Training an SVM in the primal

based on [Chapelle, 2007]

Preamble

In this practical session, the goal is to code (in Octave) a Support Vector Machine for binary classification. The method implemented is the one described in [Chapelle, 2007], which considers an unconstrained, convex and twice-differentiable formulation of the 2-norm soft-margin SVM. In order to solve the learning problem a Newton descent strategy is used.

The effectiveness of the approach will be tested against a 2-dimensional toy dataset.

1 Data preparation

All the code and data will be stored in a directory called `'primalsvm'`.

1. Download the 2-dimensional training and test data at the following locations:
   - Training data and corresponding labels
     - `http://www.lif.univ-mrs.fr/~liva/DONNEES/banana_train_data.asc`
     - `http://www.lif.univ-mrs.fr/~liva/DONNEES/banana_train_labels.asc`
   - Test data and corresponding labels
     - `http://www.lif.univ-mrs.fr/~liva/DONNEES/banana_test_data.asc`
     - `http://www.lif.univ-mrs.fr/~liva/DONNEES/banana_test_labels.asc`
2. Using Octave's `find` and `save` commands, store the set of positive and negative training data in the files `'bananaplus.asc'` and `'bananaminus.asc'`, respectively.
3. Using the `plot` command, plot the training data and, using the `print` command, save the plot in a file called `'banana.eps'` or `'banana.png'` (or whatever extension is suitable with the type of format you prefer).

2 RBF kernels

Program a function `rbfkernel(U,V,sigma)` that computes the RBF kernel of width `sigma` between the n_u data points of `U` and the n_v data points of `V` (each row corresponds to an example): the output of `rbfkernel(U,V,sigma)` is an `n1`×`n2` matrix:

$$\texttt{rbfkernel}\left(U=\begin{bmatrix}\mathbf{u}_1^\top\\ \mathbf{u}_2^\top\\ \vdots\\ \mathbf{u}_{n_u}^\top\end{bmatrix}, V=\begin{bmatrix}\mathbf{v}_1^\top\\ \mathbf{v}_2^\top\\ \vdots\\ \mathbf{v}_{n_v}^\top\end{bmatrix}, \sigma\right) = \underbrace{\left.\begin{pmatrix} k_\sigma(\mathbf{u}_1,\mathbf{v}_1) & k_\sigma(\mathbf{u}_1,\mathbf{v}_2) & \cdots & \cdots & k_\sigma(\mathbf{u}_1,\mathbf{v}_{n_v})\\ k_\sigma(\mathbf{u}_2,\mathbf{v}_1) & k_\sigma(\mathbf{u}_2,\mathbf{v}_2) & \cdots & \cdots & k_\sigma(\mathbf{u}_2,\mathbf{v}_{n_v})\\ \vdots & \vdots & \cdots & \cdots & \vdots\\ k_\sigma(\mathbf{u}_{n_u},\mathbf{v}_1) & k_\sigma(\mathbf{u}_2,\mathbf{v}_2) & \cdots & \cdots & k_\sigma(\mathbf{u}_{n_u},\mathbf{v}_{n_v})\end{pmatrix}\right\}}_{n_v \text{ columns}} n_u \text{ rows}$$

Recall that the RBF kernel k_σ of width σ is such that $k_\sigma(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$.

3 SVM learning

Given a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, the unconstrained 2-norm soft-margin SVM formulation that we are going to work with is:

$$\min_{f \in \mathbb{H}_k, b \in \mathbb{R}} \lambda \|f\|^2 + \sum_{i=1}^{n} |1 - y_i [f(\mathbf{x}_i) + b]|_+^2 ,$$

whose solution provides us with the function

$$\mathbf{x} \mapsto f(\mathbf{x}) + b = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b.$$

The difference with what we saw during the lecture is that, here, we directly work in the Reproducing Kernel Hilbert space associated with kernel k: in the lecture, k was assumed to be the usual (linear) kernel inner product.

Thanks to the representer theorem, we now that:

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$$

(where the $\mathbf{x}_i$'s are from the training set). The minimization problem can be rewritten as

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \lambda \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + \sum_{i=1}^{n} \left|1 - y_i \left[\mathbf{k}_i^\top \boldsymbol{\alpha} + b\right]\right|_+^2 , \tag{1}$$

where $\mathbf{k}_i$ is the i-th column of the Gram matrix of k associated with S, i.e.:

$$\mathbf{k}_i^\top = [k(\mathbf{x}_i, \mathbf{x}_1) \cdots k(\mathbf{x}_i, \mathbf{x}_n)].$$

Let us call F the objective function of (1), that is,

$$F\left(\boldsymbol{\beta} := \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}\right) := \lambda \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + \sum_{i=1}^{n} \left|1 - y_i \left[\mathbf{k}_i^\top \boldsymbol{\alpha} + b\right]\right|_+^2 . \tag{2}$$

F is convex and twice-differentiable with respect to $\boldsymbol{\beta}$.

In order to solve (1), we are going to implement a Newton descent procedure. Such minimization scheme relies on the minimization of successive second order approximations of the objective function under consideration. Namely, it is an iterative process that generates a sequence of points $\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots, \boldsymbol{\beta}^t, \ldots$ such that $\boldsymbol{\beta}^{t+1}$ is the minimum of the following second-order approximation $\tilde{F}$ of F at $\boldsymbol{\beta}^t$:

$$\tilde{F}_t(\boldsymbol{\beta}) = F\left(\boldsymbol{\beta}^t\right) + (\boldsymbol{\beta} - \boldsymbol{\beta}^t)^\top \nabla_t + \frac{1}{2} \left(\boldsymbol{\beta} - \boldsymbol{\beta}^t\right)^\top H_t \left(\boldsymbol{\beta} - \boldsymbol{\beta}^t\right), \tag{3}$$

where

- ∇_t is the gradient of F (i.e. the vector of all partial derivatives) at $\boldsymbol{\beta}^t$:

$$\nabla_t = \left(\left. \frac{\partial F}{\partial \beta_i} \right|_{\boldsymbol{\beta} = \boldsymbol{\beta}^t} \right)_i ,$$

- H_t is the Hessian of F (i.e. the matrix of second order derivatives) at $\boldsymbol{\beta}^t$:

$$H_t = \left(\left. \frac{\partial^2 F}{\partial \beta_i \partial \beta_j} \right|_{\boldsymbol{\beta} = \boldsymbol{\beta}^t} \right)_{ij} .$$

As F is convex, H_t is (semi-)positive definite and finding the minimum of $\tilde{F}_t$ (cf Equation 3) is just easy: it suffices to compute the gradient and to make it be equal to $\mathbf{0}$. This gives:

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t - H_t^{-1}\nabla_t. \tag{4}$$

This equation defines the iterative scheme of the optimization procedure: starting from an arbitrary $\boldsymbol{\beta}^0$, it is used to compute the iterates $\boldsymbol{\beta}^t$ that converge to the solution of (2).

In order to program an SVM, follow the following steps.

1. At every step of the optimization process, there will be n_{sv} points $\mathbf{x}_i$ such that $y_i[\mathbf{k}_i^\top \boldsymbol{\alpha}^t + b^t] < 1$ (recall that $\boldsymbol{\beta}^t = [b\ \boldsymbol{\alpha}^\top]^\top$), that we call *support vectors*. At step t, the matrix I_t is the $n \times n$ diagonal matrix having ones only on the n_{sv} diagonal entries corresponding to the indices of the support vectors.

   (a) Show that the Hessian matrix H is given by

   $$H_t = 2 \begin{pmatrix} \mathbf{1}^\top I_t \mathbf{1} & \mathbf{1}^\top I_t K \\ K I_t \mathbf{1} & \lambda K + K I_t K \end{pmatrix}.$$

   (b) Show that the gradient ∇_t is given by

   $$\nabla_t = H_t \boldsymbol{\beta}^t - 2 \begin{pmatrix} \mathbf{1}^\top \\ K \end{pmatrix} I_t Y.$$

   Here, $\mathbf{1}$ is an n-dimensional vector of 1's, Y is the n-dimensional vector of labels, and K is the Gram matrix of the input data.

2. Implement the Newton optimization procedure. Given some small $\varepsilon > 0$, for instance, $\varepsilon = 10^{-5}$, the algorithm stops when $\nabla_t^\top H_t^{-1} \nabla_t \leq \varepsilon$.

Note: there might be instability problems because of a bad conditioning of H_t, this might be solved by adding a small ridge—i.e. a positive value on the diagonal— on the Gram matrix K.

4 Testing, scaling, sparsifying

1. Measure the classification accuracy of your learned SVM on the test data. Try different values of kernel width σ and regularization parameter λ.

2. Switch the training data and the test data. (The number of test data is 10 times as big as the number of training data.) Evaluate the speed of the learning procedure.

3. (A little bit of thinking.) At the end of the procedure, the learned function

   $$\mathbf{x} \mapsto \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b$$

   will probably be expressed in terms of all training data, i.e. many α_i's will be nonzero. Here is a quick (and dirty?) way to sparsify the solution:

   (a) Choose $n_0 < n$: this will define the size of the kernel expansion we are going to look for. Randomly select n_0 points $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{n_0}}$ from S and find parameters $\boldsymbol{\theta} = [\theta_1 \cdots \theta_{n_0}]^\top$ that minimize

   $$\left\| \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot) - \sum_{j=1}^{n_0} \theta_j k(\mathbf{x}_{i_j}, \cdot) \right\|^2.$$

(b) Test the quality of the resulting classifier

$$\mathbf{x} \mapsto \sum_{j=1}^{n_0} \theta_j k(\mathbf{x}_{i_j}, \mathbf{x}) + b.$$

References

[Chapelle, 2007] Chapelle, O. (2007). Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178.