

WW-M-SVM for protein sequence processing

User's guide

October 13, 2025

Contents

1	Introduction	3
2	Architecture of the software	3
2.1	Programs	3
2.2	LP solver	3
2.3	How to compile	4
3	Solving problems	4
3.1	Simple examples	4
3.2	Training the M-SVM	4
3.3	Testing the M-SVM	5
3.4	Stopping criteria	6
4	Model selection	7
5	General comments	7

1 Introduction

This application dedicates the multi-class SVM (M-SVM) [4] introduced by Weston and Watkins (WW-M-SVM) [7] to protein sequence processing. Contrary to the implementation considered in [1], the machine is actually affine (and not only linear) in the RKHS induced by the kernel. The central part of the dedication, the kernel, is the one introduced in [5]. This dedication was made with the aim to ensure that minor changes should also make the application suitable for DNA or RNA sequence processing problems (recognition of protein-coding segments, exon recognition, etc.).

The programs are written in C ANSI (with one single python script), and thus can be used under the various releases of UNIX, Linux, IRIX, etc. The training algorithm is a *chunked* version of the Frank-Wolfe algorithm [3].

2 Architecture of the software

2.1 Programs

This application is made up of four main programs. `train_SVM` performs the training of the machine. `eval_SVM` is used to test it on labelled data, whereas `apply_SVM` is used to test it on unlabelled data. At last, `train_SVM` calls a linear programming (LP) solver.

Furthermore, two programs are provided to parameterize the kernel: `project_matrix` and `tune_kernel`. These programs, located in the directory `MSVMpred/Tools`, will be detailed in Section 4.

2.2 LP solver

By default, the solver selected is `lp_solve`, which has been developed by M. Berkelaar and implements an algorithm described in [6]. It is released under the LGPL license. However, the user can replace it with his own favourite solver. The most straightforward way to do so consists in replacing, in the `main` function of `train_SVM`, the call to the function `write_lp_file` with a call to a function specifying the LP problem in another format, for instance a MPS file. Such a file could then be used as input by a large variety of solvers (HOPDM, CPLEX, etc.). In order to use one of them, it suffices to execute in `train_SVM` the corresponding command with a system call (as is done for `lp_solve`), and to adapt in accordance the function `read_lp_sol` which reads the optimal solution of the problem (output of the solver).

This approach, in which the M-SVM program and the LP solver exchange data through files, has been implemented with the idea that this modularity would make things as easy as possible for the user. Obviously, integrating the LP solver significantly speeds up the application.

2.3 How to compile

`tune_kernel`, `train_SVM`, `eval_SVM` and `apply_SVM` can be compiled thanks to the commands: `compile_tune_kernel`, `compile_train_SVM`, `compile_eval_SVM` and `apply_eval_SVM` (the corresponding makefiles are in the respective sub-directories `make`). If the binary `lp_solve` is to be rebuilt, then the corresponding source code can be found at the following address: https://osdn.net/projects/sfnet_lpsolve/downloads/lpsolve/5.5.2.11/lp_solve_5.5.2.11_source.tar.gz/.

3 Solving problems

3.1 Simple examples

A simple way to become familiar with the use of the software consists in running it on the data sets located in the directory `MSVMpred/Data` (see the user's guide of `MSVMpred`). In order to select any of the corresponding problems, it suffices to use the corresponding script, named `configure.name`, where `name` is the name of the problem. Once this is done, the files `Fichcom/train_SVM.com` and `Fichcom/eval_SVM.com` (see below) contain the appropriate parameters, and the file of dual variables is initialized with a feasible solution. Suffice it to use the commands `execute_train_SVM` and `execute_eval_SVM` to start training and evaluate the machine respectively.

3.2 Training the M-SVM

Training is initiated with the command

```
execute_train_SVM
```

In order to specify the nature of the problem to be solved, the file

```
Fichcom/train_SVM.com
```

must preliminary be filled. It is made up of seven lines. Its structure, illustrated on the first training set of the IPM problem, is as follows:

`8.0` \leftarrow value of the soft margin parameter C of the objective function
 (see Problem 1 in the technical documentation)
`8` \leftarrow size of the *chunk* (see Section 5.3 of the technical documentation)
`../Data/IMP1.app` \leftarrow name of the file where the training data is
 stored
`Alpha/IMP.alpha.init` \leftarrow name of the file containing the initial values
 of the dual variables α
`Alpha/IMP1.alpha` \leftarrow name of the file where the updated values of the
 dual variables α are stored during training
`../Theta/theta_IPM_new.txt` \leftarrow name of the file containing the weights
 on the positions of the analysis window
`../Substitution_matrices/substitution_IPM.txt` \leftarrow name of the
 file containing the size of the alphabet and the matrix of dot products be-
 tween amino acids

The function implementing the kernel, named `gaussian`, is located in the
 library `algebre.c`. A feasible solution to initialize the values of the dual
 variables α consists in setting them all equal to 0. Care must be taken to the
 fact that the dummy variables α_{i,y_i} (see for instance [4] and the technical
 documentation) must always remain equal to 0.

3.3 Testing the M-SVM

Testing is initiated with the command
`execute_eval_SVM`

The structure of the file

`Fichcom/eval_SVM.com`

containing the parameters used by the program `eval_SVM`, is similar to the
 structure of the file `Fichcom/train_SVM.com`. Here is an example, still cor-
 responding to the IPM problem (first fold of the cross-validation):

`8.0` \leftarrow value of the parameter C of the objective function
`../Data/IPM1.app` \leftarrow name of the file where the training data is
 stored
`../Data/IPM1.test` \leftarrow name of the file where the test data is stored
`Save_alpha/IPM1.alpha` \leftarrow name of the file where the values of the
 dual variables α are read
`../Theta/theta_IPM_new.txt` \leftarrow name of the file containing the weights
 on the positions of the analysis window

`../Substitution_matrices/substitution_IPM.txt` \leftarrow name of the file containing the size of the alphabet and the matrix of dot products between amino acids

`SV/IPM1.sv` \leftarrow name of the file where the *margin support vectors* and the corresponding categories will be stored

`Data/IPM1.output` \leftarrow name of the file where the output of the M-SVM on the test set will be stored

`Save_alpha/IPM1.b` \leftarrow name of the file where the vector b of the component functions will be stored

The program `eval_SVM` displays different pieces of information. The first of them regard the satisfaction of the constraints of the learning problem (Problem 2 in the technical documentation). Then come elements used to estimate the components of vector b by means of Formula 5 in the technical documentation. This estimation is based on an identification of *margin support vectors*. At last, the program displays the training and test performances.

3.4 Stopping criteria

In order not to slow down the training algorithm, no test is made in the program `train_SVM` regarding the satisfaction of optimality conditions (Kuhn-Tucker conditions...) [2]. However, every 100 iterations, the program displays the number of examples identified as margin support vectors. In the case that the initial (feasible) solution is the null vector, then this number usually increases almost monotonically during the first iterations, before reaching a plateau and then decreasing. A rule of thumb is that as soon as the decrease occurs, it is possible to start testing (use the `eval_SVM` program) and obtain a performance close to that associated with the optimal solution (although this solution could be reached far later).

In the program `eval_SVM`, the ratio between the dual and the primal objective function is estimated. More precisely, the value of the dual objective function is computed, whereas an estimate of (upper bound on) the primal objective function is obtained. Indeed, this latter program can be used at any stage of the training process, including with the initial values of the dual variables, with the drawback that in this case, there is no closed form expression for the optimal value of vector b (Formula (2) in the technical documentation does not hold true). To sum up, given the specificities of the application, we suggest the following three-step strategy to monitor training:

1. start the program `train_SVM`;
2. if the number of examples identified as margin support vectors has started to decrease, then (with `train_SVM` still running) use `eval_SVM` to check whether the feasibility gap is low enough;
3. stop training when the ratio of the objective functions (dual / primal) is superior to 0.99.

4 Model selection

The programs performing the parameterization of the Gaussian kernel are located in the directory `Tools` (just below the root). The command `commande` calls a python script that takes in input the substitution matrix `matrix.txt` and outputs the matrix `projected_matrix.txt` of dot product between amino acids. One value must be added at the beginning of this file prior to its use by the M-SVM: the size of the alphabet/matrix. The program `tune_kernel` computes the positional weighting of the analysis window (for a given matrix of dot product). The structure of its file of parameterization, `Fichcom/tune_kernel.com`, is the following one:

```

22 ← number of symbols of the alphabet
2 ← number of categories
500 ← size of the chunk
../Data/IPM1.app ← name of the file containing the training set
../M-SVM/Theta/theta_IPM1.txt ← name of the file containing the
initial values of the weights
../M-SVM/Theta/theta_IPM_new.txt ← name of the file in which the
new values of the weights will be stored
../M-SVM/Substitution_matrices/substitution_IPM.txt ← name
of the file containing the size of the alphabet and the matrix of dot products
between amino acids

```

5 General comments

Please, feel free to report any suggestions you could have to improve the programs, this document or the technical documentation, to the following address: `Yann.Guermeur@cnrs.fr`.

Acknowledgments The original version of the script deriving the matrix of dot product between amino acids from a substitution matrix is due to T. Malliavin. The program computing the weighting of the analysis window was initially developed by R. Vert. Thanks are also due to A. Iouditski for interesting discussions on the subject and to A. Brun, D. Eveillard, W. da Rocha, N. Sapay and N. Wicker for testing the software and suggesting improvements.

References

- [1] U. Dogan, T. Glasmachers, and C. Igel. A unified view on multi-class support vector classification. *Journal of Machine Learning Research*, 17(45):1–32, 2016.
- [2] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Chichester, second edition, 1987.
- [3] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Res. Logist. Quart.*, 3:95–110, 1956.
- [4] Y. Guermeur. A generic model of multi-class support vector machine. *International Journal of Intelligent Information and Database Systems*, 6(6):555–577, 2012.
- [5] Y. Guermeur, A. Lifchitz, and R. Vert. A kernel for protein secondary structure prediction. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, chapter 9, pages 193–206. The MIT Press, Cambridge, MA, 2004.
- [6] W. Orchard-Hays. *Advanced Linear Programming Computing Techniques*. McGraw-Hill, 1968.
- [7] J. Weston and C. Watkins. Multi-class Support Vector Machines. Technical Report CSD-TR-98-04, Royal Holloway, University of London, Department of Computer Science, 1998.