Support Vector Machines

Elbarbari Mahmoud 1773784, Feola Luigi 1772197

January 10, 2018

Introduction

In this assignment we implement training methods for Support Vector Machines (supervised learning) applied to classification problems, namely for an health application . We optimize the model using different optimization applications e.g. quadratic problem optimization, or decomposing the problem, setting a working set with the most violating pair (MVP - *Most Violating Pair*) or more than one (SMO - *Sequential Minimal Optimization*). The data set is given, and in order to train the machine we split the dataset in training and test set (where in percentage are respectively 80 - 20 %)

1 Dual quadratic problem

Problem description In this part, we compute the parameters α that minimizes the following non linear SVM dual problem:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to} \quad & y^T\alpha = 0 \\ & 0 \leq \alpha \leq Ce \end{aligned}$$

Training For the training phase we split the input dataset to 80% for the training set and 20% for the test set, splitting in a consistent way also the percentage of labels equals to "+1" or "-1". Scaling before applying SVM is very important. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges, and also to avoid numerical difficulties during the calculation. We apply a scaling that center the dataset to the mean and component wise scale to unit variance.
The model is trained using scipy.optimize.minimize routine using "SLSQP" (Sequential Least Squares Programming) optimization method.
As the number of features is small, one often maps data to higher dimensional spaces, i.e. using nonlinear kernels. For this reason we implement rbf kernel. We start with $\alpha_0 = [0, ..., 0]^T$, in this way the gradient at first iteration is $\nabla f(\alpha_0) = Q\,\alpha_0 - e = -e$, where e is a vector of all ones, as big as α. We can do this because the function is convex, so we can start wherever, and for sure we can reach the minimum of this function.

Hyper-parameters tuning To choose the optimal set of the hyper-parameters (C,γ), we do grid search over reasonable range values. First in order to get the feeling of how each hyper-paramter value affects the optimization problem, we first pick some values of hyper-paramters which we think is reasonable and observe the results and then pick some extreme cases and again observe the results. This procedure helps us to define the reasonable ranges on which we will do the grid search and they are as following:

$$C = [10^{-2}, ..., 10^{6}] \qquad \gamma = [10^{-4}, ..., 10^{4}]$$

Underfitting/Overfitting After grid search is possible to see events where underfitting/overfitting happen. For example when C is too small (namely C=[0.01,0.02]) looking accuracy on training and test set, where both are equals to 58%, we must state that there is underfitting. When C is again too small,i.e. 0.258086154042 while γ is 0.581709132937, the accuracy on training and test set are respectively equals to 100% and 63%, this means that we have overfitting, because the machine learns too much the training set, so becomes unable to classify different value from it, the test set. This behavior appears when C = 0.258086154042 and for $\gamma \geq$0.0666. When C is too big, namely when C = 10^6, there is again overfitting, with high accuracy on training set (100%), and small accuracy on test set (about 60%). There are lots of non general cases in which we can discuss about overfitting/underfitting, but are omitted for simplicity.

Result We do this preceding procedure for all the combination of values of hyper-parameters and we choose the one with the best test set accuracy value, each time fixing one hyper-parameter improving the other one. The best combination of hyper-parameters is:

$$C = 6.5 \qquad \gamma = 0.24$$

Then we use this set of hyper-paramters values to train the model on the full training set. And then we evaluate the accuracy on the test set. So we will obtain:

- Optimization terminated successfully
- Training objective function: -102.05649768135328
- Training accuracy: 0.991935483871 Test accuracy: 0.861702
- Training computing time: 2.499383
- Function evaluations: 21 Gradient evaluations: 18

2 Decomposition methods: SVM^{light}

Problem description In this part we consider solving the SVM dual problem by decomposing the original problem into subproblems. Each subproblem consists of selecting a working set W^k of cardinality equals to q which must be an even number, and solving the dual problem in only a subset of the parameters α, that in the working set, while fixing the rest, so it chooses the q/2 most violating pairs. The algorithm proceeds by solving a subproblem in each iteration until the stopping criteria met or maximum iterations (2000) exceeded.

Defining the following entities:

$$R = [i : \alpha_i < C, y_i = +1] \cup [i : 0 < \alpha_i, y_i = -1]$$

$$S = [i : \alpha_i < C, y_i = -1] \cup [i : 0 < \alpha_i, y_i = +1]$$

$$m(\alpha^k) = \max_{i \in R(\alpha^k)} \left\{ -y_i (\nabla f(\alpha^k))_i \right\} \qquad M(\alpha^k) = \min_{i \in S(\alpha^k)} \left\{ -y_i (\nabla f(\alpha^k))_j \right\}$$

the stopping criteria is verified when the difference between m and M is small enough, namely with a tolerance of 10^{-3}, this means that KKT conditions are satisfied so the function reaches the optimal solution. In each iteration we choose the working set picking up the first q/2 element from the sets R and S. Notice that these two vectors, that are indexes arrays, are ordered in a specific way. R is an array of q/2 elements, ordered such that its values correspond to the decreasing order of the quantity $[-\nabla f(\alpha^k)_{i(k)}/y_{i(k)}]$ with $i(k) \in R(\alpha^k)$. While S, again an array of q/2 elements, is ordered such that its values correspond to the increasing order of the quantity $[-\nabla f(\alpha^k)_{j(k)}/y_{j(k)}]$ with $j(k) \in S(\alpha^k)$ (much details are available on Teaching Notes, or just looking the code).
Then solve the following subproblem:

$$\begin{aligned} \underset{\alpha_w}{\text{minimize}} \quad & \frac{1}{2} \alpha_w^T Q_{ww} \alpha_w + (\alpha_{\overline{w}}^T Q_{\overline{w}w} - e_w^T) \alpha_w \\ \text{subject to} \quad & y_w^T \alpha_w = -y_{\overline{w}}^T \alpha_{\overline{w}} \\ & 0 \leq \alpha_w \leq C e_w \end{aligned}$$

and updating α according to:

$$\begin{cases} \alpha_i^{k+1} = \alpha_i^*, & \text{if } i \in W^k \\ \alpha_i^{k+1} = \alpha_i^k, & \text{otherwise} \end{cases}$$

and the gradient according to:

$$\nabla f(\alpha^{k+1}) = \nabla f(\alpha^k) + \sum_{i \in W^k} Q_i (\alpha^{k+1} - \alpha^k)$$

Training We use SLSQP method for the constrained optimization subproblems with the default parameters gradient tolerance and maximum iteration.
We compute the gradient of the subproblems manually according to:

$$\nabla f(\alpha_w^k) = \alpha_w^T Q_{ww} + (\alpha_{\overline{w}}^T Q_{\overline{w}w} - e_w^T) \alpha_w$$

Results

- Optimization terminated successfully (maximum iteration not exceeded)
- Training objective function: -102.056497
- Training accuracy: 0.991935483871 Test accuracy: 0.861702
- Training computing time: 2.039375
- Function evaluations: 568 Gradient evaluations: 551
- Outer iterations: 142

Comparison The default value of q in SVM^{light} is 10, for this reason we run SVM^{light} algorithm with q = [2,8,10,12] (2 is chosen to compare SVM^{light} with MVP algorithm, deep discussion is in the following). We notice that the objective function and test accuracy are equals for each value of q. The difference are visible looking fig(1). Increasing the value of q we notice that each subproblems becomes bigger, so we have less outer iterations, function evaluations and gradient evaluations. When q, so the subproblems, becomes too small or too big, the computing time increases a lot.

3 MVP

Problem description Imposing q = 2 SVM^{light} degenerate in MVP algorithm, both of them reach the same solutions. Here we implement the ASP (Analytic Solution Procedure) algorithm in order to find the analytic solution of the subproblems. So given α_w (component of α in the working set W_k=[i, j]) and the feasible direction $\overrightarrow{d}^{i,j} = (y_i, -y_j)^T$, we can compute the solution of each subproblem, namely we can update α according to:

$$\alpha^{k+1} = \alpha^k + t^* d^*$$

where t^* and d^* are respectively the best value, step size and direction, to the solution for that subproblem.

Training So we implement the MVP (Most Violating Pair) algorithm. The working set is selected building the R and S sets, as in the previous point. The working set now contains only two indexes. As stopping criterion for optimality and as maximum iteration allowed we choose the same as in point 2, namely when M is close enough to m, and the outer iteration does not exceeded 2000.

Results

- Optimization terminated successfully (maximum iteration not exceeded)
- Training objective function: -102.056497
- Training accuracy: 0.991935483871 Test accuracy: 0.861702
- Training computing time: 11.671749
- Outer iterations: 877

Comparisons

Looking the following tabular, fig(2), and as discussed before, we see that the three algorithms reach the optimum, so the same values for objective function, training and test accuracy. The difference are visible especially looking the computational time needed. This is due to the dimension of the problem and to the outer iteration. Of course bigger is the dimension of the problem, bigger will be the time needed to solve it. On the other hand decomposing too much means run the algorithm too much time, so we need more iterations for convergence.

q	**Training time**[s]	**Func eval**	**Grad eval**	**Outer its**
2	11.795293	1924	1898	883
8	3.771996	746	728	220
10	3.002348	652	637	182
12	1.966193	568	561	142

Figure 1: SVM^{light} comparison

		C,γ	**Training accuracy**	**Test accuracy**	**Training time**[s]	**Func eval**	**Grad eval**	**Outer its**
Q1.1	Full QP	6.5, 0.24	0.991935483871	0.861702	2.690672	21	18	//
Q1.2	Dec. QP	6.5, 0.24	0.991935483871	0.861702	2.236544	568	551	142
Q1.3	MVP	6.5, 0.24	0.991935483871	0.861702	11.277205	//	//	877

Figure 2: Method comparison