Acceleration Techniques for Classic Convex Optimization Problems

Alberto Torres Barrán

Universidad Autónoma de Madrid

September 12, 2017
Outline

1. Introduction

2. Lasso

3. SVM

4. Algorithms

5. Solving the Lasso via SVMs

6. Acceleration methods for SMO

7. Experiments

8. Conclusions
Outline

1. Introduction

2. Lasso

3. SVM

4. Algorithms

5. Solving the Lasso via SVMs

6. Acceleration methods for SMO

7. Experiments

8. Conclusions
Predictive modelling

- Assume we have $n$ observations (also samples or examples) of $d$ variables (also features, explanatory variables, covariates, independent variables, ...)

- We gather them in a $n \times d$ matrix, $X$, called the data matrix

- For each observation we have an outcome (also output, target, response variable, dependent variable, ...)

- We gather all the outcomes in a $n \times 1$ vector $y$

- Our goal is predicting the outcome for new observations
Linear regression

- If the outcome is continuous, the simplest model is linear regression:

  \[ y = Xw \]

- The goal is to estimate the optimal weights (also coefficients)

- The most popular method is Ordinary Least Squares (OLS):

  \[ \hat{w} = \arg\min_w \|Xw - y\|_2^2 \]

- Under some assumptions, it has an unique closed-form solution:

  \[ \hat{w} = (X^\top X)^{-1}X^\top y \]
Regularization

- Regularization helps with **overfitting** by limiting the complexity of the model.

- The simplest example is Ridge Regression,

  \[
  \hat{w} = \arg\min_w \|Xw - y\|_2^2 + \lambda\|w\|_2^2
  \]

- \( \lambda \) is a **hyper-parameter** that controls the amount of regularization.

- Regularized learning is a very flexible framework and many models in ML can be expressed as minimizing a loss function + penalty:

  \[
  \min_w \ell(w) + r(w)
  \]
Maximum-margin classification

▶ If the target \( y \) is discrete, then we have a classification problem

▶ A common case is binary classification, where there are only 2 possible values (classes) for the target

▶ SVMs are a natural way to solve binary classification problems

▶ They find the maximum-margin separating hyperplane, assuming \( y_i \in \{-1, 1\} \),

\[
\min_{w, b} \quad \frac{1}{2} \|w\|^2_2 \\
\text{s.t.} \quad y_i(x_i^\top w + b) - 1 \geq 0 \quad \forall i
\]
Maximum-margin hyperplane

Duality

- For every constrained optimization problem we can define another equivalent one, called dual problem

- It often has complementary properties

- In some cases the original problem and the dual have the exact same solution (strong duality):
  - convex objective and affine constraints, $h(x) = Ax - b$
  - convex objective and Slater’s condition is satisfied

- In those cases we can choose to solve one or the other indistinctly
Outline

1. Introduction

2. Lasso

3. SVM

4. Algorithms

5. Solving the Lasso via SVMs

6. Acceleration methods for SMO

7. Experiments

8. Conclusions
Lasso problem

- **Lagrange formulation (U-Lasso):**

  \[
  \min_w \|Xw - y\|_2^2 + \lambda \|w\|_1
  \]

- **Original formulation (C-Lasso):**

  \[
  \min_w \|Xw - y\|_2^2 \quad \text{s.t.} \quad \|w\|_1 \leq t
  \]

- They are both equivalent (more later)

- The \(\ell_1\)-norm induces sparsity in the final coefficients

- Regression model, but also performs feature selection at the same time
Constraint regions

Lasso (a) and Ridge (b) estimates [Tibshirani, 1994]
Extreme cases of Lasso solutions

- Let \( \mathbf{w}_0 \) be the OLS estimator for the Linear Regression problem and \( t_0 = \| \mathbf{w}_0 \|_1 \)

- For every \( t > t_0 \) the solution of C-Lasso is also \( \mathbf{w}_0 \)

- To obtain the same solution in U-Lasso simply take \( \lambda = 0 \)

- On the other hand, if \( t = 0 \), the solution of C-Lasso is 0

- Increasing the value \( \lambda \) in U-Lasso gives sparser solutions

- There is a critical value \( \lambda_{\text{max}} \) such that the solution is the vector with all zeros
Equivalence between C-Lasso and U-Lasso

▶ What happens for all the values of $t$ and $\lambda$ in the middle, $0 < t < t_0$ and $0 < \lambda < \lambda_{\text{max}}$?

▶ **U-Lasso to C-Lasso**: if $w^*$ is a solution of U-Lasso, then taking $t = \|w^*\|_1$ is also a solution of C-Lasso

▶ **C-Lasso to U-Lasso**: strong duality holds since the objective function and the constraints are convex and it is strictly feasible. Using Lagrange theorem, there exists some $\lambda > 0$ such that any solution $w^*$ of C-Lasso minimizes

$$\|Xw - y\|_2^2 + \lambda(\|w\|_1 - t),$$

so $w^*$ is also a solution of U-Lasso.
Outline

1. Introduction
2. Lasso
3. SVM
4. Algorithms
5. Solving the Lasso via SVMs
6. Acceleration methods for SMO
7. Experiments
8. Conclusions
Soft-margin SVM

- The SVM can be extended to the case where the classes are not separable by an hyperplane

- We introduce slack variables $\xi_i \geq 0$ that allow for a point to be inside de margin, $0 < \xi_i \leq 1$, or missclassified $\xi_i > 1$

- We want to still maximize the margin while minimizing the missclassified points, which are upper-bounded by $\sum \xi_i$,

$$
\begin{align*}
\min_{\mathbf{w}, b, \xi} & \quad \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum \xi_i \\
\text{s.t.} & \quad y_i (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 - \xi_i \quad \forall i \\
& \quad \xi_i \geq 0 \quad \forall i
\end{align*}
$$

Alberto Torres Barrán
Slack variables

\[ x_2 \]

\[ \frac{\xi_3}{\|w\|} \]

\[ \frac{\xi_2}{\|w\|} \]

\[ \frac{\xi_1}{\|w\|} \]

Source
SVM regularized formulation

- Soft-margin SVMs can also be written in the loss + penalty formulation,

$$\min_{\mathbf{w}, b} \sum_{i=1}^{n} \max\{1 - y_i f(\mathbf{x}_i), 0\} + \lambda \|\mathbf{w}\|_2^2,$$

where $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} + b$ and $\lambda = 1/(2C)$.

- The loss function is the “hinge-loss”, defined for an output of $y_i \in \{-1, 1\}$ as

$$\ell(\mathbf{z}) = \max\{1 - y_i \mathbf{z}, 0\}$$
SVM dual formulation

- The dual formulation of the soft-margin SVM is

\[
\min_{\alpha} \frac{1}{2} \alpha^\top Q \alpha - \alpha^\top 1 \\
\text{s.t.} \quad \sum \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \; i = 1, \ldots, n
\]

where \( Q \) is a \( n \times n \) matrix with entries \( Q_{ij} = y_i y_j x_i^\top x_j \)

- It can be easily extended to the non-linear case replacing the dot products by a kernel,

\[
k(x_i, x_j) = \phi(x_i)^\top \phi(x_j),
\]

where \( \phi(\cdot) \) are non-linear functions
Nearest Point Problem

- The convex hull of a set $\mathcal{X} = \{x_i\}^n_1$ of points is the set of all convex combinations of points in $\mathcal{X}$,

$$\text{conv}(\mathcal{X}) = \left\{ \sum_{i=1}^{n} \alpha_i x_i \middle| 0 \leq \alpha_i \leq 1, \forall i, \sum_{i=1}^{n} \alpha_i = 1 \right\}$$

- The Nearest Point Problem can be defined as finding the closest point between two convex hulls,

$$\min_{\alpha} \left\{ \frac{1}{2} \left\| \sum_{y_i=1} \alpha_i x_i - \sum_{y_j=-1} \alpha_j x_j \right\|^2 \right\} \quad \text{s.t.} \quad 0 \leq \alpha \leq 1$$
SVM geometric interpretation

- The maximal-margin hyperplane bisects the segment joining the two closest points in the convex hulls spanned by the positive and negative data points.

- It can be shown that the dual SVM and the NPP are essentially equivalent problems.
Outline

1. Introduction
2. Lasso
3. SVM
4. Algorithms
5. Solving the Lasso via SVMs
6. Acceleration methods for SMO
7. Experiments
8. Conclusions
Gradient Descent

- Computing the inverse in the closed-form solution of Ridge Regression is very expensive

- If $d$ is very big, it is better to use iterative methods

- Given a differentiable convex function $f(x)$, Gradient Descent iterates,

$$x^{k+1} = x^k - \eta \nabla f(x^k)$$

- If $\eta$ is small enough the value of the objective function decreases monotonically at every iteration

- The sequence $\{x^k\}$ is also guaranteed to converge to the minimum $x^*$ with rate, $O(1/k)$
Coordinate Descent

- Gradient Descent only works for smooth functions

- The $\ell_1$-norm is not differentiable

- One of the most popular algorithms is Coordinate Descent

- Selects an index $j \in \{1, \ldots, d\}$ and partially optimizes with respect to that coordinate

- If we cycle through all the coordinates several times it converges to the minimum

- In the case of the Lasso the partial optimization problem has a closed-form solution [Friedman et al., 2010]
Coordinate Descent for the Lasso

- Assume we have coefficients $\tilde{w}_l$ for $l \neq j$

- If $\tilde{w}_j \neq 0$, the partial derivative at $\tilde{w}_j$ is

$$\frac{\partial F}{\partial w_j} = -\frac{1}{n} \sum_{i=1}^{n} x_{ij} (y_i - x_i^T \tilde{w}) + \lambda \text{sign}(w_j)$$

- The update is

$$\tilde{w}_j = S_\lambda \left( \frac{1}{n} \sum_{i=1}^{n} x_{ij} \left( y_i - \sum_{l \neq j} x_{il} \tilde{w}_l \right) \right)$$

where $S_\lambda(\cdot)$ is the soft-thresholding operator

$$S_\gamma(z) = \text{sign}(z) \max\{(|z| - \gamma), 0\} = \text{prox}_{\gamma|.|}(z)$$
Sequential Minimal Optimization

- SMO is probably the state-of-the-art algorithm to solve non-linear SVMs [Chang and Lin, 2011]
- Solves the dual formulation, since the constraints are simpler and it is easy to include the non-linear version
- Similarly to Coordinate Descent, it optimizes only two coefficients at every iteration
- However, the coefficients are selected using heuristic rules
- Those rules try to maximize approximately the decrease in the objective function
Outline

1. Introduction
2. Lasso
3. SVM
4. Algorithms
5. **Solving the Lasso via SVMs**
6. Acceleration methods for SMO
7. Experiments
8. Conclusions
Reparametrizing the $\ell_1$-ball

- We can parametrize the $\ell_1$-ball,

$$\diamondsuit_d = \left\{ x \in \mathbb{R}^d \mid \|x\|_1 \leq 1 \right\}$$

as the $2d$-dimensional simplex of probability vectors

$$\triangle_{2d} = \left\{ \tilde{x} \in \mathbb{R}^{2d} \mid 0 \leq \tilde{x} \leq 1, \sum \tilde{x}_i = 1 \right\}$$

- Formally, any vector $x \in \diamondsuit_d$ can be written as

$$x = (I_d \mid -I_d) \tilde{x}$$

where $\tilde{x} \in \triangle_{2d}$ and $(A \mid B)$ denotes the horizontal concatenation of matrices $A$ and $B$
After scaling the C-Lasso by $1/t$, we can apply the previous trick to obtain the equivalent problem,

$$
\min_{\tilde{w}} \left\{ \| \tilde{X}\tilde{w} - y \|_2^2 \right\} \quad \text{s.t.} \quad 0 \leq \tilde{w} \leq 1, \quad \sum_{i=1}^{2d} \tilde{w}_i = 1
$$

where $\tilde{X} = (X \mid -X)$ and $\tilde{w} = (I_d \mid -I_d)^\top w$

Note that $\tilde{X}\tilde{w}$ lies in the convex hull spanned by the columns of $\tilde{X}$,

$$
S = \left\{ \sum_{j=1}^{2d} \alpha_j \tilde{X}_j \right\} \quad 0 \leq \alpha \leq 1, \quad \sum_{j=1}^{2d} \alpha_j = 1
$$
The previous problem is equivalent to finding the closest point between the convex hull $S$ and the set $\{y\}$. This is the Nearest Point Problem:

It can be solved with SVM algorithms, and that may lead to faster convergence.
Kernel-aware SMO

- First approach: straight application of SMO

- Due to the artificial nature of the data matrix, the kernel matrix contains many redundancies:

\[
\begin{pmatrix}
X^\top X & -X^\top X & X^\top y \\
-X^\top X & X^\top X & -X^\top y \\
y^\top X & -y^\top X & y^\top y \\
\end{pmatrix}
\]

\[2d+1\]

- Only need to compute \(X^\top X\), \(X^\top y\) and \(y^\top y\)

- We implemented a “kernel-aware” SMO version that takes into account the previous structure and avoids repeating computations
Outline

1. Introduction
2. Lasso
3. SVM
4. Algorithms
5. Solving the Lasso via SVMs
6. Acceleration methods for SMO
7. Experiments
8. Conclusions
Heavy Ball

- The Heavy Ball method [Polyak, 1964] adds a “momentum” term to the Gradient Descent updates,

\[ x^{k+1} = x^k - \eta \nabla f(x^k) + \beta (x^k - x^{k-1}) \]

- Improves the convergence of Gradient Descent for “good” choices of \( \eta \) and \( \beta \)

- Intuitively, it uses the information about the previous two iterates to smooth the descent direction

- Conjugate Gradient Descent is a more precise version of Heavy Ball where \( \eta \) and \( \beta \) are partially optimized at every step
HB example
Nesterov’s Accelerated Gradient

- Nesterov’s Accelerated Gradient (NAG) [Nesterov, 1983] updates are,

\[ y^k = x^k + \mu_k(x^k - x^{k-1}), \]
\[ x^{k+1} = y^k - \eta \nabla f(y^k) \]

- Improves the convergence of Gradient Descent to \( O(1/k^2) \) for convex functions

- The sequence \( \mu_k \) is key to the fast convergence, but not very intuitive:

\[ t_0 = 1, \quad t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}, \quad \text{and} \quad \mu_k = \frac{t_k - 1}{t_{k+1}} \]
Heavy Ball vs Nesterov’s Acceleration

- NAG is just another form of momentum

\[ x^{k+1} = x^k + \mu_k (x^k - x^{k-1}) - \eta \nabla f (x^k + \mu_k (x^k - x^{k-1})) \]

- In contrast to Heavy Ball, NAG uses information not only about the previous coefficients but also the previous gradients

- They both achieve linear convergence rate for strongly convex functions

- NAG does not guarantee monotone descent of the objective value
NAG example

![Graph showing the objective function over iterations for different algorithms. The x-axis represents iterations, and the y-axis represents the objective function. Two lines are shown: a red line labeled GD and a cyan line labeled NAG. The objective function values range from $10^1$ to $10^{-5}$.](image-url)
SMO updates

- The question is, can we do the same in SMO?

- After selecting the two indexes, the SMO step has the form,

\[ \alpha^{k+1} = \alpha^k + \rho_k(y_i e_i - y_j e_j) = \alpha^k + \rho_k d \]

where \( e_k \) is the vector with all 0 but a 1 in position \( k \)

- Only updates two coefficients

- \( \rho_k \) is computed by minimizing the function in the direction \( d \) (line search)

- Need to clip \( \rho_k \) to satisfy the constraints
Nesterov’s Accelerated SMO (NAS)

- First we compute the intermediate point as in NAG and then perform an SMO update

\[ x^k = \alpha^k + \mu_k (\alpha^k - \alpha^{k-1}), \]
\[ \alpha^{k+1} = x^k + \rho_k d \]

- \( \mu_k \) is Nesterov’s sequence

- \( \rho_k \) is computed performing a line search, as in SMO

- Both may need to be clipped to satisfy the constraints
The convergence is not guaranteed since NAG is not monotone

At every iteration, we test for decrease of the objective function

If it does not hold, we compute a new intermediate point, with the exact $\mu_k$ such that $f(\alpha^{k+1}) < f(\alpha^k)$

Computing SVC’s dual objective function is quite expensive

Iteration complexity is twice as standard SMO
Conjugate SMO (CS)

- CS replaces SMO descent direction $\mathbf{d}$ by a conjugate direction, which is equivalent to adding a momentum term,

$$
\mathbf{p}^k = \mathbf{d}^k + \gamma_k \mathbf{p}^{k-1},
$$

$$
\alpha^{k+1} = \alpha^k + \rho_k \mathbf{p}^k
$$

- $\rho_k$ is computed as in SMO

- $\gamma_k$ is selected to approximately maximize the gain

- Again, they may need to be clipped

- Iteration complexity is $\frac{4}{3}$ of SMO’s (less than MNAS)
Outline

1. Introduction
2. Lasso
3. SVM
4. Algorithms
5. Solving the Lasso via SVMs
6. Acceleration methods for SMO
7. Experiments
8. Conclusions
Lasso to SVM experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\lambda^* (\times 10^{-3})$</th>
<th>Observations</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>0.453</td>
<td>46,215</td>
<td>90</td>
</tr>
<tr>
<td>trajectory</td>
<td>10.210</td>
<td>20,000</td>
<td>298</td>
</tr>
<tr>
<td>ctscan</td>
<td>3.748</td>
<td>53,500</td>
<td>385</td>
</tr>
<tr>
<td>mnist_reg</td>
<td>7.760</td>
<td>784</td>
<td>5000</td>
</tr>
</tbody>
</table>

- 4 datasets, time is the average of 100 experiments

- 3 different $\lambda$ values: “optimal” one selected by cross-validation $\lambda^*$, $2\lambda^*$ and $\lambda^*/2$

- Algorithms:
  - Cyclic Coordinate Descent (CCD), *scikit-learn*
  - Sequential Minimal Optimization (SMO), *scikit-learn*
  - Kernel-aware SMO (K-LSMO)
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Iterations</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CCD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>year</td>
<td></td>
<td>5153</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5148</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4998</td>
</tr>
<tr>
<td>trajectory</td>
<td></td>
<td>8555</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2588</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1313</td>
</tr>
<tr>
<td>ctscan</td>
<td></td>
<td>101130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>78710</td>
</tr>
<tr>
<td></td>
<td></td>
<td>53630</td>
</tr>
<tr>
<td>mnist_reg</td>
<td></td>
<td>1040395</td>
</tr>
<tr>
<td></td>
<td></td>
<td>635452</td>
</tr>
<tr>
<td></td>
<td></td>
<td>550356</td>
</tr>
</tbody>
</table>
Experiment results for different algorithms:

- **K-SVM**
- **GLMNet**
- **SVM**

The graph shows the objective function over time for these algorithms. The x-axis represents time in milliseconds, ranging from 0 to 800 ms. The y-axis represents the objective function on a logarithmic scale, ranging from $10^{-6}$ to $10^1$.

The algorithms show different convergence rates, with K-SVM and GLMNet demonstrating faster initial descent compared to SVM, which shows a more gradual decrease in the objective function over time.
Dataset \textit{ctscan}

Experiments

\textit{Alberto Torres Barrán}
Dataset trajectory

Experiments

Alberto Torres Barrán
SMO acceleration first experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Observations</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>australian</td>
<td>609</td>
<td>14</td>
</tr>
<tr>
<td>german</td>
<td>1,000</td>
<td>24</td>
</tr>
<tr>
<td>adult8</td>
<td>22,696</td>
<td>123</td>
</tr>
<tr>
<td>web8</td>
<td>49,749</td>
<td>300</td>
</tr>
</tbody>
</table>

- 4 datasets, 3 $C$ values, $\gamma = 1/d$ and $\epsilon = 0.001$

- Algorithms implemented in Python:
  - Standard SMO
  - Conjugate SMO (CS)
  - Monotone Nesterov’s Accelerated SMO (MNAS)

- The same model is obtained for the three algorithms
## Iteration results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C</th>
<th>SMO</th>
<th>CS</th>
<th>MNAS</th>
<th>Ratio</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CS</td>
<td>MNAS</td>
</tr>
<tr>
<td>australian</td>
<td>1</td>
<td>404</td>
<td>368</td>
<td>401</td>
<td>1.10</td>
<td>1.01</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1403</td>
<td>885</td>
<td>795</td>
<td>1.59</td>
<td>1.76</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>4586</td>
<td>3276</td>
<td>3668</td>
<td>1.40</td>
<td>1.25</td>
</tr>
<tr>
<td>german</td>
<td>1</td>
<td>672</td>
<td>601</td>
<td>672</td>
<td>1.12</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3189</td>
<td>2414</td>
<td>2046</td>
<td>1.32</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>12560</td>
<td>8584</td>
<td>6868</td>
<td>1.46</td>
<td>1.83</td>
</tr>
<tr>
<td>adult8</td>
<td>1</td>
<td>8316</td>
<td>7516</td>
<td>7200</td>
<td>1.11</td>
<td>1.16</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>43727</td>
<td>30965</td>
<td>24654</td>
<td>1.41</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>284637</td>
<td>185062</td>
<td>137391</td>
<td>1.54</td>
<td>2.07</td>
</tr>
<tr>
<td>web8</td>
<td>1</td>
<td>3951</td>
<td>3859</td>
<td>4014</td>
<td>1.02</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14127</td>
<td>11939</td>
<td>15889</td>
<td>1.18</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>41112</td>
<td>30286</td>
<td>43336</td>
<td>1.36</td>
<td>0.95</td>
</tr>
</tbody>
</table>
CS and MNAS are better for large $C$

CS usually better than MNAS (and also cheaper!)
SMO acceleration LIBSVM experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Observations</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>ijcnn1</td>
<td>49,990</td>
<td>22</td>
</tr>
<tr>
<td>cod-rna</td>
<td>59,535</td>
<td>8</td>
</tr>
<tr>
<td>mnist1</td>
<td>60,000</td>
<td>784</td>
</tr>
<tr>
<td>skin</td>
<td>245,057</td>
<td>3</td>
</tr>
</tbody>
</table>

- 4 new datasets, 4 $C$ values, 4 $\gamma$ values and $\epsilon = 0.001$.

- Algorithms implemented in LIBSVM:
  - Standard SMO
  - Conjugate SMO (CS)

- Running time is the average of 50 repetitions
Time vs cache size (adult9 and web8)

- Large cache ⇒ lower running time
- CS is better the lower the cache
Time vs $C$ and $\gamma$ (mnist1 and ijcnn1)

- Cache size set to LIBSVM default, 100Mb
- Measure the relative time difference as a percentage,

\[
\frac{\text{Time(SMO)} - \text{Time(CS)}}{\text{Time(SMO)}} \times 100
\]

### Dataset = ijcnn1

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>$C$</th>
<th>-5.29</th>
<th>-4.53</th>
<th>-0.87</th>
<th>-7.06</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{-1}$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Dataset = mnist1

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>$C$</th>
<th>0.25</th>
<th>3.05</th>
<th>6.26</th>
<th>3.49</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{-1}$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Time vs $C$ and $\gamma$ (cod-rna and skin)

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>Dataset = cod-rna</th>
<th></th>
<th>Dataset = skin</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>-5.07</td>
<td>8.73</td>
<td>19.44</td>
<td>25.04</td>
</tr>
<tr>
<td>0.1</td>
<td>1.16</td>
<td>18.53</td>
<td>24.08</td>
<td>31.66</td>
</tr>
<tr>
<td>1.0</td>
<td>13.22</td>
<td>19.94</td>
<td>27.41</td>
<td>34.89</td>
</tr>
<tr>
<td>10.0</td>
<td>-4.21</td>
<td>-3.37</td>
<td>-4.37</td>
<td>5.07</td>
</tr>
</tbody>
</table>

$C$ and $\gamma$ values:

- **CS** is better than SMO for a wide range of $C$ and $\gamma$ values in 3 out of 4 problems.
- CS usually better the bigger the $C$ and $\gamma$ (costlier cases).
Full hyper-parameter search

- We measure the total time for a full hyper-parameter search (average of 50 repetitions)
- $11 \times 10$ grid:
  \[
  \log_2 C = [-5, -3, \ldots, 13, 15] \\
  \log_2 \gamma = [-15, -13, \ldots, 1, 3]
  \]

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SMO</th>
<th>CS</th>
<th>Rel. Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>australian</td>
<td>0.262</td>
<td>0.213</td>
<td>18.70</td>
</tr>
<tr>
<td>german</td>
<td>0.841</td>
<td>0.705</td>
<td>16.17</td>
</tr>
<tr>
<td>adult8</td>
<td>2546.097</td>
<td>1608.115</td>
<td>36.84</td>
</tr>
<tr>
<td>web8</td>
<td>748.557</td>
<td>752.478</td>
<td>-0.52</td>
</tr>
<tr>
<td>ijcnn1</td>
<td>495.470</td>
<td>439.572</td>
<td>11.28</td>
</tr>
<tr>
<td>skin</td>
<td>2541.449</td>
<td>2484.472</td>
<td>2.24</td>
</tr>
<tr>
<td>cod-rna</td>
<td>23320.557</td>
<td>13029.203</td>
<td>44.13</td>
</tr>
<tr>
<td>mnist1</td>
<td>95460.910</td>
<td>85319.661</td>
<td>10.62</td>
</tr>
</tbody>
</table>
1. Introduction
2. Lasso
3. SVM
4. Algorithms
5. Solving the Lasso via SVMs
6. Acceleration methods for SMO
7. Experiments
8. Conclusions
Conclusions (Lasso to SVM)

- Reducing Lasso to a NPP opens the possibility of using SVM algorithms to solve it

- Some theoretical properties can also be translated between the two settings [Jaggi, 2014]

- Straight application of SMO is competitive with CCD

- There is room for improvement modifying SMO for the Lasso reduction

- Our K-SVM implementation generally outperforms CCD
Conclusions (Acceleration for SMO)

- In general, our accelerated SMO versions converge in less iterations than SMO to the same model.
- Less iterations does not imply faster execution times.
- CS is usually faster than MNAS (cheaper iterations).
- We implemented CS inside LIBSVM to perform a time comparison against standard SMO.
- CS is better in settings with low cache sizes and large $C$.
- For a full hyper-parameter search CS outperforms SMO.
References I

- **Tibshirani, R. (1994).**
  Regression Shrinkage and Selection Via the Lasso. 

- **Polyak, B. T. (1964).**
  Some methods of speeding up the convergence of iteration methods.

- **Nesterov, Y. (1983).**
  A method of solving a convex programming problem with convergence rate \(o(1/k^2)\).


Thanks!

Questions?
alberto.torres@uam.es