Support Vector Machine
Reference Manual

C. Saunders, M. O. Stitson, J. Weston
Department of Computer Science
Royal Holloway
University of London
e-mail: {C.Saunders,M.Stitson,J.Weston}@dcs.rhbnc.ac.uk

L. Bottou
AT&T Speech and Image Processing Services Research Lab
e-mail: leonb@research.att.com

B. Schölkopf, A. Smola
GMD FIRST
e-mail: {bs,smola}@first.gmd.de

October 6, 1998

The Support Vector Machine (SVM) is a new type of learning machine. The SVM is a general architecture that can be applied to pattern recognition, regression estimation and other problems. The following researchers were involved in the development of the SVM:

A. Gammerman (RHUL)	V. Vapnik (AT&T, RHUL)	Y. LeCun (AT&T)
N. Bozanic (RHUL)	L. Bottou (AT&T)	C. Saunders (RHUL)
B. Schölkopf (GMD)	A. Smola (GMD)	M. O. Stitson (RHUL)
V. Vovk (RHUL)	C. Watkins (RHUL)	J. A. E. Weston (RHUL)

The major reference is V.Vapnik, "The Nature of Statistical Learning Theory", Springer 1995.

1 Getting Started

The Support Vector Machine (SVM) program allows a user to carry out pattern recognition and regression estimation, using support vector techniques on some given data.

If you have any questions not answered by the documentation, you can e-mail us at:

`svmmanager@dcs.rhbnc.ac.uk`

2 Programs

Release 1.0 of the RHUL SV Machine comes with a set of seven programs (`sv, paragen, loadsv, transform_sv, snsv, ascii2bin, bin2ascii`).

- `sv` - the main SVM program
- `paragen` - program for generating parameter sets for the SVM
- `loadsv` - load a saved SVM and classify a new data set
- `transform_sv` - special SVM program for image recognition, that implements virtual support vectors [BS97].
- `snsv` - program to convert SN format to our format
- `ascii2bin` - program to convert our ASCII format to our binary format
- `bin2ascii` - program to convert our binary format to our ASCII format

The rest of this document will describe these programs. To find out more about SVMs, see the bibliography. We will not describe how SVMs work here.

The first program we will describe is the `paragen` program, as it specifies all parameters needed for the SVM.

3 `paragen`

When using the support vector machine for any given task, it is always necessary to specify a set of parameters. These parameters include information such as whether you are interested in pattern recognition or regression estimation, what kernel you are using, what scaling is to be done on the data, etc... `paragen` generates parameter files used by the SVM program, if no file was generated the user will be asked interactively.

`paragen` is run by the following command line :

```
paragen [ <load parameter file> [<save parameter file>] ]
```

The parameter file is optional, and obviously cannot be included the first time you run `paragen` as you have not created a parameter file before. If however, you have a parameter file which is similar to the one you want to use, by specifying that file as part of the command line the program will start with all of the parameters set to the relevant values, allowing you to make a couple of changes and then save the file under a different name. The second option is to specify the name of the file you want to save the parameters to. This can also be done by selecting a menu option within `paragen`. If no save parameter file argument is given it is assumed that you wish to save over the same file name as given in the load parameter file argument.

3.1 Traversing the menu system

paragen uses a simple text based menu system. The menus are organized in a tree structure which can be traversed by typing the number of the desired option, followed by return. Option 0 (labelled "Exit") is in each menu at each branch of the tree. Choosing option 0 always traverses up one level of the tree. If you are already at the top of the tree, it exits the program.

3.2 The top level menu

The first menu gives you the option of displaying, entering, or saving parameters. When the menu appears, if you choose the enter parameters option, this process is identical to specifying parameters interactively when running the SVM. The menu looks like this:

```
SV Machine Parameters
=====================

 1. Enter parameters
 2. Load  parameters
 3. Save  parameters (pattern_test)
 4. Save  parameters as...
 5. Show  parameters
 0. Exit
```

Options 2,3 and 4 are straight forward. Option 5 displays the chosen parameters and option 1 allows the parameters to be entered. We now describe the branches of the menu tree after choosing option 1 ("Enter parameters") in detail.

3.3 Enter Parameters

If you choose to enter parameters, then you are faced with a list of the current parameter settings, and a menu. An example of which is shown below :

```
SV Machine parameters
=====================

No kernel specified

Alphas unbounded
Input values will not be scaled.
Training data will not be posh chunked.
Training data will not be sporty chunked.
Number of parameter sets: 1
        Optimizer: 3
        SV zero threshold: 1e-16
        Margin threshold: 0.1
        Objective zero tolerance: 1e-07
```

```
 1. Set the SV Machine type
 2. Set the Kernel type
 3. Set general parameters
 4. Set kernel specific parameters
 5. Set expert parameters

 0. Exit

Please enter your choice:
```

Each of these menu options allow the users to specify different aspects of the Support Vector Machine that they wish to use, and each one will now be dealt with in turn.

3.4 Setting the SV Machine Type

When option 1 is chosen, the following menu appears:

```
Type of SV machine
==================
 1 Pattern Recognition
 2 Regression Estimation
 6 Multiclass Pattern Recognition

Please enter the machine type: (0)
```

After entering 1, 2 or 6 at the prompt (and pressing return), you are given the top-level parameter menu again. If you look at the line below "SV Machine Parameters", the type of SV machine which was selected should be displayed.

After selecting the SV machine type, the user must decide which kernel to use.

3.5 Setting the Kernel Type

Option 2 from the SV machine parameters menu allows the kernel type to be chosen. For a detailed description of the kernel types see the appendix. The menu will look like this:

```
Type of Kernel
==============
 1 Simple Dot Product
 2 Vapnik's Polynomial
 3 Vovk's Polynomial
 4 Vovk's Infinite Polynomial
 5 Radial Basis Function
 6 Two Layer Neural Network
```

```
 7 Infinite dimensional linear splines
 8 Full Polynomial (with scaling)
 9 Weak-mode Regularized Fourier
10 Semi-Local (polynomial & radial basis)
11 Strong-mode Regularized Fourier
17 Anova 1
18 Generic Kernel 1
19 Generic Kernel 2
```

Many of the kernel functions have one or more free parameters, the values of which can be set using option 4 "Set kernel specific parameters" in the SV Machine parameters menu (one branch of the tree up from this menu). For example, using the polynomial kernel (2) "Vapnik's polynomial" one can control the free parameter d, the degree of polynomial.

3.5.1 Implementing new kernel functions

Options 18 and 19 are special convenience kernel functions that have been included for experienced users who wish to implement their own kernel functions.

Kernel functions are written as a C++ class that inherits most of its functionality from a base class. When you wish to add a new kernel, instead of adding a new class and having to change various interface routines and the Makefiles you can simply change the function `kernel_generic_1_c::calcK(...)` or `kernel_generic_2_c::calcK(...)` and choose to use this kernel from the menu options after re-compilation.

Both generic kernels have five (potential) free parameters labelled `a_val`, `b_val` and so on which can be set in the usual way in the "Set kernel specific parameters" menu option.

3.6 Setting the General Parameters

Option 3 of the SVM parameter menu allows the user to set the free parameters for the SVM (in the pattern recognition case, the size of the upper bound on the Lagrangian variables, i.e. the box constraints, C, and in regression estimation, C and the choice of ϵ for the the ϵ-insensitive loss function.) Various other miscellaneous options have also been grouped together here: scaling strategy, chunking strategy and multi-class pattern recognition strategy. This menu has different options depending on the type of SV Machine chosen: pattern recognition, regression estimation or multi-class pattern recognition.

For the pattern recognition SVM the following options are given:

```
General parameters
==================
 1. Bound on Alphas (C) 0
 2. Scaling             off
 3. Chunking
```

```
 0. Exit
```

For the regression SVM the following options are given:

```
General parameters
==================
 1. Bound on Alphas (C) 0
 2. Scaling             off
 3. Chunking
 4. Epsilon accuracy    0

 0. Exit
```

For the multi-class SVM the following options are given:

```
General parameters
==================
 1. Bound on Alphas (C) 0
 2. Scaling             off
 3. Chunking            off
 7. Multi-class Method   1
 8. Multi-class Continuous Classes       0

 0. Exit
```

3.6.1 Setting the Bound on Alphas

This options sets the upper bound C on the support vector coefficients (alphas). This is the free parameter which controls the trade off between minimizing the loss function (satisfying the constraints) and minimizing over the regularizer. The lower the value of C, the more weight is given to the regularizer.

If C is set to infinity all the constraints must be satisfied. Typing 0 is equivalent to setting C to infinity. In the pattern recognition case this means that the training vectors must be classified correctly (they must be linearly separable in feature space).

Choosing the value of C needs care. Even if your data can be separated without error, you may obtain better results by choosing simpler decisions functions (to avoid over-fitting) by lowering the value of C, although this is generally problem specific and dependent on the amount of noise in your data.

A good rule of thumb is to choose a value of C that is slightly lower than the largest coefficient or alpha value attained from training with $C = \infty$. Choosing a value higher than the largest coefficient will obviously have no effect as the box constraint will never be violated. Choosing a value of C that is too low (say close to 0) will constrain your solution too much and you will end up with too simple a decision function.

Plotting a graph of error rate on the testing set against choice of parameter C will typically give a bowl shape, where the best value of C is somewhere in the middle. For inexperienced users who wish to get an intuitive grasp of how to choose C try playing with this value on toy problems using the RHUL SV applet at ``http://svm.cs.rhbnc.ac.uk''.

3.6.2 Scaling

Included in the support vector engine is a convenience function which pre-scales your data before training. The programs automatically scale your data back again for output and error measures, allowing a quick way to pre-process your data suitably to ensure the dot products (results of your chosen kernel function) give reasonable values. For serious problems, it is recommended you do your own pre-processing, but this function is still a useful tool.

Scaling can be done either globally (i.e. all values are scaled by the same factor) or locally (each individual attribute is scaled by an independent factor).

As a guideline you may wish to think of it this way; if the attributes are all of the same type (e.g. pixel values) then scale globally, if they are of different types (e,g, age, height, weight) then scale locally. When you select the scaling option, the program first asks if you want to scale the data, then it asks if all attributes are to be scaled with the same factor. Answering Y corresponds to global scaling and N corresponds to local scaling. You are then asked to specify the lower and upper bounds for the scaled data e.g. -1 and 1, or 0 and 1.

The scaling of your data is important! Incorrect scaling can make the program appear not to be working, but in fact the training is suffering because of lack of precision of the values of dot products in feature space. Secondly certain kernels require their parameters to be within certain ranges, for example the linear spline kernel requires that all attributes are positive, and the weak mode regularized Fourier kernel requires that $0 \leq |x_i - x_j| \leq 2\pi$. For a full description of the requirements of each kernel function see the appendix.

3.6.3 Chunking

This option chooses the type of optimizer training strategy. Note, the choice of strategy should not effect the learning ability of the SVM but rather the speed of training. If no chunking is selected the optimizer is invoked with all training points. Only use the 'no chunking' option if the number of training points is small (less than 1000 points).

The optimizer requires half of an n by n matrix where n is the number of training points, so if the number of points is large (≥ 4000) you will probably just run out of memory and even if you don't it will be very slow.

If you have a large number of data points the training should consider the optimization problem as solving a sequence of sub-problems - which we call chunking. There are two types of chunking method implemented, posh chunking and sporty chunking (out of respect to the Spice Girls) which follow the algorithms described in the papers [OFG97b] and [OFG97a] respectively.

Sporty chunking requires that you enter the chunk size. This represents the number of training points that are added to the chunk per iteration. A typical value for this parameter is 500.

Posh chunking requires that you enter the working set size and the pivoting size. The working set size is the number of vectors that are in each sub-problem, which is fixed (in sporty chunking this is variable). A typical value is 700. The pivoting size is the maximum number of vectors that can be moved out of the sub-problem and are replaced with fixed vectors. A typical value is 300.

3.6.4 Setting ϵ

ϵ defines the ϵ insensitive loss function. When $C = \infty$ this stipulates how far the training examples are allowed to deviate from the learnt function. As C tends to zero, the constraints become soft.

3.6.5 Setting the Multiclass Method

This selects the multi-class method to use. If we have n classes, method 0 trains n machines, each classifying one class against the rest. Method 1 trains $\frac{n(n-1)}{2}$ machines, each classifying one class against one other class.

3.6.6 Setting Multiclass Continuous Classes

This setting is designed to speed up training in the multi-class SVM. If you know your classes are a sequence of continuous integers like $2, 3, 4$, then you can enter 1 here to speed things up. If you choose this option and this is not the case the machine's behaviour is undefined. So if in doubt leave this setting at 0.

3.7 Setting Kernel Specific Parameters

This menu option allows you to enter the free parameters of the specific kernel you have chosen. If the kernel has no free parameters then you will not be prompted to enter anything, the program will just go back to the main parameter menu.

3.8 Setting the Expert Parameters

The expert parameter menu has the following options:

```
Expert parameters
=================
Usually these are ok!

 1. Optimizer (1=MINOS, 2=LOQO, 3=BOTTOU)       3
 2. SV zero threshold                           1e-16
 3. SV Margin threshold                         0.1
 4. Objective function zero tolerance           1e-07
```

```
0. Exit
```

If you are an inexperienced user, you are advised not to alter these values.

3.8.1 Optimizer

There are three optimizers that can be currently be used with the RHUL SV package. These are used to solve the optimization problems required to learn decision functions. They are:

- MINOS - a commercial optimization package written by the Department of Operations Research, Stanford University.
- LOQO - an implementation of an interior point method based on the LOQO paper [Van] written by Alex J. Smola, GMD, Berlin.
- BOTTOU - an implementation of the conjugate gradient method written by Leon Bottou, AT&T research labs.

Only the LOQO and BOTTOU optimizers are provided in the distribution of this package as the first is a commercial package. However, stubs are provided for MINOS, and should the user acquire a license for MINOS, or if the user already has MINOS, all you have to do is to place the MINOS Fortran code into the directory minos.f, change the MINOS setting in `Makefile.include` and re-make.

The LOQO optimizer is not currently implemented for regression estimation problems.

3.8.2 SV zero threshold

This value indicates the cut-off point when a double precision Lagrange multiplier is considered zero, in other words what numbers are not counted as support vectors. In theory support vectors are all vectors with a non-zero coefficient, however, in practice optimizers only deal with numbers to some precision, and as default values below 1e-16 are considered as zero. Note that for different optimizers and different problems this can change. An sv zero threshold that is too low can result in a large number of support vectors and increased training time.

3.8.3 SV Margin threshold

This value represents the "virtual" margin used in the posh chunking algorithm. The idea is that training vectors are not added to the chunk unless they are on the wrong side of the virtual margin, rather than the real margin, where the virtual margin is at distance $1 - value$ (default $1 - 0.1$) from the decision hyperplane. This is used to remove the problem of slight losses in precision that can cause vectors to cycle from being correctly to incorrectly classified in the chunking algorithm.

3.8.4 Objective function zero tolerance

Both chunking algorithms terminate when the objective function does not improve after solving an optimization sub-problem. To prevent precision problems of the objective continually being improved by extremely small amounts (again caused by precision problems with the optimizer) and the algorithm never terminating, an improvement has to be larger than this value to be relevant.

4 `sv`

The SVM is run from the command line, and has the following syntax :

```
sv <Training File> <Test File> [<Parameter File> [<sv machine file>]]
```

For a description of the file format see the appendix or for a simple introduction, see "`sv/docs/intro/sv_user.tex`".

Specifying a parameter file is optional. If no parameter file is specified, then the user will be presented with a set of menus, which will allow the user to define a set of parameters to be used. These menus are exactly the same as those used to enter parameters using the `paragen` program (see section 3).

If a parameter file is included the learnt decision function can be saved in with the file name of your choice. This can be reloaded using the program `loadsv` (section 5) to test new data at a later stage.

After calculating a decision rule based on the training set, each of the test examples are evaluated and the program outputs a list of statistics. If you do not want to test a test set you can specify "`/dev/null`" or an empty file as the second parameter. You can also specify the same file as the training and testing set.

The output from the program will depend on whether the user is using the SV Machine for pattern recognition, regression estimation, or multi-class classification. First of all the output from the optimizer is given, followed by a list of which examples in the training set are the support vectors[1]. Performance statistics involving the error on the training and testing sets are then given. Following this, each support vector is listed along with the value of its Lagrange multiplier (alpha value), and its deviation from the margin.

4.1 Output from the `sv` Program

```
SV Machine parameters
=====================
Pattern Recognition

Full polynomial

Alphas unbounded
```

[1] Does not apply to multi-class SVM.

```
        Input values will be globally scaled between 0 and 1.
        Training data will not be posh chunked.
        Training data will not be sporty chunked.
        Number of SVM: 1

        Degree of polynomial: 2.
        Kernel Scale Factor: 256.
        Kernel Threshold: 1.
----------------------------------------
Positive SVs: 12 13 16 41 111 114 157 161
Negative SVs: 8 36 126 138 155 165
There are 14 SVs (8 positive and 6 negative).
Max alpha size: 3.78932
B0 is -1.87909
Objective = 9.71091
Training set:
Total samples:          200
Positive samples:       100
of which errors:        0
Negative samples:       100
of which errors:        0

----------------------------------------
Test set:
Total samples:          50
Positive samples:       25
of which errors:        0
Negative samples:       25
of which errors:        0

There are 1 lagrangian multipliers per support vector.
No.     alpha(0)        Deviation
    8      1.86799      3.77646e-08
   12     0.057789      6.97745e-08
   13      2.75386
   16     0.889041      -1.63897e-08
   36      1.53568      5.93671e-08
   41     0.730079      3.91323e-09
  111     0.359107      -1.38041e-07
  114      0.88427      -9.06655e-08
  126     0.561356      1.15792e-07
  138        2.558      2.0497e-08
  155      2.50095      -1.24475e-08
  157     0.247452      -1.35147e-07
  161      3.78932      -1.2767e-07
  165     0.686947      1.03066e-07
Finished checking support vector accuracy.
Total   deviation is 9.30535e-07      No. of SVs: 14
Average deviation is 6.64668e-08
Minimum alpha     is 0.057789
Maximum alpha     is 3.78932
```

The SVM program uses a different type of optimizer to construct the rule, depending on which one you selected when setting the parameters. When using LOQO as the optimizer (the default) if there is an error in optimization this is stated. MINOS gives an output of the following form :

```
 ==============================
 M I N O S    5.4    (Dec 1992)
 ==============================

 Begin SV_TEST
OPTIMAL SOLUTION FOUND (0)
----------------------------------------
```

In this case, the optimizer signals that an optimal solution was found. If the data is scaled badly, or the data is inseparable (and the bound on the alphas is infinite), then an error may occur here. Therefore, you will have to ensure the scaling options are set correctly, and you may have to change the bound on the alpha values (the value of C).

The next section informs the user how many support vectors there are, and lists the example numbers of those examples which were support vectors. This section also indicates the largest alpha value (lagrangian multiplier), and the value of b0 (threshold of the decision function). This does not apply to the multi-class SVM.

This is followed by information as to how the SVM performed on both the training set and the test set. In the case of pattern recognition (as shown above), the output indicates the number of positive and negative samples, and the number of those which were misclassified in both the training and the test set. For instance, in the example above, all of the examples in the training set were classified correctly.
When running the SVM program to perform regression estimation, various measures of error are displayed here. The user is given the average (absolute) error on the training set. Also, the totals and averages are displayed for both absolute and squared error on the training set.
For the multiclass machine a table is displayed giving the number of errors on the individual classes. This contains the same information as the normal pattern recognition SVM in a slightly different form. Adding the columns gives you the total number of examples in a class. The diagonal is the number of correct classifications.

Following the performance statistics, a list of the values of the alphas (Lagrange multipliers) for each support vector is given, along with its deviation (how far away the support vector is from the boundary of the margin). If no deviation is printed, the vector was exactly distance 1 from the margin. Finally some statistics are given, indicating the minimum and maximum alpha values (useful for setting C, the scaling of your data and sometimes the SV zero threshold.)

5 loadsv

The loadsv program is used to load an SV Machine that has already been trained in order to classify new test data. The program is run from the command line, and has the following syntax :

```
loadsv <sv machine file> <Test File>
```

Classification of test data is performed in exactly the same as in the sv program (section 4).

6 transform_sv

This is a modified version of the sv program, that implements B. Schölkopf's [Sch97] ideas of transformation invariance for images. The training data must be binary classified images and only pattern recognition can be performed. The general idea is that most images are still the same, even if they are moved a pixel sideways or up or down. The program initially trains an SVM and then creates a new training set including all support vectors and their transformations in four directions. This set is used to train a second machine, which potentially may generalize better than the first machine.

Running the program works just like the sv program except that you are asked for the x and y dimensions of the images and the background intensity.

At the end you are given two sets of statistics. The first set is the usual set that the sv program produces. The second consists of the error rate on the newly created training set, the original training set and the test set.

7 snsv

Included in the RHUL SV Machine distribution is the utility program snsv which converts the SN data file format for pattern recognition problems only into our own data file format. For details on the exact format of SN files see "sv/docs/snsv/sn-format.txt". For a description of the file format see the appendix or for a simple introduction, see "sv/docs/intro/sv_user.tex".

The utility program is called in the following way:

```
snsv <sn data file> <sn truth data file> <output data file>
```

The first argument is the name of the data file in SN format (binary, ASCII or packed) and the second the SN data file containing the truth values (classifications) of the vectors described in the data file. The third argument is the name of the output file.

The program has the following menu options:

```
(1) Single class versus other classes ; or
```

```
(2) All classes
```

Option 1 takes the data and truth files and creates a binary classified data file. Examples from a single class (which you specify) are labeled as the positive examples, and all other classes are negative examples. This is useful when you have multi-class pattern recognition data, and you wish to learn a one-against-the-rest classifier.

Option 2 just saves the class data out as is. If there are more than two classes this data file can only be used with a multi-class SV Machine.

Finally you are asked whether you wish the output to be in binary or ASCII. Binary offers faster loading times and smaller file sizes, however ASCII can be useful for debugging or analyzing your data with an editor. All the SV programs automatically detect the format (binary or ASCII) of data files.

8 `ascii2bin` and `bin2ascii`

The programs are very simple. They convert between our binary and ASCII input files.They take two command line arguments:

```
ascii2bin <input file> <output file>
```

and

```
bin2ascii <input file> <output file>
```

If you have a program generating data, you might want to look at the appendix describing the data format.

9 Further Information

There is an on-line version of the support vector machine which has been developed in the department. The web site has a graphical interface which allows you to plot a few points and see what decision boundary is produced. The page also provides links to other SVM sites. The web address of the page is :

`http://svm.cs.rhbnc.ac.uk`

If you have any further questions e-mail us at:

`svmmanager@dcs.rhbnc.ac.uk`

10 Acknowledgements

We would like to thank A. Gammerman, V. Vapnik, V. Vovk and C. Watkins at Royal Holloway, K. Müller at GMD and Y. LeCun, P. Haffner and P. Simard at AT&T for their support in this project.

A SV Kernels

This is a list of the kernel functions in the RHUL SV Machine:

- ◇ 1. The simple dot product:

$$K(x, y) = x \cdot y$$

- ◇ 2. The simple polynomial kernel:

$$K(x, y) = ((x \cdot y) + 1)^d$$

where d is user defined.
(Taken from [Vap95])

- ◇ 3. Vovk's real polynomial:

$$K(x, y) = \frac{1 - (x \cdot y)^d}{1 - (x \cdot y)}$$

where d is user defined and where $-1 < (x \cdot y) < 1$.
(From private communications with V. Vovk)

- ◇ 4. Vovk's real infinite polynomial:

$$K(x, y) = \frac{1}{1 - (x \cdot y)}$$

where $-1 < (x \cdot y) < 1$.
(From private communications with V. Vovk)

- ◇ 5. Radial Basis function:

$$\exp(-\gamma |x - y|^2)$$

where γ is user defined.
(Taken from [Vap95])

- ◇ 6. Two layer neural network:

$$\tanh(\frac{b(x \cdot y)}{1} - c)$$

where b and c are user defined.
(Taken from [Vap95])

- ◇ 7. Linear splines with an infinite number of points:

For the one-dimensional case:

$$1 + x_i x_j + x_i x_j \min(x_i, x_j) - \frac{x_i + x_j}{2}(\min(x_i x_j))^2 + \frac{(\min(x_i, x_j))^3}{3}$$

For the multi-dimensional case $K(x, y) = \prod_{k=1}^{n} K_k(x^k, y^k)$
(Taken from [VGS])

- ⋄ 8. Full polynomial kernel:

$$\left(\frac{x \cdot y}{a} + b\right)^d$$

where a, b and d are user defined.

(From [Vap95] and generalized)

- ⋄ 9. Regularized Fourier (weaker mode regularization)

For the one-dimensional case:

$$\frac{\pi}{2\gamma}\frac{\cosh\frac{\pi-|x_i-x_j|}{\gamma}}{\sinh\frac{\pi}{\gamma}}$$

where $0 \leq |x_i - x_j| \leq 2\pi$ and γ is user defined.

For the multi-dimensional case $K(x, y) = \prod_{k=1}^{n} K_k(x^k, y^k)$

(From [VGS] and [Vapng])

- ⋄ 10. Semi Local Kernel

$$[(x_i \cdot x_j) + 1]^d \exp(-||x_i - x_j||^2\sigma^2)$$

where d and σ are user defined and weight between global and local approximation.

(From private communications with V. Vapnik)

- ⋄ 11. Regularized Fourier (stronger mode regularization)

For the one-dimensional case:

$$\frac{1-\gamma^2}{2(1-2\gamma\cos(x_i - x_j) + \gamma^2)}$$

where $0 \leq |x_i - x_j| \leq 2\pi$ and γ is user defined.

For the multi-dimensional case $K(x, y) = \prod_{k=1}^{n} K_k(x^k, y^k)$

(From [VGS] and [Vapng])

- ⋄ 17. Anova 1

$$K(x, y) = \left(\sum_{k=1}^{n} \exp(-\gamma(x^k - y^k)^2)\right)^d$$

where the degree d and γ are user defined.

(From private communications with V. Vapnik)

- ⋄ 18. Generic Kernel 1

This is a kernel intended for experiments, just modify the appropriate function in kernel_generic_1_c.C. You can use the parameters a_val, b_val, c_val, d_val and e_val.

- ⋄ 19. Generic Kernel 2

This is a kernel intended for experiments, just modify the appropriate function in kernel_generic_2_c.C. You can use the parameters a_val, b_val, c_val, d_val and e_val.

B Input file format

This is just a brief description of the input file format for the training and testing data. A detailed description is given in the next section.

B.1 ASCII input

The input files consist of a simple header and the actual data. When saving files additional data is added to the header, but this can be safely ignored.

The simplest input files are pure ASCII and only contain numbers. The first number specifies the number of examples in the file, the second number specifies how many attributes there are per example. The third number determines whether or not an extended header is used. Set this to 1, unless you want to use an extended header from the next section. This is followed by the data. Each example is given in turn, first its attributes then its classification or value.

Say we have four examples in two dimensional input space and the classification follows the function $f(x_1, x_2) = 2 \times x_1 + x_2$. The input file should look something like this:

```
4
2
1
1   1   3
1.5 3.4 6.4
1.2 0   2.4
0   3   3
```

B.2 Binary input

It is also possible to create binary input files, if you are worried about loss of accuracy. We will describe a simplified version here which corresponds to the above ASCII file.

All binary input files start with a magic number which consists of four bytes: 1e 3d 4c 53.

This is followed by `int` and `double` variables saved using the C++ `ofstream.write(void *, int size)` function or the C function `write(int file_descriptor, void *, int size)`.

The header consists of the number of examples (`int`), attributes per example (`int`), 1 (`int`), 1 (`int`), 0 (`int`), 0 (`int`).

The rest of the file simply consists of examples. First the attributes of an example then its classification as doubles.

C Sample List

The sample list is either an ASCII file or a binary file. The ASCII file is portable the binary file may not be.

The sample list file contains only numbers. The first few numbers indicate the exact format followed by the data.

The sample list can load several formats but only saves one format.

C.1 ASCII Version pre-0

The first number (`int`) of the sample list file always contains the number of examples in the file.

The second number (`int`) of the sample list file always contains the dimensionality of the input space, i.e. the number of attributes.

The third number (`int`) of the sample list file determines the format of the file. In this case, this number is set to 1, to indicate we are using ASCII Version pre-0[2].

The rest of the file simply consists of examples. First the input values of an example then its classification.

Say we have four examples in two dimensional input space and the classification follows the function $f(x_1, x_2) = 2 \times x_1 + x_2$. The input file should look something like this:

```
4
2
1
1   1   3
1.5 3.4 6.4
1.2 0   2.4
0   3   3
```

C.2 ASCII Version 0

The first three numbers have the same meaning as in version pre-0: Number of examples, number of attributes, version (0).

The fourth number (`int`) indicates the dimensionality of the classification of the examples.

The fifth number (0/1) indicates whether or not the data has been pre-scaled. This is useful if other data should be scaled in the same way this data has been

[2]Note: This number is referred to as the version number. For the ASCII pre-0 format, this number is 1. With each later version of the ASCII file format, however, this number decreases; i.e. when using ASCII Version 0 this number should be set to 1, and for ASCII Version 1, the number should have a value of -1.

scaled. The sixth number (0/1) indicates whether or not the classifications have been scaled. The seventh number indicates the lower bound of the scaled data. The eighth number indicates the upper bound of the scaled data. Then follows a list of the thresholds used for scaling (`double`). It has as many elements as there are dimensions in input space plus the number of dimensions of the classification. Then follows a list of scaling factors (`double`). It has as many elements as the previous list. For an exact explanation on how scale factors and threshold are calculated see the section on scaling. Note that these scale factors are the factors that have previously been applied to the data. They will not be applied to the data when loading.

The rest of the file simply consists of examples. First the input values of an example then its classification.

Say we have four examples in two dimensional input space and the classification follows the function $f(x_1, x_2) = 2 \times x_1 + x_2$. The data was scaled before being put into the list between -1 and 1. The original data points are the same as in the version -1 example. The input file should look something like this:

```
4
2
0
1
1
0
-1
1
-0.75    -1.7       0
1.333333 0.58823529 1
0.333333 -0.4117647 3
1.5      1          6.4
1.2      -1         2.4
0        0.76470588 3
```

C.3 ASCII Version 1

The first three numbers have the same meaning as in version pre-0: Number of examples, number of attributes, version (-1).

The fourth number (`int`) indicates the dimensionality of the classification of the examples.

The fifth number (0/1) indicates whether or not the data has individual epsilon values per example. This is only relevant for regression.

The sixth number (0/1) indicates whether or not the data has been pre-scaled. This is useful if other data should be scaled in the same way this data has been scaled. The scale factors following will only be saved if the scaling is 1 above. The seventh number (0/1) indicates whether or not the classifications have been scaled. The eighth number indicates the lower bound of the scaled data. The ninth number indicates the upper bound of the scaled data. Then follows a list of the thresholds used for scaling (`double`). It has as many elements as there are dimensions in input space plus the number of dimensions of the classification. Then follows a list of scaling factors (`double`). It has as many elements as the previous list. For an exact explanation on how scale factors and threshold are calculated see the section on scaling. Note that these scale factors are the factors that have previously been applied to the data. They will not be applied to the data when loading.

The rest of the file simply consists of examples. First the input values of an example then its classification.

Say we have four examples in two dimensional input space and the classification follows the function $f(x_1, x_2) = 2 \times x_1 + x_2$. The data was scaled before being put into the list between -1 and 1. The original data points are the same as in the version 0 example. The input file should look something like this:

```
4
2
-1
1
1
1
0
0
0
0   0   0
1   1   1
1   1   3   0.1
1.5 3.4 6.4 0.2
1.2 0   2.4 0.1
0   3   3   0.2
```

C.4 Binary Version 1

All binary sample list files start with a magic number which consists of four bytes: 1e 3d 4c 53.

This is followed by `int` and `double` variables saved using the C++ ofstream.write(void *, int size) function.

The format exactly follows the ASCII version 1: Number of examples (`int`), number of attributes (`int`), version (int, should be 1), dimensionality of the classification of the examples (`int`), individual epsilon values per example (`int`).

The sixth number (`int`) indicates whether (1) or not (0) the data has been prescaled.

If the data has been scaled the following will appear: The seventh number (`int`) indicates whether (1) or not (0) the classifications have been scaled. The eighth number indicates the lower bound of the scaled data (`double`). The ninth number indicates the upper bound of the scaled data (`double`). Then follows a list of the thresholds used for scaling (`double`). It has as many elements as there are dimensions in input space plus the number of dimensions of the classification. Then follows a list of scaling factors (`double`). It has as many elements as the previous list. For an exact explanation on how scale factors and threshold are calculated see the section on scaling. Note that these scale factors are the factors that have previously been applied to the data. They will not be applied to the data when loading. If no scaling has been used the above scale factors do not appear.

This is followed by the examples as in the ASCII version 1, but saved as doubles.

C.5 Scaling

Scaling has to be used when values become unmanageable for the optimizer used in the SV Machine. Some values reduce the numerical accuracy to such an extent that no solution can be found anymore.

Scaling a set of numbers N works as follows:

We are given the lower and upper bound (lb,ub) between which the scaling should occur. Find the maximum and minimum value in N: $\max(N)$, $\min(N)$ Calculate the scaling factor: $s = \frac{ub-lb}{\max(N)-\min(N)}$ Calculate the threshold: $t = \frac{lb}{s} - \min(N)$

Scale all samples x:

$x_s = (x + t) \times s$

References

[BS97] C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. *Neural Information Processing Systems*, 9:7, 1997.

[OFG97a] E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines. *NNSP'97*, 1997.

[OFG97b] E. Osuna, R. Freund, and F. Girosi. Training support vector machines:an application to face detection. *CVPR'97*, 1997.

[Sch97] B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997.

[Van] R. J. Vanderbei. Loqo: An interior point code for quadratic programming.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[Vapng] Vladimir N. Vapnik. *Statistical Learning Theory*. J. Wiley, forthcoming.

[VGS] Vladimir N. Vapnik, Steven E. Golowich, and Alex Smola. Support vector method for function approximation, regression estimation and signal processing.