Large-Scale Support Vector Machines: Algorithms and Theory

Research Exam

Aditya Menon

UCSD

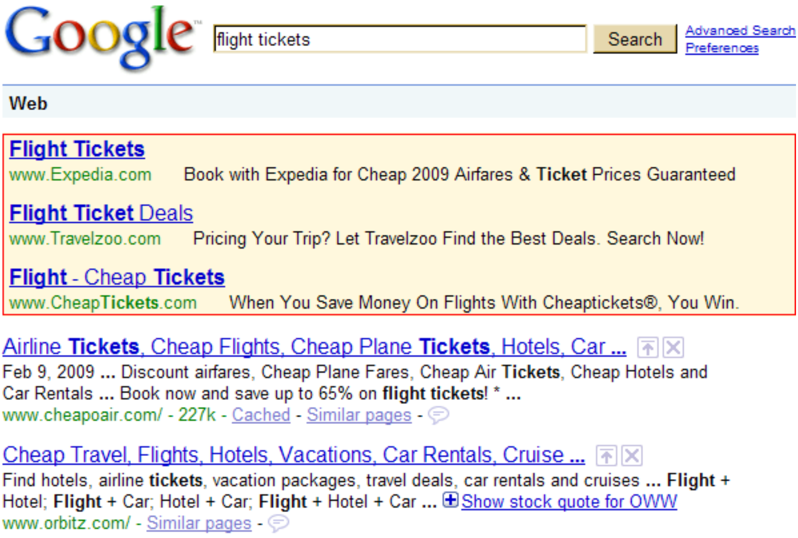
February 27, 2009

Outline

Motivation: the growth of data

- Dataset sizes have been growing at a rapid rate the past few years
- In supervised learning, this growth affects the size of training sets
- **Benefit**: More information, can make useful predictions
- **Concern**: Can our methods of analysis scale to such datasets?
  - A learning algorithm that scales superlinearly in the size of the training set will be infeasible
  - $\implies$ Need methods that scale at worst linearly in the number of examples

Example: ad click data



Defining large-scale learning

- While large-scale can simply mean a large number of examples, a more technical definition follows
- In a learning problem, we want to minimize generalization error subject to two constraints
  - There is some maximum number of examples we can pick
  - There is some maximum amount of time available
- Active constraint defines scale of problem
  - Number of examples $\implies$ small- or medium-scale
  - Time available $\implies$ large-scale

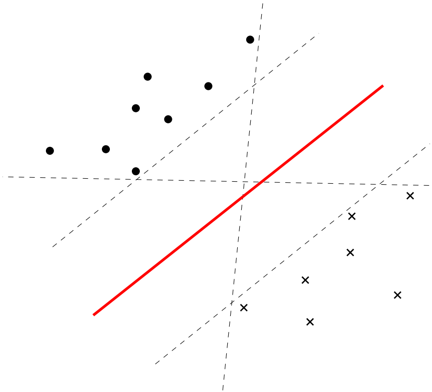
This talk

- We focus on methods for large-scale support vector machines (SVMs)
- One of the most popular approaches for binary classification tasks
  - Strong theoretical underpinnings
  - Good performance in practice
- Interested in the training algorithms and some of the theory behind them
  - What are some techniques for large-scale SVMs?
  - Why do they work? (specifically stochastic gradient descent)

Outline

Support vector machines

- A support vector machine (SVM) is a binary classifier that finds a maximum margin separating hyperplane
- Intuitively, we expect such a hyperplane to generalize the best



Primal problem for SVMs

- Suppose we have a training set $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^n$
- How do we find the maximum margin separating hyperplane?
- If we allow for misclassifications at the expense of some penalty, the SVM problem is:

Primal SVM problem

$$\text{minimize } \frac{\lambda}{2}||w||^2 + \frac{1}{n}\sum_{i=1}^{n}[1 - y_i(w \cdot x_i)]_+$$

where $[\cdot]_+$ denotes the hinge-loss:

$$[x]_+ = \max(0, x)$$

and w denotes the normal to the hyperplane

Interpreting the primal problem

- Can think of the primal problem as ℓ_2 regularized hinge-loss minimization
- Falls in the general class of functions $\hat{g}(w) = \ell_{\text{emp}}(w) + r(w)$
  - ℓ_{emp} measures loss on the training set
  - r is a regularization term
  - Both functions are convex, and hence the optimization is tractable

Dual formulation

- The dual SVM problem is the following:

Dual SVM problem

$$\text{maximize } \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$\text{subject to } 0 \leq \alpha_i \leq \frac{1}{\lambda n}$$

- According to the representer theorem, the optimal primal and dual solutions w^* and α^* satisfy

$$w^* = \sum_i \alpha_i^* y_i x_i$$

Primal vs dual formulation

- Historically, SVMs have been solved using the dual
- Reasons?
  - Naturally extend to kernels
  - Precedent set by "hard margin" case: simple dual optimization constraints
- The primal problem that we stated renders both points largely moot
  - Can also handle kernels
  - Unconstrained!
- So, no a-priori reason to eschew the primal form
  - Opens up some new techniques

Kernelized dual formulation

Standard dual problem

$$\text{maximize} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \quad \text{subject to } 0 \leq \alpha_i \leq \frac{1}{\lambda n}$$

$$\Downarrow$$

Kernelized dual problem

$$\text{maximize} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad \text{subject to } 0 \leq \alpha_i \leq \frac{1}{\lambda n}$$

- Can do the same for the primal form...

Kernelized primal formulation

Standard primal problem

$$\underset{w}{\text{minimize}} \; \frac{\lambda}{2}||w||^2 + \frac{1}{n}\sum_{i=1}^{n}[1 - y_i(w \cdot x_i)]_+$$

$$\Downarrow$$

Kernelized primal problem

$$\underset{f \in H}{\text{minimize}} \; \frac{\lambda}{2}||f||_H^2 + \frac{1}{n}\sum_{i=1}^{n}[1 - y_i f(x_i)]_+$$

where H is a reproducing kernel Hilbert space (RKHS) with kernel K.

- A Hilbert space is a complete inner product space, and an RKHS can be written $\{f : f(x) = \sum_i \beta_i K(x, x_i)\}$

Solving the SVM problem

- Suppose we want to solve the SVM problem with a quadratic programming (QP) solver
- The dual problem requires access to the matrix Q, defined as

$$Q_{ij} = y_i y_j K(x_i, x_j)$$

- Q's size is $n \times n$, which for even moderately large training sets is too expensive to store in memory
  - An off-the-shelf solver is insufficient
  - We need to design more specialized QP solvers

Classical SVM solvers: SVM$^{\text{light}}$ and SMO

- SVM$^{\text{light}}$: instead of solving the (large) QP problem, focus on a subset of variables (the working set)
  - Pick a subset of variables that are "likely" to change
  - Now solve this reduced problem using a standard QP solver
- SMO is a special case of SVM$^{\text{light}}$ where we optimize only two variables at once
  - Advantage: this optimization can be done analytically; does not require an external QP solver
  - But we need heuristics for choosing the variables to optimize
- Can implement caching of Q values to improve performance

Problems with classical SVM solvers

- **Problem**: These algorithms scale like n^2 in the worst case
  - Infeasible on large datasets
- **Question**: Can we design algorithms that solve the problem more efficiently?
  - We will see some algorithms that scale either linearly or are independent of the number of examples!
  - There is a catch, however...

Linear or kernel SVM?

- Most nascent large-scale solvers looked at the linear SVM case
- Why?
  - Simpler!
  - Oft-cited argument is *"In some applications, data appear in a rich dimensional feature space,the performances are similar with/without nonlinear mapping"*, with the canonical example being text classification [HCL+08]
  - Most text classification tasks are linearly separable [Joa98]
- Focus on linear SVMs alone is obviously fundamentally limiting: a major weakness of several of these techniques
  - Those that do consider kernels do so in passing; for many, viability for arbitrary kernels is an open question

Approximate SVM solution

- Newer solvers find approximate solutions to the SVM problem
- If the objective function is $f(w)$, then instead of finding $w^* = \text{argmin}\ f(w)$, they find $\tilde{w}$ such that

$$f(w^*) \leq f(\tilde{w}) \leq f(w^*) + \rho$$

- The constant ρ is the (user-controllable) optimization tolerance
  - Runtime is explicitly analyzed in terms of this
  - We call $\tilde{w}$ a ρ-optimal solution
- Approximate solutions are meaningful for the SVM problem because the optimization is a surrogate anyway
  - Important point that we discuss later

Outline

Stochastic gradient for SVMs

- Stochastic gradient descent (SGD) underlies at least three SVM training methods: SVM-SGD, NORMA, and Zhang's algorithm
- Idea is simply to apply SGD on the primal SVM problem
  - **Advantage**: Runtime is independent of number of examples
- Seems obvious, so why was it not tried earlier?
  - Historical favouring of the dual over the primal; dual SGA tried in the kernel adatron
  - Association of SGD with backpropagation in multi-layer perceptrons (non-convex problem)
  - Slow convergence rate

Stochastic gradient for SVMs

- Stochastic gradient descent (SGD) underlies at least three SVM training methods: SVM-SGD, NORMA, and Zhang's algorithm
- Idea is simply to apply SGD on the primal SVM problem
  - **Advantage**: Runtime is independent of number of examples
- Seems obvious, so why was it not tried earlier?
  - Historical favouring of the dual over the primal; dual SGA tried in the kernel adatron
  - Association of SGD with backpropagation in multi-layer perceptrons (non-convex problem)
  - Slow convergence rate ← will discuss this subsequently

Review: stochastic gradient descent (SGD)

- SGD uses a randomized gradient estimate to minimize a function $f(w)$
  - Instead of ∇f, use $\tilde{\nabla} f$ where $\mathbb{E}[\tilde{\nabla} f] = \nabla f$
- For empirical loss $\ell_{\text{emp}}(w) = \frac{1}{n} \sum_i \ell(x_i, y_i; w)$:

$$w_{t+1} \leftarrow w_t - \eta \nabla \ell(x_{i(t)}, y_{i(t)}; w_t)$$

where η is the learning rate and $i(t) \in \{1, \ldots, n\}$ uniformly at random

Pros	Cons
Fast: "instantaneous" gradient	Have to tune learning rate Slow convergence

SGD update for SVMs

- Recall the primal problem

$$\min \frac{\lambda}{2}||w||^2 + \frac{1}{n}\sum_{i=1}^{n}[1 - y_i(w \cdot x_i)]_+$$

- The SGD update is:

$$w_t \leftarrow (1 - \eta_t \lambda) w_{t-1} - \begin{cases} \eta_t y_{i(t)} x_{i(t)} & \text{if } y_{i(t)}(w_{t-1} \cdot x_{i(t)}) < 1 \\ 0 & \text{otherwise} \end{cases}$$

where $i(t) \in \{1, 2, \dots, n\}$ uniformly at random

SVM-SGD: the algorithm

for $t = 1 \ldots T$

Pick a random example (x_i, y_i)

$\eta_t \leftarrow \frac{1}{\lambda(t+t_0)}$

$w_t \leftarrow (1 - \eta_t \lambda) w_{t-1}$ `// weight decay`

`// not correctly classified with confidence`

if $y_i(w \cdot x_i) < 1$

$w_t \leftarrow w_t - \eta_t y_i x_i$

return w_T

- Note 1: t_0 is a heuristically chosen constant
- Note 2: Learning rate of $\eta_t = \frac{1}{\lambda(t+t_0)}$ initially mysterious...

Efficient sparse implementation

- We can represent the weight vector w by a tuple (v, s), where v is a vector and s a scalar
- Then the w update is

$$s \leftarrow (1 - \eta_t \lambda) s$$

$$v \leftarrow v - \eta_t y_i x_i$$

- Runtime per iteration is proportional to number of non-zero feature values

SVM-SGD results

- SVM-SGD can be orders of magnitude faster than methods like SVM^{light}, SVM^{perf}
- Results on `ccat` data (781,265 examples with 47,152 features):

Algorithm	**Training Time**	**Primal cost**	**Test Error**
SVM^{light}	23642 secs	0.2275	6.02%
SVM^{perf}	66 secs	0.2278	6.03%
SVM-SGD	1.4 secs	0.2275	6.01%

Table: Results as reported in [Bot07].

Handling kernels: NORMA

- To apply SGD with a kernel SVM, we notice that, similar to the representer theorem, our learned weight is always of the form

$$w = \sum \alpha_i y_i \Phi(x_i)$$

  - $\implies$ We can implicitly represent w by storing the (non-zero) α_i's
- Now the update for each example $(x_{i(t)}, y_{i(t)})$ becomes

$$(\forall 1 \leq j \leq n)\alpha_j \leftarrow (1 - \eta_t \lambda)\alpha_j$$

$$\alpha_{i(t)} \leftarrow \alpha_{i(t)} - \begin{cases} \eta_t & \text{if } y_{i(t)}(w_{t-1} \cdot x_{i(t)}) < 1 \\ 0 & \text{otherwise} \end{cases}$$

Drawbacks of SGD based methods?

- Both methods converge to ρ-optimal solution in $O(1/\rho^2)$ iterations
  - Slow: usually expect optimizers to converge in e.g. $O(\log 1/\rho)$ iterations
- Learning rate tuning
  - Common complaint about gradient methods!
- **Question**: Fundamental limit to what we can do with SGD?
- **Answer**: No, simple extensions make it powerful

Pegasos: extending SGD

- The Pegasos solver extends SGD in two ways:
  - Aggressively decrease the learning rate: use $\eta_t = \frac{1}{\lambda t}$, no parameter sweep required
  - Project the weight vector onto $\{x : ||x|| \leq 1/\sqrt{\lambda}\}$ (stochastic gradient projection)
- Can prove that these changes allow for convergence in $\tilde{O}\left(\frac{d}{\lambda\rho}\right)$ time
  - Inverse dependence on λ accounts for problem difficulty
- Can also work with kernels, similar to NORMA

Pegasos: the algorithm

for $t = 1 \ldots T$

Pick random $A_t \subseteq \mathcal{T}$ such that $|A_t| = k$

```
// not correctly classified with confidence
```

$\mathcal{M} := \{(x, y) \in A_t : y(w \cdot x) < 1\}$

$\nabla_t := \lambda w_t - \frac{1}{|\mathcal{M}|} \sum_{(x,y) \in \mathcal{M}} yx$

Update $w_{t+\frac{1}{2}} \leftarrow w_t - \frac{1}{\lambda t} \cdot \nabla_t$ `// SGD update`

Let $w_{t+1} \leftarrow \min\left(1, \frac{1}{\sqrt{\lambda}||w_{t+\frac{1}{2}}||}\right) w_{t+\frac{1}{2}}$ `// Projection step`

return w_{T+1}

- Note: k does not appear in runtime, and so effectively can be chosen to be 1!

Pegasos' convergence

- The projection step makes the learning rate $\propto \frac{1}{t}$ feasible
- The reason the projection makes sense is the following theorem:

Theorem

The optimal SVM solution w^* *satisfies* $||w^*|| \leq \frac{1}{\sqrt{\lambda}}$

Proof of theorem

- By the strong duality theorem, the values of the optimal primal and dual solutions are equal
- Rescaling dual problem so that $\alpha_i \in [0, 1]$, we get:

$$\frac{\lambda}{2}||w^*||^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i; w^*) = \frac{1}{n}\sum_{i=1}^{n} \alpha_i^* - \frac{1}{2\lambda n^2}\sum_{i,j} \alpha_i^* \alpha_j^* y_i y_j (x_i \cdot x_j).$$

- But by the representer theorem, $w^* = \sum \alpha_i^* y_i x_i$:

$$\frac{\lambda}{2}||w^*||^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i; w^*) = \frac{||\alpha^*||_1}{n} - \frac{\lambda}{2}||w^*||^2.$$

- Rearranging,

$$\lambda||w^*||^2 = \frac{||\alpha^*||_1 - \sum_{i=1}^{n} \ell(x_i, y_i; w^*)}{n} \leq 1$$

Pegasos results

Runtime comparison of Pegasos (in seconds):

Algorithm	Dataset		
	ccat	covertype	astro-ph
$\text{SVM}^{\text{light}}$	20,075	25,514	80
SVM^{perf}	77	85	5
Pegasos	2	6	2

Table: Results as reported in [SSSS07].

ccat: 804414×47236, 0.16% dense
covertype: 581012×54, 22% dense
astro-ph: 62369×99757, 0.08% dense

Stochastic gradient descent: verdict?

- Observed performance of various methods is good
  - In no small part because individual updates are fast
- But even with Pegasos, convergence rate is only $1/\rho$
  - Not competitive in terms of optimization
- So why is SGD useful for learning?
- **Answer**: Poor optimization does not necessarily mean poor generalization
  - If SGD can optimize "enough", then we can process more examples and get a good generalization
  - Will discuss this more later
- We now quickly look at a couple of recent SGD-based methods

Recent work: FOLOS

- FOLOS is a general solver of convex regularized risk minimization problems i.e. $\ell_{\text{emp}}(w) + r(w)$
- Idea is to do SGD, and an analytic minimization (c.f. projection):

$$w_{t+\frac{1}{2}} = w_t - \eta_t \tilde{\nabla} \ell_{\text{emp}}(w_t)$$

$$w_{t+1} = \operatorname*{argmin}_w \left(\frac{1}{2} ||w - w_{t+\frac{1}{2}}||^2 + \eta_{t+\frac{1}{2}} r(w) \right)$$

- Can be shown that the update is "forward looking", and implicitly imposes the correct regularization term:

$$w_{t+1} = w_t - \eta_t \tilde{\nabla} \ell_{\text{emp}}(w_t) - \eta_{t+\frac{1}{2}} \boxed{\nabla r(w_{t+1})}$$

- Similar update to Pegasos for ℓ_2 regularization, discovers sparsity for ℓ_1 regularization

Recent work: SGD-QN

- SGD-QN combines stochastic gradient descent and quasi-Newton methods
- Instead of using the inverse Hessian H^{-1}, use a diagonal scaling matrix D to approximate it

$$w_{t+1} \leftarrow w_t - \eta_t D \cdot \tilde{\nabla}(\ell_{\text{emp}} + r)(w_t)$$

- No theoretical bound or formal experiments
  - Only briefly described as part of an ICML workshop (where it was the winning method!)
  - Unclear whether projection can be replaced (or augmented) with diagonal scaling

Outline

Dual coordinate descent

- We can use coordinate descent to solve the optimization problem in the dual (DCD)
  - Algorithm underlying LibLinear
- Idea is to simply solve the problem using a univariate minimization
  - Pick some α_i, and hold $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots$ constant
  - Find the optimal value of α_i given the other values
- Fortunately, it turns out that the second step is easy to do for SVMs

Dual coordinate descent

- If we let $f(\alpha)$ be the dual objective function, then we want to find

$$\min f(\alpha + de_i) \text{ subject to } 0 \le \alpha_i + d \le C$$

where $C = 1/n\lambda$. This can be solved with the following:

Solution of the minimization

Let $\nabla_i := (\nabla f(\alpha))_i$. Then, the solution to the univariate minimization is either α_i or

$$\alpha_i \leftarrow \min(\max(\alpha_i - \nabla_i/||x_i||^2, 0), C)$$

Further, by the representer theorem, if we explicitly store w,

$$\nabla_i = y_i(w \cdot x_i) - 1$$

DCD algorithm

while α is not optimal

Pick an index $i \in \{1, \ldots, n\}$ `// potentially stochastic`

$\alpha_i^{\text{old}} \leftarrow \alpha_i$

$\nabla_i \leftarrow y_i(w \cdot x_i) - 1$

$$\nabla_P \leftarrow \begin{cases} \min(\nabla_i, 0) & \text{if } \alpha_i = 0 \\ \max(\nabla_i, 0) & \text{if } \alpha_i = C \\ \nabla_i & \text{otherwise} \end{cases}$$

if $\nabla_P \neq 0$ `// check for non-trivial minimizer`

$\alpha_i \leftarrow \min(\max(\alpha_i - \nabla_i/||x_i||^2, 0), C)$

$w \leftarrow w + (\alpha_i - \alpha_i^{\text{old}})y_i x_i$

Analyzing DCD

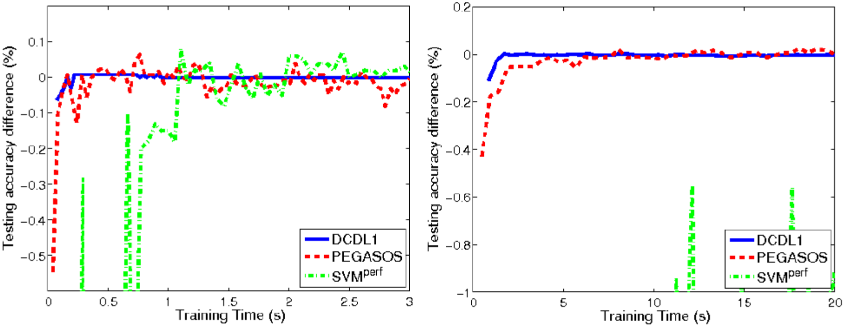
- We can interpret the update solely in terms of w

$$w_t = w_{t-1} - (\alpha - \alpha^{\mathsf{old}}) y_i x_i$$

- If i is chosen randomly, we can think of the algorithm as a form of stochastic gradient descent!
  - Learning rate is chosen via analytic minimization; "optimal" in some sense
  - Superior in general to updates for Pegasos? SGD-QN?
- Convergence in batch case in $O(\log 1/\rho)$ passes over training set
  - Stochastic case is not clear

DCD results

Paper's results indicate it is faster than Pegasos; but some issues about choice of C



(a) Results on `astro-ph` (b) Results on `rcv1`

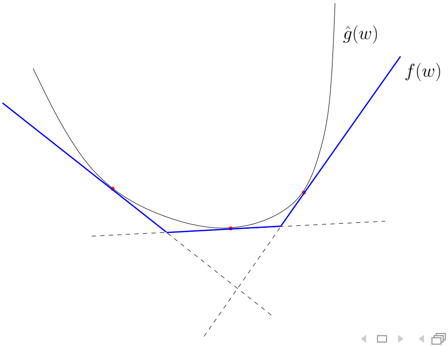
Figure: Results from [HCL+08].

`astroph`: 622369 $\times$ 99757, 0.08% dense

`rcv`: 677399 $\times$ 47236, 0.1% dense

Bundle method

- General optimization technique for convex functions: bundle methods
  - Idea is to lower bound a function by an envelope of hyperplanes
  - Regularize solution for stability
- Can apply this idea to SVMs with BMRM (bundle method for risk minimization)
  - Aside: misnomer? Bundle method vs cutting plane



BMRM update

- Suppose $\hat{g}(w) = \ell_{\text{emp}}(w) + r(w)$
- $f(w) = b_t + \nabla^{(s)} \ell_{\text{emp}}(w_t) \cdot w$ defines a hyperplane tangential to $\hat{g}(w)$ at $w = w_t$, where $\nabla^{(s)}$ denotes a subgradient
- Update the offset b_t with

$$b_{t+1} = \ell_{\text{emp}}(w_t) - \nabla^{(s)} \ell_{\text{emp}}(w_t) \cdot w_t$$

- The iterates of w are simply taken to be the minimizers of the current approximation:

$$w_{t+1} = \operatorname*{argmin}_w \left\{ r(w) + \max_{t' \leq t+1} \left[b_{t'} + (\nabla^{(s)} \ell_{\text{emp}}(w_{t'})) \cdot w \right] \right\}$$

BMRM update

- Suppose $\hat{g}(w) = \ell_{\text{emp}}(w) + r(w)$
- $f(w) = b_t + \nabla^{(s)} \ell_{\text{emp}}(w_t) \cdot w$ defines a hyperplane tangential to $\hat{g}(w)$ at $w = w_t$, where $\nabla^{(s)}$ denotes a subgradient
- Update the offset b_t with

$$b_{t+1} = \ell_{\text{emp}}(w_t) - \nabla^{(s)} \ell_{\text{emp}}(w_t) \cdot w_t$$

- The iterates of w are simply taken to be the minimizers of the current approximation:

$$w_{t+1} = \operatorname*{argmin}_w \left\{ r(w) + \boxed{\max_{t' \leq t+1} \left[b_{t'} + (\nabla^{(s)} \ell_{\text{emp}}(w_{t'})) \cdot w \right]} \right\}$$

  - Upper envelope

BMRM optimization

- Fortunately, the update admits a simple dual formulation

Dual bundle problem

The optimization problem for the bundle update is

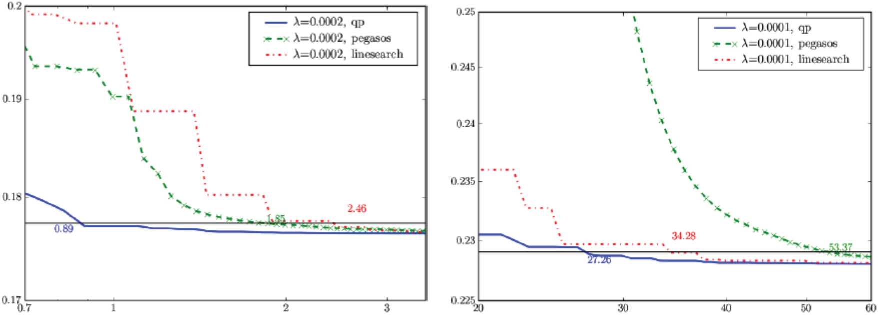
$$\max_{\alpha} -\frac{1}{2\lambda}\alpha^T Q\alpha + \alpha \cdot b \text{ such that } ||\alpha||_1 = 1, \alpha_i \geq 0,$$

where $Q_{ij} = \nabla^{(s)}\ell_{\text{emp}}(w_i) \cdot \nabla^{(s)}\ell_{\text{emp}}(w_j)$.

- This is a problem whose size at iteration t is $t \times t$
  - Worst case, $t = O(1/\rho)$; and for smooth ℓ_{emp}, $t = O(\log 1/\rho)$
  - Conjecture that an average of piecewise linear functions (e.g. SVMs) also has roughly $t = O(\log 1/\rho)$!
- Can be solved with a QP solver, or with a line search

BMRM results

Results only presented for reducing objective value; not as informative as test error



(a) Results on `astro-ph` (b) Results on `ccat`

Figure: Objective value vs time results from [SVL07].

BMRM and SVM$^{\text{perf}}$

- Do the results comparing BMRM and Pegasos contradict those in the Pegasos paper?
  - Authors claim SVM$^{\text{perf}}$ can be seen as a special case of BMRM
- So is SVM$^{\text{perf}}$ superior to Pegasos?
  - Should compare their generalization ability rather than optimization
  - But `ccat` results are surprising; Pegasos greatly outperformed till $\sim$ 50 seconds? Hard to imagine Pegasos has comparable generalization!
  - Disappointingly, authors do not discuss this issue at all
- Issue is potentially moot...results in OCAS contradict the ones here!

Extension: OCAS

- OCAS is an extension of the BMRM approach
- Recall that we try to minimize the regularized risk $\hat{g}(w)$ with a lower envelope $f(w)$
- The BMRM iterates are guaranteed to satisfy

$$f(w_{t+1}) < f(w_t) \text{ but } \textbf{not } \hat{g}(w_{t+1}) < \hat{g}(w_t)$$

  - Some iterates are undesirable
- OCAS ensures monotonicity of $\hat{g}(w_t)$ by a line search
  - Keeps track of best solution, and combines this with current iterate
  - Aside: truer bundle method, but calls itself cutting plane!
- Discusses potential for parallelization

Outline

Why is SGD successful?

- We've seen SVM training algorithms based on SGD
- Advantage: faster processing of each example
- Disadvantage: slower convergence in general
- **Question**: Doesn't the slow convergence rate seriously hamper its viability?

Why is SGD successful?

- We've seen SVM training algorithms based on SGD
- Advantage: faster processing of each example
- Disadvantage: slower convergence in general
- **Question**: Doesn't the slow convergence rate seriously hamper its viability?
- **(Surprising) Answer**: Not if we look at the optimization process more closely

Learning and optimization

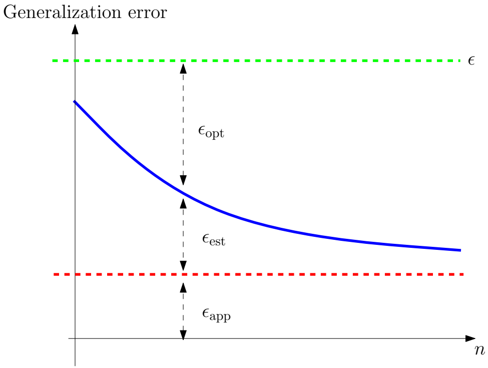
- We have studied SVMs through their optimization problem
  - In particular, we looked at runtime as a function of ρ
- But recall that optimization is only a surrogate for generalization
- We hope that minimizing the training error will minimize the generalization error
- **Question**: Is it necessary to perform strenuous optimization?

The three components of error

- [BB07] explicitly considers the role of optimization error
- Suppose that we obtain a weight $\tilde{w}$ using an optimization algorithm run to some finite tolerance
- They decompose the generalization error $\epsilon(\tilde{w}) = \mathbb{E}[\ell(x, y; \tilde{w})]$ into three components
  - Approximation error ϵ_{app}: minimum error due to hypothesis class
  - Estimation error ϵ_{est}: minimum error due to training set
  - Optimization error ϵ_{opt}: minimum error due to optimization

Minimizing generalization error

- If $n \to \infty$, then $\epsilon_{\text{est}} \to 0$
- **Implication**: For a fixed generalization error ϵ, as n increases we can increase the optimization tolerance ρ



The role of estimation error

- We look at the behaviour of estimation error as n gets large
- This lets us connect the behaviour of n and ρ to the generalization error ϵ

Estimation error bound

Let ϵ^* denote the minimum possible generalization error. Then,

$$0 \leq \epsilon - \epsilon^* \leq c \cdot \left(\epsilon_{\text{app}} + \frac{d}{n} \log \frac{n}{d} + \rho \right)$$

- **Intuition**: Estimation error behaves like $\log n / n$

The role of estimation error

- Now let's bound the individual terms by $\mathcal{E}$, the excess error w.r.t. the approximation error:

$$\rho = \Theta(\mathcal{E})$$

$$n = \Theta(d \log(1/\mathcal{E})/\mathcal{E})$$

- That implies

$$\epsilon - \epsilon^* \leq c \cdot (\epsilon_{\mathsf{app}} + \mathcal{E})$$

- So, we have a way to compare convergence to the generalization optimum

Estimation error and SGD

- How quickly do GD and SGD get a bound of $c \cdot (\epsilon_{\text{app}} + \mathcal{E})$?

Algorithm	Optimization time	Generalization time
GD	$O\left(nd \log \frac{1}{\rho}\right)$	$O\left(\frac{d^2}{\mathcal{E}} \log^2 \frac{1}{\mathcal{E}}\right)$
SGD	$O\left(\frac{d}{\rho}\right)$	$O\left(\frac{d}{\mathcal{E}}\right)$
2GD	$O\left((d^2 + nd) \log \log \frac{1}{\rho}\right)$	$O(\frac{d^2}{\mathcal{E}} \log \frac{1}{\mathcal{E}} \log \log \frac{1}{\mathcal{E}})$
2SGD	$O\left(\frac{d^2}{\rho}\right)$	$O\left(\frac{d^2}{\mathcal{E}}\right)$

- **Conclusion**: SGD generalizes asymptotically faster than GD!
- **Key point**: SGD's runtime does not depend on the number of examples

Applying to SVMs

- What implications does this have for SVMs?
  - Note: minimizing a regularized loss term
- SGD-based solvers (e.g. Pegasos) should be able to leverage decreased estimation error
  - But can we more accurately characterize this?

Applying to SVMs

- What implications does this have for SVMs?
  - Note: minimizing a regularized loss term
- SGD-based solvers (e.g. Pegasos) should be able to leverage decreased estimation error
  - But can we more accurately characterize this?
- **(Very surprising) Implication**: The training time should decrease as the number of examples increases

SVM generalization

- Given more examples, SVM training time should ideally not increase if we want the same generalization error
  - Suppose in time t, we achieve 5% generalization error with 10,000 examples
  - With 100,000 examples, we can sample to get the same error
- But [SSS08] goes a step further
  - Argues that as number of examples increases, runtime should decrease as a result of decreased estimation error
  - That is, it takes us $< t$ time to get 5% generalization error with 100,000 examples

SVM generalization bound

- Foundation of the analysis is the following theorem

Theorem

Let w be the learned weight vector when SVM optimization is done up to tolerance ρ, and let w_0 be any other weight vector. Then,

$$g(w) \leq g(w_0) + 2\rho + \frac{\lambda}{2}||w_0||^2 + \tilde{O}(1/\lambda n),$$

where $g(w)$ is the generalization error with weight vector w,

$$g(w) = \mathbb{E}_{(x,y)\sim P(\mathcal{X},\mathcal{Y})}[\ell(x, y; w)]$$

Proof of generalization bound

- Let $f(w) := g(w) + \frac{\lambda}{2}||w||^2$ denote the regularized generalization error
- Now decompose $g(w)$ as

$$\begin{aligned} g(w) &= f(w) - \frac{\lambda}{2}||w||^2 \\ &= f(w) - \frac{\lambda}{2}||w||^2 - \left(f(w_0) - g(w_0) - \frac{\lambda}{2}||w_0||^2 \right) \\ &= g(w_0) + (f(w) - f(w_0)) + \frac{\lambda}{2}||w_0||^2 - \frac{\lambda}{2}||w||^2 \\ &= g(w_0) + (f(w) - f(w^*)) + (f(w^*) - f(w_0)) \\ &+ \frac{\lambda}{2}||w_0||^2 - \frac{\lambda}{2}||w||^2. \end{aligned}$$

where $w^* = \text{argmin } f(w)$.

Proof of generalization bound contd.

- Recall that $\hat{g}(w)$ is the SVM training error
- Second term is a difference of expected losses, can be bounded using the corresponding empirical losses:

$$f(w) - f(w^*) \leq 2\max(0, \hat{g}(w) - \hat{g}(w^*)) + O\left(\frac{\log 1/\delta}{\lambda n}\right)$$
$$= 2\rho + O\left(\frac{\log 1/\delta}{\lambda n}\right) \text{ by definition of } w$$

- $f(w^*) - f(w_0) \leq 0$ by the optimality of w^*
- Combining these facts,

$$g(w) \leq g(w_0) + 2\rho + \frac{\lambda}{2}||w_0||^2 + \tilde{O}(1/\lambda n)$$

Applying the bound

- We can use this bound to find $T(n, \mathcal{E})$, the time needed for training with n examples to get excess error $\mathcal{E}$
- Rewrite ρ in terms of T, λ: e.g. for Pegasos,

$$g(w) \leq g(w_0) + \tilde{O}(d/\lambda T) + \frac{\lambda}{2}||w_0||^2 + \tilde{O}(1/\lambda n)$$

- Choosing λ to minimize this,

$$\begin{aligned} g(w) &\leq g(w_0) + \tilde{O}(||w_0||\sqrt{d/T}) + O(||w_0||/\sqrt{n}) \\ &= g(w_0) + \mathcal{E} \end{aligned}$$

- Now express T as a function of n and $\mathcal{E}$...

Pegasos and SVM$^{\text{perf}}$

- Easy to get runtime bounds for Pegasos and SVM$^{\text{perf}}$
- Pegasos:

$$T(n, \mathcal{E}) = \tilde{O}\left(\frac{d}{\left(\mathcal{E}/||w_0|| - O(1/\sqrt{n})\right)^2}\right)$$

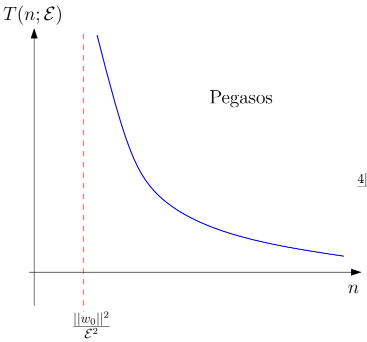
- SVM$^{\text{perf}}$:

$$T(n, \mathcal{E}) = O\left(\frac{{\color{red}n}d}{\left(\mathcal{E}/||w_0|| - O(1/\sqrt{n})\right)^2}\right)$$

- Implications?

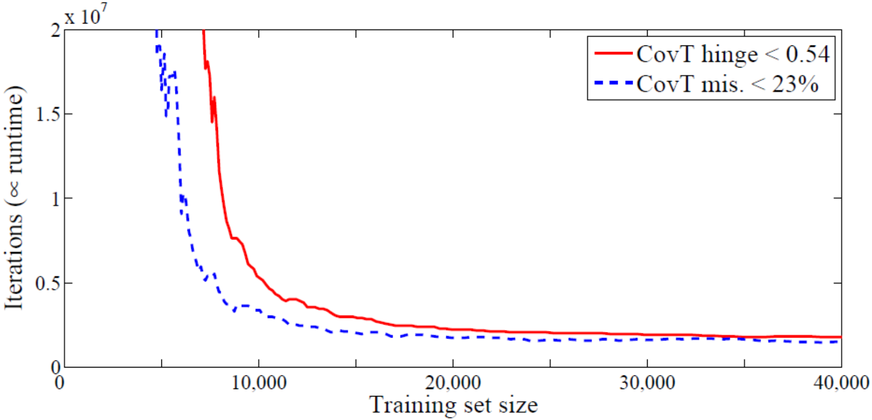
Pegasos and SVM$^{\text{perf}}$

- Pegasos' runtime monotonically decreases as a function of n!
- SVM$^{\text{perf}}$ has a turning point, but after that the runtime increases
  - Turning point where decrease in estimation is offset by increase in iteration cost



Verifying Pegasos' runtime

Results on `covertype`:



Outline

Summary

- We have seen several solvers for SVMs targetting large training sets
  - Primal methods based on SGD
  - Dual solvers based on optimization "tricks"
- Saw why SGD is poor at optimization but good at generalization
  - Runtime agnostic about number of examples
  - Still manages to leverage decreased estimation error
- Runtime for Pegasos decreases with increase in training set size
  - Fundamental limit to dual QP methods?
  - SGD for dual c.f. DCD?

Comparison of methods?

- Experimental comparisons by no means comprehensive
  - Pegasos' results directly contradict those in BMRM and OCAS!
  - Insufficient detail on parameter choices
- Motivation for ICML workshop
  - SGD-QN performed well, but so did some batch algorithms
  - Optimized interior point method did extremely well!
- Role of loading time
  - If method stores training set in memory, loading time is usually the bottleneck!
  - Online algorithms mix parsing and learning (e.g. Vowpal Wabbit)

Comment on plausibility

- **Question**: Is supervised learning realistic when the training set is very large?
  - In some domains like bioinformatics, labelling examples is expensive; impossible to completely label a large training set
  - A more realistic setting is semi-supervised learning
- **Answer**: Not always, but there are domains where large training sets are completely labelled
  - Labelling may be a natural byproduct of user actions e.g. Google ad clicks
  - Large user-bases can be leveraged/"tricked" into doing the labelling e.g. reCAPTCHA
- Nonetheless, (large-scale) semi-supervised SVMs is an important future direction

Other research directions

- Other approaches to large-scale SVMs (and learning)
  - Parallelization e.g. Cascade SVM, OCAS
  - Training set reduction, e.g. active learning, clustering
- Stochastic gradient for kernels?
  - NORMA was more interested in moving target setting; no comparison to other methods
  - Caching? Storing truncated kernel in memory?
- SVMs when data does not fit in memory
  - Completely precludes batch algorithms; incremental SVMs?
- Multi-class SVMs
  - In domains where we might expect large-scale data to arise naturally, classification is usually more complex than binary
  - Need efficient multi-class SVMs
  - Some nascent work e.g. LaRank

Questions?

Outline

[BB07] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20, 2007.

[Bot07] Léon Bottou. SVM-SGD. `http://leon.bottou.org/projects/sgd`, 2007.

[HCL+08] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of ECML-98: 10th European Conference on Machine Learning*, pages 137–142. Springer Verlag, 1998.

[SSS08] Shai Shalev-Shwartz and Nathan Srebro. SVM optimization: inverse dependence on training set size. In *ICML '08: Proceedings of the 25th International Conference on Machine learning*, pages 928–935, New York, NY, USA, 2008. ACM.

[SSSS07] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *ICML '07: Proceedings of the 24th International Conference on Machine learning*, pages 807–814, New York, NY, USA, 2007. ACM.

[SVL07] A. J. Smola, S. V. N. Vishwanathan, and Q. V. Le. Bundle methods for machine learning. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, Cambridge MA, 2007. MIT Press.