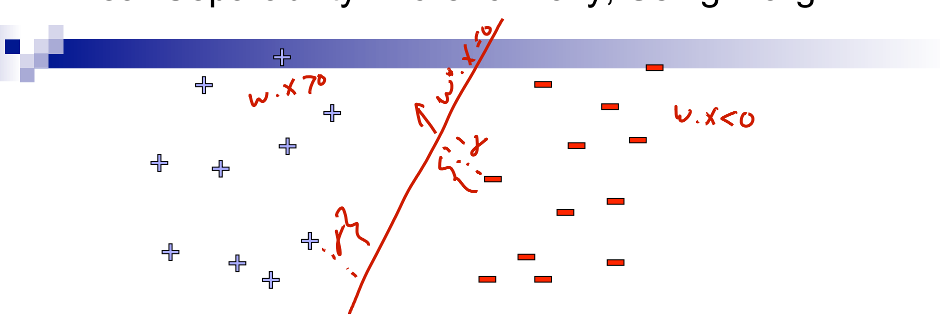
Kernels

Machine Learning – CSE446
Carlos Guestrin
University of Washington

October 28, 2013

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector $\exists\ w^*,\ \|w^*\|=1$
  - a margin $\gamma>0$
- Such that all points are at least γ away from $w.x=0$

$\forall t$: if $y^{(t)}=+1$, $w^*.x^{(t)} \geq \gamma$; Linearly separable: $y^{(t)}\ w^*.x^{(t)} \geq \gamma$

$y^{(t)}=-1$, $w^*.x^{(t)} \leq -\gamma$

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $(x^{(1)}, y^{(1)}) \dots (x^{(T)}, y^{(T)})$
    any
    not iid
  - Each feature vector has bounded norm: $\forall t \ \|x^{(t)}\| \le R$
  - If dataset is linearly separable:

$$\exists w^*, \ \|w^*\| = 1, \ \forall t \quad y^{(t)} w^* \cdot x^{(t)} \geq \gamma \quad \text{for } \gamma > 0$$

- Then the number of mistakes made by the online perceptron on any such sequence is bounded by

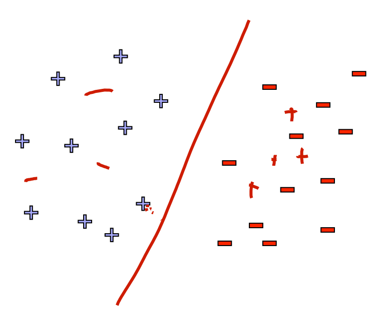
$$\left(\frac{R}{\gamma}\right)^2$$

Wow!! :)

constant, doesn't grow with T!! :)

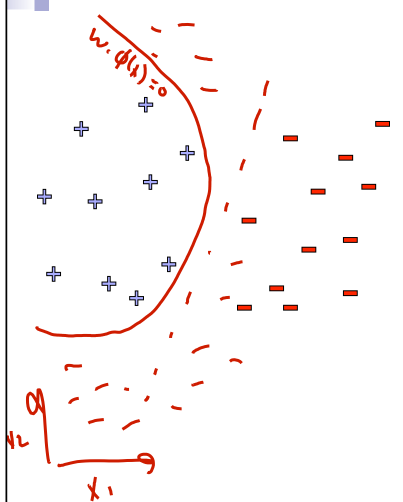
Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data
- However, real world not linearly separable
  - Can't expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire
  - Converges, but ultimately may not give good accuracy (make many many many mistakes)



& if data is very very non-linearly separable

What if the data is not linearly separable?



Use features of features of features of features….

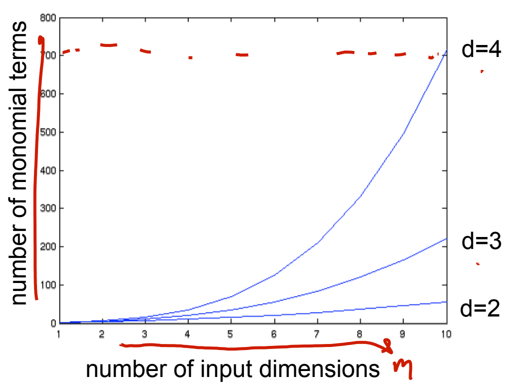
$$\Phi(\mathbf{x}) : R^m \mapsto F$$

$$\phi(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \\ e^{x_1 x_2} \\ e^{\sin x_1}, \log x_2 \\ \vdots \end{pmatrix}$$

Feature space can get really large really quickly!

Higher order polynomials

$$\text{num. terms} = \binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!}$$



m – input features
d – degree of polynomial

Kernels:
e.g. fit polynomials of any degree $\approx$ same cost

grows fast!
d = 6, m = 100
about 1.6 billion terms

Perceptron Revisited

- Given weight vector $w^{(t)}$, predict point **x** by:

$$\hat{y} = \text{sign}\left(w^{(t)} \cdot x\right)$$

- Mistake at time *t*: $w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)}$

for simplicity $w^{(0)} = 0$

- Thus, write weight vector in terms of mistaken data points only:
  - Let $M^{(t)}$ be time steps up to *t* when mistakes were made:

$$w^{(t)} = \sum_{j \in M(t)} y^{(j)} x^{(j)}$$

only depends on dot prod. between x & mistakes

- Prediction rule now:

$$\text{sign}\left(x \cdot w^{(t)}\right) = \text{sign}\left(x \cdot \sum_{j \in M(t)} y^{(j)} x^{(j)}\right) = \text{sign}\left(\sum_{j \in M(t)} y^{(j)} x \cdot x^{(j)}\right)$$

- When using high dimensional features:

$$\text{sign}\left(\phi(x) \cdot w^{(t)}\right) = \text{sign}\left(\sum_{j \in M(t)} y^{(j)} \underbrace{\phi(x) \cdot \phi(x^{(j)})}_{\text{can often be comp efficiently, even in high dim}}\right)$$

Dot-product of polynomials

$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \quad v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$ ~~=~~ for polynomials of degree exactly d

$$d=1 \quad \phi(u) \cdot \phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u \cdot v$$

$$d=2 \quad \phi(u) \cdot \phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} \cdot \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2 u_1 u_2 v_1 v_2 + u_2^2 v_2^2 = (u_1 v_1 + u_2 v_2)^2 = (u \cdot v)^2$$

Proof by single of induction

For poly. of degree exactly d : $\quad \phi(u) \cdot \phi(v) = (u \cdot v)^d = K(u,v)$

Kernel trick!!

Finally the Kernel Trick!!! (Kernelized Perceptron

- Every time you make a mistake, remember $(x^{(t)},y^{(t)})$

↳ keep index M(t) of all mistakes up to time t

- Kernelized Perceptron prediction for **x**:

$$\underbrace{sign(\mathbf{w}^{(t)} \cdot \phi(\mathbf{x}))}_{\hat{y}} = \sum_{j \in M^{(t)}} y^{(j)} \phi(\mathbf{x}^{(j)}) \cdot \phi(\mathbf{x})$$

$$= \sum_{j \in M^{(t)}} y^{(j)} k(\mathbf{x}^{(j)}, \mathbf{x})$$

remember mistakes & use kernels for dot prods

Polynomial kernels

- All monomials of degree d in O(d) operations:

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d =$ polynomials of degree exactly d

- How about all monomials of degree up to d?
  - Solution 0: $\phi(u) \cdot \phi(v) = \sum_{i=0}^{d} \binom{d}{i} (u \cdot v)^i$
  - Better solution:

$(u \cdot v)^1 + (u \cdot v)^2 + (v \cdot u)^1 + (\cancel{u \cdot v})^0 = (u \cdot v + 1)^2$

proof by "induction"

$\phi(u) \cdot \phi(v) = K(u, v) = (u \cdot v + 1)^d$

wow!! ☺

Common kernels

MANY KERNELS ARE EFFICIENT

- Polynomials of degree exactly d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian (squared exponential) kernel

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||^2}{2\sigma^2}\right)$$

Radial basis function
RBF kernel
Gaussian kernel
infinite dim feature space

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proofs
- The kernel trick
- Kernelized Perceptron
- Derive polynomial kernel
- Common kernels
- In online learning, report averaged weights at the end

Your Midterm…

- Content: Everything up to last Wednesday (Perceptron)…
- Only 80mins, so arrive early and settle down quickly, we'll start and end on time
- "Open book"
  - Textbook, Books, Course notes, Personal notes, your HWs
- Bring a calculator that can do log ☺
- No:
  - Computers, tablets, phones, other materials, internet devices, wireless telepathy or wandering eyes…
- The exam:
  - Covers key concepts and ideas, work on understanding the big picture, and differences between methods

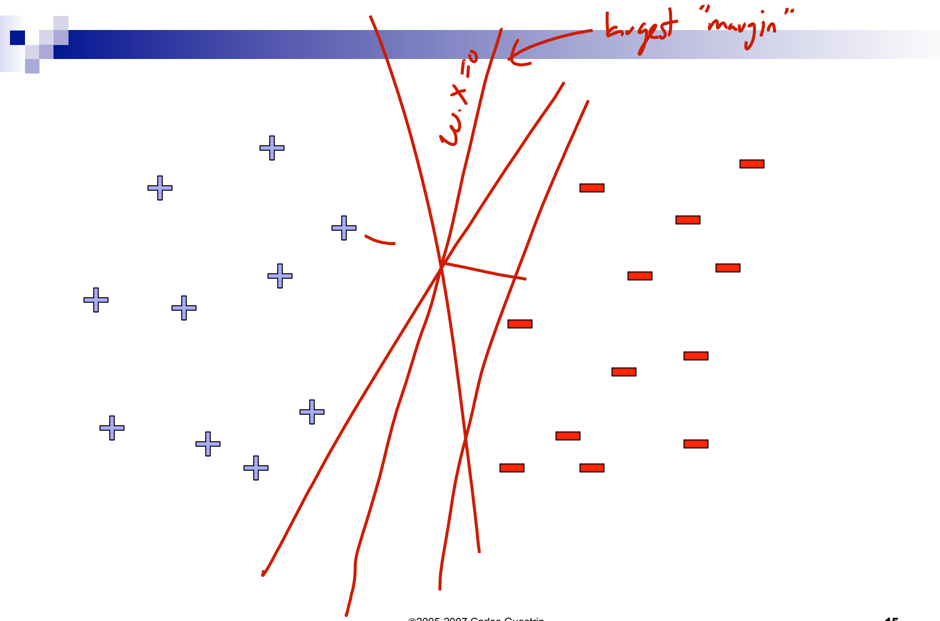
≈ perceptron + L2 regularization.

Support Vector Machines

Machine Learning – CSE446
Carlos Guestrin
University of Washington

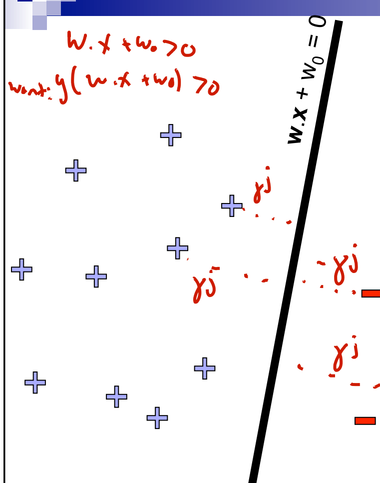
October 28, 2013

Linear classifiers – Which line is better?



Pick the one with the largest margin!

$$\text{“confidence”} = y^j(\mathbf{w} \cdot \mathbf{x}^j + w_0)$$



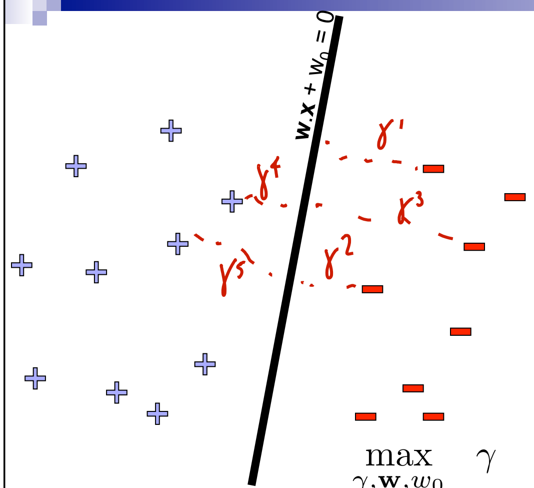
$w \cdot x + w_0 < 0$

want $y(w \cdot x + w_0) > 0$

$\gamma^j = y^j(w \cdot x^j + w_0)$

Maximize the margin

SVMs: max worst case γ^j
min γ^j



$$\max_{w, w_0} \min_j \gamma^j$$

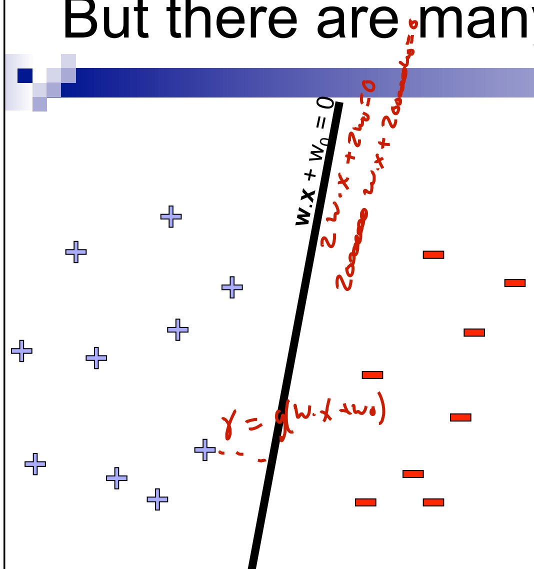
$$\gamma^j = y^j (w \cdot x^j + w_0)$$

find me w, w_0 such that all points have margin at least γ

$$\max_{\gamma,\mathbf{w},w_0} \gamma$$

$$y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0) \geq \gamma, \forall j \in \{1,\ldots,N\}$$

But there are many planes…



γ will increase in magnitude as we increase w, w_0

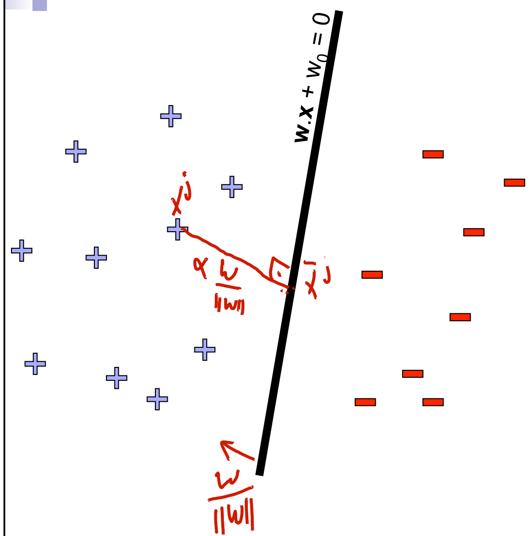
$\Rightarrow$ unbounded problem

Review: Normal to a plane

$$\mathbf{x}^j = \bar{\mathbf{x}}^j + \alpha \frac{\mathbf{w}}{||\mathbf{w}||}$$

projection onto plane

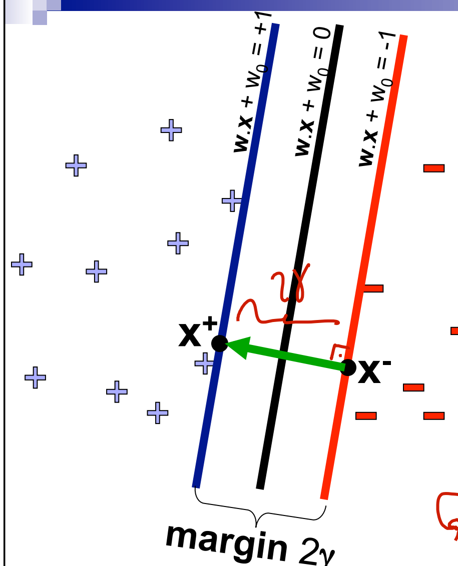
normal direction



A Convention: Normalized margin – Canonical hyperplanes

$w.w = \|w\|^2$

$$\mathbf{x}^j = \bar{\mathbf{x}}^j + \alpha \frac{\mathbf{w}}{||\mathbf{w}||}$$



plug

$w \cdot x^+ + w_0 = 1$ } from canonical planes

$w \cdot x^- + w_0 = -1$

$x^+ = x^- + 2\gamma \frac{w}{\|w\|}$ } because of projection

$w(x^- + 2\gamma \frac{w}{\|w\|}) + w_0 = 1$

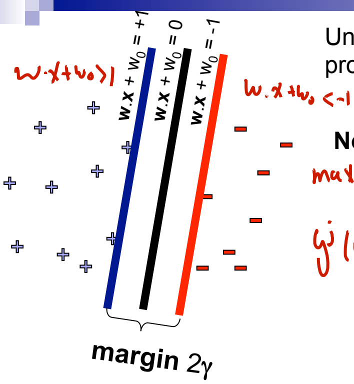
$\underbrace{w \cdot x^- + w_0}_{-1} + 2\gamma \frac{w.w}{\|w\|} = 1$

$\Rightarrow 2\gamma \frac{w.w}{\|w\|} = 2 \Rightarrow \gamma = \frac{1}{\|w\|}$

margin increases as $\|w\| \to 0$

Margin maximization using canonical hyperplanes

$\max \frac{1}{f} \equiv \min f \equiv \min f^2$, $f > 0$



Unnormalized problem:

$$\max_{\gamma,\mathbf{w},w_0} \gamma$$
$$y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0) \geq \gamma, \forall j \in \{1,\ldots,N\}$$

Normalized Problem:

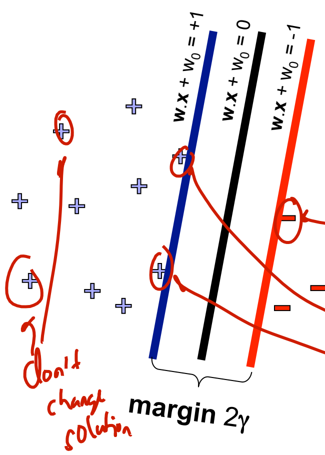
$\max \gamma \equiv \max \frac{1}{||w||} \equiv \min ||w|| \equiv \min ||w||^2$

$y^j(w \cdot x^j + w_0) \geq 1$

SVMs

$$\min_{\mathbf{w},w_0} ||w||_2^2$$
$$y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0) \geq 1, \forall j \in \{1,\ldots,N\}$$

Support vector machines (SVMs)

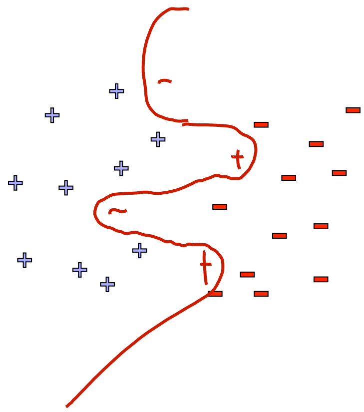


$$\min_{\mathbf{w},w_0} ||w||_2^2$$
$$y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0) \geq 1, \forall j \in \{1,\ldots,N\}$$

- Solve efficiently by many methods, e.g.,
  - quadratic programming (QP)
    - Well-studied solution algorithms
  - Stochastic gradient descent
- Hyperplane defined by support vectors

Could forget all data except for support vectors & get same solution

What if the data is not linearly separable?

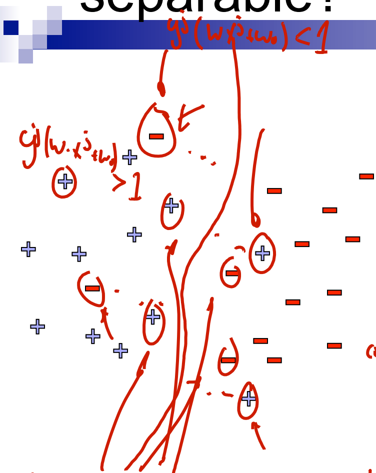


Use features of features of features of features….

What if the data is still not linearly separable?

$$\min_{\mathbf{w},w_0} \quad ||w||_2^2$$

$$y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0) \geq 1 \qquad ,\forall j$$



Support vectors: all j where $y^j(w\cdot x^j+w_0)\leq 1$

- If data is not linearly separable, some points don't satisfy margin constraint:

$\exists j \; y^j(w\cdot x^j + w_0) < 1 \Leftrightarrow \underbrace{1 - y^j(w\cdot x^j+w_0)}_{\text{a violation of constraint}} > 0$

- How bad is the violation?

$$\text{constraint violation} = \left.\begin{cases} 0 & 1-y^j(w x^j+w_0)\leq 0 \\ 1-y^j(w x^j+w_0) & \text{otherwise}\end{cases}\right\} \left(1-y^j(w x^j+w_0)\right)_+$$

- Tradeoff margin violation with ||**w**||:

$$\min_{w,w_0} ||w||_2^2 + C\sum_{j=1}^{N}\left(1-y^j(w x^j+w_0)\right)_+$$

$C>0$ slack penalty

SVMs for Non-Linearly Separable meet my friend the Perceptron…

- Perceptron was minimizing the hinge loss:

$$\sum_{j=1}^{N} \left(-y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0)\right)_+$$

hinge loss

- SVMs minimizes the regularized hinge loss!!

convention in SVMs

$$\underbrace{||\mathbf{w}||_2^2}_{\text{regularization}} + C\sum_{j=1}^{N} \left(1 - y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0)\right)_+$$

Stochastic Gradient Descent for SVMs

(!)

- Perceptron minimization:

$$\sum_{j=1}^{N} \left(-y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0)\right)_+$$

- SGD for Perceptron:

$$\underbrace{\mathbf{w}^{(t+1)}}_{\text{update weights}} \leftarrow \mathbf{w}^{(t)} + \eta\, \overbrace{\underbrace{\mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)}\cdot\mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}}_{D(-y^{(t)}(w^{(t)}\cdot x^{(t)} + w_0))_+}}^{\text{mistake}}$$

$\eta = 1$

- SVMs minimization:

$$||\mathbf{w}||_2^2 + C\sum_{j=1}^{N} \left(1 - y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0)\right)_+$$

- SGD for SVMs:

$$w^{(t+1)} \leftarrow w^{(t)} + \eta\left(C\,\mathbb{1}\left(1 - y^{(t)}(w^{(t)}\cdot x^{(t)} + w_0^{(t)}) \leq 0\right) y^{(t)} x^{(t)} - 2w^{(t)}\right)$$

step size is important in SVMs

pick C?

by cross validation

What you need to know

- Maximizing margin
- Derivation of SVM formulation
- Non-linearly separable case
  - Hinge loss
  - A.K.A. adding slack variables
- SVMs = Perceptron + L2 regularization
- Can also use kernels with SVMs
- Can optimize SVMs with SGD
  - Many other approaches possible