Large Scale Learning

Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

Topics Covered

- Gradients and Subgradients
- Gradient descent based learning
  - PEGASOS
  - SARAH
  - Coordinate Descent
- Kernel Approximation
  - Fourier Approximation
- Randomized Algorithms
  - Random Kitchen Sinks
  - Random Projections
  - Extreme Learning Machine
  - Doubly Stochastic Learning

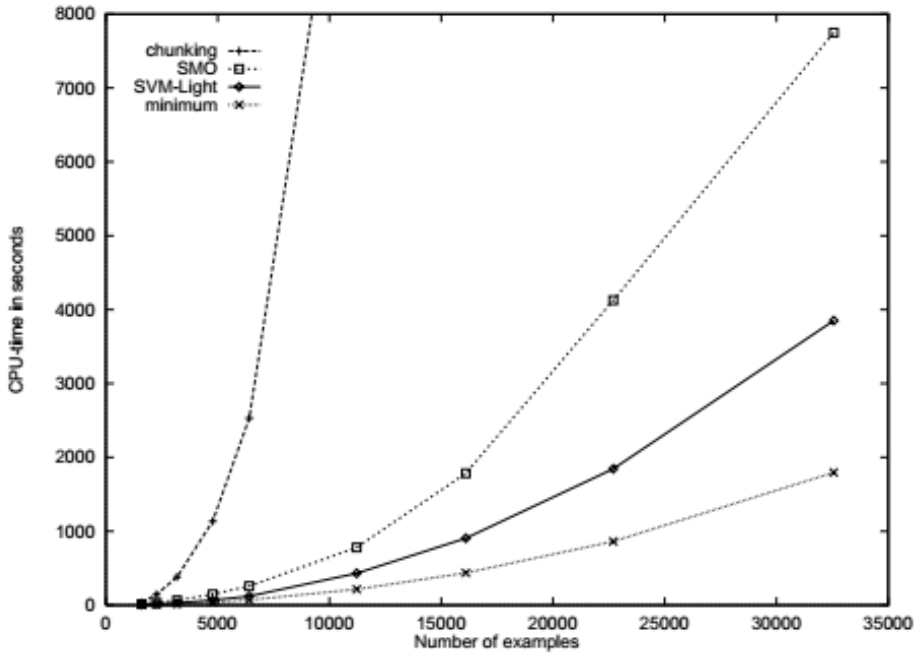
Learning from Big Data

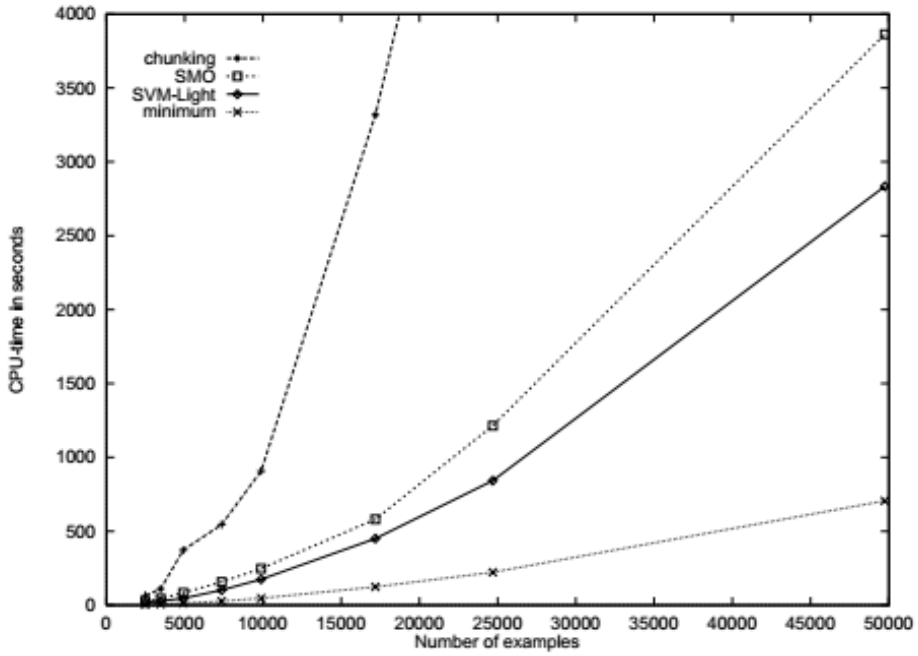
- Classification of large data sets
- Requires super-efficient learning algorithms
  - These problems are constrained by the total computation time
  - In small scale learning problems, the error is constrained by the number of training examples
- QP solver for SVM
  - Given n examples, the computational cost is $O(n^2)$ to $O(n^3)$ given a precomputed kernel matrix for small and large values of C respectively
    - Bottou and Lin, “Support Vector Machine Solvers”, JMLR

Big Data issues

- Kernel computation
  - Computing the kernel is expensive
  - Only a few kernel values have any impact on the solution (support vectors!)
  - The kernel does not fit in the memory

Computation time





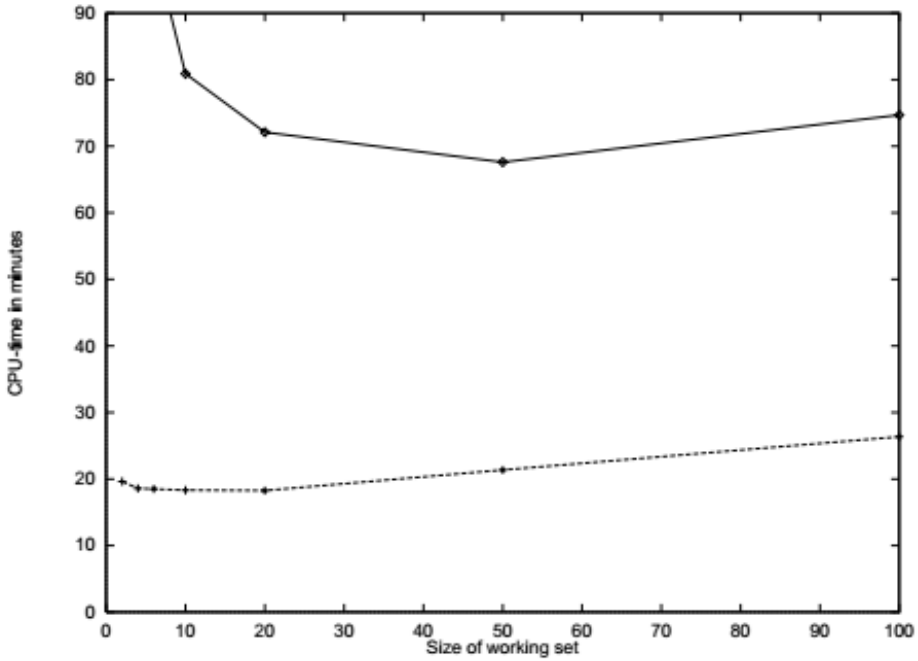
- Joachims, “Making Large-Scale SVM Learning Practical” in: 'Advances in Kernel Methods - Support Vector Learning', Bernhard Scholkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.),
MIT Press, Cambridge, USA, 1998

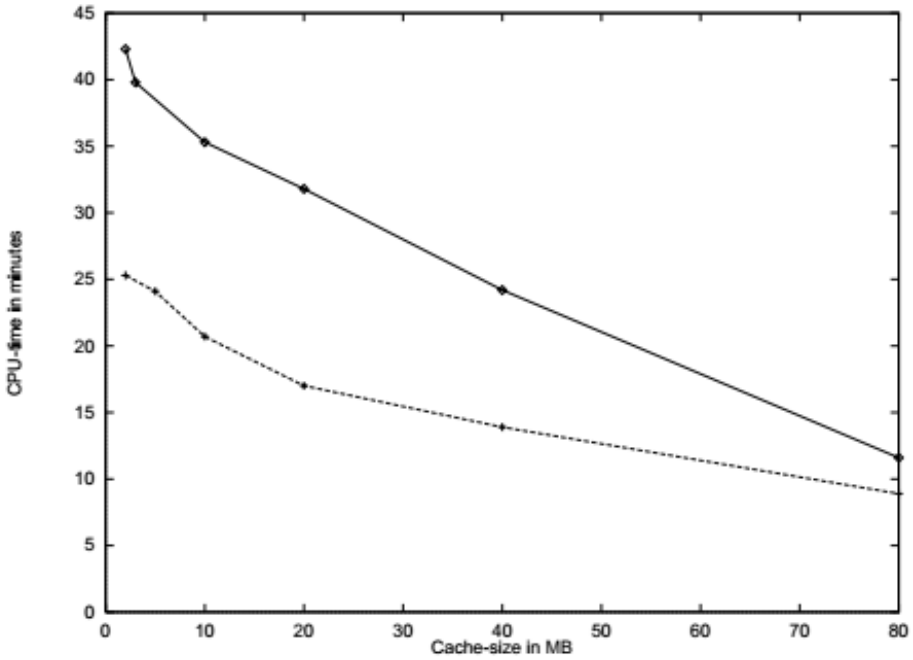
Computation time

- Scikit-SVM has complexity of $O(dn^2)$ to $O(dn^3)$ and uses libsvm
  - Can benefit from sparse data
  - Can benefit from caching the kernel
  - Linear SVM is faster (sklearn.svm.LinearSVC) and uses liblinear for larger number of examples
- Check out: Tips on practical use
  - Data Copying
  - Kernel Caching
  - Data Scaling and Preprocessing (badly scaled data will cause issues!)
  - Uses Joblib (built-in support for parallelization!)
  - Use Stochastic Gradient Descent (sklearn.linear_model.SGDClassifier)

http://scikit-learn.org/stable/modules/svm.html#tips-on-practical-use

Effect of Kernel Caching





Representation

- SVM style algorithms can be represented as:

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\mathbf{x},y))$$

$$\ell(\mathbf{w};(\mathbf{x},y)) \;=\; \max\{0, 1 - y\,\langle \mathbf{w}, \mathbf{x}\rangle\}$$

  - The loss functions and the regularizers can be changed
  - We solved the dual of the above formulation using a QP Solver
  - However, to develop new classifiers, we need to know about some “easy” and fast optimization methods

Types of SVM Solvers

- Interior Point Methods: $O(n^2)$, reaches an ϵ-accurate solution in $O(\log(\log(1/\epsilon))$
- Decomposition based methods: SMO, SVM-Light, typically super-linear in the number of examples
- Cutting plane methods: reach an ϵ-accurate solution in $O(nd / \lambda\epsilon)$ time
- Primal solvers: Use the representation theorem $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ to solve the nonlinear classification proble in the dual with conjugate gradient or newton optimization methods
- Stochastic gradient descent

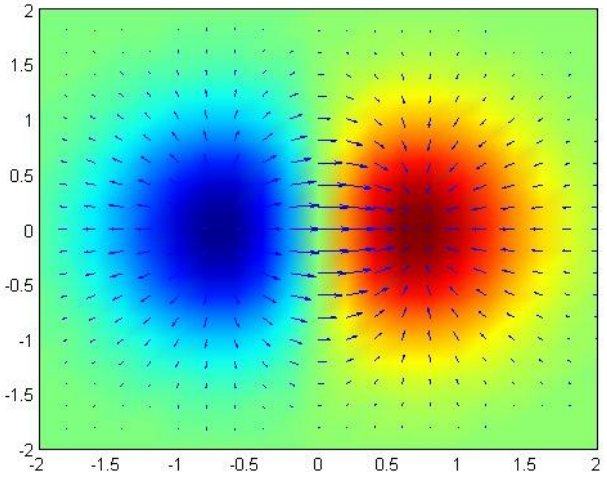
Efficient Algorithms for your own learning machines

- PEGASOS
  - Shalev-Shwartz, Shai, Yoram Singer, Nathan Srebro, and Andrew Cotter. “Pegasos: Primal Estimated Sub-Gradient Solver for SVM.” *Mathematical Programming* 127, no. 1 (March 1, 2011): 3–30. doi:10.1007/s10107-010-0420-4.
  - Gradient based method
  - Solves the SVM in the primal
  - The number of iterations is not dependent upon the number of examples
  - Easy implementation!

Gradient

- a generalization of the usual concept of derivative of a function in one dimension to a function in several dimensions.
- Given a function $f(\boldsymbol{x}) = f(x_1, x_2, \ldots, x_n)$, its gradient is given by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

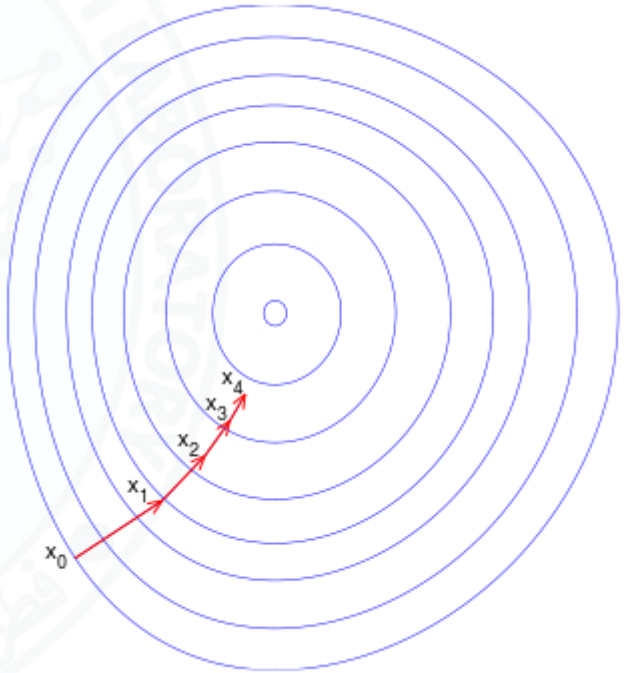


$$f(\boldsymbol{x}) = x_1 e^{{x_1}^2 + {x_2}^2}$$

Gradient Descent

- A derivative based optimization method to find local minimum

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

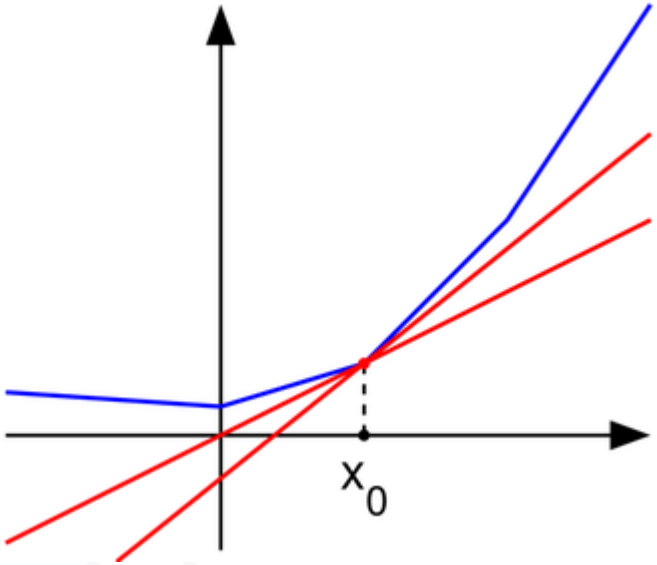


Subgradient

– Generalizes the gradient to functions which are not differentiable.

– Given a function f(x), its subderivative is any value of ‘c’ for which

$$f(x) - f(x_0) \geq c(x - x_0)$$



Subgradient

- the set of sub-derivatives at x_0 for a convex function is a nonempty closed interval [a, b]

$$a = \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0}$$

$$b = \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0}$$

- The set [*a*, *b*] of all subderivatives is called the subdifferential of the function *f* at x_0. If *f* is convex and its subdifferential at contains exactly one subderivative, then *f* is differentiable at x_0.

Subgradient

- The Taylor series approximation of a function at ‘a’ is given by

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots.$$

- The first order approximation is

$$f(a) + \frac{f'(a)}{1!}(x-a)$$

- Thus, if the first order approximation f(a)+c(x-a) of the function under-approximates the function, then ‘c’ is a subderivative of the function!

Examples

Loss function	Subgradient
$\ell(z, y_i) = \max\{0, 1 - y_i z\}$	$\mathbf{v}_t = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i z < 1 \\ \mathbf{0} & \text{otherwise} \end{cases}$
$\ell(z, y_i) = \log(1 + e^{-y_i z})$	$\mathbf{v}_t = -\frac{y_i}{1+e^{y_i z}} \mathbf{x}_i$
$\ell(z, y_i) = \max\{0, \lvert y_i - z \rvert - \epsilon\}$	$\mathbf{v}_t = \begin{cases} \mathbf{x}_i & \text{if } z - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - z > \epsilon \\ \mathbf{0} & \text{otherwise} \end{cases}$
$\ell(z, y_i) = \max_{y \in \mathcal{Y}} \delta(y, y_i) - z_{y_i} + z_y$	$\mathbf{v}_t = \phi(\mathbf{x}_i, \hat{y}) - \phi(\mathbf{x}_i, y_i)$ where $\hat{y} = \arg\max_y \delta(y, y_i) - z_{y_i} + z_y$
$\ell(z, y_i) = \log\left(1 + \sum_{r \neq y_i} e^{z_r - z_{y_i}}\right)$	$\mathbf{v}_t = \sum_r p_r \phi(\mathbf{x}_i, r) - \phi(\mathbf{x}_i, y_i)$ where $p_r = e^{z_r} / \sum_j e^{z_j}$

Derivation of Pegasos

- The basic optimization problem of SVM can be written as:

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\mathbf{x},y)) \ ,$$

where

$$\ell(\mathbf{w};(\mathbf{x},y)) \ = \ \max\{0, 1 - y\,\langle \mathbf{w},\mathbf{x}\rangle\} \ ,$$

- Pegasos operates by picking one example randomly at each iteration and computing the value of the objective function with respect to that (Stochastic!)

$$f(\mathbf{w};i_t) \ = \ \frac{\lambda}{2}\|\mathbf{w}\|^2 + \ell(\mathbf{w};(\mathbf{x}_{i_t},y_{i_t}))$$

Pegasos Derviation

- The subgradient of this function is given by

$$\nabla_t = \lambda \mathbf{w}_t - \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1] \, y_{i_t} \mathbf{x}_{i_t}$$

- The weight update equation can be written as

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t$$

- With the learning rate given by $\eta_t = 1/(\lambda t)$
  - The learning rate is gradually reduced ("annealing")
- The complete equation thus becomes

$$\mathbf{w}_{t+1} \leftarrow (1 - \frac{1}{t}) \mathbf{w}_t + \eta_t \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1] \, y_{i_t} \mathbf{x}_{i_t}$$

Projection step (optional)

- Limit the weight vector to a ball of radius $1/\sqrt{\lambda}$.
  - Better control of generalization

- Done by

$$\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$$

Pegasos Algorithm

INPUT: S, λ, T, k
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
Choose $A_t \subseteq [m]$, where $|A_t| = k$, uniformly at random
Set $A_t^+ = \{i \in A_t : y_i \langle \mathbf{w}_t, \mathbf{x}_i \rangle < 1\}$
Set $\eta_t = \frac{1}{\lambda t}$
Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$
[Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

Mini-batch version

- Select 'k' examples at each step in a set A_t
- Potentially better approximation of the gradient through averaging over examples in A_t

INPUT: S, λ, T, k
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
Choose $A_t \subseteq [m]$, where $|A_t| = k$, uniformly at random
Set $A_t^+ = \{i \in A_t : y_i \langle \mathbf{w}_t, \mathbf{x}_i \rangle < 1\}$
Set $\eta_t = \frac{1}{\lambda t}$
Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$
[Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

Convergence properties

- To reach an $\boldsymbol{\varepsilon} > 0$ accurate solution in which $\boldsymbol{f(w) \leq f(w^*) + \varepsilon}$, the number of iterations is $\boldsymbol{O\left(\frac{d}{\lambda\varepsilon}\right)}$
  - $\boldsymbol{d}$ is the dimensionality of the data

For Mercer's Kernels

- The weight update equation can be written as

$$\mathbf{w}_{t+1} \leftarrow (1 - \frac{1}{t})\mathbf{w}_t + \eta_t \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1] \, y_{i_t} \mathbf{x}_{i_t}$$

- Let's take: $\mathbf{v}_t = \frac{1}{|A_t|} \sum_{i \in A_t} \mathbb{1}[y_i \langle \mathbf{w}_t, \mathbf{x}_t \rangle < 1] \, y_i \mathbf{x}_i$

- Thus: $\mathbf{w}_{t+1} = \left(1 - \frac{1}{t}\right) \mathbf{w}_t + \frac{1}{t\lambda} \mathbf{v}_t$

This equation (12) in the paper has a mistake with a '-' when it should have been '+'

- Assume that at iteration 'i' we choose example 'i' and then try to see what its impact on the weight vector is uptil iteration 't' independent of other examples (assume only one example!)
- For this purpose we take
  - $w_i = 0$
  - The initial multiplier of v_t is thus $\frac{1}{\lambda i}$

Using Mercer's Conditions

- We can now solve the recurrence with
  - $\boldsymbol{w_i = 0}$
  - $\boldsymbol{w_{i+1}} = \left(1 - \frac{1}{i}\right)\boldsymbol{w_i} + \frac{1}{\lambda i}\boldsymbol{v_i} = \frac{1}{\lambda i}\boldsymbol{v_i}$
  - $\boldsymbol{w_{i+2}} = \left(1 - \frac{1}{i+1}\right)\boldsymbol{w_{i+1}} + \frac{1}{\lambda(i+1)}\boldsymbol{v}_{i+1} = \left(\frac{i}{i+1}\right)\frac{1}{\lambda i}\boldsymbol{v_i} + \frac{1}{\lambda(i+1)}\boldsymbol{v}_{i+1} = \frac{1}{\lambda(i+1)}(\boldsymbol{v_i} + \boldsymbol{v}_{i+1})$

- In general, $\boldsymbol{w_{t+1}} = \frac{1}{\lambda t}\sum_{i=1}^{t}\boldsymbol{v_i}$

- Substituting v_i $$\mathbf{w}_{t+1} = \frac{1}{\lambda t}\sum_{i=1}^{t}\mathbb{1}\left[y_{i_t}\left\langle \mathbf{w}_t, \phi(\mathbf{x}_{i_t})\right\rangle < 1\right] y_{i_t}\phi(\mathbf{x}_{i_t})$$

- Let, denote the number of times, uptil the current iteration, the example i_t has been chosen and it has violated the margin

$$\alpha_{t+1}[j] = \left|\left\{t' \leq t : i_{t'} = j \wedge y_j \left\langle \mathbf{w}_{t'}, \phi(\mathbf{x}_j)\right\rangle < 1\right\}\right|$$

Using Mercer's Conditions

- With this definition and $$\mathbf{w}_{t+1} = \frac{1}{\lambda t} \sum_{i=1}^{t} \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \phi(\mathbf{x}_{i_t}) \rangle < 1] \, y_{i_t} \phi(\mathbf{x}_{i_t})$$
- We get: $\boldsymbol{w_{t+1}} = \frac{1}{\lambda t} \alpha_{t+1}[j] y_{i_t} \boldsymbol{\phi}(x_{i_t})$
- Since we have a total of m' examples we get

$$\mathbf{w}_{t+1} = \frac{1}{\lambda t} \sum_{j=1}^{m} \alpha_{t+1}[j] \, y_j \phi(\mathbf{x}_j)$$

- The final weight vector will thus be: $\boldsymbol{w} = \frac{1}{\lambda T} \sum_{j=1}^{m} \alpha_{T+1}[j] y_j \boldsymbol{\phi}(x_j)$

- This is inline with the Representer Theorem
  - The weight vector is being expressed as a linear combination of the given examples

Using Mercer's Conditions

- With this, we can now evaluate the output score of an example as:
  - $\boldsymbol{w^T \phi(x)} = \frac{1}{\lambda T}\sum_{j=1}^{m} \alpha_{T+1}[j] y_j \boldsymbol{\phi}^T(x_j)\boldsymbol{\phi(x)} = \frac{1}{\lambda T}\sum_{j=1}^{m} \alpha_{T+1}[j] y_j K(x_j, x)$
- Thus for learning, all we need are the α for each example
- Note that the calculation of α requires us to see if the chosen example violates the margin or not which in turn requires us to compute the loss function which is dependent on the output score for that example at that iteration
- Thus, each iteration now becomes O(m)
  - Overall, runtime with kernels thus becomes $\boldsymbol{O\left(\frac{m}{\lambda \varepsilon}\right)}$
  - As a consequence of this, the runtime is dependent upon the number of examples even though the number of iterations is not
  - Also note that the problem is still being solved in the primal as the optimization is still with respect to w and not the lagrange variables in the dual representation

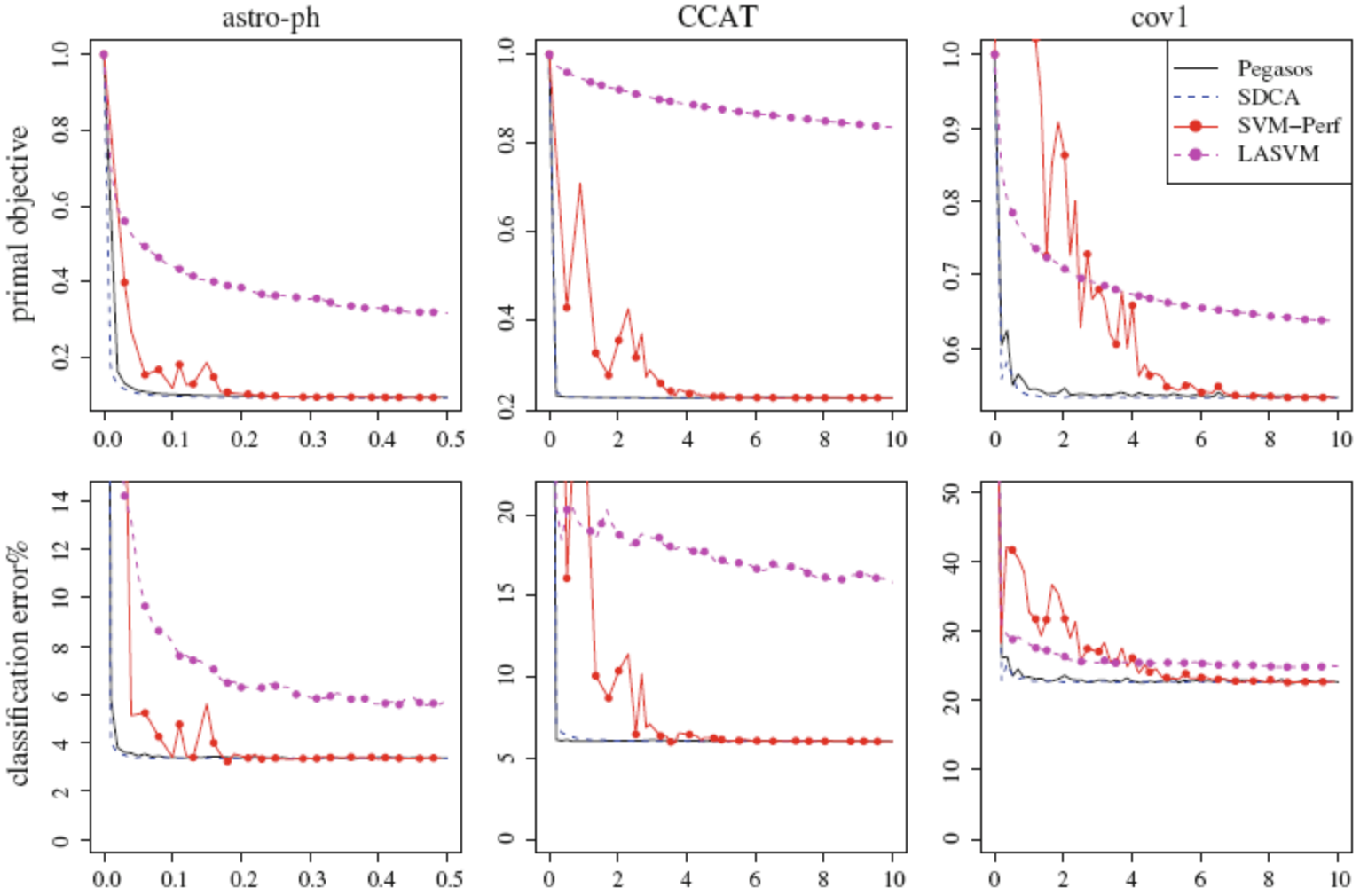


Fig. 4 Comparison of linear SVM optimizers. Primal suboptimality (*top row*) and testing classification error (*bottom row*), for one run each of Pegasos, stochastic DCA, SVM-Perf, and LASVM, on the astro-ph (*left*), CCAT (*center*) and cov1 (*right*) datasets. In all plots the horizontal axis measures runtime in seconds

Timing results (linear)

- Superfast!

Table 1 Training runtime and test error achieved (in parentheses) using various optimization methods on linear SVM problems

Dataset	Pegasos	SDCA	SVM-Perf	LASVM
astro-ph	0.04*s* (3.56%)	0.03*s* (3.49%)	0.1*s* (3.39%)	54*s* (3.65%)
CCAT	0.16*s* (6.16%)	0.36*s* (6.57%)	3.6*s* (5.93%)	$> 18000s$
cov1	0.32*s* (23.2%)	0.20*s* (22.9%)	4.2*s* (23.9%)	210*s* (23.8%)

Kernelized

- Slower!

Dataset	Pegasos	SDCA	SVM-Light	LASVM
Reuters	15*s* (2.91%)	13*s* (3.15%)	4.1*s* (2.82%)	4.7*s* (3.03%)
Adult	30*s* (15.5%)	4.8*s* (15.5%)	59*s* (15.1%)	1.5*s* (15.6%)
USPS	120*s* (0.457%)	21*s* (0.508%)	3.3*s* (0.457%)	1.8*s* (0.457%)
MNIST	4200*s* (0.6%)	1800*s* (0.56%)	290*s* (0.58%)	280*s* (0.56%)

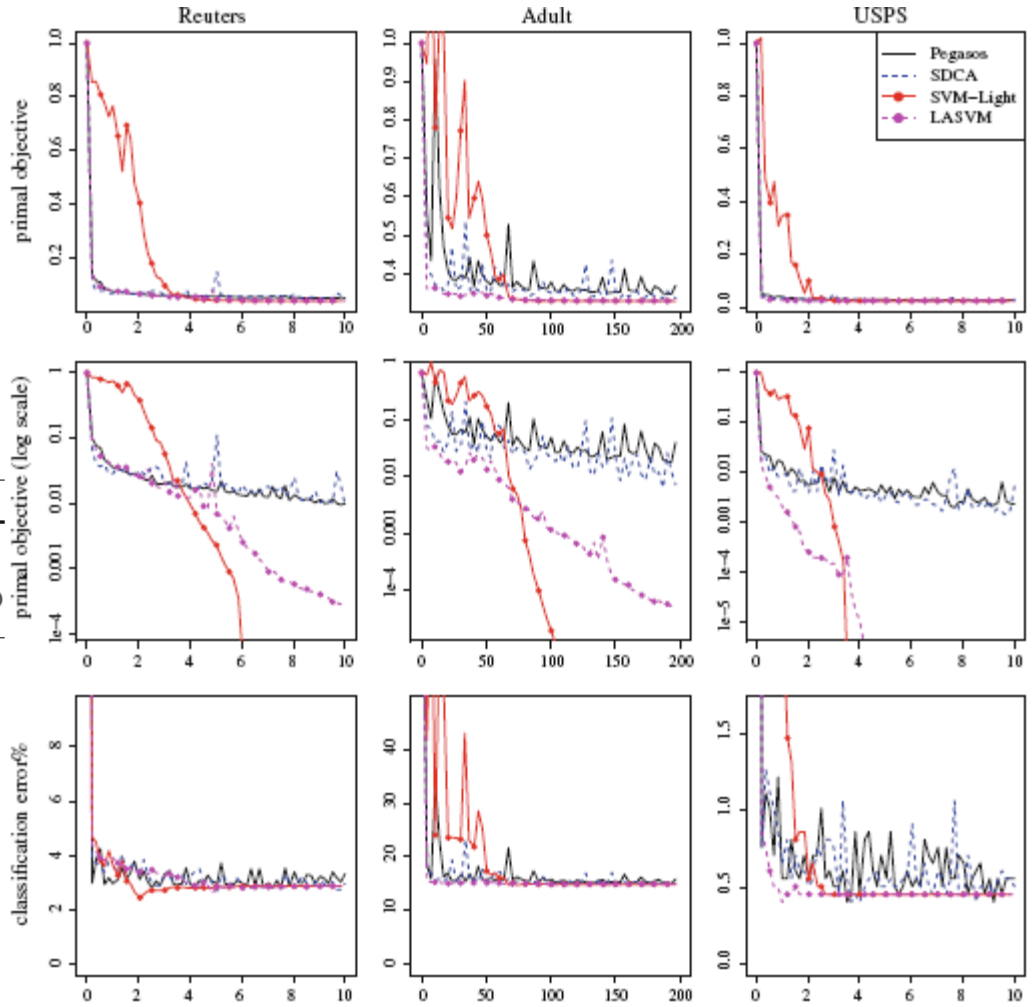


Fig. 5 Comparison of kernel SVM optimizers. Primal suboptimality (*top row*), primal suboptimality in log scale (*middle row*) and testing classification error (*bottom row*), for one run each of Pegasos, stochastic DCA, SVM-Light, and LASVM, on the Reuters (*left column*), Adult (*center column*) and USPS (*right column*) datasets. Plots of traces generated on the MNIST dataset (not shown) appear broadly similar to those for the USPS dataset. The horizontal axis is runtime in seconds

Effect of batches

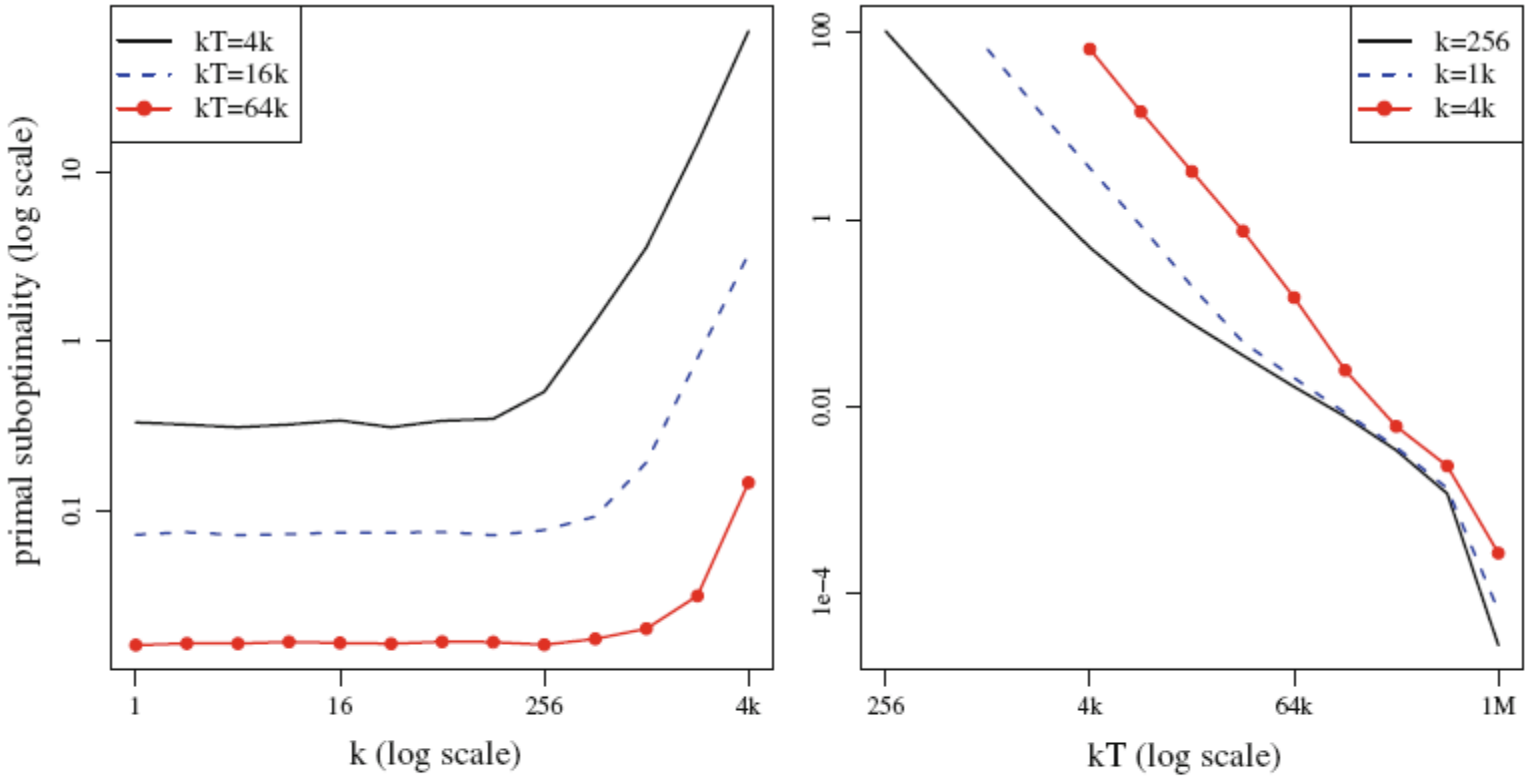
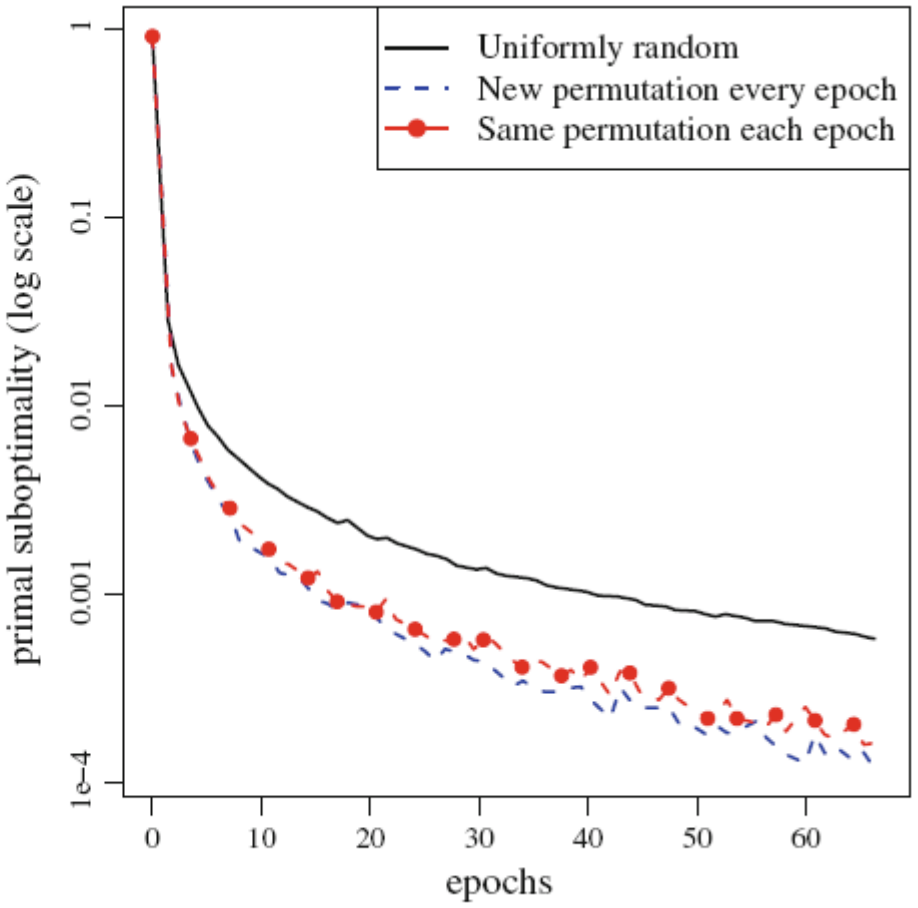


Fig. 7 The effect of the mini-batch size on the runtime of Pegasos for the astro-ph dataset. The first plot shows the primal suboptimality achieved for certain fixed values of overall runtime kT, for various values of the mini-batch size k. The second plot shows the primal suboptimality achieved for certain fixed values of k, for various values of kT. Very similar results were achieved for the CCAT dataset

Effect of Sampling

Fig. 8 The effect of different sampling methods on the performance of Pegasos for the astro-ph dataset. The curves show the primal suboptimality achieved by uniform i.i.d. sampling, sampling from a fixed permutation, and sampling from a different permutation for every epoch



- One epoch is when the classifier has used all input examples once

Solving other problems

- A number of other problems can be solved using SGD just by choosing the appropriate loss function

Loss function	Subgradient
$\ell(z, y_i) = \max\{0, 1 - y_i z\}$ **Binary classification**	$\mathbf{v}_t = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i z < 1 \\ \mathbf{0} & \text{otherwise} \end{cases}$
$\ell(z, y_i) = \log(1 + e^{-y_i z})$ **Binary classification with log-loss**	$\mathbf{v}_t = -\frac{y_i}{1+e^{y_i z}} \mathbf{x}_i$
$\ell(z, y_i) = \max\{0, \lvert y_i - z \rvert - \epsilon\}$ **Regression**	$\mathbf{v}_t = \begin{cases} \mathbf{x}_i & \text{if } z - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - z > \epsilon \\ \mathbf{0} & \text{otherwise} \end{cases}$
$\ell(z, y_i) = \max_{y \in \mathcal{Y}} \delta(y, y_i) - z_{y_i} + z_y$ **Multi-class classification**	$\mathbf{v}_t = \phi(\mathbf{x}_i, \hat{y}) - \phi(\mathbf{x}_i, y_i)$ where $\hat{y} = \arg\max_y \delta(y, y_i) - z_{y_i} + z_y$
$\ell(z, y_i) = \log\left(1 + \sum_{r \neq y_i} e^{z_r - z_{y_i}}\right)$ **Multi-class classification**	$\mathbf{v}_t = \sum_r p_r \phi(\mathbf{x}_i, r) - \phi(\mathbf{x}_i, y_i)$ where $p_r = e^{z_r} / \sum_j e^{z_j}$

Using SGD in Scikit-Learn

- Available: http://scikit-learn.org/stable/modules/sgd.html
  - Need to specify the number of epochs/iterations
  - Sensitive to feature scaling (normalize and standardize your features via preprocessing)

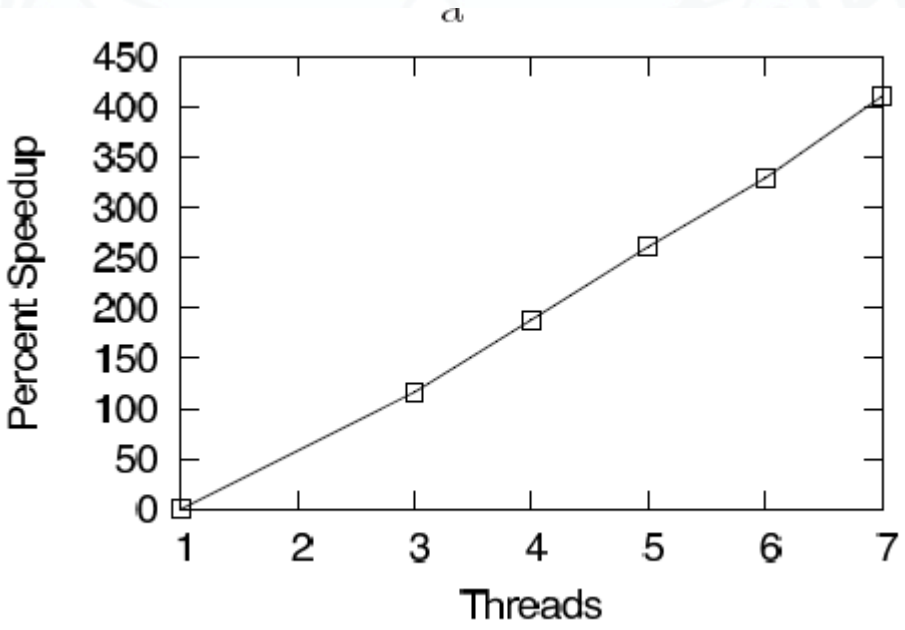
Reading

- http://ciml.info/ "Efficient Learning" and "Linear Models"

Large Scale Learning

- **PEGASOS Super good for linear SVMs when d<<n**
  - We discuss how the runtime of SVM optimization should **decrease** as the size of the training data increases. We present theoretical and empirical results demonstrating how a simple subgradient descent approach ndeed displays such behavior, at least for linear kernels.
    - SVM Optimization: Inverse Dependence on Training Set Size by Shalev-Shwartz and Srebro et al.
  - Faster convergence with SARAH: A Novel Method for Machine Learning Problems Using Stochastic Recursive Gradient
- **Parallelization**
  - Langford, John, Alexander Smola, and Martin Zinkevich. "Slow Learners Are Fast." *arXiv:0911.0491 [math, Stat]*, November 3, 2009. http://arxiv.org/abs/0911.0491.
- **Need better methods for when d>>n**
  - S. Shalev-Shwartz, A. Tewari Stochastic Methods for L1-regularized Loss Minimization [Jul.] J. Mach. Learn. Res., 12 (2011), pp. 1865–1892
- **Locally Linear Methods**
  - CAFÉ-Map: Context Aware Feature Mapping for mining high dimensional biomedical data by Minhas, Asif and Arif
  - Wang, Zhuang, Nemanja Djuric, Koby Crammer, and Slobodan Vucetic. "Trading Representability for Scalability: Adaptive Multi-Hyperplane Machine for Nonlinear Classification." In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 24–32. KDD '11. New York, NY, USA: ACM, 2011. doi:10.1145/2020408.2020420.
  - Ladicky, Lubor, and Philip Torr. "Locally Linear Support Vector Machines." In Proceedings of the 28th International Conference on Machine Learning (ICML-11), edited by Lise Getoor and Tobias Scheffer, 985–92. ICML '11. New York, NY, USA: ACM, 2011.

- Slow learners are fast



- Lee, Sangkyun, and Stephen J. Wright. "Stochastic Subgradient Estimation Training for Support Vector Machines." In *Mathematical Methodologies in Pattern Recognition and Machine Learning*, edited by Pedro Latorre Carmona, J. Salvador Sánchez, and Ana L. N. Fred, 67–82. Springer Proceedings in Mathematics & Statistics 30. Springer New York, 2013. http://link.springer.com/chapter/10.1007/978-1-4614-5076-4_5.

- Gives a faster kernelized algorithm

Table 3 Training CPU time (in seconds, h:hours) and test error rate (%) in parentheses

	Subgradient methods				Cutting-plane				Decomposition			
$s = 512$	ASSET_M		$\text{ASSET}^{\star}_M$		CPSP		CPNY		LASVM		SVM-Light	
ADULT	23	(15.1±.06)	24	(15.1±.06)	3,020	(15.2)	8.2h	(15.1)	1,011	(18.0)	857	(15.1)
MNIST	97	(4.0±.05)	101	(4.0±.04)	550	(2.7)	348	(4.1)	588	(1.4)	1323	(1.2)
CCAT	95	(8.2±.08)	99	(8.3±.06)	800	(5.2)	62	(8.3)	2,616	(4.7)	3423	(4.7)
IJCNN	87	(1.1±.02)	89	(1.1±.02)	727	(0.8)	320	(1.1)	288	(0.8)	1331	(0.7)
COVTYPE	697	(18.2±.06)	586	(18.2±.07)	1.8h	(17.7)	1,842	(18.2)	38.3h	(13.5)	52.7h	(13.8)
$s = 1024$	ASSET_M		$\text{ASSET}^{\star}_M$		CPSP		CPNY		LASVM		SVM-Light	
ADULT	78	(15.1±.05)	83	(15.1±.04)	3,399	(15.2)	7.5h	(15.2)	1,011	(18.0)	857	(15.1)
MNIST	275	(2.7±.03)	275	(2.7±.02)	1,273	(2.0)	515	(2.7)	588	(1.4)	1323	(1.2)
CCAT	265	(7.1±.05)	278	(7.1±.04)	2,950	(5.2)	123	(7.2)	2,616	(4.7)	3423	(4.7)
IJCNN	307	(0.8±.02)	297	(0.8±.01)	1,649	(0.8)	598	(0.8)	288	(0.8)	1,331	(0.7)
COVTYPE	2,259	(16.5±.04)	2064	(16.5±.06)	4.1h	(16.6)	3,598	(16.5)	38.3h	(13.5)	52.7h	(13.8)

Kernel approximation dimension is varied by setting $s = 512$ and $s = 1,024$ for ASSET_M, $\text{ASSET}^{\star}_M$, CPSP, and CPNY. Decomposition methods do not depend on s, so their results are the same in both tables

- Trading Representability for Scalability: Adaptive Multi-Hyperplane Machine for Nonlinear Classification
  - Super Simple!
  - Cannot handle implicit feature representations
  - BudgetedSVM: AMM
    - http://www.dabi.temple.edu/budgetedsvm/index.html

Algorithm	Training time	Prediction time	Model size
Pegasos	$O(N \cdot C \cdot S)$	$O(C \cdot S)$	$O(C \cdot D)$
AMM	$O(N \cdot S \cdot B)$	$O(S \cdot B)$	$O(D \cdot B)$
LLSVM	$O(N \cdot S \cdot B^2 + N \cdot S \cdot B)$	$O(S \cdot B^2 + S \cdot B)$	$O(D \cdot B + B^2)$
BSGD	$O(N \cdot (C + S) \cdot B)$	$O((C + S) \cdot B)$	$O((C + D) \cdot B)$
RBF-SVM	$O(I \cdot N \cdot C \cdot S)$	$O(N \cdot C \cdot S)$	$O(N \cdot C \cdot S)$

Datasets	Error rate (%)					Training time (seconds)[1]				
	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM
a9a	15.03±0.11	16.44±0.23	15.04±0.07	14.94	14.97	2	0.2	1	2	99
ijcnn	2.40±0.11	3.02±0.14	7.76±0.19	2.16	1.31	2	0.1	1	11	27
webspam	4.50±0.24	6.14±1.08	7.28±0.09	1.56	0.80	80	4	12	3,228	15,571
mnist_bin	0.53±0.05	0.54±0.03	2.03±0.04	NA	0.43[2]	3084	300	277	NA	2 days[2]
mnist_mc	3.20±0.16	3.36±0.20	8.41±0.11	NA	0.67[3]	13864	1200	1180	NA	8 days[3]
rcv1_bin	2.20±0.01	2.21±0.02	2.29±0.01	NA	NA	1100	80	25	NA	NA
url	1.34±0.21	2.87±1.49	1.50±0.39	NA	NA	400	24	100	NA	NA

Better Kernelized Methods

- **Approximation methods**
  - **Fourier based methods: Random features for large-scale kernel machines, On the Error of Random Fourier Features**
    - **Used by Amina for Learning with rejection**
    - **https://channel9.msdn.com/Events/Neural-Information-Processing-Systems-Conference/Neural-Information-Processing-Systems-Conference-NIPS-2016/Orthogonal-Random-Features**
    - **Orthogonal Random Features https://arxiv.org/abs/1610.09072 , Implementation: https://github.com/NICTA/revrand**
  - Nystrom
  - Xi-Squared estimation
  - Le, Quoc Viet, Tamas Sarlos, and Alexander Johannes Smola. "Fastfood: Approximate Kernel Expansions in Loglinear Time." *arXiv:1408.3060 [cs, Stat]*, August 13, 2014. http://arxiv.org/abs/1408.3060.
  - Lee, Sangkyun, and Stephen J. Wright. "Stochastic Subgradient Estimation Training for Support Vector Machines." In *Mathematical Methodologies in Pattern Recognition and Machine Learning*, edited by Pedro Latorre Carmona, J. Salvador Sánchez, and Ana L. N. Fred, 67–82. Springer Proceedings in Mathematics & Statistics 30. Springer New York, 2013. http://link.springer.com/chapter/10.1007/978-1-4614-5076-4_5.
- **Randomized algorithms**
  - **Random Kitchen Sinks**
  - **Scalable Kernel Methods via Doubly Stochastic Gradients (Matlab code available, Julia Implementation by Dr. Fayyaz Minhas)**
  - **Extreme Learning Machines**
  - **The Unreasonable Effectiveness of Random Orthogonal Embeddings**
  - How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets
    - **GPU based implementation**
  - Fast randomized kernel methods with statistical guarantees
  - Deep Semi-Random Features for Nonlinear Function Approximation
  - Steps toward deep kernel methods from infinite neural networks
  - A comparative study on large scale kernelized support vector machines
  - Stochastic Methods for l1-regularized Loss Minimization
  - Distributed Coordinate Descent Method for Learning with Big Data
  - The Tradeoffs of Large Scale Learning
  - Large-Scale Support Vector Machines: Algorithms and Theory

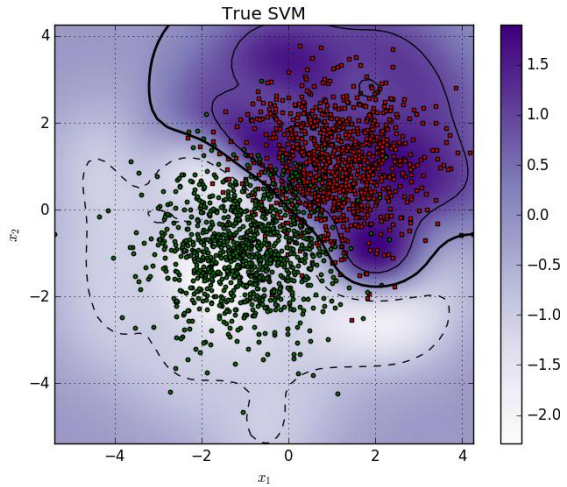
Deep learning papers

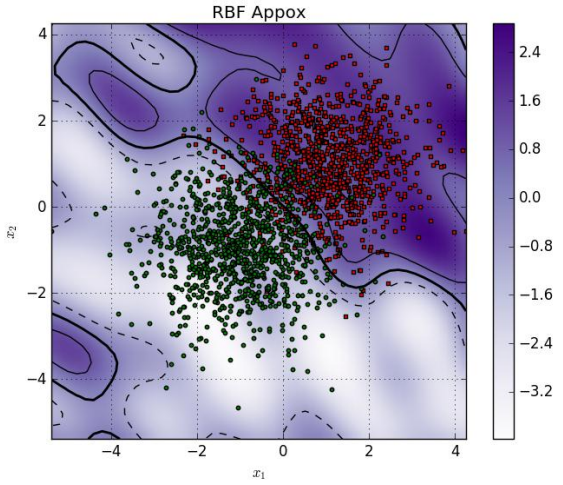
- Deep Semi-Random Features for Nonlinear Function Approximation
- http://nuit-blanche.blogspot.com/2017/01/understanding-deep-learning-requires.html
- Deep Nets Don't Learn via Memorization
- https://medium.com/@phelixlau/deep-nets-dont-learn-via-memorization-6fd692dea63e

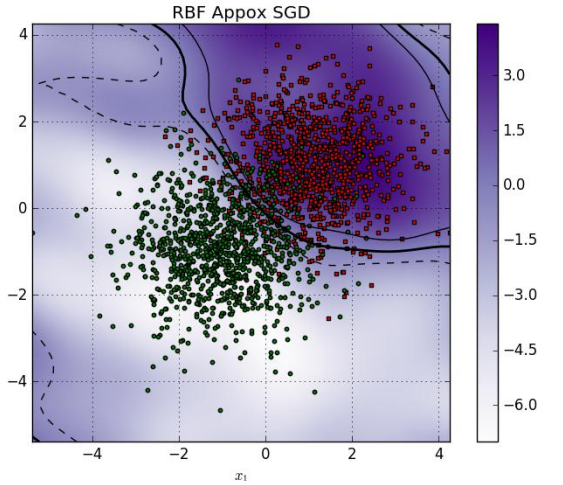
Structured Prediction

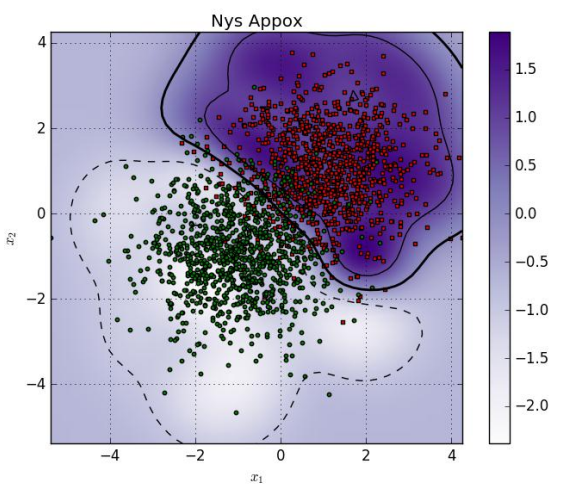
- http://ciml.info/

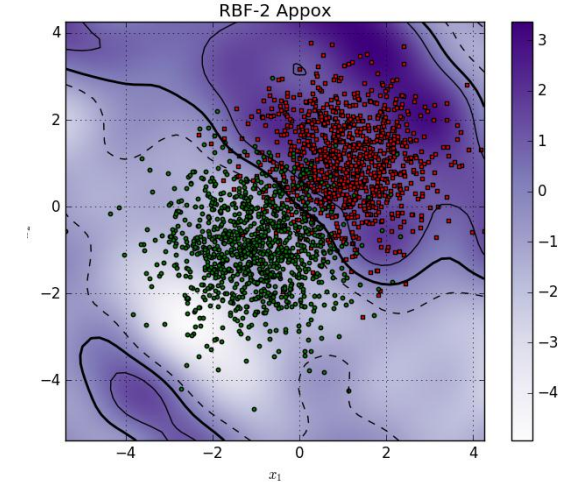
Kernel Approximations: Code in notes

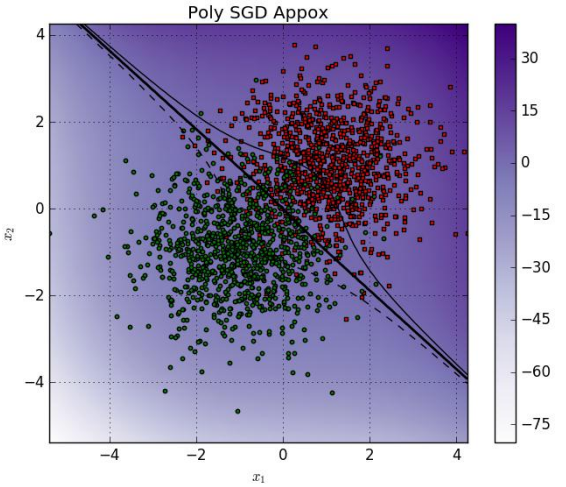












End of Lecture

I believe that learning has just started, because whatever we did before, it was some sort of a classical setting known to classical statistics as well. Now we come to the moment where we are trying to develop a new philosophy which goes beyond classical models.

- Vapnik

http://www.learningtheory.org/learning-has-just-started-an-interview-with-prof-vladimir-vapnik/