Why Large-scale Datasets?

- Data Mining

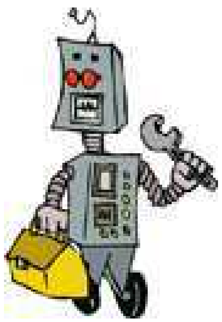
Gain competitive advantages by analyzing data that describes the life of our computerized society.

- Artificial Intelligence

Emulate cognitive capabilities of humans. Humans learn from abundant and diverse data.

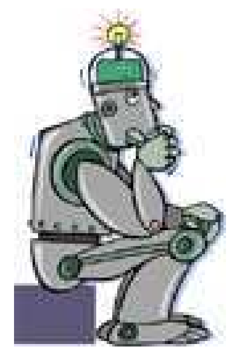
The Computerized Society Metaphor

- A society with just two kinds of computers:



$\leftarrow$ Makers do business and generate revenue. They also produce data in proportion with their activity.

Thinkers analyze the data to increase revenue by finding competitive advantages. $\rightarrow$



- When the population of computers grows:
  - The ratio #Thinkers/#Makers must remain bounded.
  - The Data grows with the number of Makers.
  - The number of Thinkers does not grow faster than the Data.

Limited Computing Resources

- **The computing resources available for learning do not grow faster than the volume of data.**
  - The cost of data mining cannot exceed the revenues.
  - Intelligent animals learn from streaming data.
- **Most machine learning algorithms demand resources that grow faster than the volume of data.**
  - Matrix operations (n^3 time for n^2 coefficients).
  - Sparse matrix operations are worse.

Part I

Statistical Efficiency versus Computational Costs.

This part is based on a joint work with Olivier Bousquet.

Simple Analysis

- **Statistical Learning Literature:**

"It is good to optimize an objective function than ensures a fast estimation rate when the number of examples increases."

- **Optimization Literature:**

"To efficiently solve large problems, it is preferable to choose an optimization algorithm with strong asymptotic properties, e.g. superlinear."

- **Therefore:**

"To address large-scale learning problems, use a superlinear algorithm to optimize an objective function with fast estimation rate.
Problem solved."

The purpose of this presentation is. . .

Too Simple an Analysis

- **Statistical Learning Literature:**

"It is good to optimize an objective function than ensures a fast estimation rate when the number of examples increases."

- **Optimization Literature:**

"To efficiently solve large problems, it is preferable to choose an optimization algorithm with strong asymptotic properties, e.g. superlinear."

- **Therefore:** *(error)*

"To address large-scale learning problems, use a superlinear algorithm to optimize an objective function with fast estimation rate.
Problem solved."

. . . to show that this is completely wrong !

Objectives and Essential Remarks

- Baseline large-scale learning algorithm

Randomly discarding data is the simplest way to handle large datasets.

  - What are the statistical benefits of processing more data?
  - What is the computational cost of processing more data?

- We need a theory that joins Statistics and Computation!

– 1967: Vapnik's theory does not discuss computation.
– 1981: Valiant's learnability excludes exponential time algorithms, but (i) polynomial time can be too slow, (ii) few actual results.
– We propose a simple analysis of approximate optimization. . .

Learning Algorithms: Standard Framework

- Assumption: examples are drawn independently from an unknown probability distribution $P(x,y)$ that represents the rules of Nature.
- Expected Risk: $E(f) = \int \ell(f(x),y)\,dP(x,y)$.
- Empirical Risk: $E_n(f) = \frac{1}{n}\sum \ell(f(x_i),y_i)$.
- We would like f^* that minimizes $E(f)$ among all functions.
- In general $f^* \notin \mathcal{F}$.
- The best we can have is $f^*_{\mathcal{F}} \in \mathcal{F}$ that minimizes $E(f)$ inside $\mathcal{F}$.
- But $P(x,y)$ is unknown by definition.
- Instead we compute $f_n \in \mathcal{F}$ that minimizes $E_n(f)$.
  Vapnik-Chervonenkis theory tells us when this can work.

Learning with Approximate Optimization

Computing $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ is often costly.

Since we already make lots of approximations,

why should we compute f_n exactly?

Let's assume our optimizer returns $\tilde{f}_n$

such that $E_n(\tilde{f}_n) < E_n(f_n) + \rho$.

For instance, one could stop an iterative
optimization algorithm long before its convergence.

Decomposition of the Error (i)

$$
\begin{aligned}
E(\tilde{f}_n) - E(f^*) &= E(f^*_{\mathcal{F}}) - E(f^*) && \text{Approximation error} \\
&+ E(f_n) - E(f^*_{\mathcal{F}}) && \text{Estimation error} \\
&+ E(\tilde{f}_n) - E(f_n) && \text{Optimization error}
\end{aligned}
$$

Problem:

Choose $\mathcal{F}$, n, and ρ to make this as small as possible,

subject to budget constraints $\left\{ \begin{array}{l} \text{maximal number of examples } n \\ \text{maximal computing time } T \end{array} \right.$

Decomposition of the Error (ii)

Approximation error bound: (Approximation theory)

– decreases when $\mathcal{F}$ gets larger.

Estimation error bound: (Vapnik-Chervonenkis theory)

– decreases when n gets larger.

– increases when $\mathcal{F}$ gets larger.

Optimization error bound: (Vapnik-Chervonenkis theory plus tricks)

– increases with ρ.

Computing time T: (Algorithm dependent)

– decreases with ρ

– increases with n

– increases with $\mathcal{F}$

Small-scale vs. Large-scale Learning

We can give *rigorous definitions*.

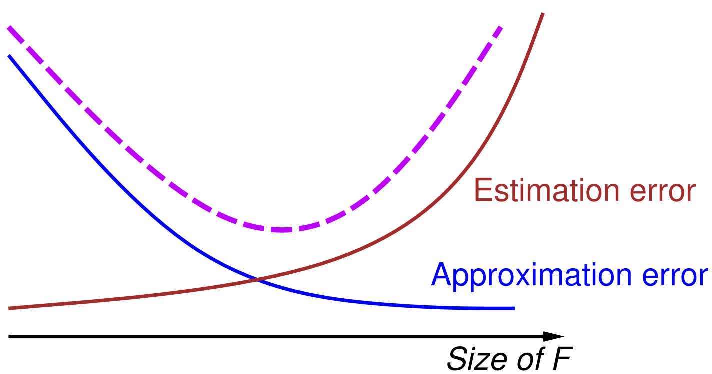
- **Definition 1:**
  We have a **small-scale learning** problem when the **active budget constraint is the number of examples n.**

- **Definition 2:**
  We have a **large-scale learning** problem when the **active budget constraint is the computing time T.**

Small-scale Learning

The active budget constraint is the number of examples.

- To reduce the estimation error, take n as large as the budget allows.
- To reduce the optimization error to zero, take $\rho = 0$.
- We need to adjust the size of $\mathcal{F}$.



See Structural Risk Minimization (Vapnik 74) and later works.

Large-scale Learning

The active budget constraint is the computing time.

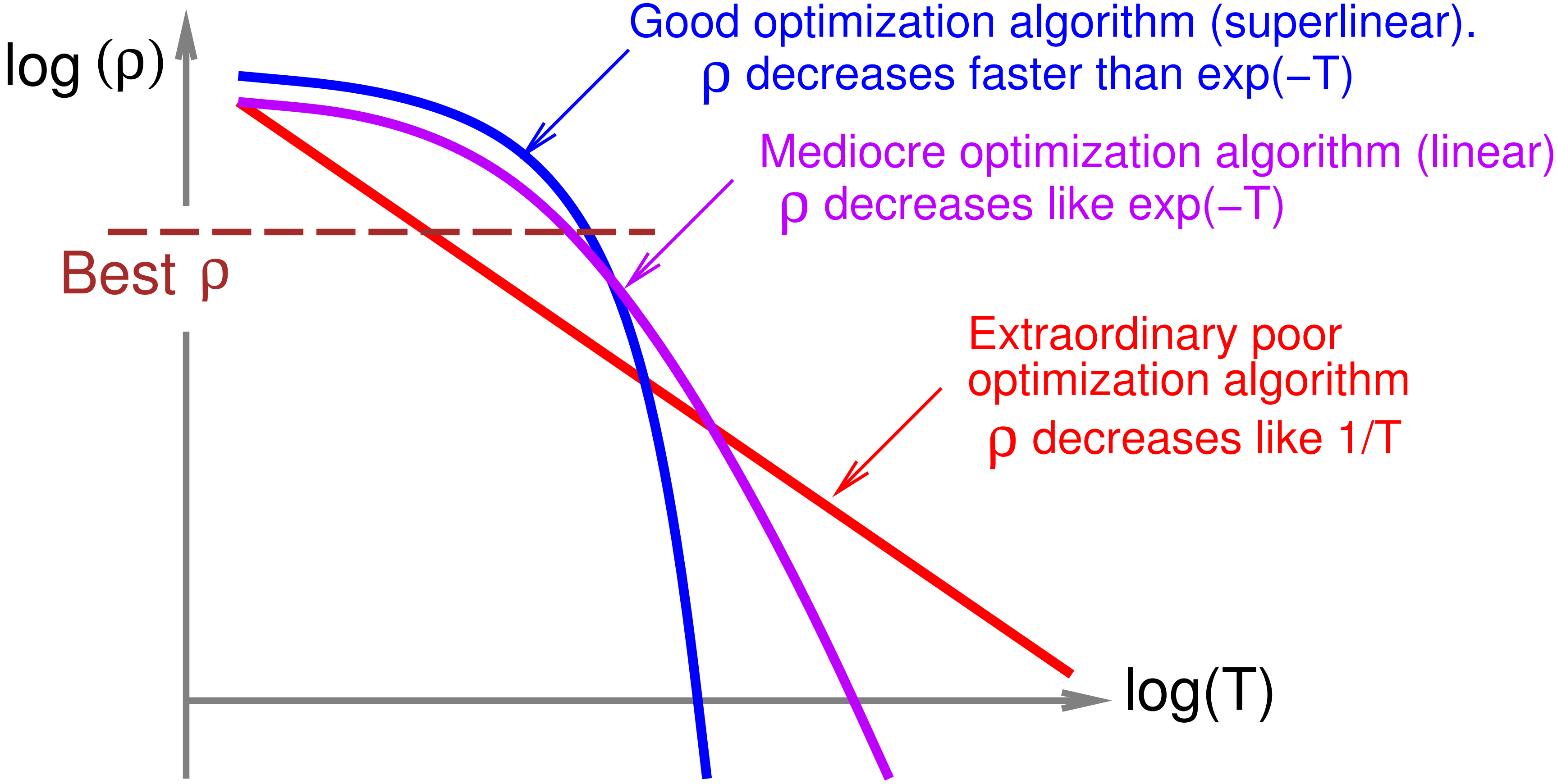
- More complicated tradeoffs.
  The computing time depends on the three variables: $\mathcal{F}$, n, and ρ.

- Example.
  If we choose ρ small, we decrease the optimization error. But we must also decrease $\mathcal{F}$ and/or n with adverse effects on the estimation and approximation errors.

- The exact tradeoff depends on the optimization algorithm.

- We can compare optimization algorithms rigorously.

Executive Summary



Asymptotics: Estimation

Uniform convergence bounds (with capacity $d+1$)

Estimation error $\leq \mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha}\right)$ with $\frac{1}{2} \leq \alpha \leq 1$.

There are in fact three types of bounds to consider:

- Classical V-C bounds (pessimistic): $\mathcal{O}\left(\sqrt{\frac{d}{n}}\right)$
- Relative V-C bounds in the realizable case: $\mathcal{O}\left(\frac{d}{n}\log\frac{n}{d}\right)$
- Localized bounds (variance, Tsybakov): $\mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha}\right)$

Fast estimation rates are a big theoretical topic these days.

Asymptotics: Estimation+Optimization

Uniform convergence arguments give

$$\text{Estimation error} + \text{Optimization error} \leq \mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha} \; + \; \rho\right) \; .$$

This is true for all three cases of uniform convergence bounds.

Scaling laws for ρ when $\mathcal{F}$ is fixed

The approximation error is constant.

– No need to choose ρ smaller than $\mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha}\right)$.

– Not advisable to choose ρ larger than $\mathcal{O}\left(\left[\frac{d}{n}\log\frac{n}{d}\right]^{\alpha}\right)$.

. . . Approximation+Estimation+Optimization

When $\mathcal{F}$ is chosen via a λ-regularized cost

– Uniform convergence theory provides bounds for simple cases
(Massart-2000; Zhang 2005; Steinwart et al., 2004-2007; . . .)

– Computing time depends on both λ and ρ.

– Scaling laws for λ and ρ depend on the optimization algorithm.

When $\mathcal{F}$ is realistically complicated

Large datasets matter

– because one can use more features,

– because one can use richer models.

Bounds for such cases are rarely realistic enough.

Luckily there are interesting things to say for $\mathcal{F}$ fixed.

Case Study

Simple parametric setup

– $\mathcal{F}$ is fixed.
– Functions $f_w(x)$ linearly parametrized by $w \in \mathbb{R}^d$.

Comparing four iterative optimization algorithms for $E_n(f)$

1. Gradient descent.
2. Second order gradient descent (Newton).
3. Stochastic gradient descent.
4. Stochastic second order gradient descent.

Quantities of Interest

- Empirical Hessian at the empirical optimum w_n.

$$H = \frac{\partial^2 E_n}{\partial w^2}(f_{w_n}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial^2 \ell(f_n(x_i), y_i)}{\partial w^2}$$

- Empirical Fisher Information matrix at the empirical optimum w_n.

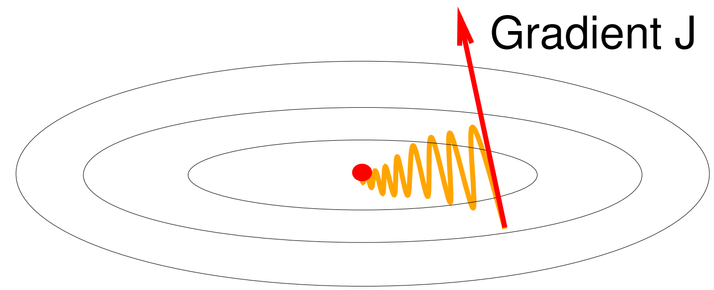
$$G = \frac{1}{n}\sum_{i=1}^{n}\left[\left(\frac{\partial \ell(f_n(x_i), y_i)}{\partial w}\right)\left(\frac{\partial \ell(f_n(x_i), y_i)}{\partial w}\right)'\right]$$

- **Condition number**

  We assume that there are $\lambda_{\min}$, $\lambda_{\max}$ and ν such that
  - $\mathrm{trace}\big(GH^{-1}\big) \approx \nu$.
  - $\mathrm{spectrum}\big(H\big) \subset [\lambda_{\min}, \lambda_{\max}]$.

  and we define the condition number $\kappa = \lambda_{\max}/\lambda_{\min}$.

Gradient Descent (GD)



Iterate

- $w_{t+1} \leftarrow w_t - \eta \frac{\partial E_n(f_{w_t})}{\partial w}$

Best speed achieved with fixed learning rate $\eta = \frac{1}{\lambda_{\max}}$.
(e.g., Dennis & Schnabel, 1983)

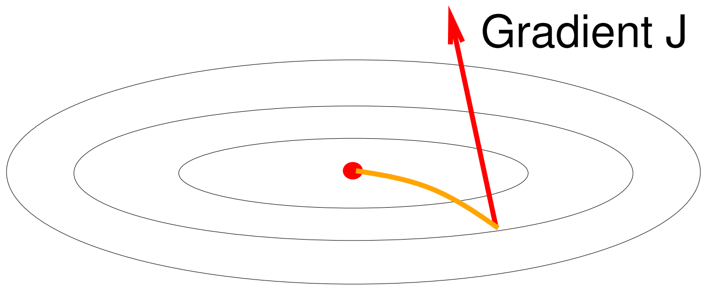
	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
GD	$\mathcal{O}(nd)$	$\mathcal{O}\left(\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(nd\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2 \kappa}{\varepsilon^{1/\alpha}} \log^2 \frac{1}{\varepsilon}\right)$

– In the last column, n and ρ are chosen to reach ε as fast as possible.
– Solve for ε to find the best error rate achievable in a given time.
– Remark: abuses of the $\mathcal{O}()$ notation

Second Order Gradient Descent (2GD)

Iterate

- $w_{t+1} \leftarrow w_t - H^{-1} \dfrac{\partial E_n(f_{w_t})}{\partial w}$



We assume H^{-1} is known in advance.
Superlinear optimization speed (e.g., Dennis & Schnabel, 1983)

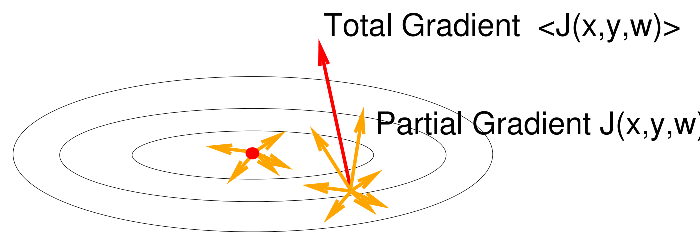
	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
2GD	$\mathcal{O}\big(d(d+n)\big)$	$\mathcal{O}\left(\log\log\frac{1}{\rho}\right)$	$\mathcal{O}\left(d(d+n)\log\log\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2}{\varepsilon^{1/\alpha}}\log\frac{1}{\varepsilon}\log\log\frac{1}{\varepsilon}\right)$

– Optimization speed is much faster.
– Learning speed only saves the condition number $\boldsymbol{\kappa}$.

Stochastic Gradient Descent (SGD)

Iterate

- Draw random example (x_t, y_t).
- $$w_{t+1} \leftarrow w_t - \frac{\eta}{t}\,\frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}$$



Best decreasing gain schedule with $\eta = \frac{1}{\lambda_{\min}}$.
(see Murata, 1998; Bottou & LeCun, 2004)

	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
SGD	$\mathcal{O}(d)$	$\frac{\nu k}{\rho} + o\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d\,\nu\,k}{\rho}\right)$	$\mathcal{O}\left(\frac{d\,\nu\,k}{\varepsilon}\right)$

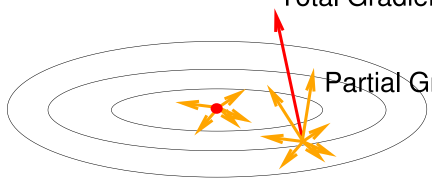
With $1 \leq k \leq \kappa^2$

– Optimization speed is *catastrophic*.
– Learning speed does not depend on the statistical estimation rate $\boldsymbol{\alpha}$.
– Learning speed depends on condition number $\boldsymbol{\kappa}$ but *scales very well*.

Second order Stochastic Descent (2SGD)

Iterate

- Draw random example (x_t, y_t).
- $w_{t+1} \leftarrow w_t - \frac{1}{t} H^{-1} \frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}$



Replace scalar gain $\frac{\eta}{t}$ by matrix $\frac{1}{t} H^{-1}$.

	Cost per iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $E(\tilde{f}_n) - E(f^*_{\mathcal{F}}) < \varepsilon$
2SGD	$\mathcal{O}\left(d^2\right)$	$\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2\,\nu}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2\,\nu}{\varepsilon}\right)$

– Each iteration is d times more expensive.
– The number of iterations is reduced by κ^2 (or less.)
– Second order only changes the constant factors.

Part II

Learning with Stochastic Gradient Descent.

Benchmarking SGD in Simple Problems

- The theory suggests that SGD is very competitive.
  - Many people associate SGD with trouble.
- SGD historically associated with back-propagation.
  - Multilayer networks are very hard problems (nonlinear, nonconvex)
  - What is difficult, SGD or MLP?

- **Try PLAIN SGD on simple learning problems.**
  - Support Vector Machines
  - Conditional Random Fields

Download from `http://leon.bottou.org/projects/sgd`.
These simple programs are very short.

See also (Shalev-Schwartz et al., 2007; Vishwanathan et al., 2006)

Text Categorization with SVMs

- **Dataset**
  - Reuters RCV1 document corpus.
  - 781,265 training examples, 23,149 testing examples.
  - 47,152 TF-IDF features.

- **Task**
  - Recognizing documents of category `CCAT`.
  - Minimize $E_n = \frac{1}{n}\sum_i \left(\frac{\lambda}{2} w^2 + \ell(w\, x_i + b,\, y_i) \right)$.
  - Update $w \leftarrow w - \eta_t \nabla(w_t, x_t, y_t) = w - \eta_t \left(\lambda w + \frac{\partial \ell(w\, x_t + b,\, y_t)}{\partial w} \right)$

Same setup as (Shalev-Schwartz et al., 2007) but plain SGD.

Text Categorization with SVMs

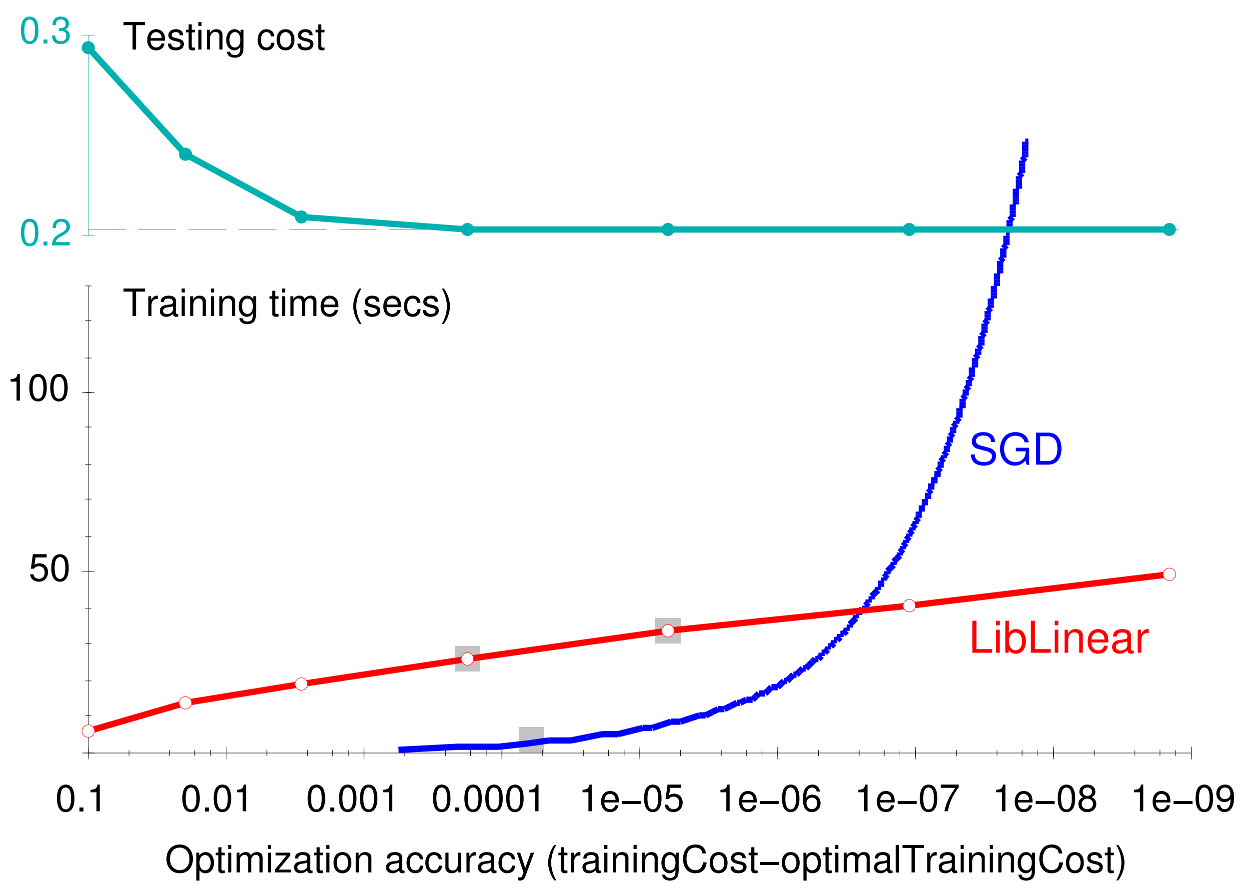
- **Results: Linear SVM**
  $\ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\} \quad \lambda = 0.0001$

	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	*0.2275*	*6.02%*
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	*0.2275*	*6.02%*

- **Results: Log-Loss Classifier**
  $\ell(\hat{y}, y) = log(1 + exp(-y\hat{y})) \quad \lambda = 0.00001$

	Training Time	Primal cost	Test Error
LibLinear ($\varepsilon = 0.01$)	30 secs	0.18907	*5.68%*
LibLinear ($\varepsilon = 0.001$)	44 secs	*0.18890*	5.70%
SGD	2.3 secs	0.18893	*5.66%*

The Wall


0.3
Testing cost
0.2
Training time (secs)
100
50
SGD
LibLinear
0.1
0.01
0.001
0.0001
1e−05
1e−06
1e−07
1e−08
1e−09
Optimization accuracy (trainingCost−optimalTrainingCost)

More SVM Experiments

From: Patrick Haffner

Date: Wednesday 2007-09-05 14:28:50

. . . I have tried on some of our main datasets. . .

I can send you the example, it is so striking!

– Patrick

Dataset	Train size	Number of features	% non-0 features	LIBSVM (SDot)	LLAMA SVM	LLAMA MAXENT	SGDSVM
Reuters	781K	47K	0.1%	210,000	3930	153	7
Translation	1000K	274K	0.0033%	days	47,700	1,105	7
SuperTag	950K	46K	0.0066%	31,650	905	210	1
Voicetone	579K	88K	0.019%	39,100	197	51	1

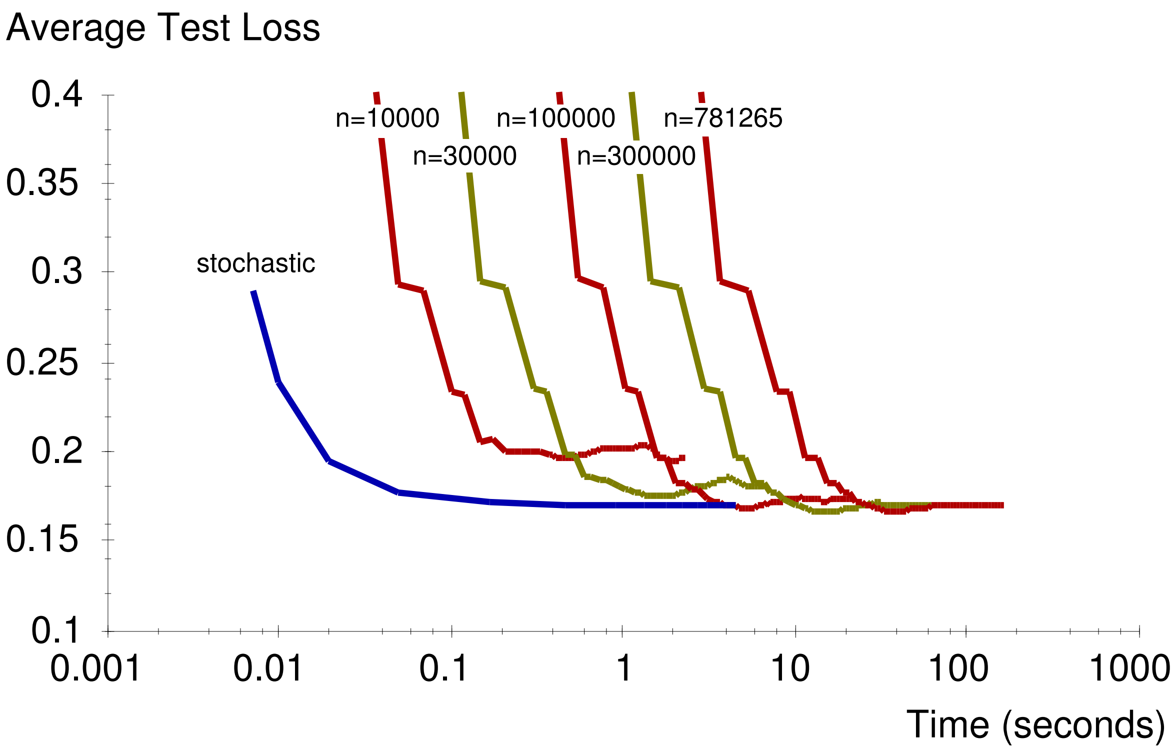
More SVM Experiments

From: Olivier Chapelle

Date: Sunday 2007-10-28 22:26:44

. . . you should really run batch with various training set sizes . . .

– Olivier



Log-loss problem

Batch Conjugate Gradient on various training set sizes

Stochastic Gradient on the full set