Convex Optimization Project

Please send your report and code by email to aspremon@ens.fr

1 Support vector machines solvers

Given m data points $x_i \in \mathbf{R}^n$ with labels $y_i \in \{-1, 1\}$. Write a function to solve the classification problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|_2^2 + C\mathbf{1}^T z \\ \text{subject to} & y_i(w^T x_i) \geq 1 - z_i, \quad i = 1, \ldots, m \\ & z \geq 0 \end{array}$$

in the variables $w \in \mathbf{R}^n$, $z \in \mathbf{R}^m$, and its dual (warning : this problem is a bit different from the one in exercise 1). Solving this problem trains a classifier vector w such that, up to some errors

$$\begin{array}{rcll} w^T x_i & > & 0 & \text{when } x_i = 1 \\ w^T x_i & < & 0 & \text{when } x_i = -1. \end{array}$$

This classifier can then be used to classify new points x as positives or negatives by simply computing the scalar product $w^T x$. [1]

— Use the barrier method to solve both primal and dual problems. Test your code on random clouds of points (e.g. generate two classes of data points by picking two bivariate Gaussian samples with different moments). Try various values of $C > 0$ and measure out-of-sample performance (*i.e.* classification errors on points the algorithm has not seen). Plot duality gap versus iteration number as well as a separation example in 2D (you may add a constant coefficient to the data points x to allow classifiers that do not go through the origin).

— *(Optional)* Use CVX (MATLAB or OCTAVE) or CVXOPT (python), as well as LIBSVM and/or LIBLINEAR to check your results and compare performance.

— *(Optional)* Use the coordinate descent method to solve the dual. Plot duality gap versus iteration number and compare performance with the barrier method for various problem sizes (vary the number of samples and record to total CPU time required by each code to reduce the gap by a factor 10^{-3}).

— *(Optional)* Use the logarithmic barrier code you wrote in HW3 to solve a small random instance of the primal problem using the ACCPM algorithm. Plot an upper bound on the distance to optimality in semilog scale and try various constraint dropping strategies. Compare convergence with the two other methods.

NOTE : Please use *graphics and tables* to illustrate your results as much as possible. You can use MATLAB/OCTAVE or general purpose languages such as PYTHON or JULIA.

1. See http://www.di.ens.fr/~aspremon/PDF/MVA/ApplicationsOne.pdf for details.