Large Scale Support Vector Machines

Olivier Chapelle

YAHOO! RESEARCH

Oct 23rd 2007

Outline

1. Linear SVM
2. Nonlinear SVM
3. Extensions & Applications

Outline

1. Linear SVM
   - Primal minimization
   - Experiments
2. Nonlinear SVM
3. Extensions & Applications

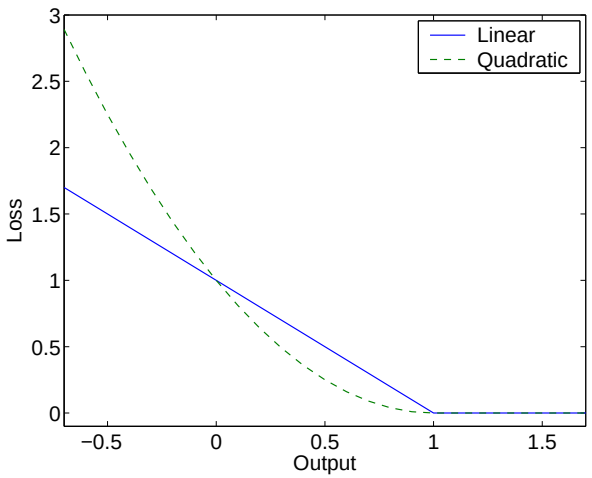
Linear SVM classifier

- Training set $(\mathbf{x}_i, y_i),\ 1 \leq i \leq n,\ \mathbf{x}_i \in \mathbb{R}^d,\ y_i = \pm 1.$
- Linear classifiers: $\mathbf{x} \rightarrow \mathbf{w} \cdot \mathbf{x}$.

Objective function

$$\min \quad ||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i^p$$

under constraints $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i$.



Linear penalization of the slacks ($p = 1$) or quadratic ($p = 2$).

At this point, most SVMs textbooks introduce Lagrange multipliers and solve the dual problem.

Simpler is better! Just write the unconstrained objective function

$$||\mathbf{w}||^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i))^2.$$

and minimize it directly in the primal.

Take $p = 2$:

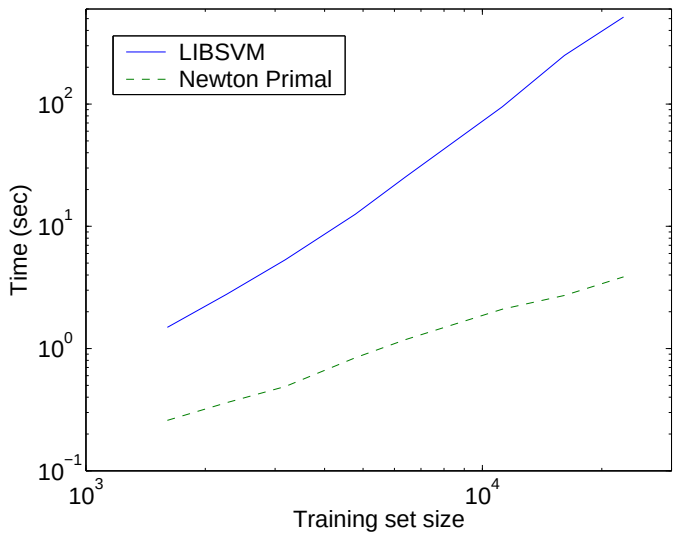
- Numerically easier: the objective function is differentiable.
- Statistically: no particular reason to prefer $p = 1$ or $p = 2$.

At this point, most SVMs textbooks introduce Lagrange multipliers and solve the dual problem.
Simpler is better! Just write the unconstrained objective function

$$||\mathbf{w}||^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i))^2.$$

and minimize it directly in the primal.
Take $p = 2$:

- Numerically easier: the objective function is differentiable.
- Statistically: no particular reason to prefer $p = 1$ or $p = 2$.



3 standard and efficient minimization algorithms:

1. Nonlinear conjugate gradient.
2. Truncated Newton.
3. Stochastic gradient descent.

Nonlinear conjugate gradient

- Only need to compute the gradient:

$$\nabla_p = w_p - C \sum_{i=1}^{n} x_{ip} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i)).$$

  Complexity of $O(nd)$ per iteration.
- Line search: function evaluation is dominated by the computation of $X\mathbf{w}$.

$$X(\mathbf{w} + t\mathbf{s}) = X\mathbf{w} + t\ X\mathbf{s}.$$

  $\longrightarrow$ After precomputing $X\mathbf{w}$ and $X\mathbf{s}$, the evaluation of the function along $\mathbf{s}$ is $O(n)$.
  In practice, line search done with 1D Newton steps.

Truncated Newton

- Newton step:

$$\mathbf{w} \leftarrow \mathbf{w} - H^{-1}\nabla$$

$$H_{pq} = \delta_{pq} + C\sum_{i=1}^{n} x_{ip}x_{iq}1_{y_i(\mathbf{w}\cdot\mathbf{x}_i)\leq 1}.$$

- As before, one can do line search.
- Complexity per iteration:
  - $O(nd)$ to find the n_{sv} support vectors.
  - $O(n_{\text{sv}}d^2)$ to compute the Hessian
  - $O(d^3)$ to invert the Hessian.
  - Only couple of iterations are required.
- Not feasible for large d.

Instead solve the linear system by *linear conjugate gradient*:

repeat ▷ Newton Loop
  sv $\leftarrow \{i,\ y_i(\mathbf{w}\cdot\mathbf{x}_i) < 1\}$. $X_{\text{sv}} :=$ submatrix of X.
  repeat ▷ Solve by CG $(I + CX_{\text{sv}}^\top X_{\text{sv}})^{-1}\nabla$.
    $\mathbf{v} \leftarrow \mathbf{s} + CX_{\text{sv}}^\top(X_{\text{sv}}\mathbf{s})$
    Update $\mathbf{s}$ based on $\mathbf{v}$.
  until Convergence
  $\mathbf{w} \leftarrow \mathbf{w} + t\mathbf{s}$ (t found by line search).
until Convergence

- Each iteration is a matrix vector multiplication: $O(n_{\text{sv}}d)$ per iteration; and much less when the training data is sparse.
- Similar to nonlinear conjugate gradient but can be faster when $n_{\text{sv}} \ll n$.

Stochastic gradient descent

Objective function as a sum over the training samples:

$$\frac{1}{n}\sum_{i=1}^{n}\underbrace{||\mathbf{w}||^2 + nC\max(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i))^2}_{\ell_i(\mathbf{w})}.$$

SGD update

At iteration t, choose a random i and update:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{t}\nabla\ell_i(\mathbf{w})$$

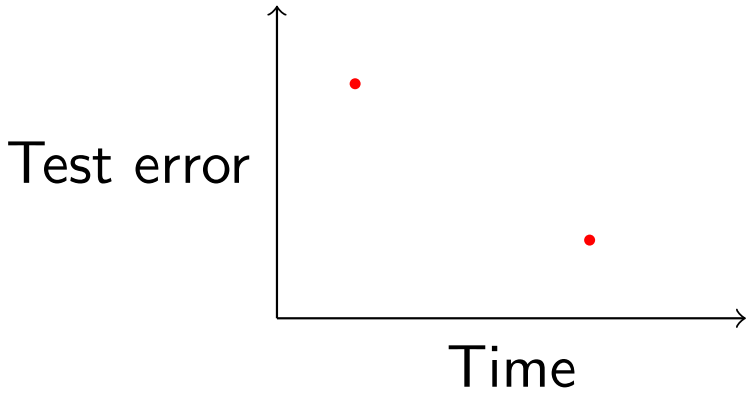
The learning rate η has to be chosen carefully. According to [Bottou, Bousquet '07], one should take $\eta \sim \frac{1}{\lambda_{min}}$, with λ_{min} the smallest eigenvalue of the Hessian. Because of the regularizer, $\lambda_{min} \geq 1$ and we take $\eta = 1$.

Experimental comparison

Claim

Large scale learning algorithms should be compared on a test error vs training time plot.

- Plot obtained by varying stopping criterion, number of examples, dimensionality, regularization parameter,...
- Objective function value and training error can be interesting quantities for debugging, but ultimately only the test error matters.



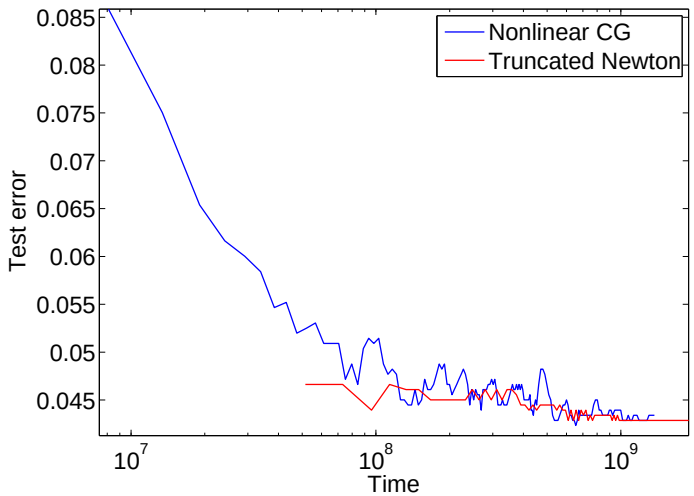
Dataset

MNIST, digit 5 vs 8.
About 6k (resp 1k) examples per classes for training (resp testing).
28×28 pixels.

- For simplicity, C chosen for best performance on the test set (but other values might achieve better accuracy/complexity trade-off!)
- For a fair comparison, time complexity is the number of multiplications / additions.

Comparison Nonlinear CG vs Truncated Newton

Nonlinear CG is Polak-Ribiere; Flecher-Reeves gives similar results.
Truncated Newton: linear CG limited at 5 iterations.

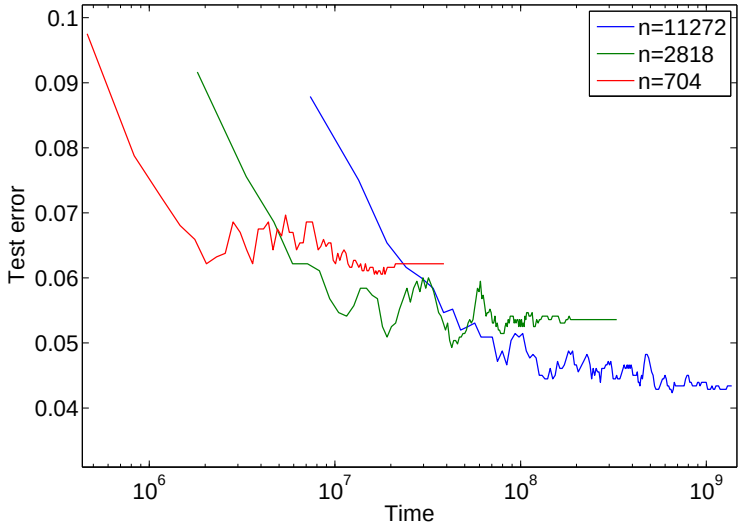


Nonlinear CG is usually faster than a "basic" truncated Newton.
However, truncated Newton takes advantage of the sparsity
$\longrightarrow$ in practice, roughly the same performance.

Not clear when to truncate the linear CG.
From now on, we consider nonlinear CG.

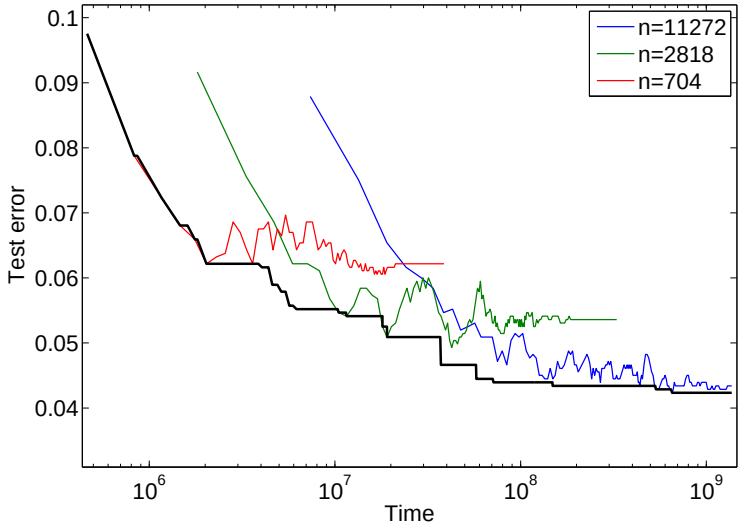
Influence of the training set size

Subsampling is an obvious (but often forgotten!) strategy to speed up training.



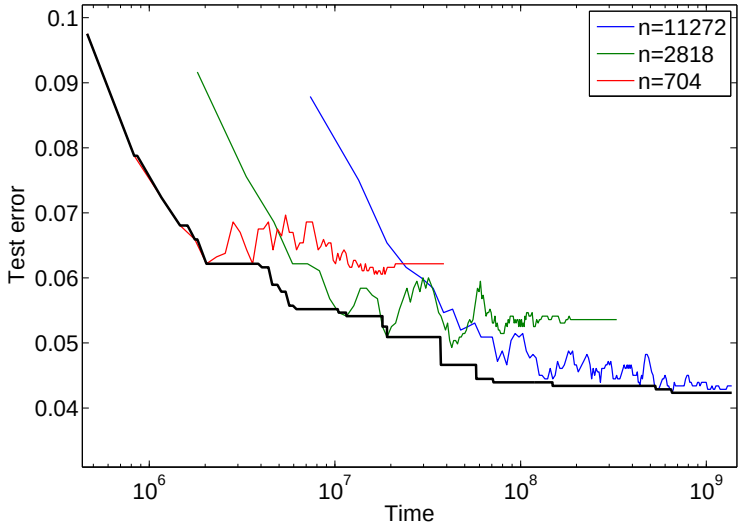
Influence of the training set size

Subsampling is an obvious (but often forgotten!) strategy to speed up training.

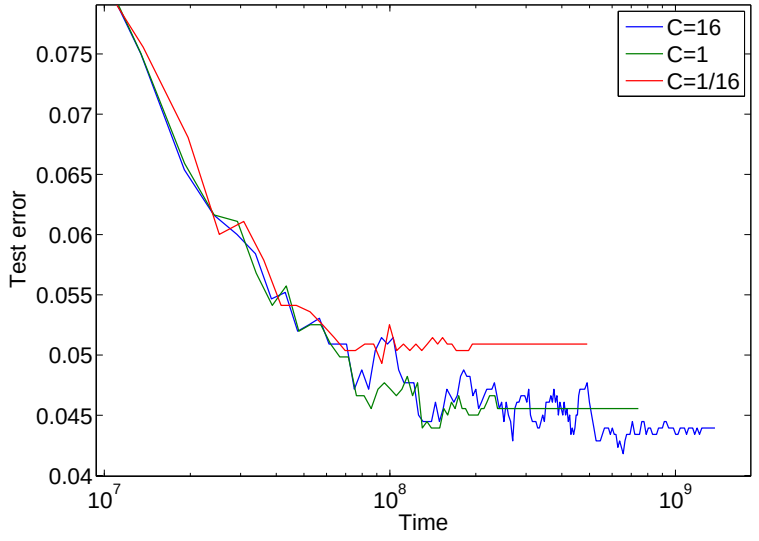


Influence of the training set size

Subsampling is an obvious (but often forgotten!) strategy to speed up training.



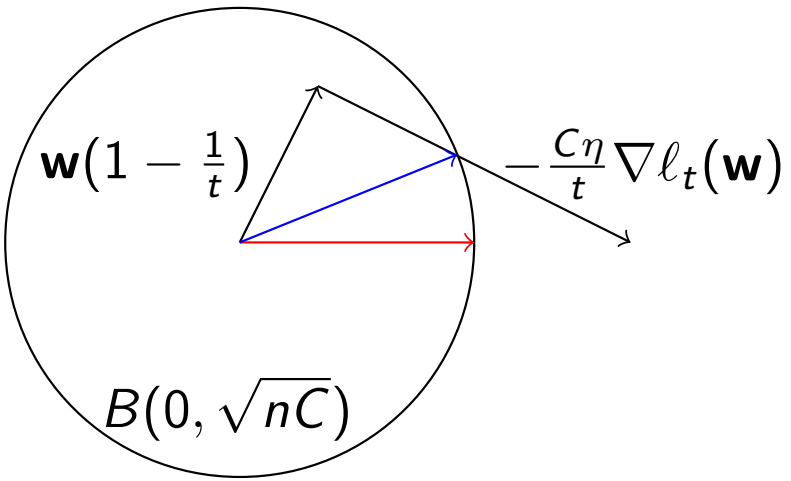
Influence of C



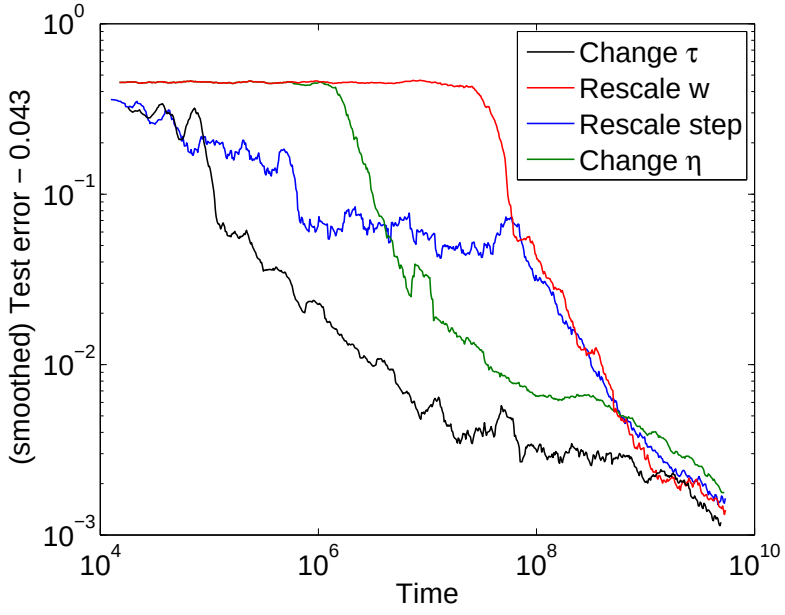
Stochastic gradient descent

Main difficulty: the optimal asymptotic learning learning rate in $1/t$ is often too large at the beginning.

First note that the optimal $\mathbf{w}$ satisfies $||\mathbf{w}|| \leq \sqrt{nC}$.
$\longrightarrow$ At each step, either rescale (if necessary) $\mathbf{w}$ or the step to make sure that $||\mathbf{w}|| \leq \sqrt{nC}$



1. Change $1/t$ by $1/(t+\tau)$
2. Rescale $\mathbf{w}$ at each iteration [Shalev-Schwartz et al. '07]
3. Rescale the gradient
4. Change $1/t$ by η/t.



- 1,2,3 do not change the asymptotic learning rate, but 4 does.
- In 2 and 3, the rescaling is done such that $\mathbf{w}$ stays in the ball of radius $\sqrt{nC}$.

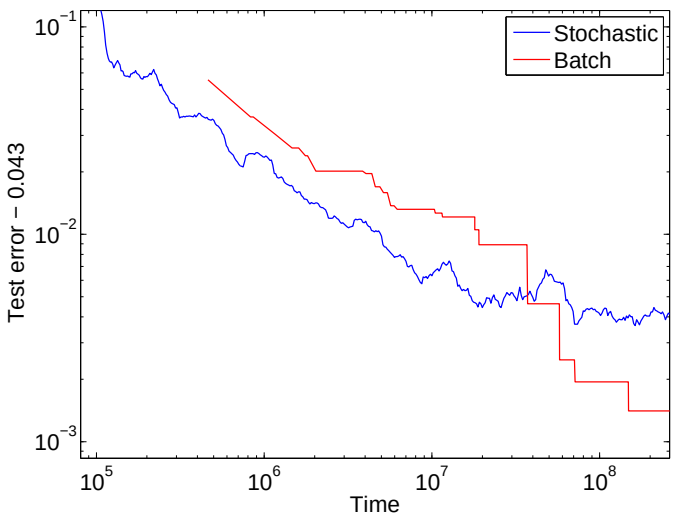
Stochastic vs batch

In both cases, one difficult adjustable parameter:

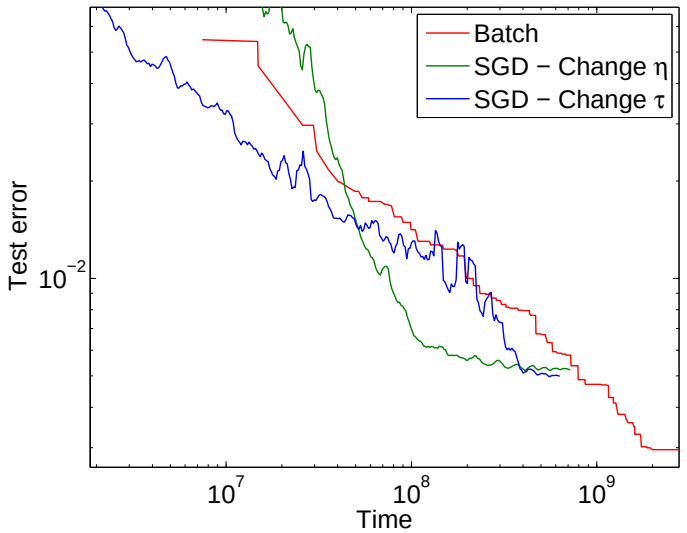
Batch Subsample rate

Stochastic Learning rate

But a well tuned stochastic method seems faster to achieve a reasonable accuracy.



Mnist 5 vs 8



Text classification

$n = 150k, d = 250k$

Sparsity: 0.1%

Outline

1. Linear SVM
2. Nonlinear SVM
   - Reduced set expansion
   - Nonlinear conjugate gradient
3. Extensions & Applications

Nonlinear SVMs

- "Kernel trick": Introduction of an implicit mapping $\Phi(\mathbf{x})$ and a *kernel* function k such that $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$.
- At the optimum $\mathbf{w}$, the gradient vanishes and $\mathbf{w} = \sum_{i=1}^{n} \beta_i \Phi(\mathbf{x}_i)$.

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_{i=1}^{n} \beta_i k(\mathbf{x}, \mathbf{x}_i). \quad \text{[representer theorem]}$$

- Minimize over $\boldsymbol{\beta} \in \mathbb{R}^n$ the *primal* objective function,

$$\underbrace{\boldsymbol{\beta}^\top K \boldsymbol{\beta}}_{\text{regularizer}} + C \underbrace{\sum_{i=1}^{n} \max(0, 1 - y_i \overbrace{[K\boldsymbol{\beta}]_i}^{f(\mathbf{x}_i)})^2}_{\text{loss}}, \qquad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

$\longrightarrow$ Again, no need of duality theory.

Newton step

The Newton step turns out to be of the following form:

$$
\begin{aligned}
H^{-1}\nabla &= \beta - (K + CKI^0K)^{-1}KI^0Y \qquad I^0_{pp} = I(y_p f(\mathbf{x}_p) < 1) \\
&= \beta - (\lambda I_n + I^0K)^{-1}I^0Y \\
&= \beta - \begin{pmatrix} (\lambda I_{n_{\text{sv}}} + K_{\text{sv}})^{-1}Y_{\text{sv}} \\ 0 \end{pmatrix}.
\end{aligned}
$$

Complexity is $O(n_{\text{sv}}n + n_{\text{sv}}^3)$

- Exactly the same as standard dual SVM solvers.
- Prohibitive if n_{sv} is large.
  $\longrightarrow$ Need for approximate solutions.

Sparse expansion

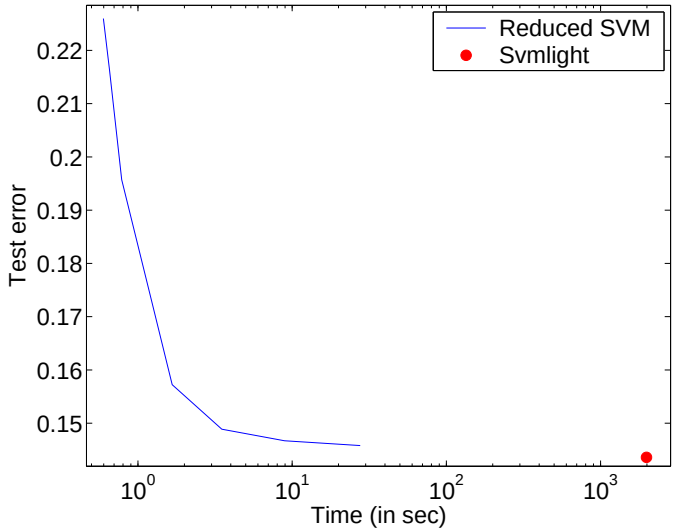
- Fix the complexity (both in the learning and computational sense) by *choosing a subset of points on which to expand the solution*.
- Let $S \subset [1...n]$. Minimize

$$\beta_S^\top K_{SS} \beta_S + C \sum_{i=1}^{n} \max(0, 1 - y_i K_{i,S} \beta_S)^2.$$

- If $|S| = k$, the complexity is $O(k^2 n)$.
- Very similar to RSVM [Mangasarian '00].

- In feature space, this is equivalent to train a linear SVM after projection of points on the subspace spanned by $\{\Phi(\mathbf{x}_i)\}_{i \in S}$.
- If the "effective dimension" of the feature space (or the effective rank of the Gram matrix) is around $k \longrightarrow$ not a big loss in approximation.
- If it's larger, loss in accuracy, but computational speed-up.

Experimental results on Adult ($n = 22697, d = 123$), RBF kernel. Varying the number of (random) basis functions.



- Having more than 100 basis function does not help a lot (the spectrum decays rapidly).
- Comparison with standard SVM learning: training time reduced of several orders of magnitude, without significant loss in accuracy.

Remarks

- Objective function and test error are strongly related $\longrightarrow$ we are in an *underfitting* situation
- The Adult dataset is quite noisy $\longrightarrow$ no need for an accurate solution (and S can be small).
- For some other datasets (e.g. Mnist), need for accuracy.

Greedy selection of the basis functions; borrow ideas from the regression case.

Matching pursuit style algorithm

1: $S \leftarrow \emptyset$
2: **for** $k = 1$ to d **do**
3: Choose a random set of p basis function candidates
4: Select the one which decreases the most the objective function and add it to S.
5: Update the solution
6: **end for**

- Step 4 takes $O(np)$ (see below)
- Step 5 takes $O(nk)$ (rank 1 update of the invert Hessian)
- If $p \sim d$, total cost is $O(nd^2)$

Let S be the current set of basis function and j be the index of a candidate basis function. Let $\mathcal{R}(\boldsymbol{\beta}_S, \beta_j)$ the objective function value and $\hat{\boldsymbol{\beta}}_S = \arg\min \mathcal{R}(\boldsymbol{\beta}_S, 0)$ the current solution.

Two possible ways for scoring a candidate basis function:

1. $\mathcal{R}(\hat{\boldsymbol{\beta}}_S, 0) - \min_{\boldsymbol{\beta}_S, \beta_j} \mathcal{R}(\boldsymbol{\beta}_S, \beta_j)$.
2. $\mathcal{R}(\hat{\boldsymbol{\beta}}_S, 0) - \min_{\beta_j} \mathcal{R}(\hat{\boldsymbol{\beta}}_S, \beta_j)$.

Both quantities can be approximated by the Newton decrement which can be computed efficiently: $O(nk)$ for 1 and $O(n)$ for 2.

Detailed comparison of basis selection strategies in:

Keerthi, Chapelle, DeCoste, *Building Support Vector Machines with Reduced Classifier Complexity*, JMLR 2006

$\longrightarrow$ 2 turns out to be the most efficient method (because it is cheap) and random selection works also surprisingly well.
$p = 10$ candidates at each iteration seems to be a good trade-off.

Let S be the current set of basis function and j be the index of a candidate basis function. Let $\mathcal{R}(\boldsymbol{\beta}_S, \beta_j)$ the objective function value and $\hat{\boldsymbol{\beta}}_S = \arg\min \mathcal{R}(\boldsymbol{\beta}_S, 0)$ the current solution.

Two possible ways for scoring a candidate basis function:

1. $\mathcal{R}(\hat{\boldsymbol{\beta}}_S, 0) - \min_{\boldsymbol{\beta}_S, \beta_j} \mathcal{R}(\boldsymbol{\beta}_S, \beta_j)$.
2. $\mathcal{R}(\hat{\boldsymbol{\beta}}_S, 0) - \min_{\beta_j} \mathcal{R}(\hat{\boldsymbol{\beta}}_S, \beta_j)$.

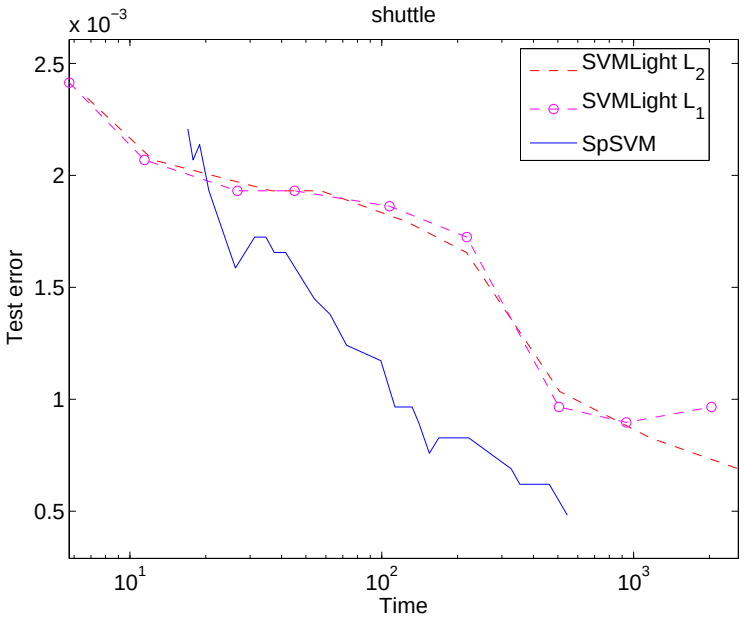
Both quantities can be approximated by the Newton decrement which can be computed efficiently: $O(nk)$ for 1 and $O(n)$ for 2.

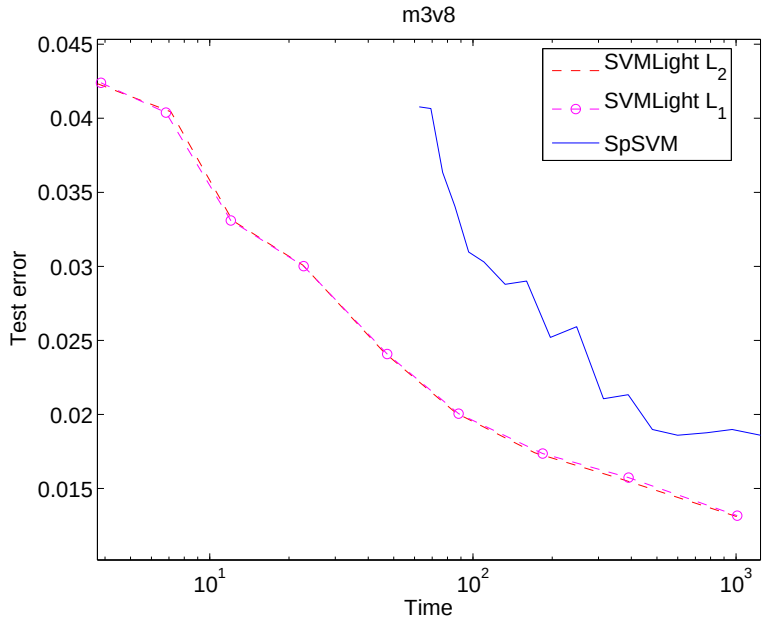
Detailed comparison of basis selection strategies in:

Keerthi, Chapelle, DeCoste, *Building Support Vector Machines with Reduced Classifier Complexity*, JMLR 2006

$\longrightarrow$ 2 turns out to be the most efficient method (because it is cheap) and random selection works also surprisingly well.
$p = 10$ candidates at each iteration seems to be a good trade-off.

Comparison with SVMLight with subsampling.





$\longrightarrow$ A lot of variability across datasets. But typically, very sparse solutions and 3-4 times reduction in training time.

- This method enables the user to control the time complexity of the algorithm; and to stop after a certain time limit.
- With few basis functions, the regularizer is not needed $\longrightarrow \approx$ RBF networks. But with more basis functions, it approaches the full SVM solution.

Nonlinear conjugate gradient

As in the linear case, simply use nonlinear conjugate gradient.
Two difficulties:

1. The gradient calculation requires $K\beta \longrightarrow O(n^2)$ complexity.
2. The Hessian is $K + CKI^0K$ and not as well conditioned as in the dual (for which the Hessian is K) $\longrightarrow$ slower convergence.

Computation of the matrix vector multiplication can be made faster with:

Sparse kernel such as a compactly supported RBF kernel.

Low rank approximation of the kernel matrix: if $K \approx UU^\top, K\beta \approx U(U^\top \beta)$.

Fast multipole method and KD tree in case of an RBF kernel

- Fast multipole methods work well for large bandwidth.
- KD trees work well for small bandwidth.
- But interesting bandwidth are in between! In that case, both methods work only in small dimension (≤ 5). But when n is large and d is small, SVMs are not necessary: k-NN is good enough.

Computation of the matrix vector multiplication can be made faster with:

Sparse kernel such as a compactly supported RBF kernel.

Low rank approximation of the kernel matrix: if $K \approx UU^\top, K\beta \approx U(U^\top \beta)$.

Fast multipole method and KD tree in case of an RBF kernel

- Fast multipole methods work well for large bandwidth.
- KD trees work well for small bandwidth.
- But interesting bandwidth are in between! In that case, both methods work only in small dimension (≤ 5). But when n is large and d is small, SVMs are not necessary: k-NN is good enough.

- Bad conditioning of the Hessian: $K + CKI^0K$.
- Solution: precondition by K^{-1}
- Preconditioned gradient of the form $K^{-1}K\mathbf{v}$
  $\longrightarrow$ straightforward to compute.
- Preconditioning in the β space is equivalent to standard conjugate gradient in the RKHS.

Complexity analysis:

- Number of iterations to reach a certain accuracy is $\sim \sqrt{\kappa}$, with κ the condition number of the (preconditioned) Hessian.
- Let $\tilde{\kappa}$ the condition number of K:
  - Without preconditioning: $\kappa \approx \tilde{\kappa}^2$
  - With preconditioning: $\kappa \approx \tilde{\kappa}$

$\longrightarrow$ The number of iterations goes from t to $\sqrt{t}$ (verified experimentally).

- Bad conditioning of the Hessian: $K + CKI^0K$.
- Solution: precondition by K^{-1}
- Preconditioned gradient of the form $K^{-1}K\mathbf{v}$
  $\longrightarrow$ straightforward to compute.
- Preconditioning in the β space is equivalent to standard conjugate gradient in the RKHS.

Complexity analysis:

- Number of iterations to reach a certain accuracy is $\sim \sqrt{\kappa}$, with κ the condition number of the (preconditioned) Hessian.
- Let $\tilde{\kappa}$ the condition number of K:
  - Without preconditioning: $\kappa \approx \tilde{\kappa}^2$
  - With preconditioning: $\kappa \approx \tilde{\kappa}$

$\longrightarrow$ The number of iterations goes from t to $\sqrt{t}$ (verified experimentally).

Outline

1. Linear SVM
2. Nonlinear SVM
3. Extensions & Applications
   - Semi-supervised web spam classification
   - Ranking as structured output learning
   - Boosting for complex objective functions

Semi-supervised learning

- a set of l labeled examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$.
- a set of u unlabeled examples, $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_n$, with $n = l + u$.
- a (weighted directed) graph whose nodes are $\mathbf{x}_1, \ldots, \mathbf{x}_n$. a_{ij} is the weight of the link from $\mathbf{x}_i$ to $\mathbf{x}_j$.

The goal is to learn a linear classifier in an *inductive* mode (it can be evaluated on a un seen test example).

Minimize:

$$\underbrace{||\mathbf{w}||^2 + C \sum_{i=1}^{l} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i))^2}_{\text{Standard SVM}} + \underbrace{\Omega(\mathbf{w} \cdot \mathbf{x}_1, \ldots, \mathbf{w} \cdot \mathbf{x}_n)}_{\approx \text{ negative log prior}}.$$

No problem as long as Ω is convex and differentiable.

Web spam classification

- Standard graph regularization:
  $\Omega(f_1, \ldots, f_n) = \sum_{i \sim j} (f_i - f_j)^2 = f^\top L f$.
- But we have prior knowledge: a non spam host is unlikely to link to a spam host:

$$\Omega(f_1, \ldots, f_n) = \sum_{i \to j} \max(0, f_j - f_i)^2 \qquad [\text{spam} = 1]$$

$\longrightarrow$ Only penalizes links to hosts with higher spamicity.

Challenge

- We took part in a web spam challenge challenge and won :-)
- Results (at the host level):

	Features & Graph	Features only	Graph only
AUC	0.934	0.892	0.922

- Features were bag of words and did not help significantly.

Web spam classification

- Standard graph regularization:
  $\Omega(f_1, \ldots, f_n) = \sum_{i \sim j} (f_i - f_j)^2 = f^\top L f$.
- But we have prior knowledge: a non spam host is unlikely to link to a spam host:

$$\Omega(f_1, \ldots, f_n) = \sum_{i \to j} \max(0, f_j - f_i)^2 \qquad [\text{spam} = 1]$$

$\longrightarrow$ Only penalizes links to hosts with higher spamicity.

Challenge

- We took part in a web spam challenge challenge and won :-)
- Results (at the host level):

	Features & Graph	Features only	Graph only
AUC	0.934	0.892	0.922

- Features were bag of words and did not help significantly.

Structured output learning

Pairs $(\mathbf{x}_i, y_i)$ where y_i can be arbitrary complex.
Example: $\mathbf{x}_i$ is a sentence; y_i is a sequence of tags.

Learning for structured outputs (Tsochantaridis et al. '04)

- Learn a mapping $\mathbf{x} \longrightarrow y$
- Joint feature map: $\Psi(\mathbf{x}, y)$.
- Prediction rule:

$$\hat{y} = \arg\max_y \mathbf{w}^\top \Psi(\mathbf{x}, y).$$

Constraints for correct predictions on the training set:

$$\forall i, \forall y \neq y_i, \mathbf{w}^\top \Psi(\mathbf{x}_i, y_i) - \mathbf{w}^\top \Psi(\mathbf{x}_i, y) > 0.$$

SVM-like optimization problem:

$$\min_{\mathbf{w}, \xi_i} \quad \frac{\lambda}{2} ||\mathbf{w}||^2 + \sum_{i=1}^{n} \xi_i,$$

under constraints:

$$\forall i, \forall y \neq y_i, \mathbf{w}^\top \Psi(\mathbf{x}_i, y_i) - \mathbf{w}^\top \Psi(\mathbf{x}_i, y) \geq \Delta(y, y_i) - \xi_i,$$

where $\Delta(y, y_i)$ is the *loss* incurred while predicting y instead of y_i.

As before, perform a primal unconstrained optimization:

Objective function

$$\frac{\lambda}{2}||\mathbf{w}||^2 + \sum_{i=1}^{n} \max_y \ \Delta(y, y_i) + \mathbf{w}^\top \Psi(\mathbf{x}_i, y) - \mathbf{w}^\top \Psi(\mathbf{x}_i, y_i)$$

- Intrinsically non differentiable because of the max.
- Can be solve by stochastic gradient descent [Ratliff et al. '07].
- Other possibility: nonsmooth optimization techniques, such as the *bundle* method [Smola et al. '07]

Ranking application

Editorial data of the form:

$$(\underbrace{\text{query, url}}_{\mathbf{x}_{qi}}, \underbrace{\text{relevance grade}}_{s_{qi} \in \{1,2,3,4,5\}})$$

Notations

- $\mathbf{x}_q$ is a *set* of documents associated with query q; $\mathbf{x}_{qi}$ the i-th document.
- y_q is a *ranking* (i.e. a permutation): y_{qi} is the rank of the i-th document (obtained by sorting the scores s_{qi}).

Goal is to maximize the DCG:

$$\frac{1}{Q} \sum_{q=1}^{Q} \sum_{y_{qi} \leq 5} \frac{2^{s_{qi}} - 1}{\log_2(1 + y_{qi})}$$

We take: $\Psi(\mathbf{x}_q, y_q) = \sum_i \mathbf{x}_{qi} A(y_{qi})$.

- $A : \mathbb{N} \to \mathbb{R}$ is a user defined non increasing funct ion.
- Ranking is given by the order of $\mathbf{w}^\top \mathbf{x}_{qi}$ because $\mathbf{w}^\top \Psi(\mathbf{x}, y) = \sum_i \mathbf{w}^\top \mathbf{x}_{qi} A(y_{qi})$.

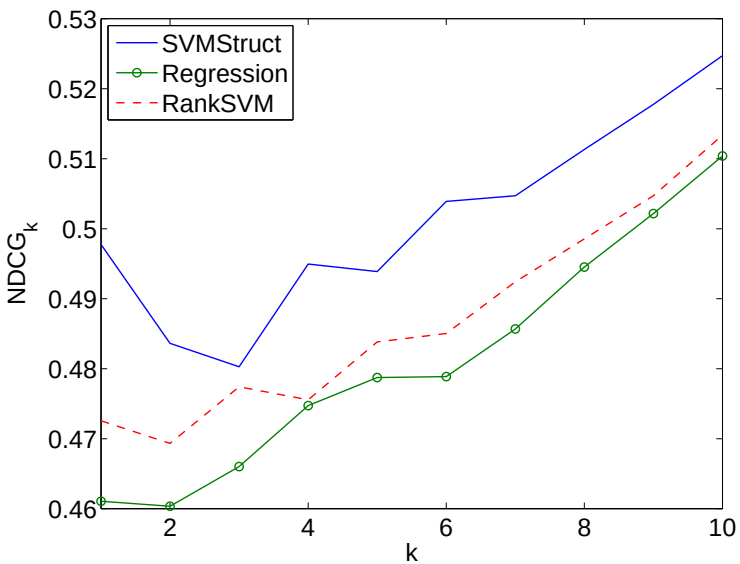
$$\begin{array}{c|ccccccc} \mathbf{w}^\top \mathbf{x}_{qi} & 2.5 & & 3.7 & & -0.5 & & \\ A(y) & \times & + & \times & + & \times & = 15.2 \to \max \\ & A(2) = 2 & & A(1) = 3 & & A(3) = 1 & & \end{array}$$

The loss $\Delta(y, y_q)$ is the *DCG loss* defined as the difference between the optimal DCG and the DCG obtained with the ranking y.

Experiments

Dataset

- Several hundred features.
- ∼50k (query,urls) pairs from an international market.
- ∼1500 queries randomly split in training / test (80% / 20%).
- 5 levels of relevance.



- Normalized DCG for different truncation levels
- λ chosen on a validation set
- $A(r) = \max(r + 1 - k, 0)$.
- Training time ∼20 minutes.
- About 2% improvement (p-value $= 0.03$ vs regression, 0.07 vs RankSVM).

Non-linear extension

Gradient boosted decision trees (or generalized linear model)

- Frideman's functional gradient boosting framework.

$$\min_{f \in \mathcal{F}} \mathcal{R}(f) = \sum_{j=1}^{n} \ell(y_j, f(\mathbf{x}_j)), \quad \mathcal{F} = \left\{ \sum_{i=1}^{N} \alpha_i h_i \right\}.$$

- Typically h_i is a tree and N is infinite.
- Cannot do direct optimization on $\mathcal{F}$.

$f \leftarrow 0$
repeat
  $g_j \leftarrow -\frac{\partial \ell(f(\mathbf{x}_j), y_j)}{\partial f(\mathbf{x}_j)}$. ▷ Functional gradient
  $\hat{\imath} \leftarrow \arg\min_{i,\lambda} \sum_{j=1}^{n} (g_j - \lambda h_i(\mathbf{x}_j))^2$. ▷ Steepest component
  $\hat{\rho} \leftarrow \arg\min \mathcal{R}(f + \rho h_{\hat{\imath}})$. ▷ Line search
  $f \leftarrow f + \eta \hat{\rho} \hat{h}$. ▷ "Shrinkage" when $\eta < 1$.
until Max iterations reached.

Non-linear extension

Gradient boosted decision trees (or generalized linear model)

- Frideman's functional gradient boosting framework.

$$\min_{f \in \mathcal{F}} \mathcal{R}(f) = \sum_{j=1}^{n} \ell(y_j, f(\mathbf{x}_j)), \quad \mathcal{F} = \left\{ \sum_{i=1}^{N} \alpha_i h_i \right\}.$$

- Typically h_i is a tree and N is infinite.
- Cannot do direct optimization on $\mathcal{F}$.

$f \leftarrow 0$
repeat
  $g_j \leftarrow -\frac{\partial \ell(f(\mathbf{x}_j), y_j)}{\partial f(\mathbf{x}_j)}$. ▷ Functional gradient
  $\hat{\imath} \leftarrow \arg\min_{i,\lambda} \sum_{j=1}^{n} (g_j - \lambda h_i(\mathbf{x}_j))^2$. ▷ Steepest component
  $\hat{\rho} \leftarrow \arg\min \mathcal{R}(f + \rho h_{\hat{\imath}})$. ▷ Line search
  $f \leftarrow f + \eta \hat{\rho} \hat{h}$. ▷ "Shrinkage" when $\eta < 1$.
until Max iterations reached.

- The objective function can be much more general:

$$\mathcal{R}(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$$

- $g_j = -\frac{\partial \mathcal{R}}{\partial f(\mathbf{x}_j)}$.
- For ranking via structured output learning:

$$\mathcal{R}(f) = \sum_q \max_y \left(\Delta(y, y_q) + \sum_{i=1}^{Q} f(\mathbf{x}_{qi})(A(y_i) - A(y_{qi})) \right).$$

Summary

Standard unconstrained minimization methods can be very efficient for large scale machine learning.