Accelerating SVM training: beyond SMO

Álvaro Barbero
Joint work with Jorge López and José R. Dorronsoro

Dpto. de Ingeniería Informática and Instituto de Ingeniería del Conocimiento
Universidad Autónoma de Madrid, 28049 Madrid, Spain

October 29, 2010

Support Vector Machines

- Standard, robust method for classification.
- Extensions for regression and novelty detection.
- Very fast algorithms available (PEGASOS, LIBLINEAR) for the linear case.
- For the non-linear case, a dual optimization problem is solved.

SVM optimization problem

$$\begin{array}{ll} \min_x & \frac{1}{2}x^T K x - x \cdot p \\ \text{s.t} & \begin{cases} 0 \leq x \leq C \\ x \cdot y = \Delta \end{cases} \end{array}$$

- Very simple problem: quadratic objective and linear constraints.
- Standard, well understood algorithms from optimization theory available: Inner Point methods, Projected Newton, etc.
- However...

Problems

- Methods with fast convergence require using Hessian or inverse of Hessian information.
- Hessian K size of the dataset and non-sparse, K^{-1} costly to compute ($O(N^3)$).
- Prohibitive for medium-sized problems.

$$\min_x \quad \frac{1}{2}x^T K x - x \cdot p$$
$$\text{s.t} \quad \begin{cases} 0 \leq x \leq C \\ x \cdot y = \Delta \end{cases}$$

Sequential Minimal Optimization

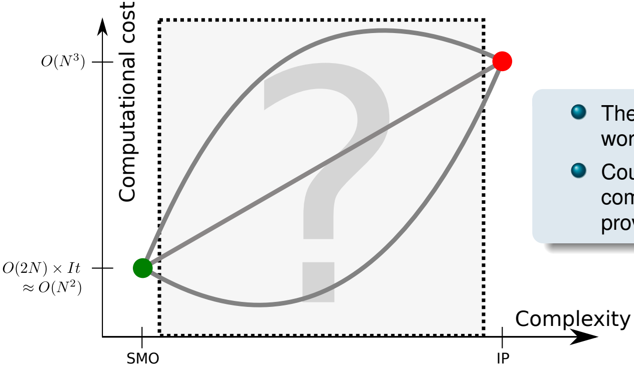
- State of the art algorithm, implemented in LIBSVM.
- At each iteration, update only the two “most violating” entries of x.
- Large number of iterations, but each of them at linear cost.
- Only 2 rows of K are used at each iteration: allows for K larger than memory.

The SMO algorithm

1. $x \leftarrow 0$, compute $\nabla f(x)$.
2. Find aprox. “best” updating direction d with 2 non-zero entries $(O(2N))$.
3. Compute optimal stepsize δ $(O(1))$.
4. Update $x' = x + \delta d$ $(O(2))$.
5. Update gradient $\nabla f(x)$ $(O(2N))$.
6. Back to 2 until convergence.

- Step 2 can be (roughly) done by looking for the largest entries of $\nabla f(x)$ taking restrictions into account, and selecting the two best ones.

The big picture



- The more complex the worse?
- Could intermediate complexity algorithms provide better results?

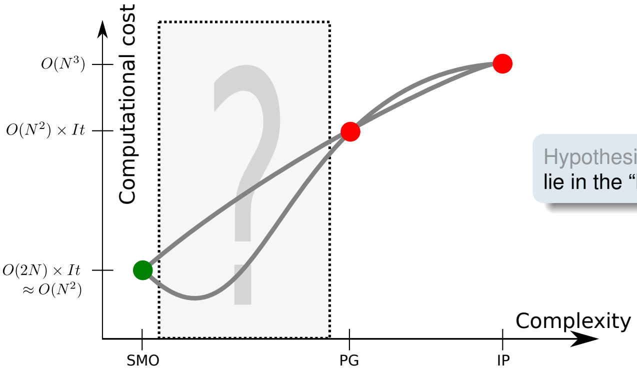
Projected Gradient

- IP is costly because it requires full Hessian information.
- SMO is very simple because it only uses 2 entries of the gradient.
- Natural intermediate algorithm: **projected gradient**.

1. $x \leftarrow 0$, compute $\nabla f(x)$.
2. Compute optimal stepsize δ ($O(N^2)$).
3. Update and project back: $x' = [x + \delta d]_P$ ($O(N)$).
4. Update gradient $\nabla f(x)$ ($O(N^2)$).
5. Back to 2 until convergence.

- Steps 2 and 4 involve a cost $O(NM)$, M number of non-zero components in d.
- Moral: sparsity in the updating direction is desirable. PG non-sparse.

The big picture revisited



Hypothesis: improvements should lie in the “low-complexity” area.

We should...

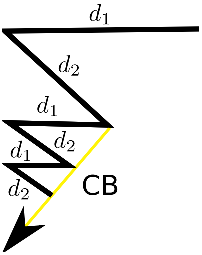
- ... avoid using the full Hessian.
- ... generate sparse updating directions d.
- ... find a balance between sparsity and usefulness of d.

Two algorithms proposed

Cycle-Breaking $\longrightarrow$ Momentum SMO

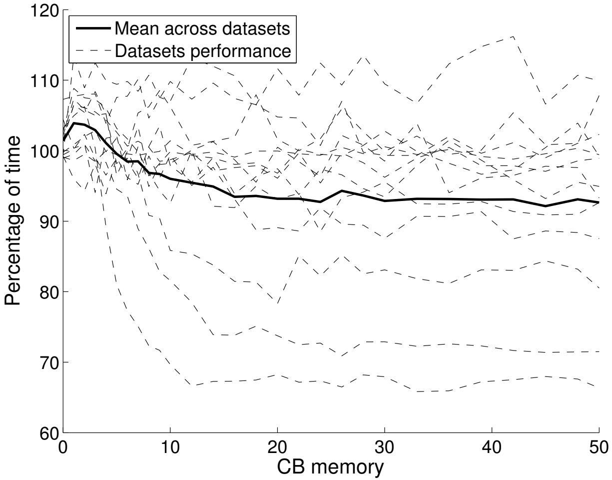
Cycle-Breaking

- "Zigzagging" is common in SMO.
- A sequence of updating directions $d_1, d_2, \ldots, d_M$ appears repeatedly during the run of the algorithm $\longrightarrow$ Cycles.
- If after doing updates along $d_1, d_2, \ldots, d_M$, SMO selects d_1 again for update, it might well happen that afterwards we will have again $d_2, \ldots, d_M$.



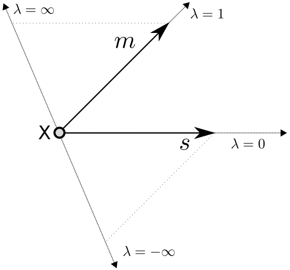
- Keep track of the τ last updating directions in a queue.
- If current updating direction is present in the queue, suppose a cycle is going on.
- Update following the direction of the cycle v (sum of previous updates).
- Sparsity is guaranteed through τ.
- Cost of a cycle-breaking update: $O(N \times \tau^2)$.

Cycle-Breaking results



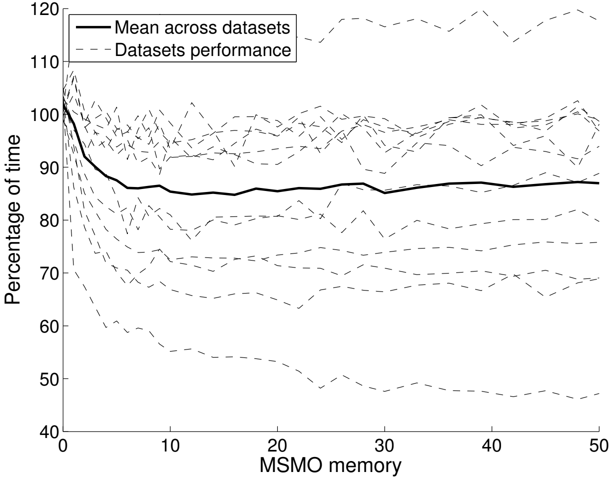
Momentum SMO

- Neural Networks: a momentum term helps to capture the "general direction" of movement.
- Classic momentum: $d_t = (1 - \lambda_t)s_t + \lambda_t m_t$, s_t SMO update, $m_t = x_t - x_{t-1}$.
- m_t non-sparse for t large.
- Limited momentum: only τ past updates, $m_t = \sum_{r=t-\tau}^{t-1}(1 - \lambda_t)\delta_t s_t$.

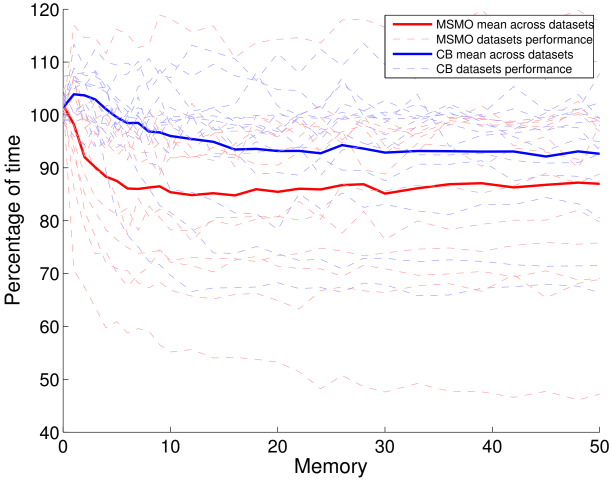


- Update as $x_{t+1} = x_t + \delta((1 - \lambda_t)s_t + \lambda_t m_t)$.
- Both the tradeoff parameter λ_t and the updating step δ_t computed in closed form.
- Optimization along a 2D halfspace.
- By storing calculations from τ previous iterations, cost $\approx O(5N)$ per iteration.

Momentum SMO results

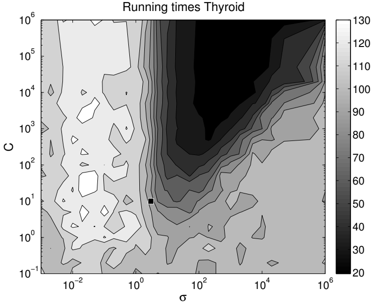


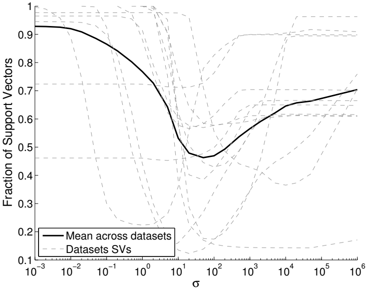
Methods comparison



Why the methods behave better in some datasets?

$\longrightarrow$ Structure of the kernel space

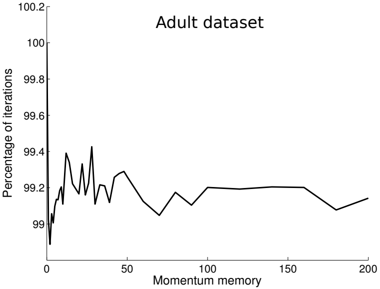


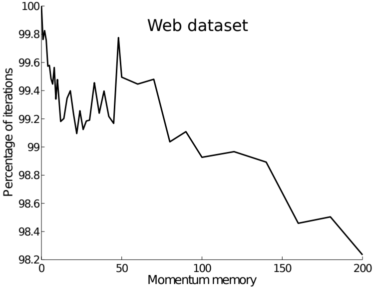


- Large C $\longrightarrow$ "unbounded problem" $\longrightarrow$ fewer hits with boundaries.
- Intermediate σ $\longrightarrow$ less SV $\longrightarrow$ smaller effective dimension.

Drawbacks

- The savings are not large enough to overthrow standard SMO.
- These methods seem to work poorly for large datasets.





Currently working on

- Adding shrinking techniques to the method $\longrightarrow$ reduce the effective dimensionality of the problem.
- For quadratic functions (like SVM) momentum with specific choices of δ, λ can be shown to be equivalent to the Conjugate Gradient method. Might be applicable here.

I will appreciate any suggestions / feedback.

Thanks for your attention

Escuela Politécnica Superior - Universidad Autónoma de Madrid