Large Scale Support Vector Machines

Olivier Chapelle & S. Sathiya Keerthi

YAHOO! RESEARCH

Jul 09, 2008
ICML workshop on large scale learning

General thoughts

Goal = find a good function.
Minimizing an objective function is only a *means* to this end.

See: Bottou and Bousquet 2008, *The Tradeoffs of Large Scale Learning*, NIPS 2008:

- Objective function value and training error can be interesting quantities for debugging, but ultimately only the test error matters.
- There is nothing "holy" in the minimum of the objective function.

$\longrightarrow$ From our perspective, the wild track was much more interesting than the SVM track.

The *number of examples* has to be optimized.

- No need to consider a million examples to solve a 3D linear problem!
- For the competition, 100k examples are more than enough to estimate a reasonable linear decision boundary.

The *stopping criterion* has to be optimized.

- Very loose stopping criterion is usually good enough.
- Example: on the Forest dataset, changing the stopping criterion of LIBSVM from $\varepsilon = 10^{-3}$ to $\varepsilon = 1$ reduces the training time by a factor 10 without loss of accuracy!
- When comparing your new learning algorithm to existing ones, you *need* to optimize the stopping criteria.

The *number of examples* has to be optimized.

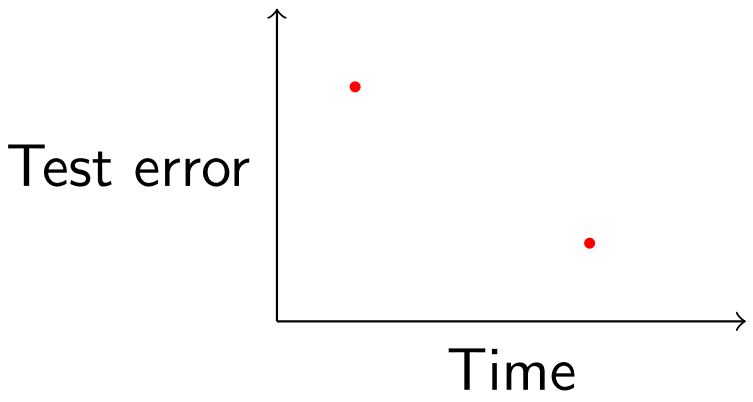
- No need to consider a million examples to solve a 3D linear problem!
- For the competition, 100k examples are more than enough to estimate a reasonable linear decision boundary.

The *stopping criterion* has to be optimized.

- Very loose stopping criterion is usually good enough.
- Example: on the Forest dataset, changing the stopping criterion of LIBSVM from $\varepsilon = 10^{-3}$ to $\varepsilon = 1$ reduces the training time by a factor 10 without loss of accuracy!
- When comparing your new learning algorithm to existing ones, you *need* to optimize the stopping criteria.

Claim

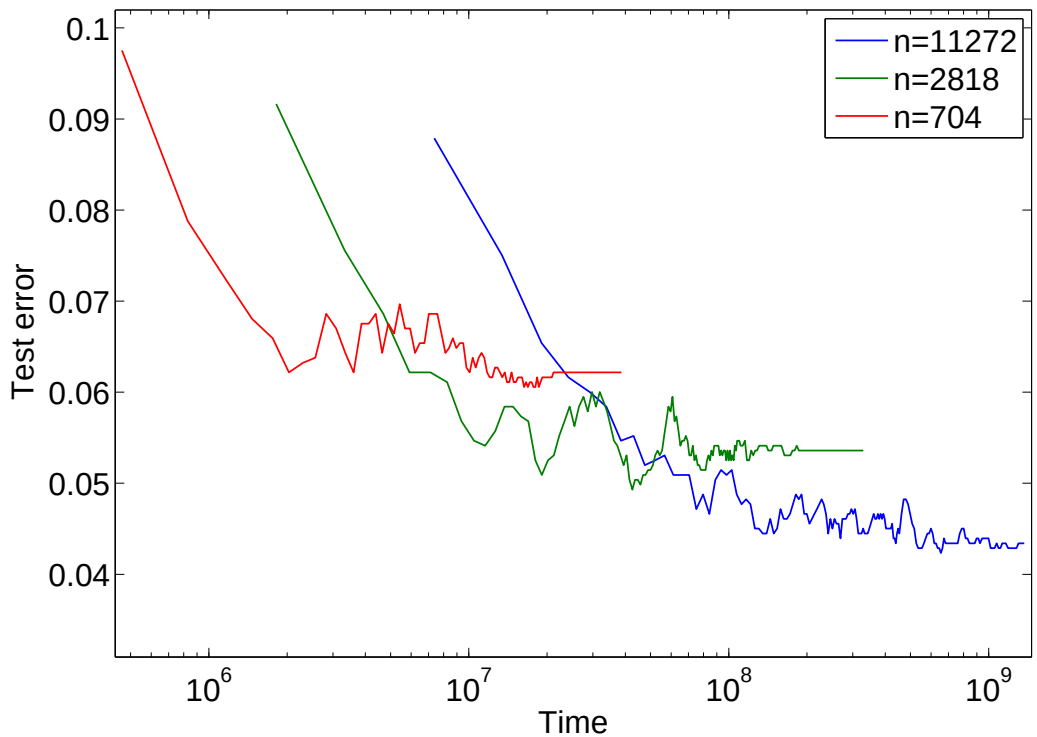
Large scale learning algorithms should be compared on a test error vs training time plot.



Plot obtained by varying stopping criterion, number of examples, dimensionality, regularization parameter,...
Then take the *lower envelope*.

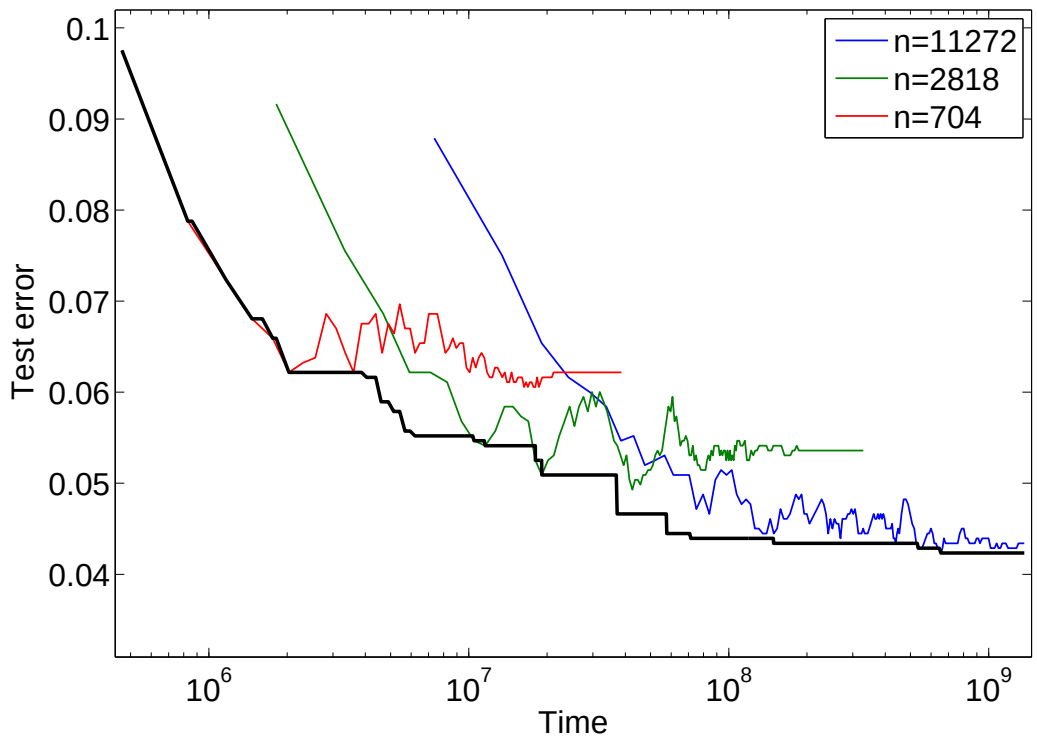
Example: MNIST

Each curve corresponds to a different training set size and is drawn for various number of iterations in a non-linear conjugate gradient minimization.



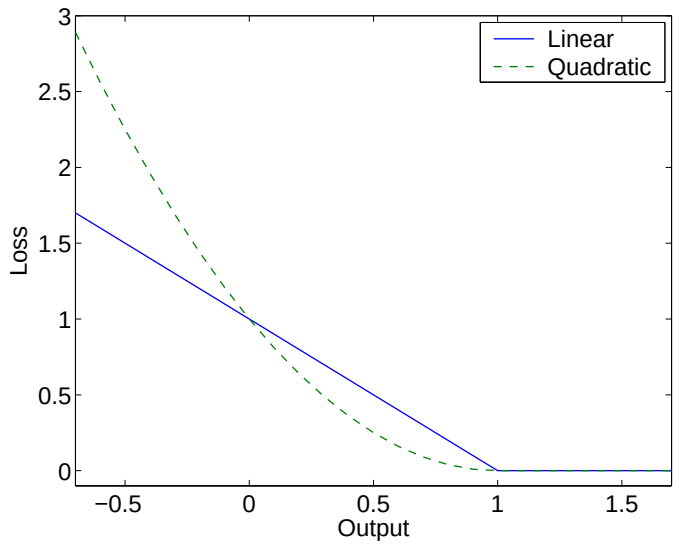
Example: MNIST

Each curve corresponds to a different training set size and is drawn for various number of iterations in a non-linear conjugate gradient minimization.



Use *differentiable* objective functions

- Numerically: usually easier to optimize.
- Statistically: no particular reason to prefer hinge loss to squared hinge loss.



SVM optimization in the primal

See article and Matlab code at: http://olivier.chapelle.cc/primal/

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 $n_{\text{sv}} \ll n$.

Competition

- For the competition, we simply did *one* Newton step.
- Since the number of dimensions was rather small, we could compute and invert the Hessian explicitly (no linear conjugate gradient).
- Since we did only one step, the line search was not needed (but we still did it).
- Matlab code available at: http://olivier.chapelle.cc/primal/
  But one Newton steps is very easy to code: one Matlab line!
  (this is just linear regression)

Caching

One Newton step takes $O(nd^2)$ operations for the Hessian computation

In contrast, we tried to take k steps of non-linear conjugate gradient with $k < d$. Time complexity is $O(knd)$. But this was slower than Newton! $[n = 10^5, k = 100, d = 500]$.

The reason is probably that matrix-matrix multiplications are relatively faster than matrix-vector multiplications due to a better cache optimization.

$\longrightarrow$ More generally, this suggests that mini-batch algorithms should be faster than online / sequential methods.

Similar observation was made by Yoshua Bengio last year at a Nips workshop: *Speeding Up Stochastic Gradient Descent*, `http://videolectures.net/eml07_bengio_ssg/`

Sequential Dual Method (SDM SVM L1/L2)

- Based on *Hsieh et al, ICML 2008*. Method is very close to "`-s 3`" of LIBLINEAR.
- L1=hinge loss; L2=squared hinge loss
- *Basic iteration:* Select a subset of examples, order them randomly and update the dual variable of one example at a time.
- Two types of iterations: *Full* and *Shrunk*
  - *Full:* all examples are chosen for update.
  - *Shrunk:* only examples whose dual variable is away from boundary (0 and C in L1 and only 0 in L2) are chosen for update.

Implementation notes

- Stopped when one of the following occurs:
  1. dual optimality violation is within 0.1.
  2. total number of effective epochs is at most 20.

  Post-competition realization: Stopping based on stationarity of *aoPRC* on a subset of training examples would have yielded better results.
- When training with dataset size n, C is taken to be a function of n as: $C = \tilde{C}\sqrt{N/n}$, where N is the size of full training set.
- For each dataset the value of $\tilde{C}$ was tuned using 100,000 training examples and optimizing *aoPRC* on the remaining labeled examples.

Two final thoughts

The main bottleneck in training linear classifiers is reading the data: more than 90% of the time during the competition was spent reading the data!

Future challenge:

1. Consider datasets which do not fit in memory.
2. Include reading time in the evaluation.

$\longrightarrow$ *Need for algorithms which try to minimize the number of passes on the data*.

A robust method should be fully automated.
$\longrightarrow$ Time needed for *model selection* should also be included in the evaluation.

The code submitted by a competitor should be oblivious to the datasets it is tested on.