WW-M-SVM user's guide

February 23, 2024

Contents

1 Introduction

This application is an implementation of the multi-class SVM (M-SVM) introduced by Weston and Watkins (WW-M-SVM) [8]. Contrary to the implementation considered in [3], the machine is actually affine (and not only linear) in the RKHS induced by the kernel. The programs are written in C ANSI, 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 [5].

2 Architecture of the software

2.1 Programs

This application is made up of three main programs. `train_SVM` performs the training of the machine, whereas `eval_SVM` is used to test it. `train_SVM` calls `lp_solve`, a solver for linear programming (LP) problems. `lp_solve` has been developed by M. Berkelaar and implements an algorithm described in [7]. It is released under the LGPL license.

2.2 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`).

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 multi-class problems

3.1 Simple examples

A simple way to become familiar with the use of the software consists in running it on some of the eleven benchmarks provided. They correspond to the data sets from the UCI repository [1] used to evaluate the *consolidatin kernel* [2]. In order to select any of them, it suffices to use the corresponding script, named `configure.name`, where `name` is the name of the data set as used in the directory `Data` (`abalone`, `car`, `glass`, ...). 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 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 one of the five training sets of the Abalone problem. The name of the corresponding file is `Data/DataXfold/abalone0.app`.

`3342` ⟵ number of points in the set

`8` ⟵ number of components of the vectors coding the input data (descriptions)

`3` ⟵ number of categories

`0.58 0.44 0.175 1.2255 0.5405 0.2705 0.3265 10.0 3` ⟵ description of the first example plus the label of its category (here 3).

`...`

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 six lines. Its structure, illustrated on the aforementioned Abalone problem, is as follows:

`4` ⟵ nature of the kernel

`1.0` ⟵ value of the soft margin parameter C of the objective function (see Problem 1 in the technical documentation)

`10` ⟵ size of the *chunk* (see Section 5.3 of the technical documentation)

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

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

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

The functions implementing the kernels are located in the library `algebre.c`. The only exception is the consolidation kernel. In that case, the corresponding functions are located in the programs `train_SVM` and `eval_SVM`. The values of the hyperparameters are set directly in the code. Four kernels are implemented by default. They are listed in the order of the corresponding value of the variable "nature of the kernel":

1. linear kernel (Euclidean dot product);
2. Gaussian kernel;
3. polynomial kernel;
4. consolidation kernel.

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 [6] and the technical documentation) must always remain equal to 0.

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

`4` ⟵ nature of the kernel

`1.0` ⟵ value of the parameter C of the objective function

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

`Data/abalone.test` ⟵ name of the file where the test data is stored

`Save_alpha/abalone.alpha` ⟵ name of the file where the values of the dual variables α are read

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

`Data/abalone.output` ⟵ name of the file where the outputs of the M-SVM 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.

3.5 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...) [4]. 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 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 choice of the algorithms implemented by this application is borrowed from A. Elisseeff. 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] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] A. El Dakdouki, Y. Guermeur, and N. Wicker. Consolidation kernel. 2024. (in revision).

[3] 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.

[4] R. Fletcher. *Practical Methods of Optimization.* Wiley, 1987.

[5] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Res. Logist. Quart.*, 3:95–110, 1956.

[6] Y. Guermeur. A generic model of multi-class support vector machine. *International Journal of Intelligent Information and Database Systems*, 6(6):555–577, 2012.

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

[8] 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.