A Small User Guide for *SVMTorch II*

December 18, 2000

1 General Introduction

SVMTorch is a new implementation of Vapnik's Support Vector Machine that works both for classification and regression problems, and that has been specifically tailored for large-scale problems (such as more than 20000 examples, even for input dimensions higher than 100). A simple extension to multiclass problems (using the one-over-all mechanism) is also provided.

2 Arguments of `SVMTorch`

The command line is

SVMTorch [options] example_file model_file

Where "example_file" is the file containing your training examples and "model_file" will be the output model generated by the program.

The options are :

`-help` prints a little message. It prints the same message when you launch *SVMTorch* without argument.

`-multi` for multiclass problems. Your data must be in the multiclass format. See section 4 below.

`-mfc <int>` for multiclass problems. You can specify here the first class to learn. (For example if you train your SVM on a cluster you can train a class on each computer with this option and the next option).

`-mlc <int>` for multiclass problems. You can specify here the last class to learn.

`-rm` for regression problems.

`-c <float> <float>` is the trade-off between training error and margin. Default = 100.

`-eps <float> <float>` is the width of the error pipe for regression problems. Default = 0.5.

`-unshrink` if you want to check the optimality of removed variables at the end of the learning phase. It will possibly restart the optimization. Note that *SVMTorch* will be slower with this option.

`-t <int> <int>` defines the kernel:

- `0` is the linear kernel.
- `1` is the polynomial kernel $(a, b) \mapsto (s\ a \cdot b + r)^d$.
- `2` is the gaussian kernel $(a, b) \mapsto \exp(\frac{-\|a-b\|^2}{std^2})$.
- `3` is the sigmoidal kernel $(a, b) \mapsto \tanh(s\ a \cdot b + r)$.

- 4 is a user defined kernel. See an example in kernel_userc.

-d <int> where <int> is the parameter d in polynomial kernel.

-std <float> where <float> is the parameter *std* in the gaussian kernel.

-s <float> where <float> is the parameter s in the sigmoidal or in the polynomial kernel.

-r <float> where <float> is the parameter r in sigmoidal or in the polynomial kernel.

-u <string> where <string> is a parameter for the user defined kernel.

-m <int> ≥ 1 <int> is the size of the cache (in Mb), for saving kernel evaluations. The more you can put, the faster it should go. Default = 50.

-e <float> <float> is the precision required to stop learning. The default is 0.01, you can try 0.001 but lower values increase *significantly* the training time.

-h <int> <int> is the minimal number of iterations a variable needs to be stuck at bounds to be shrinked. If you set it to a low value, you may have a poor solution. If you set it to a large value, it increases the training time. The default value (100) seems to give good results.

-xs <float> > 0 epsilon value related to shrinking[1]. Default = 10^{-9} in double and 10^{-4} in float.

-xb <float> > 0 is the internal precision to decide if a variable is at a bound. It must be strictly positive. Default = 10^{-12} in double and 10^{-4} in float.

-sparse for the sparse mode. Use it when your data is in the sparse format. See section 4 below.

-bin for binary data format. See section 4.

-load <int> load only <int> examples for training.

On UNIX systems you can make less <model> where <model> is a model generated by *SVMTorch* to watch three lines of information at the beginning of the file. (The rest of the file is in binary).

Note that the default format is in ASCII. No test is made to check if your data is in the correct format. Moreover no test is made to check if your data is in the sparse format or in the default format. Therefore, don't forget to set the correct parameters, otherwise you will have a error message, a segmentation fault, or a strange behaviour.

When you train an SVM machine in multiclass mode, whith n classes, the program will create n models, each model named as the generic name you gave in the command line plus an extension $.i$ where i corresponds to the model for class i $(+1)$ against the others (-1). Note that each generated model can be tested separatly.

Last, if you use training data in sparse mode, the model will be saved in sparse format.

3 Arguments of SVMTest

The command line is

SVMTest [options] model_file test_file

Where "model_file" is an output model generated by the svm_torch and "test_file" is the file containing your test examples.

The options are :

[1] You can have a look at ftp://ftp.idiap.ch/pub/reports/2000/rr00-17.ps.gz for additional information.

-help prints a little message. It prints the same message when you launch *SVMTest* without argument.

-multi for multiclass problems. Your data must be in the multiclass format. See section 4 below.

-norm when in multiclass mode, this option normalizes (with the inverse of the margin norm) the output of each model before deciding in which class the example should be. To avoid time consuming the margin is now approximated by the sum of the α_i, where α_i is the coefficient related to the i$^{\text{th}}$ support vector. (In the separable case it is the same as the margin)

-sparse for the sparse mode. Use it when your data is in the sparse format. See section 4 below.

-bin if your data is in binary, set this flag.

-load <int> load only <int> examples in the test set.

-no when you want to take a decision for data for which you do not have any target (and use the corresponding format without target), set this flag. You must use this option with -oa or -ob.

-oa <file> Prints the output of the model in <file>, in ASCII.

-ob <file> Prints the output of the model in <file>, in binary.

Note that in sparse mode, the data should be in sparse format *as well as* the model.

In multiclass mode the program searches for model files named as the generic name you gave in the command line plus an extension *.i*, as the files generated by SVMTorch. Therefore, it is possible to train separatly several classes against the others, and to save each result to a file as "model.i". When you then launch SVMTest in multiclass mode with model file parameter "model", it will test as if it had been trained in one time. It outputs errors for each model as if they were trained separatly, as well as an overall classification performance.

If you are not in multiclass mode the ouput of the model obtained by -oa or -ob is a column of floating point values corresponding to the output of the SVM, without any modification. In multiclass mode, the floating point values are the predicted class for each example.

4 Input file formats

In this section we use the notation a_{ij} for *the floating point value* corresponding to the j-th value of the i-th example, and t_i *the floating point value* [2] corresponding to the desired target of the i-th example. Moreover n is the number of examples and d is the input dimension of each example. For sparse data, we denote z_{ij} the number of the j-th non-null column [3] of the i-th example.

In standard classification t_i is $+1.$ or $-1.$. In multiclass mode, the classes are numbered [4] $0., 1., \ldots$.

4.1 ASCII mode

There is four different possibilities (Respect the carriage returns!!!) :

- **Standard:**

$$\begin{array}{cccc} n & (d+1) & & \\ a_{11} & \ldots & a_{1d} & t_1 \\ & \vdots & & \\ a_{n1} & \ldots & a_{nd} & t_n \end{array}$$

[2] The type is important, especially in binary !

[3] The column numbers start at 0 and are expressed as integers.

[4] Therefore, if you have five classes, they *must* be numbered $0., 1., 2., 3., 4., 5.$

- **Standard without target:** for `SVMTest` with the `-no` option.

$$\begin{array}{ccc} n & d & \\ a_{11} & \dots & a_{1d} \\ & \vdots & \\ a_{n1} & \dots & a_{nd} \end{array}$$

- **Sparse:**

$$\begin{array}{cccccc} n & (d+1) & & & & \\ d_1 & \dots & z_{1j} & a_{1z_{1j}} & \dots & t_1 \\ & \vdots & & & & \\ d_n & \dots & z_{nj} & a_{nz_{nj}} & \dots & t_n \end{array}$$

  where d_i is the number of non-null values in the i^{th} vector.

- **Sparse without target:** for `SVMTest` with the `-no` option.

$$\begin{array}{ccccc} n & d & & & \\ d_1 & \dots & z_{1j} & a_{1z_{1j}} & \dots \\ & \vdots & & & \\ d_n & \dots & z_{nj} & a_{nz_{nj}} & \dots \end{array}$$

  where d_i is the number of non-null values in the i^{th} vector.

4.2 Binary mode

Now the binary format is exactly the same as the ASCII format, without carriage returns. (But of course the data is in binary!)