recall. XiAlpha-estimates [Joachims, 2002a, Joachims, 2000b] can be computed at essentially no computational expense, but they are conservatively biased. Almost unbiased estimates provides leave-one-out testing. SVM*light* exploits that the results of most leave-one-outs (often more than 99%) are predetermined and need not be computed [Joachims, 2002a].

New in this version is an algorithm for learning ranking functions [Joachims, 2002c]. The goal is to learn a function from preference examples, so that it orders a new set of objects as accurately as possible. Such ranking problems naturally occur in applications like search engines and recommender systems.

Futhermore, this version includes an algorithm for training large-scale transductive SVMs. The algorithm proceeds by solving a sequence of optimization problems lower-bounding the solution using a form of local search. A detailed description of the algorithm can be found in [Joachims, 1999c]. A similar transductive learner, which can be thought of as a transductive version of k-Nearest Neighbor is the Spectral Graph Transducer.

SVM*light* can also train SVMs with cost models (see [Morik et al., 1999]).

The code has been used on a large range of problems, including text classification [Joachims, 1999c][Joachims, 1998a], image recognition tasks, bioinformatics and medical applications. Many tasks have the property of sparse instance vectors. This implementation makes use of this property which leads to a very compact and efficient representation.

Source Code and Binaries

The program is free for scientific use. Please contact me, if you are planning to use the software for commercial purposes. The software must not be further distributed without prior permission of the author. If you use SVM*light* in your scientific work, please cite as

- T. Joachims, Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.
  [PDF] [Postscript (gz)] [BibTeX]

I would also appreciate, if you sent me (a link to) your papers so that I can learn about your research. The implementation was developed on Solaris 2.5 with gcc, but compiles also on SunOS 3.1.4, Solaris 2.7, Linux, IRIX, Windows NT, and Powermac (after small modifications, see FAQ). The source code is available at the following location:

http://download.joachims.org/svm_light/current/svm_light.tar.gz

If you just want the binaries, you can download them for the following systems:

- Linux (32-bit): http://download.joachims.org/svm_light/current/svm_light_linux32.tar.gz
- Linux (64-bit): http://download.joachims.org/svm_light/current/svm_light_linux64.tar.gz
- Windows (32-bit): http://download.joachims.org/svm_light/current/svm_light_windows32.zip
- Windows (64-bit): http://download.joachims.org/svm_light/current/svm_light_windows64.zip
- Cygwin (32-bit): http://download.joachims.org/svm_light/current/svm_light_cygwin.tar.gz
- Mac OS X (old): http://download.joachims.org/svm_light/current/svm_light_osx.tar.gz
- Mac OS X (new): http://download.joachims.org/svm_light/current/svm_light_osx.8.4_i7.tar.gz
- Solaris: http://download.joachims.org/svm_light/current/svm_light_solaris.tar.gz

Please send me email and let me know that you got svm-light. I will put you on my mailing list to inform you about new versions and bug-fixes. SVM^{light} comes with a quadratic programming tool for solving small intermediate quadratic programming problems. It is based on the method of Hildreth and D'Espo and solves small quadratic programs very efficiently. Nevertheless, if for some reason you want to use another solver, the new version still comes with an interface to PR_LOQO. The PR_LOQO optimizer was written by A. Smola. It can be requested from http://www.kernel-machines.org/code/prloqo.tar.gz.

Installation

To install SVM^{light} you need to download `svm_light.tar.gz`. Create a new directory:

```
mkdir svm_light
```

Move `svm_light.tar.gz` to this directory and unpack it with

```
gunzip -c svm_light.tar.gz | tar xvf -
```

Now execute

```
make or make all
```

which compiles the system and creates the two executables

```
svm_learn (learning module)
svm_classify (classification module)
```

If you do not want to use the built-in optimizer but PR_LOQO instead, create a subdirectory in the svm_light directory with

```
mkdir pr_loqo
```

and copy the files `pr_loqo.c` and `pr_loqo.h` in there. Now execute

```
make svm_learn_loqo
```

If the system does not compile properly, check this FAQ.

How to use

This section explains how to use the SVM^{light} software. A good introduction to the theory of SVMs is Chris Burges' tutorial.

SVM^{light} consists of a learning module (`svm_learn`) and a classification module (`svm_classify`). The classification module can be used to apply the learned model to new examples. See also the examples below for how to use `svm_learn` and `svm_classify`.

`svm_learn` is called with the following parameters:

```
svm_learn [options] example_file model_file
```

Available options are:

```
General options:
         -?          - this help
         -v [0..3]   - verbosity level (default 1)
Learning options:
         -z {c,r,p}  - select between classification (c), regression (r), and
                       preference ranking (p) (see [Joachims, 2002c])
                       (default classification)
         -c float    - C: trade-off between training error
                       and margin (default [avg. x*x]^-1)
         -w [0..]    - epsilon width of tube for regression
                       (default 0.1)
         -j float    - Cost: cost-factor, by which training errors on
                       positive examples outweight errors on negative
                       examples (default 1) (see [Morik et al., 1999])
         -b [0,1]    - use biased hyperplane (i.e. x*w+b0) instead
                       of unbiased hyperplane (i.e. x*w0) (default 1)
         -i [0,1]    - remove inconsistent training examples
                       and retrain (default 0)
Performance estimation options:
         -x [0,1]    - compute leave-one-out estimates (default 0)
                       (see [5])
         -o ]0..2]   - value of rho for XiAlpha-estimator and for pruning
                       leave-one-out computation (default 1.0)
                       (see [Joachims, 2002a])
         -k [0..100] - search depth for extended XiAlpha-estimator
                       (default 0)
Transduction options (see [Joachims, 1999c], [Joachims, 2002a]):
         -p [0..1]   - fraction of unlabeled examples to be classified
                       into the positive class (default is the ratio of
                       positive and negative examples in the training data)
Kernel options:
         -t int      - type of kernel function:
                        0: linear (default)
                        1: polynomial (s a*b+c)^d
                        2: radial basis function exp(-gamma ||a-b||^2)
                        3: sigmoid tanh(s a*b + c)
                        4: user defined kernel from kernel.h
         -d int      - parameter d in polynomial kernel
         -g float    - parameter gamma in rbf kernel
         -s float    - parameter s in sigmoid/poly kernel
         -r float    - parameter c in sigmoid/poly kernel
         -u string   - parameter of user defined kernel
Optimization options (see [Joachims, 1999a], [Joachims, 2002a]):
         -q [2..]    - maximum size of QP-subproblems (default 10)
         -n [2..q]   - number of new variables entering the working set
                       in each iteration (default n = q). Set n<q to prevent
                       zig-zagging.
         -m [5..]    - size of cache for kernel evaluations in MB (default 40)
                       The larger the faster...
         -e float    - eps: Allow that error for termination criterion
                       [y [w*x+b] - 1] = eps (default 0.001)
         -h [5..]    - number of iterations a variable needs to be
                       optimal before considered for shrinking (default 100)
         -f [0,1]    - do final optimality check for variables removed by
                       shrinking. Although this test is usually positive, there
                       is no guarantee that the optimum was found if the test is
                       omitted. (default 1)
         -y string   -> if option is given, reads alphas from file with given
```

```
                                 and uses them as starting point. (default 'disabled')
                -# int        -> terminate optimization, if no progress after this
                                 number of iterations. (default 100000)
        Output options:
                -l char       - file to write predicted labels of unlabeled examples
                                into after transductive learning
                -a char       - write all alphas to this file after learning (in the
                                same order as in the training set)
```

A more detailed description of the parameters and how they link to the respective algorithms is given in the appendix of [Joachims, 2002a].

The input file `example_file` contains the training examples. The first lines may contain comments and are ignored if they start with #. Each of the following lines represents one training example and is of the following format:

```
<line> .=. <target> <feature>:<value> <feature>:<value> ... <feature>:<value> # <info>
<target> .=. +1 | -1 | 0 | <float>
<feature> .=. <integer> | "qid"
<value> .=. <float>
<info> .=. <string>
```

The target value and each of the feature/value pairs are separated by a space character. Feature/value pairs MUST be ordered by increasing feature number. Features with value zero can be skipped. The string `<info>` can be used to pass additional information to the kernel (e.g. non feature vector data). Check the FAQ for more details on how to implement your own kernel.

In classification mode, the target value denotes the class of the example. +1 as the target value marks a positive example, -1 a negative example respectively. So, for example, the line

```
-1 1:0.43 3:0.12 9284:0.2 # abcdef
```

specifies a negative example for which feature number 1 has the value 0.43, feature number 3 has the value 0.12, feature number 9284 has the value 0.2, and all the other features have value 0. In addition, the string `abcdef` is stored with the vector, which can serve as a way of providing additional information for user defined kernels. A class label of 0 indicates that this example should be classified using transduction. The predictions for the examples classified by transduction are written to the file specified through the -l option. The order of the predictions is the same as in the training data.

In regression mode, the <target> contains the real-valued target value.

In ranking mode [Joachims, 2002c], the target value is used to generated pairwise preference constraints (see STRIVER). A preference constraint is included for all pairs of examples in the `example_file`, for which the target value differs. The special feature "qid" can be used to restrict the generation of constraints. Two examples are considered for a pairwise preference constraint only, if the value of "qid" is the same. For example, given the `example_file`

```
3 qid:1 1:0.53 2:0.12
2 qid:1 1:0.13 2:0.1
7 qid:2 1:0.87 2:0.12
```

a preference constraint is included only for the first and the second example (ie. the first should be ranked higher than the second), but not with the third example, since it has a different "qid". NOTE: SVM^{rank} is a new algorithm for training Ranking SVMs that is much faster than SVM^{light} in '-z p' mode (available here).

In all modes, the result of `svm_learn` is the model which is learned from the training data in `example_file`. The model is written to `model_file`. To make predictions on test examples, `svm_classify` reads this file. `svm_classify` is called with the following parameters:

```
svm_classify [options] example_file model_file output_file
```

Available options are:

```
-h          Help.
-v [0..3]   Verbosity level (default 2).
-f [0,1]    0: old output format of V1.0
            1: output the value of decision function (default)
```

The test examples in `example_file` are given in the same format as the training examples (possibly with 0 as class label). For all test examples in `example_file` the predicted values are written to `output_file`. There is one line per test example in `output_file` containing the value of the decision function on that example. For classification, the sign of this value determines the predicted class. For regression, it is the predicted value itself, and for ranking the value can be used to order the test examples. The test example file has the same format as the one for `svm_learn`. Again, `<class>` can have the value zero indicating unknown.

If you want to find out more, try this FAQ.

Getting started: some Example Problems

Inductive SVM

You will find an example text classification problem at

http://download.joachims.org/svm_light/examples/example1.tar.gz

Download this file into your svm_light directory and unpack it with

```
gunzip -c example1.tar.gz | tar xvf -
```

This will create a subdirectory `example1`. Documents are represented as feature vectors. Each feature corresponds to a word stem (9947 features). The task is to learn which Reuters articles are about "corporate acquisitions". There are 1000 positive and 1000 negative examples in the file `train.dat`. The file `test.dat` contains 600 test examples. The feature numbers correspond to the line numbers in the file `words`. To run the example, execute the commands:

```
svm_learn example1/train.dat example1/model
svm_classify example1/test.dat example1/model example1/predictions
```

The accuracy on the test set is printed to stdout.

Transductive SVM

To try out the transductive learner, you can use the following dataset (see also Spectral Graph Transducer). I compiled it from the same Reuters articles as used in the example for the inductive SVM. The dataset consists of only 10 training examples (5 positive and 5 negative) and the same 600 test examples as above. You find it at

http://download.joachims.org/svm_light/examples/example2.tar.gz

Download this file into your svm_light directory and unpack it with

```
gunzip -c example2.tar.gz | tar xvf -
```

This will create a subdirectory `example2`. To run the example, execute the commands:

```
svm_learn example2/train_transduction.dat example2/model
svm_classify example2/test.dat example2/model example2/predictions
```

The classification module is called only to get the accuracy printed. The transductive learner is invoked automatically, since `train_transduction.dat` contains unlabeled examples (i. e. the 600 test examples). You can compare the results to those of the inductive SVM by running:

```
svm_learn example2/train_induction.dat example2/model
svm_classify example2/test.dat example2/model example2/predictions
```

The file `train_induction.dat` contains the same 10 (labeled) training examples as `train_transduction.dat`.

Ranking SVM

For the ranking SVM [Joachims, 2002c], I created a toy example. It consists of only 12 training examples in 3 groups and 4 test examples. You find it at

http://download.joachims.org/svm_light/examples/example3.tar.gz

Download this file into your svm_light directory and unpack it with

```
gunzip -c example3.tar.gz | tar xvf -
```

This will create a subdirectory `example3`. To run the example, execute the commands:

```
svm_learn -z p example3/train.dat example3/model
svm_classify example3/test.dat example3/model example3/predictions
```

The output in the predictions file can be used to rank the test examples. If you do so, you will see that it predicts the correct ranking. The values in the predictions file do not have a meaning in an absolute sense. They are only used for ordering.

It can also be interesting to look at the "training error" of the ranking SVM. The equivalent of training error for a ranking SVM is the number of training pairs that are misordered by the learned model. To find those pairs, one can apply the model to the training file:

```
svm_classify example3/train.dat example3/model example3/predictions.train
```

Again, the predictions file shows the ordering implied by the model. The model ranks all training examples correctly.

Note that ranks are comparable only between examples with the same qid. Note also that the target value (first value in each line of the data files) is only used to define the order of the examples. Its absolute value does not matter, as long as the ordering relative to the other examples with the same qid remains the same.

NOTE: SVM*rank* is a new algorithm for training Ranking SVMs that is much faster than SVM*light* in '-z p' mode (available here).

Questions and Bug Reports

If you find bugs or you have problems with the code you cannot solve by yourself, please contact me via email.

Disclaimer

History

V6.01 - V6.02

- Fixed a floating point precision issue that might occur on 64bit AMD and Intel processors for datasets with extremely many features.
- Updated makefile to add the ability for compiling SVM-light into a shared-object library that gives external code easy access to learning and classification functions.
- Source code for SVM*light* V6.01

V6.00 - V6.01

- Small bug fixes in HIDEO optimizer.

V5.00 - V6.00

- Allows restarts from a particular vector of dual variables (option y).
- Time out for exceeding number of iterations without progress (option #).
- Allows the use of Kernels for learning ranking functions.
- Support for non-vectorial data like strings.
- Improved robustness and convergence especially for regression problems.
- Cleaned up code, which makes it easier to integrate it into other programs.
- Interface to SVM*struct*.
- Source code for SVM*light* V5.00

V4.00 - V5.00

- Can now solve ranking problems in addition to classification and regression.
- Fixed bug in kernel cache that could lead to segmentation fault on some platforms.
- Fixed bug in transductive SVM that was introduced in version V4.00.
- Improved robustness.
- Source code for SVM*light* V4.00

V3.50 - V4.00

- Can now solve regression problems in addition to classification.
- Bug fixes and improved numerical stability.
- Source code for SVM*light* V3.50

V3.02 - V3.50

- Computes XiAlpha estimates of the error rate, the precision, and the recall.
- Efficiently computes Leave-One-Out estimates of the error rate, the precision, and the recall.
- Improved Hildreth and D'Espo optimizer especially for low-dimensional data sets.
- Easier to link into other C and C++ code. Easier compilation under Windows.
- Faster classification of new examples for linear SVMs.

V3.01 - V3.02

- Now examples can be read in correctly on SGIs.

V3.00 - V3.01

- Fixed convergence bug for Hildreth and D'Espo solver.

V2.01 - V3.00

- Training algorithm for transductive Support Vector Machines.
- Integrated core QP-solver based on the method of Hildreth and D'Espo.
- Uses folding in the linear case, which speeds up linear SVM training by an order of magnitude.
- Allows linear cost models.
- Faster in general.

V2.00 - V2.01

- Improved interface to PR_LOQO
- Source code for SVM^{light} V2.01

V1.00 - V2.00

- Learning is much faster especially for large training sets.
- Working set selection based on steepest feasible descent.
- "Shrinking" heuristic.
- Improved caching.
- New solver for intermediate QPs.
- Lets you set the size of the cache in MB.
- Simplified output format of svm_classify.
- Data files may contain comments.

V0.91 - V1.00

- Learning is more than 4 times faster.
- Smarter caching and optimization.
- You can define your own kernel function.
- Lets you set the size of the cache.
- VCdim is now estimated based on the radius of the support vectors.
- The classification module is more memory efficient.
- The f2c library is available from here.
- Adaptive precision tuning makes optimization more robust.
- Includes some small bug fixes and is more robust.
- Source code for SVM^{light} V1.00

V0.9 - V0.91

- Fixed bug which appears for very small C. Optimization did not converge.

Extensions and Additions

- PERL Interface: a PERL interface to SVM^{light} written by Ken Williams
- Matlab Interface: a MATLAB interface to SVM^{light} written by Anton Schwaighofer (for SVM^{light} V4.00)
- Matlab Interface: a MATLAB MEX-interface to SVM^{light} written by Tom Briggs
- Python Interface: Python interface for SVM^{light} written by Daoud Clarke.
- Python Interface: Python binding for SVM^{light} written by Bill Cauchois.
- Python Interface: interface to call SVM^{light} executables from Python, written by Clint Burfoot.
- Ruby Interface: interface to call SVM^{light} from Ruby, written by Vicente Bosch.
- Ruby Interface: interface to call SVM^{light} from Ruby, written by Camilo Lopez.
- .NET Interface: .NET wrapper for SVM^{light}, written by Krishnamurthy Koduvayur Viswanathan.
- DLL Interface: a DLL that makes it easier to integrate SVM^{light} functions into other code written by Miha Grcar.
- Java Interface: JNI JAVA interface for SVM^{light}, written by Martin Theobald (for SVM^{light} V6.01)
- JNI Kernel: JAVA interface for SVM^{light}, including access to kernel functions implemented in Java, written by Stephan Bloehdorn (for SVM^{light} V6.01)
- jSVM: a JAVA interface to SVM^{light} written by Heloise Hwawen Hse (for SVM^{light} V2.01 / source code)
- Tree Kernels: kernel for classifying trees with SVM^{light} written by Alessandro Moschitti
- SVM-Dlight: allows implementing kernels for non-vectorial data more easily. Written by Paolo Frasconi (for SVM^{light} V6.01)
- A special version of SVM^{light} is integrated into the virtual file system libferris by Ben Martin
- LightDataAgent: tool to translate comma/tab-delimited data into SVM^{light} format, written by Ophir Gottlieb
- SVM-Classify TCP/IP Server: a server version of svm_classify that let's you classify examples over a TCP/IP port, written by Matthew Gerber (for SVM^{light} V6.01)

References

[Joachims, 2002a] Thorsten Joachims, *Learning to Classify Text Using Support Vector Machines*. Dissertation, Kluwer, 2002.
[B&N] [Amazon] [Kluwer] [BibTeX]

[Joachims, 2002c] T. Joachims, *Optimizing Search Engines Using Clickthrough Data*, Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM, 2002.
Online [Postscript] [PDF] [BibTeX]

[Klinkenberg, Joachims, 2000a] R. Klinkenberg and T. Joachims, *Detecting Concept Drift with Support Vector Machines*. Proceedings of the Seventeenth International Conference on Machine Learning (ICML), Morgan Kaufmann, 2000.
[Postscript (gz)] [PDF (gz)] [BibTeX]

[Joachims, 2000b] T. Joachims, *Estimating the Generalization Performance of a SVM Efficiently*. Proceedings of the International Conference on Machine Learning, Morgan Kaufman, 2000.
[Postscript (gz)] [PDF] [BibTeX]

[Joachims, 1999a] T. Joachims, 11 in: *Making large-Scale SVM Learning Practical*. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT Press, 1999.
[Postscript (gz)] [PDF] [BibTeX]

[Joachims, 1999c]

	Thorsten Joachims, *Transductive Inference for Text Classification using Support Vector Machines*. International Conference on Machine Learning (ICML), 1999. [Postscript (gz)] [PDF] [BibTeX]
[Morik et al., 1999a]	K. Morik, P. Brockhausen, and T. Joachims, *Combining statistical learning with a knowledge-based approach - A case study in intensive care monitoring*. Proc. 16th Int'l Conf. on Machine Learning (ICML-99), 1999. [Postscript (gz)] [PDF] [BibTeX]
[Joachims, 1998a]	T. Joachims, *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. Proceedings of the European Conference on Machine Learning, Springer, 1998. [Postscript (gz)] [PDF] [BibTeX]
[Joachims, 1998c]	Thorsten Joachims, *Making Large-Scale SVM Learning Practical*. LS8-Report, 24, Universität Dortmund, LS VIII-Report, 1998. [Postscript (gz)] [PDF] [BibTeX]
[Vapnik, 1995a]	Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 1995.

Last modified July 25, 2013 by Thorsten Joachims <thorsten@joachims.org>