M-SVM^2 user's guide

Yann Guermeur

November 22, 2009

Contents

1 Introduction

This software is an implementation of the multi-class SVM named M-SVM^2 introduced in [7] (the paper can be found in the subdirectory Tech). It is written in C ANSI, and thus can be used under the various releases of UNIX, Linux, IRIX, etc. The training algorithm is a variant of the Frank-Wolfe algorithm [5] which implements a decomposition strategy.

2 Architecture of the software

2.1 Programs

This application is made up of three main programs. `train_SVM` performs the training of the SVM, whereas `eval_SVM` is used to test the model. `train_SVM` calls a software solving linear programming (LP) problems. This sofware can be chosen by the user.

2.2 Linear programming solver

By default, the solver selected is `lp_solve`, which has been developed by M. Berkelaar and implements an algorithm described in [8]. It is released under the LGPL license. However, since the exchanges between this solver and the M-SVM^2 are performed through files, the user can replace it with his own favourite solver (HOPDM, CPLEX, etc.). This modular approach has been implemented with the aim to make things as easy as possible for the user. Obviously, integrating a LP solver in `train_SVM` should significantly speed up the application.

2.3 How to compile

`train_SVM` and `eval_SVM` can be compiled thanks to the commands:
`compile_train_SVM` and `compile_eval_SVM`
(the corresponding makefiles are in the subdirectory `make`).

`lp_solve` can be found in the directory `LP/lp_solve_3.2`. Suffice it to use the command `make` in this directory in order to compile it. This produces the binary executable file `lp_solve`. Once it has been produced, this file must be copied in the main directory, called `Dev`.

3 Solving multi-class problems

3.1 Three simple examples

A simple way to become familiar with the use of the software consists in running it on the three examples provided. The first one corresponds to the iris data set of Fisher, available at the UCI repository [1]. The second one is a 3-class toy problem borrowed from [3]. The program to generate the corresponding data and display it are located in the subdirectory `Toy`. The third one is the three-Gaussian problem that illustrates [6]. In order to select any of the three problems, it suffices to use the corresponding script, either `configure.iris`, `configure.toy` or `configure.gaussians`. 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 command `execute_train_SVM` to start training. While training is underway, the command `execute_eval_SVM` can be used to assess its progress and compute the training and test performance.

3.2 Structure of the files containing the data

The files containing the data must be text files, with a specific structure. We illustrate this structure on Elisseeff's toy problem. The name of the corresponding file is `Data/toy.app`.

`1000` ⟵ number of points in the set
`2` ⟵ number of components of the vectors coding the input data
`0.781323 0.298303 1` ⟵ description of the first example: two components of the input vector plus the label of the category, here 1.
`...`

3.3 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 aforementioned toy problem, is as follows:

`3` ⟵ number of categories for the problem

`1` ⟵ nature of the kernel

`100.0` ⟵ value of the soft margin parameter C

`12` ⟵ size of the *chunk* (see the technical documentation)

`Data/toy.app` ⟵ name of the file where the training data is stored

`Alpha/toy.alpha` ⟵ name of the file containing the initial values of the dual variables α

`Alpha/toy.alpha` ⟵ name of the file where the updated values of the dual variables α are stored during training

Currently, only three types of kernels are implemented: a linear kernel, a Gaussian kernel and a polynomial one. A value of 1 for the "nature of the kernel" corresponds to a linear kernel (Euclidean inner product), whereas a value of 2 corresponds to a Gaussian kernel and a value of 3 corresponds to a polynomial kernel. The parameters, width of the Gaussian kernel and degree of the polynomial kernel, can be changed by modifying adequately the functions `gaussian` and `polynomial` in the file `algebre.c`. This file can also be used to store the functions corresponding to the new kernels a user could find useful to add. A feasible solution of the dual problem (which can be used to initialize the vector of dual variables α) is the null vector. Care must be taken to the fact that the dummy variables α_{iy_i} must always remain equal to 0. During training, the following pieces of information are displayed:

`*** Iteration: 100` ⟵ number of gradient ascent steps since the beginning of training

`Number of support vectors: 347` ⟵ number of training examples for which at least one of the dual variables α_{ik} is positive (different from 0)

`Quadratic term: 716.623906` ⟵ current value of the quadratic form $\alpha^T \tilde{H} \alpha$

`Linear term: 3725.453030` ⟵ current value of the linear form $\frac{1}{Q-1} 1_{Qm}^T \alpha$

`Objective function: 3367.141077` ⟵ current value of the objective function $J_{\text{M-SVM}^2,\text{d}}(\alpha) = -\frac{1}{2}\alpha^T \tilde{H} \alpha + \frac{1}{Q-1} 1_{Qm}^T \alpha$.

3.4 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 pretty similar to the structure of the file `Fichcom/train_SVM.com`. Here is an example, corresponding once more to the toy problem:

`3` ⟵ number of categories for the problem
`1` ⟵ nature of the kernel
`100.0` ⟵ value of the soft margin parameter C
`Data/toy.app` ⟵ name of the file where the training data is stored
`Data/toy.test` ⟵ name of the file where the test data is stored
`Save_alpha/toy.alpha` ⟵ name of the file where the values of the dual variables α are read
`Data/toy.outputs` ⟵ name of the file where the outputs of the M-SVM^2 will be stored

3.5 Stopping criterion

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 [4], value of the feasibility gap [2], etc. There is only a constant named `nb_iter` which limits the number of iterations (steps in the gradient ascent). Its value can be set utterly arbitrarily. The program `eval_SVM` displays the values of several quantities characterizing the feasibility of the current solution (vector α) and its convergence towards the optimal solution α^*. The easiest way to derive a "cheap" stopping criterion (early stopping, etc.) probably consists in making use of the values of the components of the dual objective function (provided by `train_SVM`): the quadratic form $\alpha^T \tilde{H} \alpha$ and the linear form $\frac{1}{Q-1} 1_{Qm}^T \alpha$. Indeed, let $\alpha^{(t)}$ be the feasible solution obtained after t steps of gradient ascent. We have:

$$\lim_{t \longrightarrow +\infty} \frac{1_{Qm}^T \alpha^{(t)}}{(Q-1) {\alpha^{(t)}}^T \tilde{H} \alpha^{(t)}} = 1.$$

Note that a difficulty springs from the fact that the curves of the two forms as a function of t can intersect themselves before the optimum is attained. Thus, the fact that the ratio above is equal to one is a necessary but not sufficient condition of optimality. In practice, we always observed one single intersection before the optimum was attained (disregarding the initial situation $\alpha = 0$).

4 General comments

This application is intended to be used for academic research purposes only. It is intended to evolve frequently. Please, feel free to report any suggestion you could have to improve the programs or this document to the following address: `Yann.Guermeur@loria.fr`

Acknowledgments The principle of the algorithm implemented by this software is due to A. Elisseeff. Thanks are also due to A. Iouditski for interesting discussions on the subject.

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, 2000.

[3] A. Elisseeff. *Etude de la complexité et contrôle de la capacité des systèmes d'apprentissage : SVM multi-classe, réseaux de régularisation et réseaux de neurones multicouches*. PhD thesis, ENS Lyon, 2000. (in French).

[4] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Chichester, second edition, 1987.

[5] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[6] Y. Guermeur, M. Maumy, and F. Sur. Model selection for multi-class SVMs. In *ASMDA'05*, pages 507–517, 2005.

[7] Y. Guermeur and E. Monfrini. A quadratic loss multi-class SVM for which a radius-margin bound applies. *INFORMATICA*, 2009. (submitted).

[8] W. Orchard-Hays. *Advanced Linear Programming Computing Techniques*. McGraw-Hill, 1968.