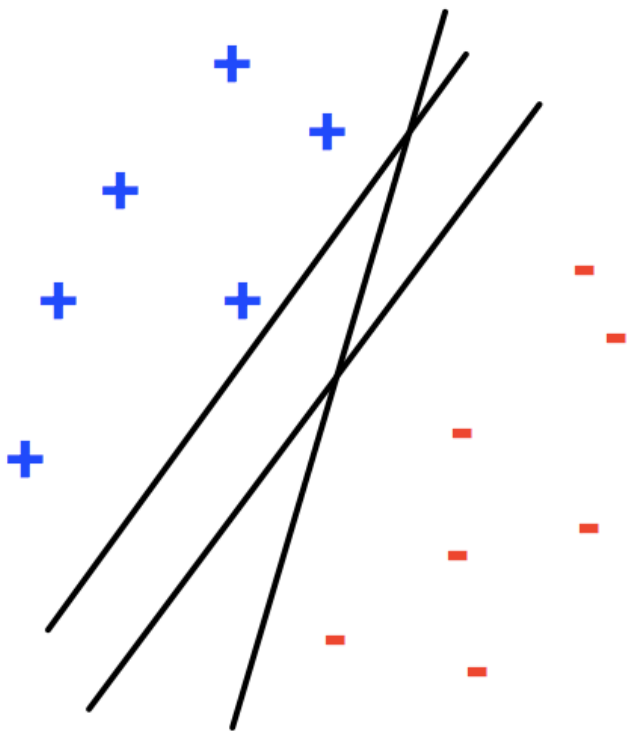
Algorithms for Support Vector Machines

http://bicmr.pku.edu.cn/~wenzw/bigdata2016.html

Acknowledgement: this slides is based on Prof. Jure Leskovec's and Prof. Stephen Wright's lecture notes

Support Vector Machines

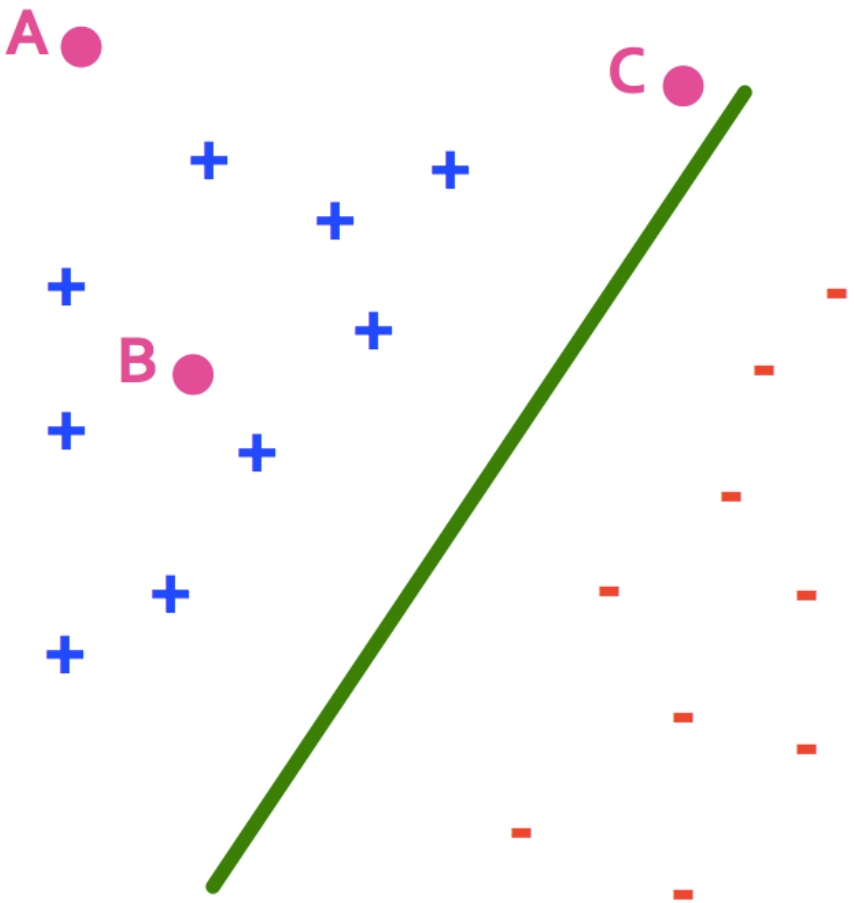
- Want to separate "+" from "-" using a line



Data:

- Training examples:
  - $(x_1, y_1) \cdots (x_n, y_n)$
- Each example i:
  - $x_i = (x_i^{(1)}, \cdots, x_i^{(d)})$
    - $x_i^{(j)}$ is real valued
  - $y_i \in \{-1, +1\}$
- Inner product:
  $w \cdot x = \sum_{j=1}^{d} w^{(j)} \cdot x^{(j)}$

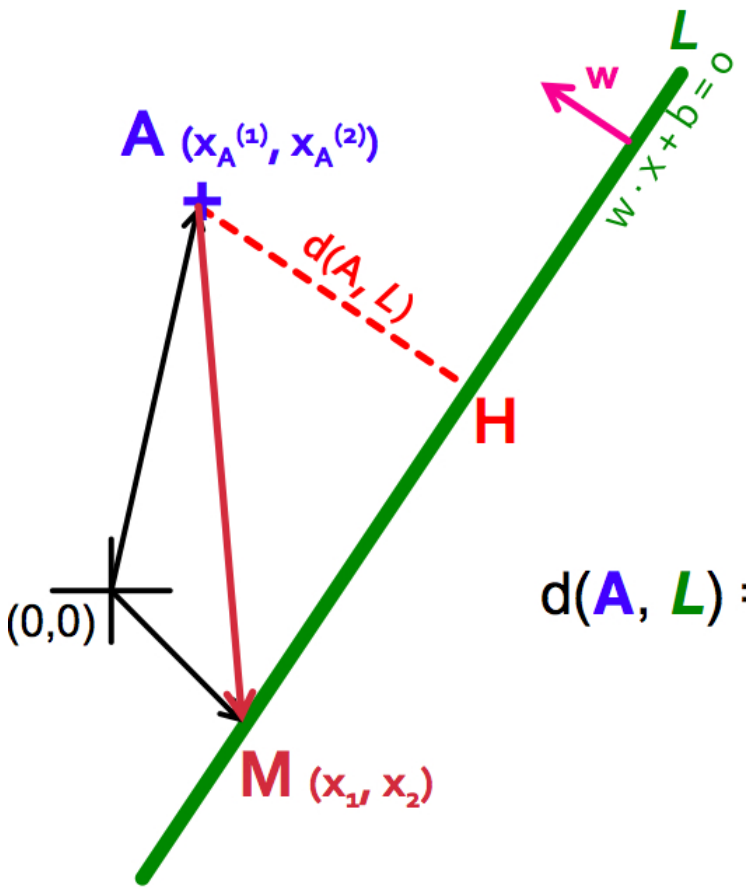
Large Margin



- Distance from the separating hyperplane corresponds to the "confidence" of prediction
- Example:
  - We are more sure about the class of A and B than of C

Margin: Distance of closest example from the decision line/hyperplane

What is the margin?

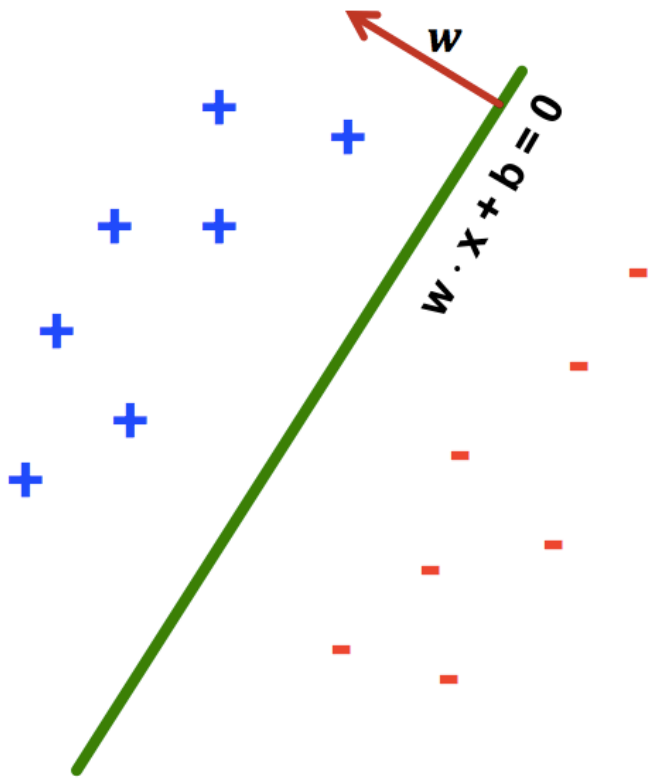


- Let $f(x) = w \cdot x$:
  - Line L:
    $w \cdot x + b = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + b = 0$
  - $w = (w^{(1)}, w^{(2)})$
  - Point $A = (x_A^{(1)}, x_A^{(2)})$
  - Point M on a line $= (x_M^{(1)}, x_M^{(2)})$

$$\begin{aligned} d(A, L) &= |f(x_A) - f(x_M)| \\ &= |(x_A - x_M) \cdot w| \\ &= |x_A \cdot w - x_M \cdot w| \\ &= |w \cdot x_A + b| \end{aligned}$$

Remember $x_M^{(1)}w^{(1)} + x_M^{(2)}w^{(2)} = -b$
since M belongs to line L

Largest Margin



- Prediction $= sign(w \cdot x + b)$
- "Confidence" $= (w \cdot x + b)y$
- For i-th datapoint:
  $\gamma_i = (w \cdot x_i + b)y_i$
- Want to solve:
  $\max_w \min_i \gamma_i$
- Can rewrite as

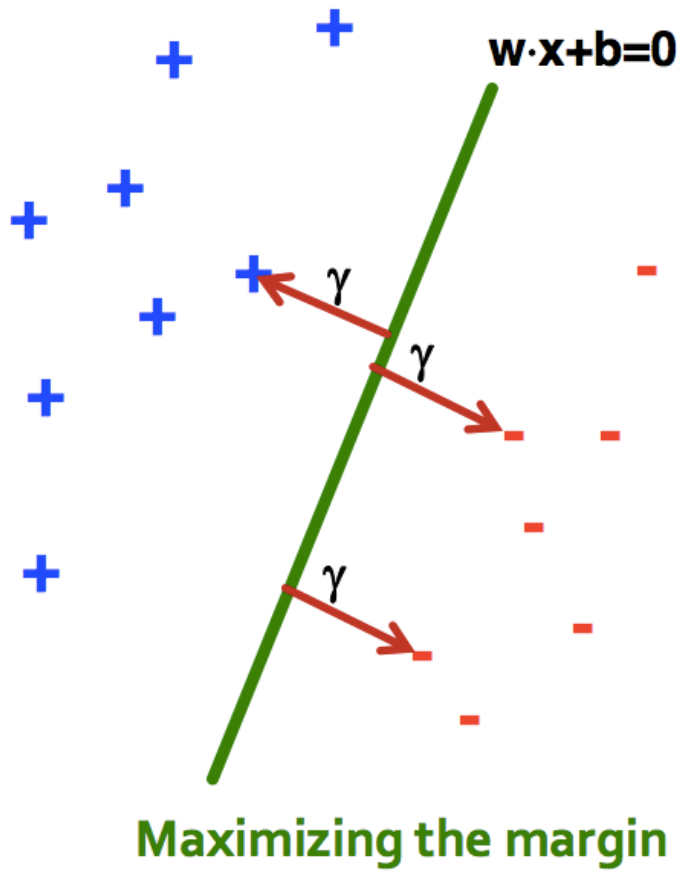
$$\max_{w,\gamma} \gamma$$
$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq \gamma, \forall i$$

Support Vector Machine

- Maximize the margin:
  - Good according to intuition, theory (VC dimension) & practice

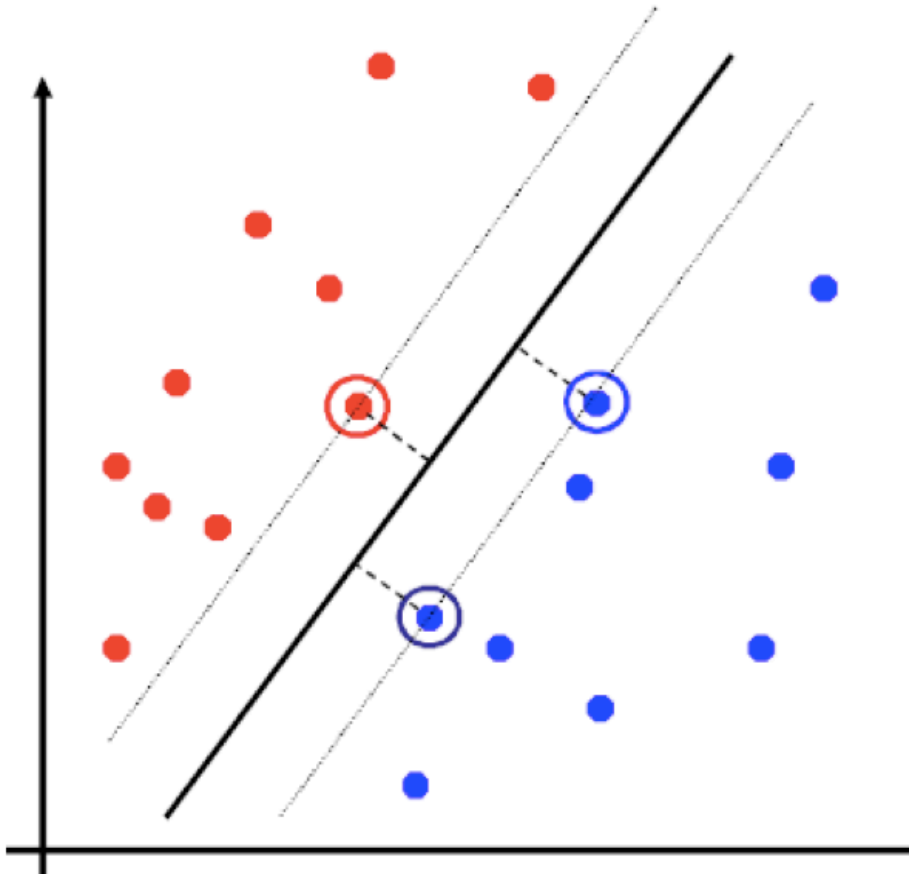
$$\max_{w,\gamma} \gamma$$
$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq \gamma, \forall i$$

  - γ is margin $\cdots$ distance from the separating hyperplane



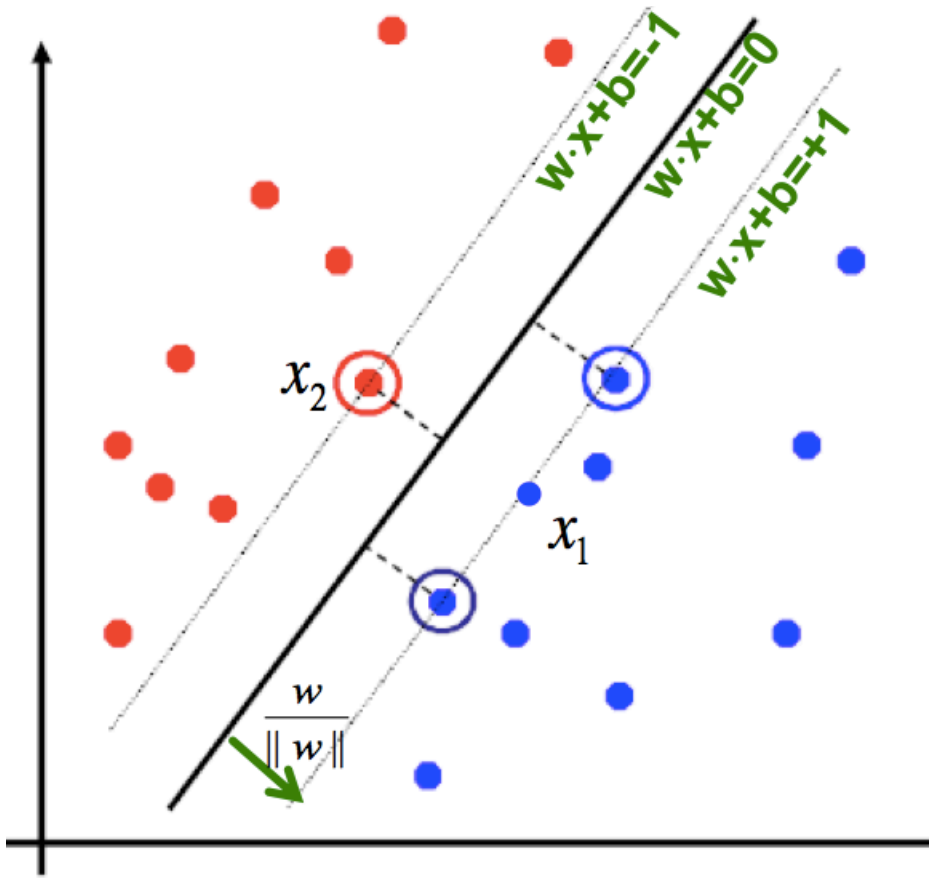
Support Vector Machines

- Separating hyperplane is defined by the support vectors
  - Points on +/- planes from the solution
  - If you knew these points, you could ignore the rest
  - If no degeneracies, d+1 support vectors (for d dimensional data)



Canonical Hyperplane: Problem

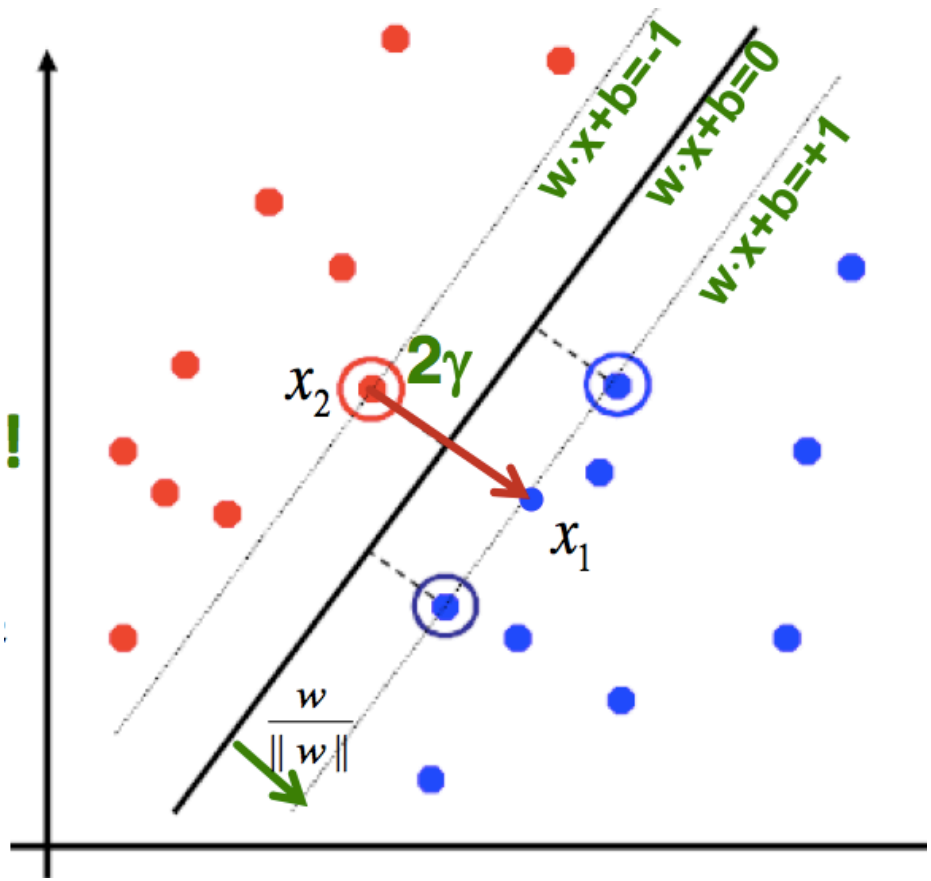
- Problem:
  - There exists x_M with $w \cdot x_M + b = 0$. Hence, $w \cdot x + b = w \cdot (x - x_M)$
  - Let $w \cdot (x - x_M)y = \gamma$, then $2w \cdot (x - x_M)y = 2\gamma$
    - Scaling w increses margin!
- Solution:
  - Work with normalized w: $\gamma = (\frac{w}{\|w\|} \cdot x + b)y$
  - Let's also require support vectors x_j to be on the plane defined by: $w \cdot x_j + b = \pm 1$



$$\|w\| = \sqrt{\sum_{j=1}^{d} (w^{(j)})^2}$$

Canonical Hyperplane: Solution

- Want to maximize margin γ!
- What is the relation between x_1 and x_2?
  - $x_1 = x_2 + 2\gamma \frac{w}{\|w\|}$
  - We also know:
    - $w \cdot x_1 + b = +1$
    - $w \cdot x_2 + b = -1$
- So:
  - $w \cdot x_1 + b = +1$
  - $w(x_2 + 2\gamma \frac{w}{\|w\|}) + b = +1$
  - $\underbrace{w \cdot x_2 + b}_{-1} + 2\gamma \frac{w \cdot w}{\|w\|} = +1$



$$\Rightarrow \gamma = \frac{\|w\|}{w \cdot w} = \frac{1}{\|w\|}$$

Note: $w \cdot w = \|w\|^2$

Maximizing the Margin

- We started with

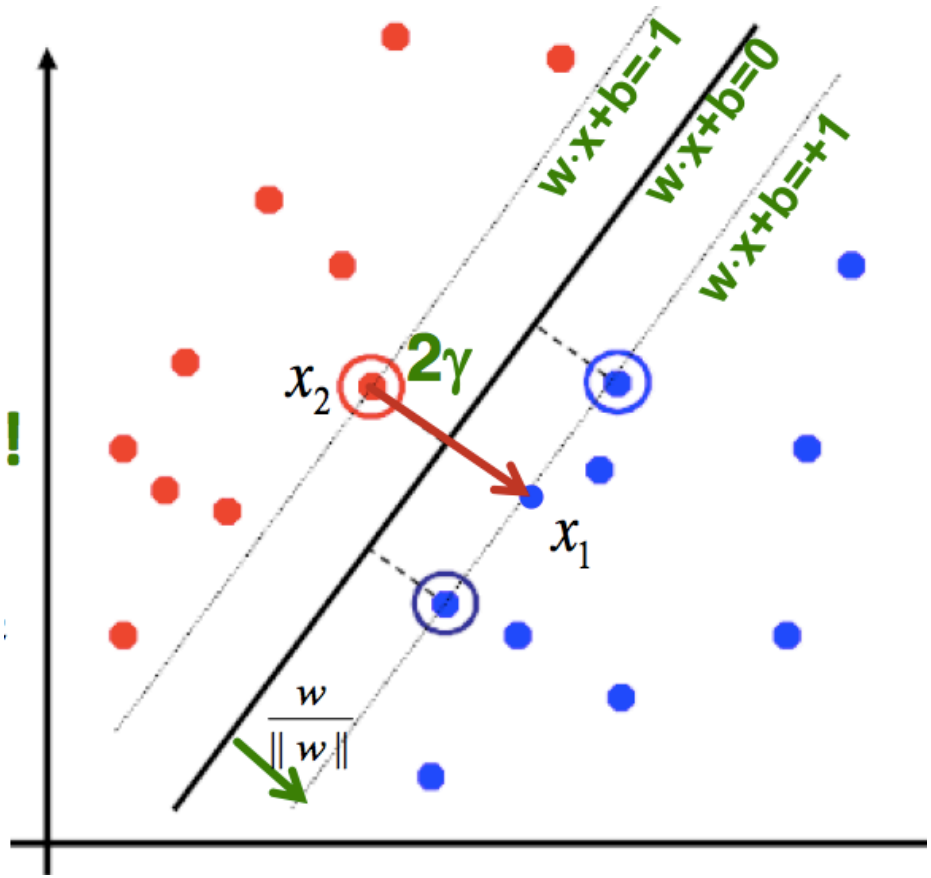
$$\max_{w,\gamma} \gamma$$
$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq \gamma, \forall i$$

But w can be arbitrarily large!

- We normalized and $\cdots$
$\max \gamma \approx \max \frac{1}{\|w\|} \approx \min \|w\| \approx \min \frac{1}{2}\|w\|^2$
- Then:

$$\min_{w} \frac{1}{2}\|w\|^2$$
$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq 1, \forall i$$

This is called SVM with "hard" constraints



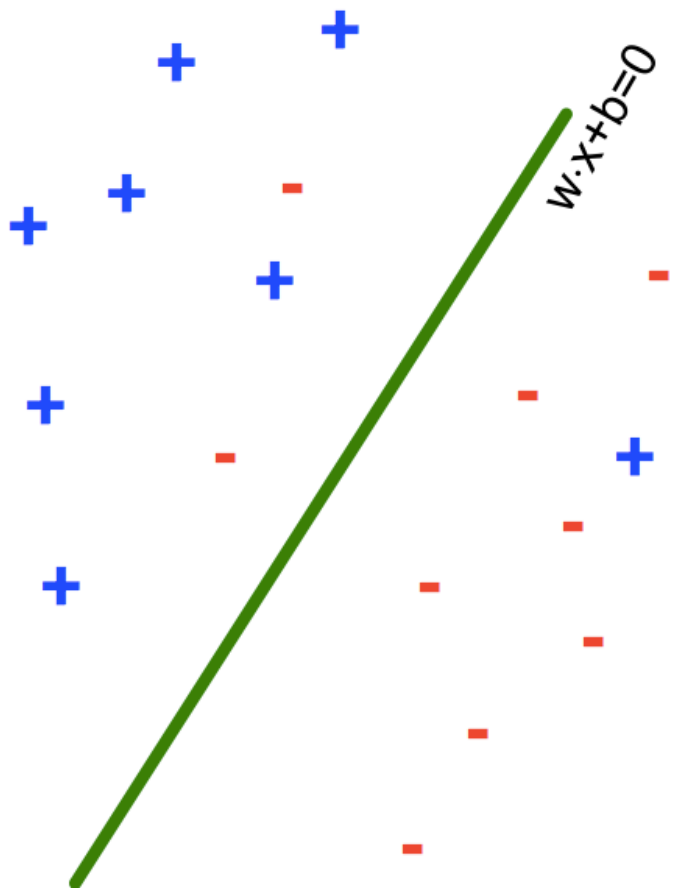
Non linearly separable data?

- If data is not separable, introduce penalty:

$$\min_w \frac{1}{2}\|w\|^2 + C \text{ (\#number of mistakes)}$$

$$\text{s.t. } y_i(w \cdot x_i + b) \geq 1, \forall i$$

  - Minimize $\|w\|^2$ plus the number of training mistakes
  - Set C using cross validation
- How to penalize mistakes?
  - All mistakes are not equally bad!



Support Vector Machines

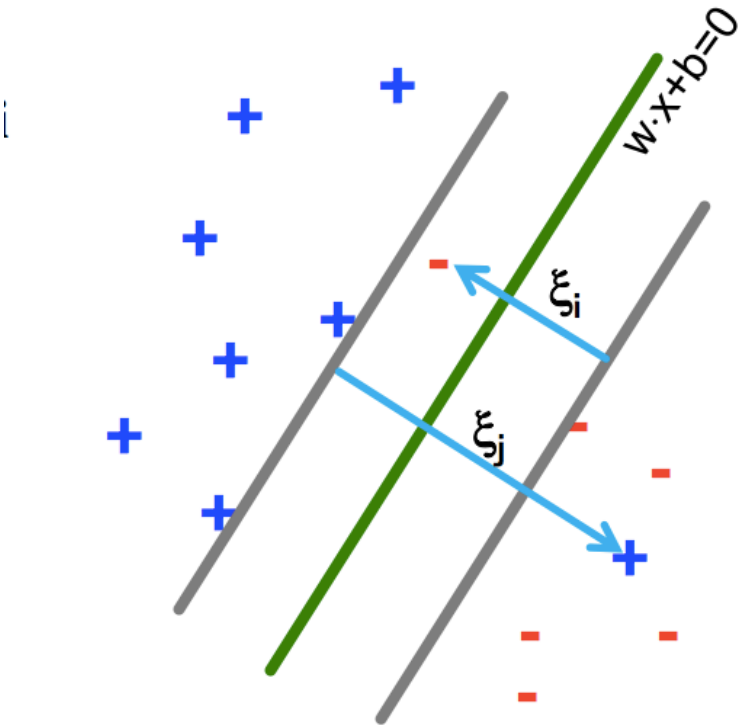
- Introduce slack variables ξ_i

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \forall i$$

$$\xi \geq 0$$

- If point x_i is on the wrong side of the margin then get penalty ξ_i



For each datapoint:
If margin ≥ 1, don't care
If margin < 1, pay linear penalty

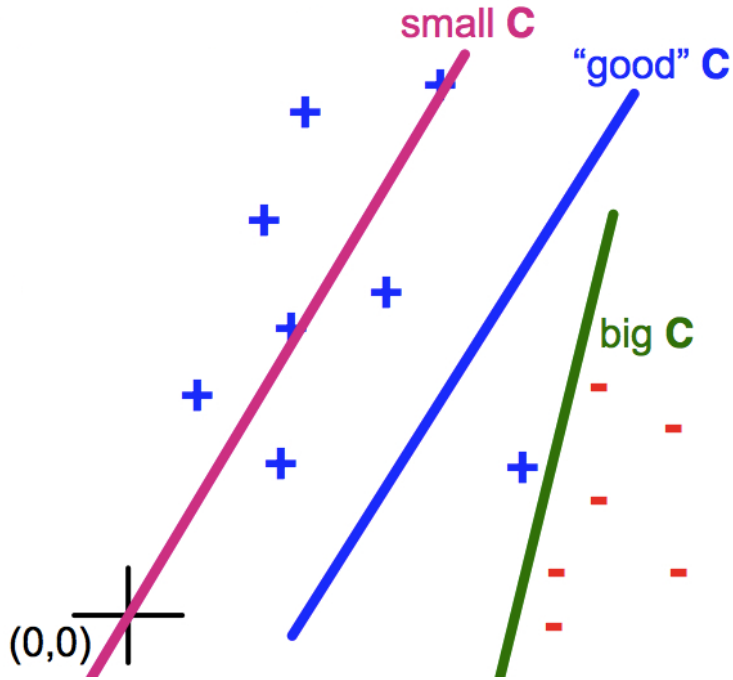
Slack Penalty C

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \forall i$$

$$\xi \geq 0$$

- What is the role of slack penalty C:
  - $C = \infty$: Only want to w,b that separate the data
  - $C = 0$: Can set ξ_i to anything, then w=0(basically ignores the data)



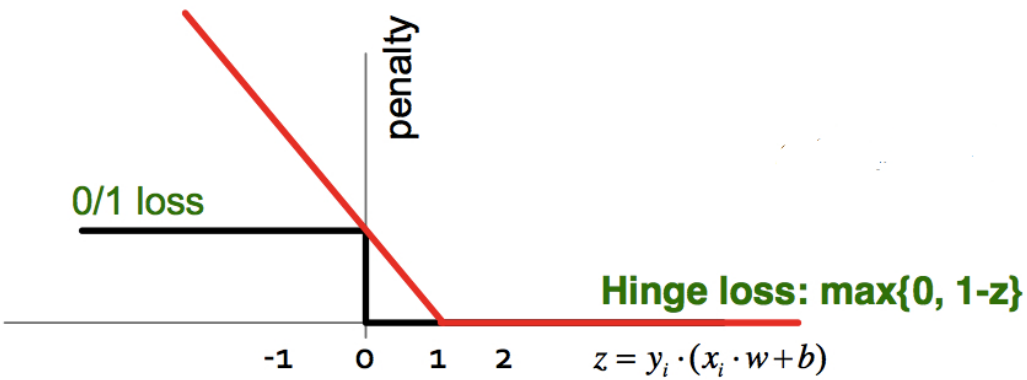
Support Vector Machines

- SVM in the "natural" form (exact penalty function)

$$\arg\min_{w,b} \frac{1}{2} \underbrace{w \cdot w}_{\text{Margin}} + C \sum_{i=1}^{n} \underbrace{\max\{0, 1 - y_i(w \cdot x_i + b)\}}_{\text{Empirical loss L (how well we fit training data)}}$$

  C is Regularization parameter

- SVM uses "Hinge Loss":



$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{s.t.} \quad y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i, \forall i$$

$$\xi \geq 0$$

SVM Classification: Primal

$$\min_{w,b,\xi} \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

$$\text{s.t. } y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i, \forall i$$

$$\xi \geq 0$$

- Want to estimate w and b!
  - Standard way: use a solver!
    - Solver:software for finding solutions to "common" optimization problems
- Use a quadratic solver:
  - Minimize quadratic function
  - Subject to linear constraints
- Challenge: solvers for problems with big data!

Dual

Dual is also a convex QP, in variable $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N)^T$:

$$\min_{\alpha} \frac{1}{2}\alpha^T K \alpha - \mathbf{1}^T \alpha \qquad \text{s.t.} \quad 0 \leq \alpha \leq C\mathbf{1},\ y^T \alpha = 0,$$

where

$$K_{ij} = (y_i y_j) x_i^T x_j,\ y = (y_1, y_2, \ldots, y_N)^T,\ \mathbf{1} = (1, 1, \ldots, 1)^T.$$

KKT conditions relate primal and dual solutions:

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i,$$

while b is Lagrange multiplier for $y^T \alpha = 0$. Leads to classifier:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i (x_i^T x) + b.$$

Kernel Trick, RKHS

For a more powerful classifier, can project feature vector x_i into a higher-dimensional space via a function $\phi : \mathbb{R}^n \to \mathbb{R}^t$ and classify in that space. **Dual formulation is the same**, except for redefined K:

$$K_{ij} = (y_i y_j)\phi(x_i)^T \phi(x_j).$$

Leads to classifier:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)^T \phi(x) + b.$$

Don't actually need to use ϕ at all, just inner products $\phi(x)^T \phi(\bar{x})$. Instead of ϕ, work with a kernel function $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.

If k is continuous, symmetric in arguments, and positive definite, there exists a Hilbert space and a function ϕ in this space such that $k(x, \bar{x}) = \phi(x)^T \phi(\bar{x})$.

Thus, a typical strategy is to choose a kernel k, form $K_{ij} = y_i y_j k(x_i, x_j)$, solve the dual to obtain α and b, and use the classifier

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i k(x_i, x) + b,$$

Most popular kernels:

- **Linear:** $k(x, \bar{x}) = x^T \bar{x}$
- **Gaussian:** $k(x, \bar{x}) = \exp(-\gamma \|x - \bar{x}\|^2)$
- **Polynomial:** $k(x, \bar{x}) = (x^T \bar{x} + 1)^d$

These (and other) kernels typically lead to K dense and ill conditioned.

Solving the Primal and (Kernelized) Dual

Many methods have been proposed for solving either the primal formulation of linear classification, or the dual (usually the kernel form).

Many are based on optimization methods, or can be interpreted using tools from the analysis of optimization algorithms.

Methods compared via a variety of metrics:

- CPU time to find solution of given quality (e.g. error rate).
- Theoretical efficiency.
- Data storage requirements.
- (Simplicity.) (Parallelizability.)

Solving the Dual

$$\min_{\alpha} \frac{1}{2}\alpha^T K\alpha - \mathbf{1}^T\alpha \quad \text{s.t.} \quad 0 \leq \alpha \leq C\mathbf{1},\ y^T\alpha = 0.$$

Convex QP with mostly bound constraints, but

a. Dense, ill conditioned Hessian makes it tricky.

b. The linear constraint $y^T\alpha = 0$ is a nuisance!

Dual SVM: Coordinate Descent

(Hsieh et al 2008) Deal with the constraint $y^T\alpha = 0$ by getting rid of it! Corresponds to removing the "intercept" term b from the classifier.

Get a convex, bound-constrained QP:

$$\min_{\alpha} \frac{1}{2}\alpha^T K\alpha - \mathbf{1}^T\alpha \quad \text{s.t.} \quad 0 \leq \alpha \leq C1.$$

Basic step: for some $i = 1, 2, \ldots, N$, solve this problem in closed form for α_i , holding all components $\alpha_j, j \neq i$ fixed.

- Can cycle through $i = 1, 2, ..., N$, or pick i at random.
- Update $K\alpha$ by evaluating one column of the kernel.
- Gets near-optimal solution quickly.

Dual SVM: Gradient Projection

(Dai, Fletcher 2006) Define $\Omega = \{0 \leq \alpha \leq C\mathbf{1}, y^T\alpha = 0\}$ and solve

$$\min_{\alpha\in\Omega} q(\alpha) := \frac{1}{2}\alpha^T K\alpha - \mathbf{1}^T\alpha$$

by means of gradient projection steps:

$$\alpha_{l+1} = P_\Omega(\alpha_l - \gamma_l \nabla q(\alpha_l)),$$

where P_Ω denotes projection onto Ω and γ_l is a steplength.
P_Ω not trivial, but not too hard to compute.
Can choose γ_l using a Barzilai-Borwein formula together with a nonmonotone (but safeguarded) procedure. Basic form of BB chooses γ_l so that $\gamma_l^{-1} I$ mimics behavior of true Hessian ∇q over the latest step; leads to

$$\gamma_l = \frac{s_l^T s_l}{s_l^T y_l}, \text{ where } s_l := \alpha_l - \alpha_{l-1},\ y_l := \nabla q(\alpha_l) - \nabla q(\alpha_{l-1})$$

.

Dual SVM: Decomposition

Many algorithms for dual formulation make use of *decomposition:* Choose a subset of components of α and (approximately) solve a subproblem in just these components, fixing the other components at one of their bounds. Usually maintain feasible α throughout.

Many variants, distinguished by strategy for selecting subsets, size of subsets, inner-loop strategy for solving the reduced problem.

SMO: (Platt 1998). Subproblem has two components.

SMV$^{\text{light}}$: (Joachims 1998). Use chooses subproblem size (usually small); components selected with a first-order heuristic. (Could use an ℓ_1 penalty as surrogate for cardinality constraint?)

PGPDT: (Zanni, Serafini, Zanghirati 2006) Decomposition, with gradient projection on the subproblems. Parallel implementation.

LIBSVM: (Fan, Chen, Lin, Chang 2005). SMO framework, with first- and second-order heuristics for selecting the two subproblem components. Solves a 2-D QP to get the step.

Heuristics are vital to efficiency, to save expense of calculating components of kernel K and multiplying with them:

- Shrinking: exclude from consideration the components α_i that clearly belong at a bound (except for a final optimality check);
- Caching: Save some evaluated elements K_{ij} in available memory.

Performance of Decomposition:

- Used widely and well for > 10 years.
- Solutions α are often not particularly sparse (many support vectors), so many outer (subset selection) iterations are required.
- Can be problematic for large data sets.

Dual SVM: Active-Set

(Scheinberg 2006)

- Apply a standard QP active-set approach to Dual, usually changing set of "free" components $\alpha_i \in (0, C)$ by one index at each iteration.
- Update Cholesky factorization of "free" part of Hessian K after each change.
- Uses shrinking strategy to (temporarily) ignore components of α that clearly belong at a bound.

(Shilton et al 2005) Apply active set to a min-max formulation (a way to get rid of $y^T\alpha = 0$:

$$\max_b \min_{0 \le \alpha \le C\mathbf{1}} \frac{1}{2} \begin{bmatrix} b \\ \alpha \end{bmatrix}^T \begin{bmatrix} 0 & y^T \\ y & K \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} - \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}^T \begin{bmatrix} b \\ \alpha \end{bmatrix}$$

Cholesky-like factorization maintained.

Active set methods good for

- warm starting, when we explore the solution path defined by C.
- incremental, where we introduce data points (x_i, y_i) one by one (or in batches) by augmenting α appropriately, and carrying on.

Dual SVM: Interior-Point

(Fine&Scheinberg 2001). Primal-dual interior-point method. Main operation at each iteration is solution of a system of the form

$$(K + D)u = w,$$

where K is kernel and D is a diagonal. Can do this efficiently if we have a low-rank approximation to K, say $K \approx VV^T$, where $V \in \mathbb{R}^{N \times p}$ with $p \ll N$.
F&S use an incomplete Cholesky factorization to find V. There are other possibilities:

- Arnoldi methods: eigs command in Matlab. Finds dominant eigenvectors / eigenvalues.
- Sampling: Nyström method (Drineas&Mahoney 2005). Nonuniform sample of the columns of K, reweight, find SVD.

Low-rank Approx + Active Set

If we simply use the low-rank approximation $K \leftarrow VV^T$, the dual formulation becomes:

$$\min_{\alpha} \frac{1}{2}\alpha^T VV^T \alpha - \mathbf{1}^T\alpha \quad \text{s.t.} \quad 0 \leq \alpha \leq C\mathbf{1}, \quad y^T\alpha = 0.$$

If we introduce $\gamma = V^T\alpha \in \mathbb{R}^p$, it becomes

$$\min_{\alpha,\gamma} \frac{1}{2}\gamma^T\gamma - \mathbf{1}^T\alpha \quad \text{s.t.} \quad 0 \leq \alpha \leq C\mathbf{1}, \quad \gamma = V^T\alpha, \quad y^T\alpha = 0,$$

For small p, can solve this efficiently with an active-set QP code (e.g. CPLEX).

Solution is unique in γ, possibly nonunique in α, but can show that the classifier is invariant regardless of which particular α is used.

Solving the Primal

$$\min_{w,b,\xi} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{N}\xi_i,$$

$$\text{subject to} \quad \xi_i \geq 0,\ y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, 2, \ldots, N.$$

Motivation: Dual solution often not particularly sparse (many support vectors - particularly with a nonlinear kernel). Dual approaches can be slow when data set is very large.

Methods for primal formulations have been considered anew recently.

Limitation: Lose the kernel. Need to define the feature space "manually" and solve a linear SVM.

But see (Chapelle 2006) who essentially replaces feature vector x_i by $[k(x_j, x_i)]_{j=1,2,\ldots,N}$, and replaces $w^T w$ by $w^T K w$. (The techniques below could be applied to this formulation.)

Primal SVM: Cutting Plane

Formulate the primal as

$$\min_{w,b} P(w,b) := \frac{1}{2}\|w\|_2^2 + R(w,b),$$

where R is a piecewise linear function of (w,b):

$$R(w,b) = C\sum_{i=1}^{N} \max(1 - y_i(w^T x_i + b), 0).$$

Cutting-plane methods build up a piecewise-linear lower-bounding approximation to $R(w,b)$ based on a subgradient calculated at the latest iterate (w^k, b^k). This approach used in many other contexts, e.g. stochastic linear programming with recourse.

In SVM, the subgradients are particularly easy to calculate.

(Joachims 2006) implemented as SVM$^{\text{perf}}$. (Franc&Sonnenburg 2008) add line search and monotonicity: OCAS. Convergence / complexity proved.

Modifications tried (Lee and Wright) by modifying OCAS code:

- partition the sum $R(w,b)$ into p bundles, with cuts generated separately for each bundle. Gives a richer approximation, at the cost of a harder subproblem.
- different heuristics for adding cuts after an unsuccessful step.

Many more ideas could be tried. In the basic methods, each iteration requires computation of the full set of inner products $w^T x_i, i = 1, 2, \ldots, N$. Could use strategies like partial pricing in linear programming to economize.

Primal SVM: How to estimate w?

- Want to estimate w,b!

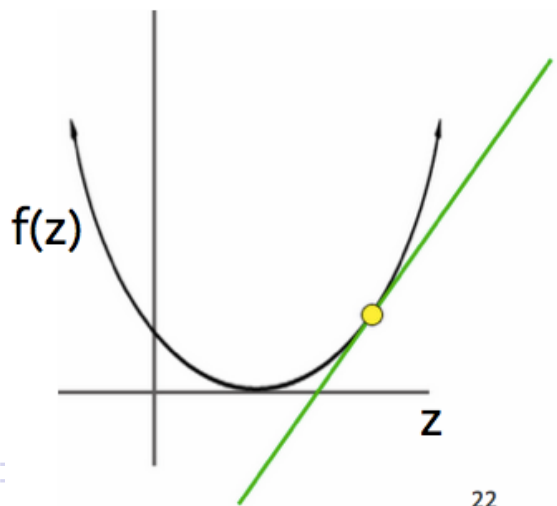
$$\min_{w,b,\xi} \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

s.t. $y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i, \xi \geq 0$

- Alternative approach:
  - Want to minimize f(w,b):

$$f(w,b) = \frac{1}{2} \sum_{j=1}^{d} (w^{(j)})^2 + C \sum_{i=1}^{n} \max\{0, 1 - y_i (\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b)\}$$

  - How to minimize convex functions $f(z)$?
  - Use subgradient method: $\min_z f(z)$
  - Iterate: $z_{t+1} \leftarrow z_t - \eta \nabla f(z_t)$



Primal SVM: How to estimate w?

- Want to minimize f(w,b):

$$f(w,b) = \frac{1}{2}\sum_{j=1}^{d}(w^{(j)})^2 + C\sum_{i=1}^{n}\underbrace{\max\{0, 1 - y_i(\sum_{j=1}^{d} w^{(j)}x_i^{(j)} + b)\}}_{\text{Empirical loss } L(x_i y_i)}$$

- Compute the subgradient $\nabla f(w^{(j)}, b)$

$$\frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

$$\frac{\partial L(x_i, y_i)}{\partial w^{(j)}} = 0 \quad \text{if } y_i(w \cdot x_i + b) \geq 1$$

$$= -y_i x_i^{(j)} \quad \text{else}$$

Primal SVM: How to estimate w?

- subgradient method:

  Iterate until convergence:

  - For $j = 1 \cdots d$
    - Evaluate: $\nabla_{w^{(j)}} f = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$
    - Update: $w^{(j)} \leftarrow w^{(j)} - \eta \nabla_{w^{(j)}} f$

  η is learning rate parameter

  C is regularization parameter

- Problem:
  - Computing $\nabla_{w^{(j)}} f$ takes $O(n)$ time!
    - n is size of the training dataset

Primal SVM: How to estimate w?

We just had: $\nabla_{w^{(j)}} f = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$

- Stochastic subgradient method
  - Instead of evaluating gradient over all examples evaluate it for each individual training example

$$(\nabla_{w^{(j)}} f)_i = w^{(j)} + C \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

- Stochastic subgradient method

Iterate until convergence:

- For $i = 1 \cdots n$
  - For $j = 1 \cdots d$
    - Evaluate: $(\nabla_{w^{(j)}} f)_i$
    - Update: $w^{(j)} \leftarrow w^{(j)} - \eta (\nabla_{w^{(j)}} f)_i$

Primal SVM: Stochastic Subgradient

(Bottou) Take steps in the subgradient direction of a few-term approximation to $P(w,b)$, e.g. at iteration k, for some subset $I_k \subset \{1,2,\ldots,N\}$, use subgradient of

$$P_k(w,b) := \frac{1}{2}\|w\|_2^2 + C\frac{N}{|I_k|}\sum_{i\in I_k}\max(1-y_i(w^T x_i + b),0),$$

Step length η_k usually decreasing with k according to a fixed schedule. Can use rules $\eta_k \sim k^{-1}$ or $\eta_k \sim k^{-1/2}$.

Cheap if $|I_k|$ is small. Extreme case: I_k is a single index, selected randomly. Typical step: Select $j(k) \in \{1,2,\ldots,N\}$ and set

$$(w^{k+1}, b^{k+1}) \leftarrow (w^k, b^k) - \eta_k g_k,$$

where

$$g_k = \begin{cases} (w,0) & \text{if } 1 - y_{j(k)}(w^T x_{j(k)} + b) \le 0, \\ (w,0) - CN_{y_{j(k)}}(x_{j(k)},1) & \text{otherwise.} \end{cases}$$

Stochastic Subgradient

(Shalev-Shwartz, Singer, Srebro 2007). Pegasos: After subgradient step, project w onto a ball $\{w| \, \|w\|_2 \leq \sqrt{CN}\}$. Performance is insensitive to $|I_k|$. (Omits intercept b.)

Convergence: Roughly, for steplenths $\eta_k = CN/k$, have for fixed total iteration count T and k randomly selected from $\{1, 2, \ldots, T\}$, the expected value of the objective f is within $O(T^{-1} \log T)$ of optimal.

Similar algorithms proposed in (Zhang 2004), (Kivinen, Smola, Williamson 2002) - the latter with a steplength rule of $\eta_k \sim k^{-1/2}$ that yields an expected objective error of $O(T^{-1/2})$ after T iterations.

There's a whole vein of optimization literature that's relevant— Russian in origin, but undergoing a strong revival. One important and immediately relevant contribution is (Nemirovski et al. 2009).

Stochastic Approximation Viewpoint

(Nemirovski et al, SIAM J Optimization 2009) consider the setup

$$\min_{x \in X} f(x) := E_\zeta[F(x,\zeta)],$$

where subgradient estimates $G(x,\zeta)$ are available such that $g(x) := E_\zeta[G(x,\zeta)]$ is a subgradient of f at x. Steps:

$$x^{k+1} \leftarrow P_X(x^k - \eta_k G(x^k, \zeta^k))$$

where ζ^k selected randomly. Some conlusions:

- If f is convex with modulus γ, steplengths $\eta_k = (\gamma k)^{-1}$ yield $E[f(x^k) - f(x^*)] = O(1/k)$.
- Slight differences to the stepsize (e.g. a different constant multiple) can greatly degrade performance.
- If f is convex (maybe weakly), the use of stepsizes $\eta_k \sim k^{-1/2}$ yields convergence at rate $k^{-1/2}$ of a weighted average of iterates in expected function value.
- This is a slower rate, but much less sensitive to the "incorrect" choices of steplength scaling. See this in practice.

Example: Text categorization

- Example by Leon Bottou:
  - Reuters RCV1 document corpus
    - Predict a category of a document
      One vs. the rest classification
  - $n = 781,000$ training examples(documents)
  - $23,000$ test examples
  - $d = 50,000$ features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

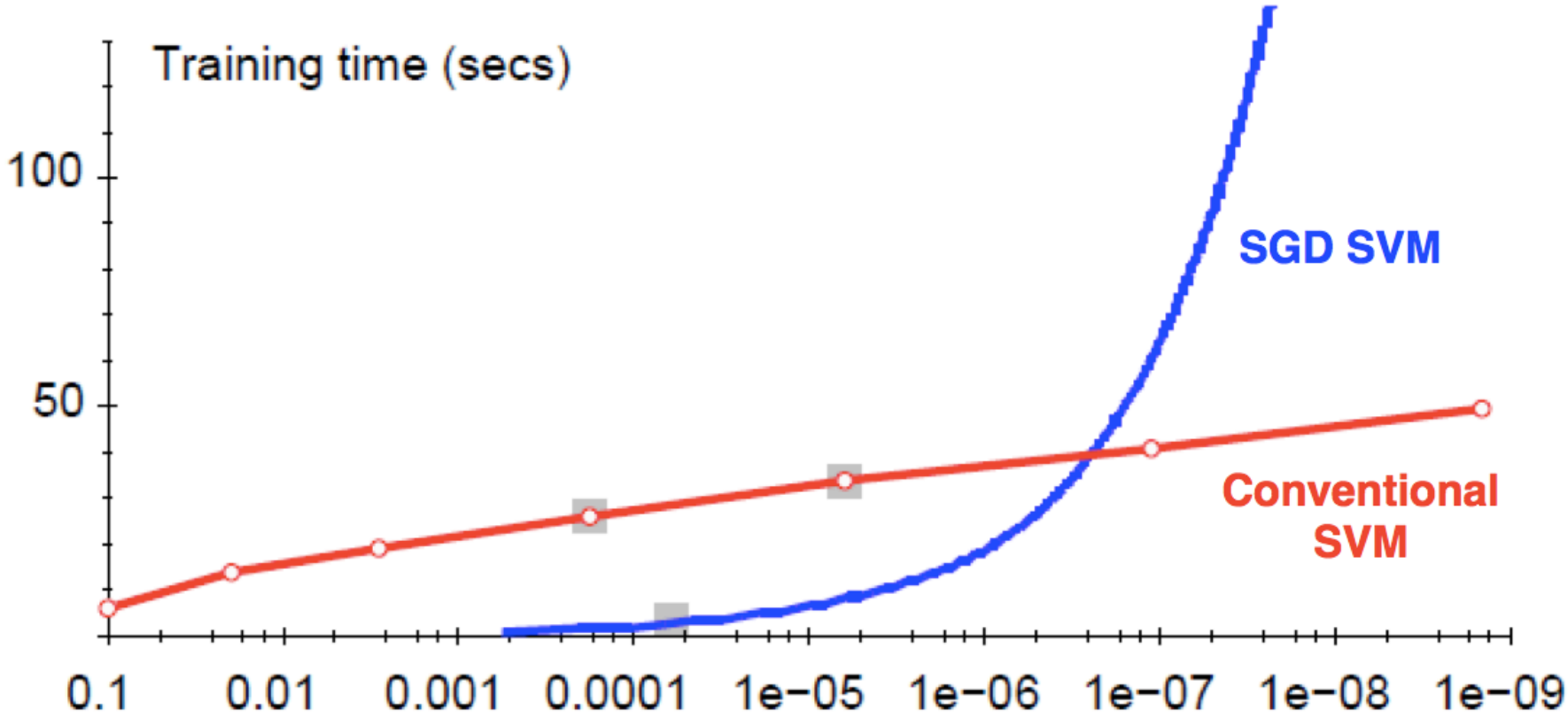
Example: Text categorization

- Questions:
  1. Is SGD successful at minimizing $f(w, b)$?
  2. How quickly does SGD find the min of $f(w, b)$?
  3. What is the error on a test set?

	Training time	***Value of f(w,b)***	***Test error***
Standard SVM	23,642 secs	*0.2275*	*6.02%*
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	*0.2275*	*6.02%*

(1) SGD-SVM is successful at minimizing the value of $f(w, b)$
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable

Optimization "Accuracy"



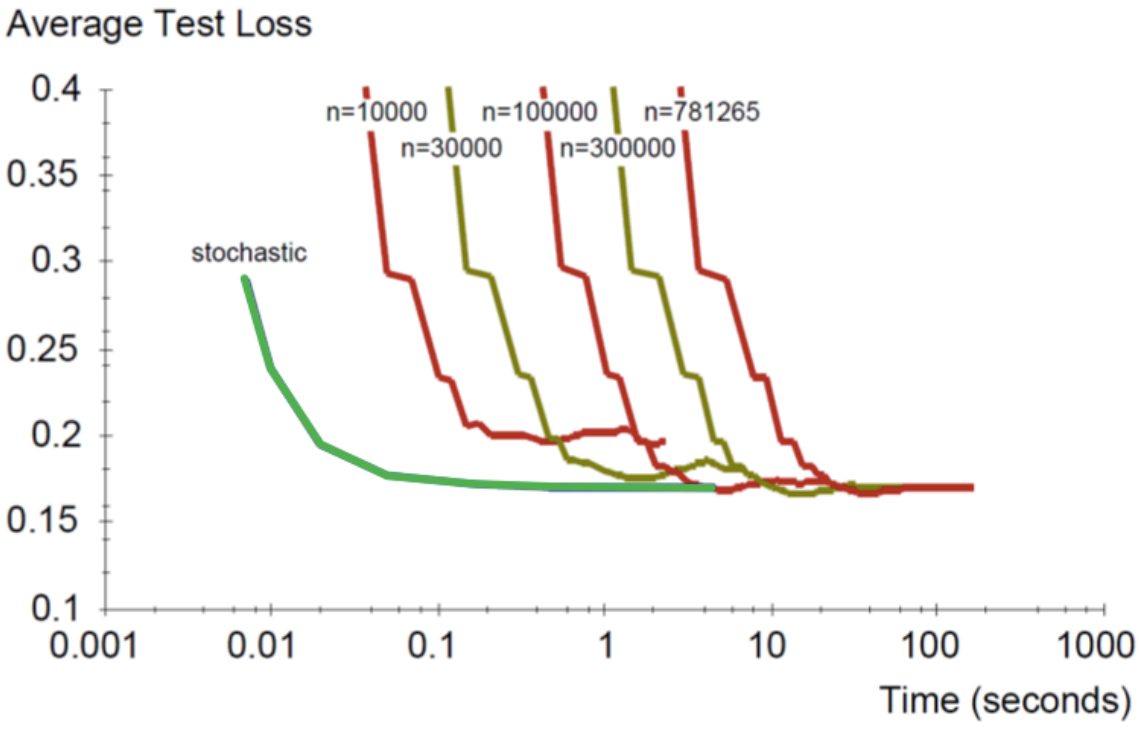
Optimization quality: $|f(w,b) - f(w^{opt}, b^{opt})|$

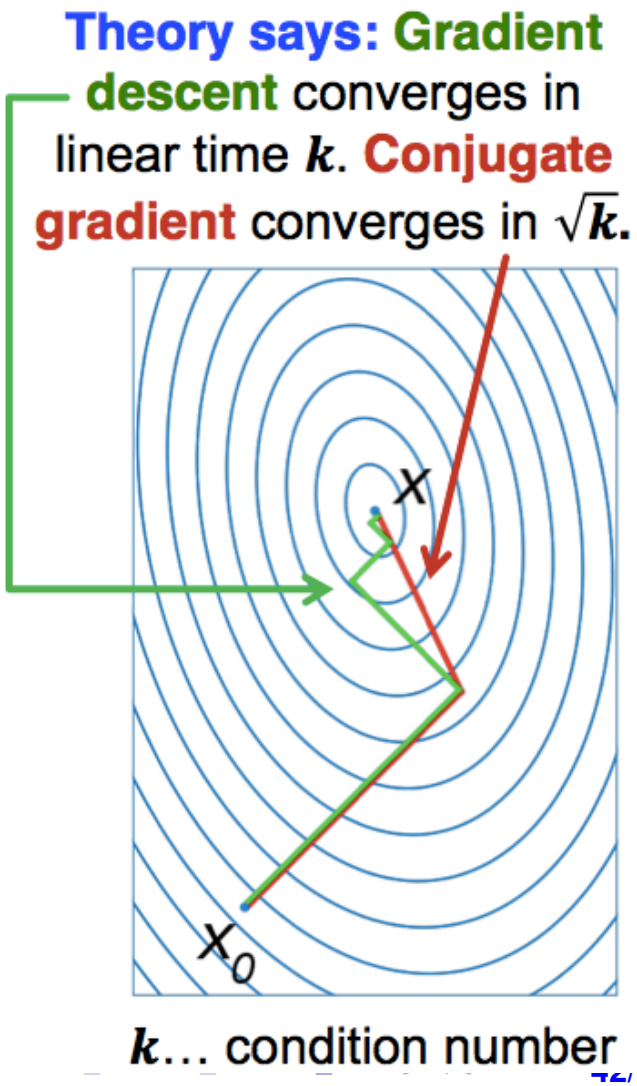
For optimizing $f(w,b)$ within reasonable quality

$SGD - SVM$ is super fast

SGD vs. Batch Conjugate Gradient

- SGD on full dataset vs. Batch Conjugate Gradient on a sample of n training examples





Bottom line:Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) BCG update a few times

Practical Considerations

- Need to choose learning rate η and t_0

$$w_{t+1} \leftarrow w_t - \frac{\eta_t}{t + t_0}(w_t + C\frac{\partial L(x_i, y_i)}{\partial w})$$

- Leon suggests:
  - Choose t_0 so that the expected initial updates are comparable with the expected size of the weights
  - Choose η :
    - Select a small subsample
    - Try various rates $\eta (e.g., 10, 1, 0.1, 0.01, \cdots)$
    - Pick the one that most reduces the cost
    - Use η for next 100k iterations on the full dataset

Practical Considerations

- Sparse Linear SVM:
  - Feature vector x_i is sparse (contains many zeros)
    - Do not do: $x_i = [0, 0, 0, 1, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, \cdots]$
    - But represent x_i as a sparse vector $x_i = [(4, 1), (9, 5), \cdots]$
  - Can we do the SGD update more efficiently?

$$w \leftarrow w - \eta(w + C\frac{\partial L(x_i, y_i)}{\partial w})$$

  - Approximated in 2 steps:

$$w \leftarrow w - \eta C\frac{\partial L(x_i, y_i)}{\partial w}$$

$$w \leftarrow w(1 - \eta)$$

cheap: x_i is sparse and so few coordinates j of w will be updated

expensive:w is not sparse,all coordinates need to be updated

Practical Considerations

- Solution 1: $\mathbf{w} = \mathbf{s} \cdot \mathbf{v}$
  - Represent vector $\mathbf{w}$ as the product of scalar $\mathbf{s}$ and vector $\mathbf{v}$
  - Then the update procedure is:
    (1) $v = v - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
    (2) $s = s(1 - \eta)$

Two step update procedure:
(1) $w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
(2) $w \leftarrow w(1 - \eta)$

- Solution 2:
  - Perform only step (1) for each training example
  - Perform step (2) with lower frequency and higher η

Practical Considerations

- Stopping criteria
  How many iterations of SGD?
  - Early stopping with cross validation
    - Create validation set
    - Monitor cost function on the validation set
    - Stop when loss stops decreasing
  - Early stopping
    - Extract two disjoint subsamples A and B of training data
    - Train on A, stop by validating on B
    - Number of epochs is an estimate of k
    - Train for k epochs on the full dataset

Alternative Formulations: $\|w\|_1$.

Replacing $\|w\|_2^2$ by $\|w\|_1$ in the primal formulation gives a **linear program** (e.g. Mangasarian 2006; Fung&Mangasarian 2004, others):

$$\min_{w,b,\xi} \|w\|_1 + C \sum_{i=1}^{N} \max(1 - y_i(w^T x_i + b), 0).$$

Sometimes called "1-norm linear SVM."

Tends to produce **sparse** vectors w; thus classifiers that depend on a small set of features.

($\|\cdot\|_1$ regularizer also used in other applications, e.g. compressed sensing).

Production LP solvers may not be useful for large data sets; the literature above describes specialized solvers.

Elastic Net

Idea from (Zou&Hastie 2005). Include both $\|w\|_1$ and $\|w\|_2$ terms in the objective:

$$\min_{w,\xi} \frac{\lambda_2}{2}\|w\|_2^2 + \lambda_1\|w\|_1 + \sum_{i=1}^{N} \max(1 - y_i(w^T x_i + b), 0).$$

In variable selection, combines ridge regression with LASSO. Good at "group selecting" (or not selecting) correlated w_i's jointly.

Is this useful for SVM?

It would be easy to extend some of the techniques discussed earlier to handle this formulation.

SpaRSA

An extremely simple approach introduced in context of compressed sensing (Wright, Figueiredo, Nowak 2008) can be applied more generally, e.g. to logistic regression. Given formulation

$$\min \mathcal{F}(x) + \lambda \mathcal{R}(x),$$

and current iterate x^k, find new iterate by choosing scalar α_k and solving

$$\min_z \frac{1}{2\alpha_k}(z - x^k)^T(z - x^k) + \nabla \mathcal{F}(x^k)^T(z - x^k) + \lambda \mathcal{R}(z).$$

Possibly adjust α_k to get descent in the objective, then set $x^{k+1} \leftarrow z$.

- Form a quadratic model of $\mathcal{F}$ around x^k , correct to first order, with simple Hessian approximation $1/\alpha_k$.
- Variants: Barzilai-Borwein, nonmonotonic.
- Useful when the subproblem is cheap to solve.
- Continuation strategy useful in solving for a range of λ values (largest to smallest). Use solution for one λ as warm start for the next smaller value.

When $R = \|\cdot\|_1$ (standard compressed sensing), can solve subproblem in $O(n)$ (closed form).

Still cheap when

$$\mathcal{R}(x) = \sum_l \|x_{[l]}\|_2, \quad \mathcal{R}(x) = \sum_l \|x_{[l]}\|_\infty$$

where $x_{[l]}$ are disjoint subvectors. (Group LASSO.)

Not so clear how to solve the subproblems cheaply when

- subvectors $x_{[l]}$ are not disjoint in the group-lasso formulation
- regularized $\mathcal{R}$ chosen to promote a hierarchical relationship between components of x
- $\mathcal{R}(x)$ is a TV-norm.