Kernels

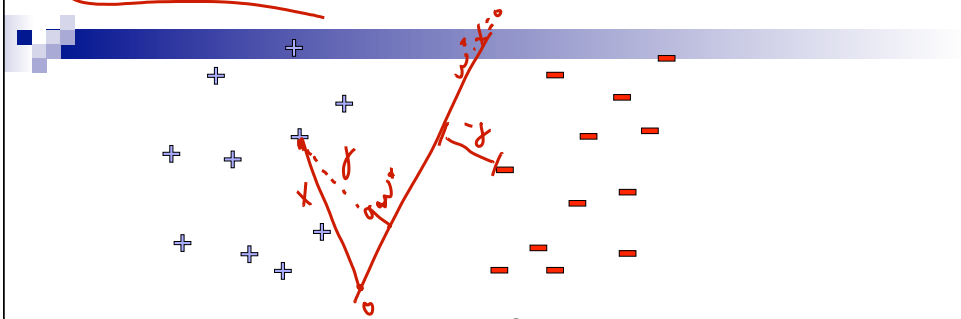
Machine Learning – CSE446

Carlos Guestrin

University of Washington

May 3, 2013

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector
  - a margin
- Such that

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \ldots (x^{(T)}, y^{(T)})$
  - Each feature vector has bounded norm: $\forall t \ \|x^{(t)}\| \le R$
  - If dataset is linearly separable: $\exists w^*, \|w^*\| = 1 \quad \forall t \quad y^{(t)} w^* \cdot x^{(t)} \ge \gamma$, for $\gamma > 0$
- Then the number of mistakes made by the online perceptron on any sequence is bounded by $\left(\frac{R}{\gamma}\right)^2$

examples need not be i.i.d. or random..

w^* is unknown!

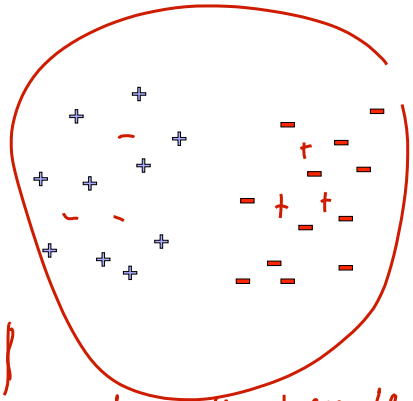
wow!!

$\le$ constant, doesn't depend on T

or dimensionality of X !!

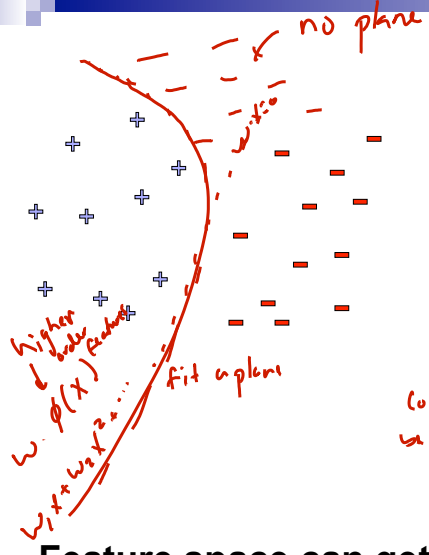
Beyond Linearly Separable Case

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



we need features that make data as linearly separable as possible

What if the data is not linearly separable?



Use features of features of features….

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

$\phi(x) =$ [1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, $x_1x_2^2$, x_2^3, e^{x}, $\sin x$, $\dots$, $e^{-\frac{x^2}{\sigma^2}}$, $\dots$]

could even be ∞ dim

space here

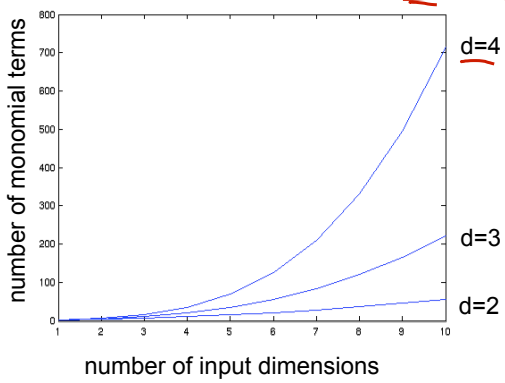
Feature space can get really large really quickly!

Higher order polynomials

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

dim of $\phi(x)$ = dim of ϕ

m – input features
d – degree of polynomial



even though dims of $\phi(x)$ are huge, fit model very quickly

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

Perceptron Revisited

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

$$\hat{y} = \text{Sign}(w^{(t)} \cdot x)$$

- Mistake at time *t*: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$
- Thus, write weight vector in terms of mistaken data points only:
  - Let $M^{(t)}$ be time steps up to *t* when mistakes were made:

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

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

how similar x is to my old mistakes

- Prediction rule now:

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

classification rule only depends on dot prod. between the points

- When using high dimensional features:

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

when can you compute this efficiently

Dot-product of polynomials

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

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

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

$= (u_1 v_1 + u_2 v_2)^2 = (u \cdot v)^2$

proof by base case (one step of induction)

for poly of degree exactly d

$\phi(u) \cdot \phi(v) = (u \cdot v)^d$ ← kernel trick in general

compute dot product extremely efficiently

Finally the Kernel Trick!!! (Kernelized Perceptron

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

↳ keep indices $M(t)$ mistakes up to time t

- Kernelized Perceptron prediction for $\mathbf{x}$:

prediction

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

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

kernel function

$K(u,v) = \phi(u) \cdot \phi(v)$

perceptron alg w. kernels
→ keep mistakes in memory
→ compute prediction
if mistake, save it

Polynomial kernels

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

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

- 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: $d=2$: $(u \cdot v)^2 + (u \cdot v)^1 + (u \cdot v)^0 = (u \cdot v + 1)^2$

for polynomials of degree d

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

Common kernels

MANY OTHERS

- 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

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

- Sigmoid

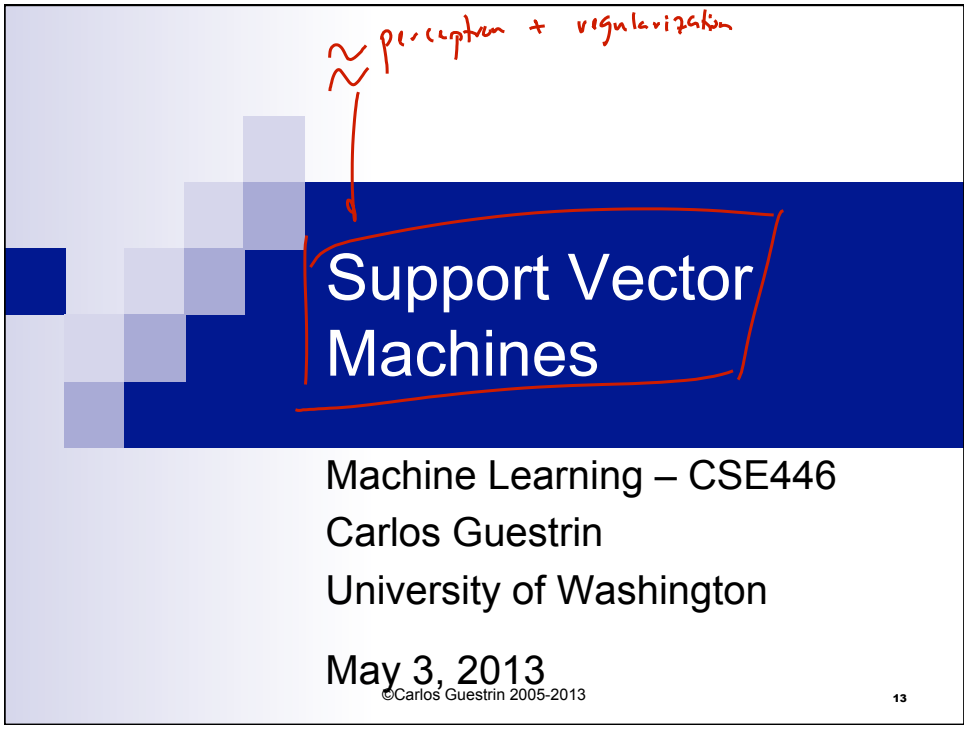
$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

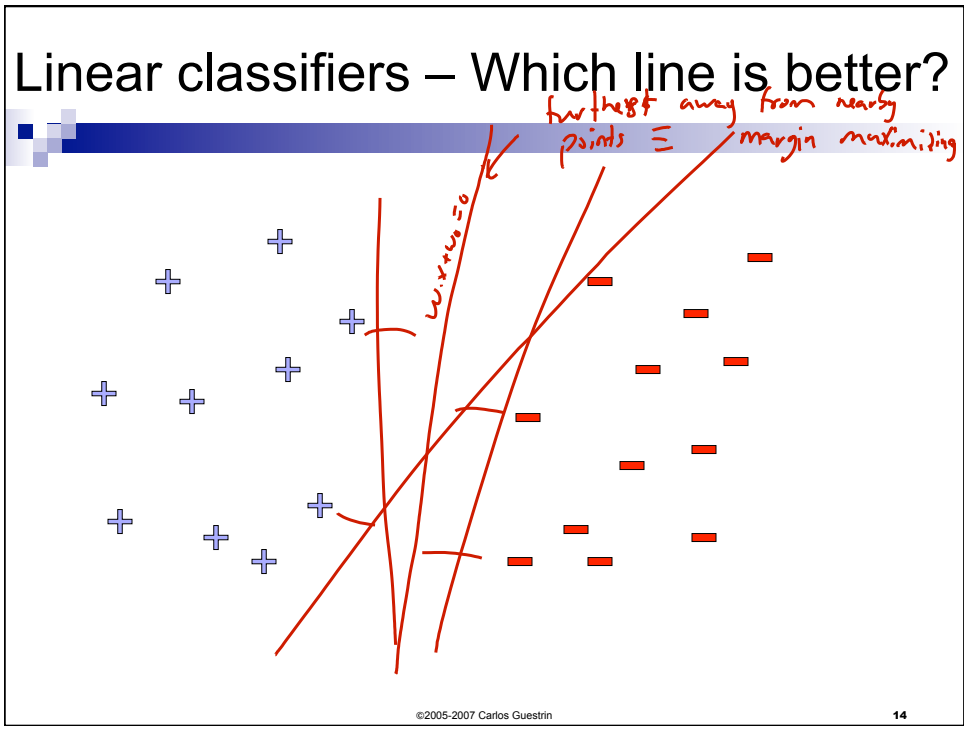
u_1, u_2 ... attributes are real
$\phi(\cdot)$ projects them into high dim space
$\phi(u) \cdot \phi(v)$
equivalent to $\phi(u) \cdot \phi(v)$ where $\phi(u)$ is infinite dimensional
← dot prod. in infinite space

0 strings
0 graph
1 :

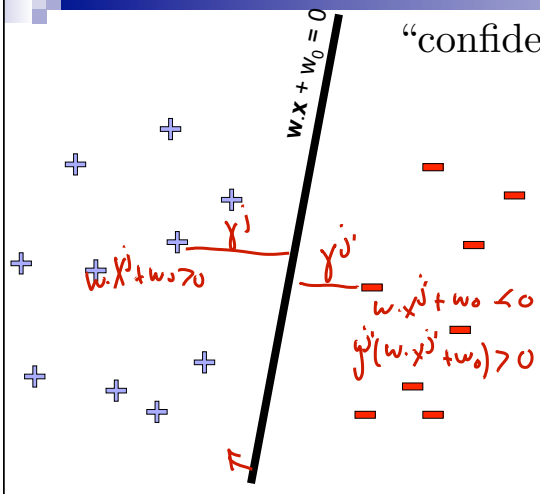
What you need to know

- 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


~ perceptron + regularization
Support Vector Machines
Machine Learning – CSE446
Carlos Guestrin
University of Washington
May 3, 2013
©Carlos Guestrin 2005-2013
13


Linear classifiers – Which line is better?
furthest away from nearby points = margin maximizing
w.x+w0=0
©2005-2007 Carlos Guestrin
14

Pick the one with the largest margin!



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

$w\cdot x^j = \sum_{i=1}^{k} w_i x_i^j$

γ^j

$w\cdot x^j + w_0 > 0$

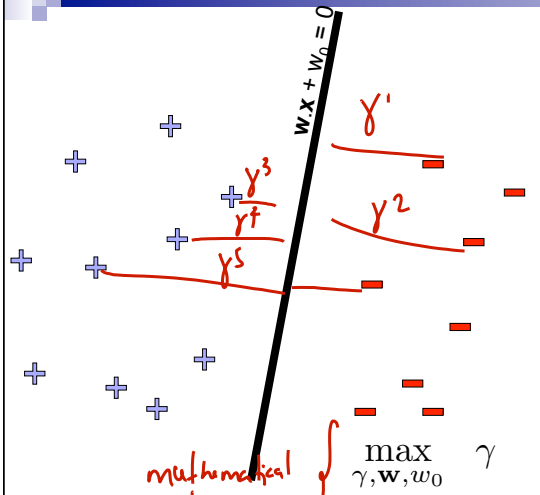
$\gamma^{j'}$

$w\cdot x^{j'} + w_0 < 0$

$y^{j'}(w\cdot x^{j'} + w_0) > 0$

Maximize the margin

maximize worst case margin:



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

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

γ^1

γ^2

γ^3

γ^4

γ^5

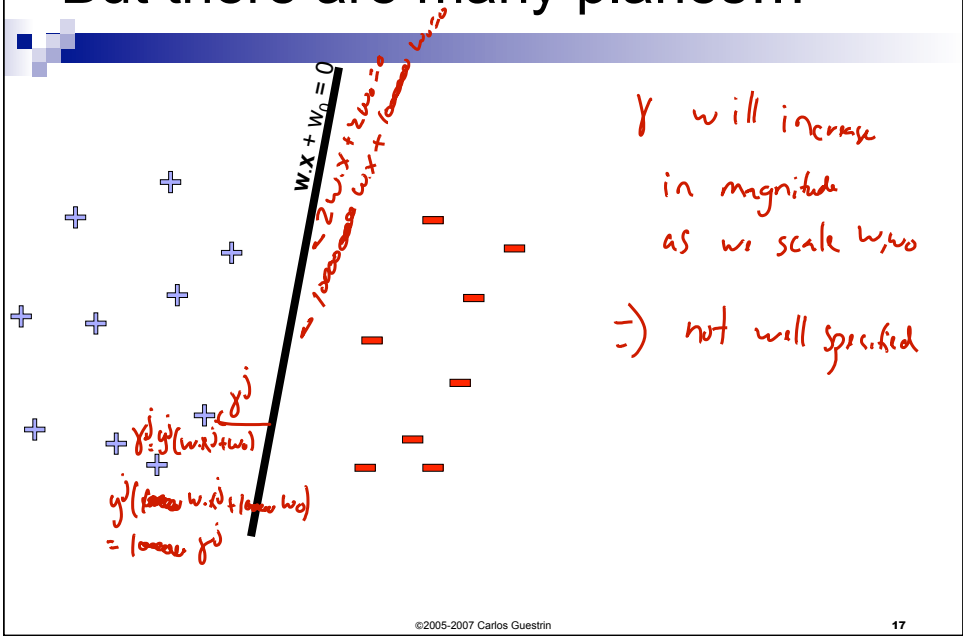
find me w, w_0 that give max margin γ for all points j

mathematical program

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

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

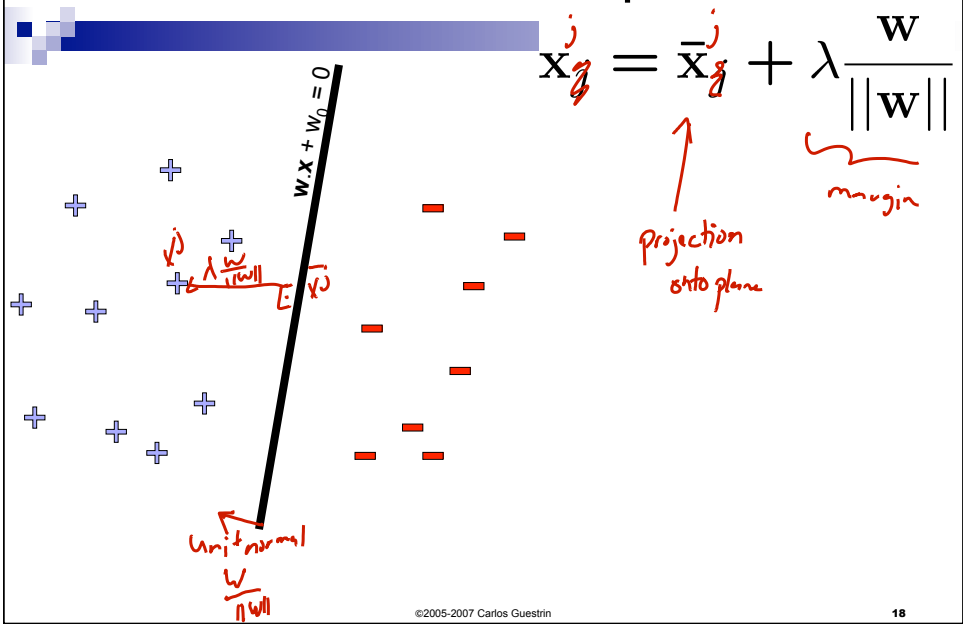
But there are many planes…



γ will increase in magnitude as we scale w, w0

:-) not well specified

Review: Normal to a plane



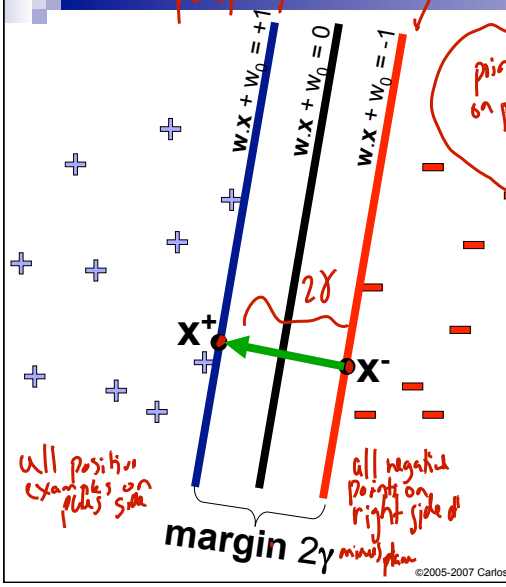
$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}$$

projection onto plane

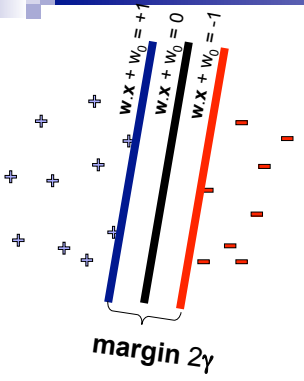
margin

A Convention: Normalized margin – Canonical hyperplanes

$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}$$



Margin maximization using canonical hyperplanes



Unnormalized problem:

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

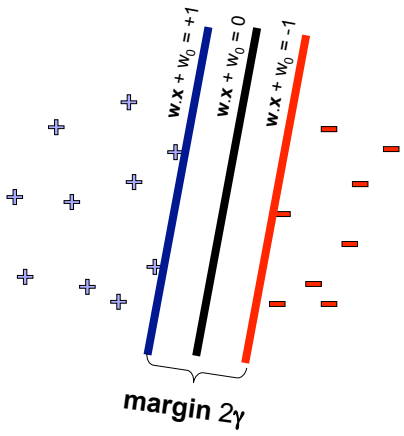
$$y^j(\mathbf{w}\cdot\mathbf{x}^j + w_0) \geq \gamma, \forall j \in \{1,\ldots,N\}$$

Normalized Problem:

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

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

Support vector machines (SVMs)

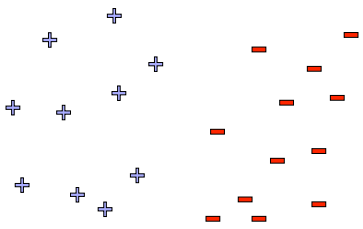


$$\min_{\mathbf{w},w_0} \quad ||w||_2^2$$

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

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

What if the data is not linearly separable?

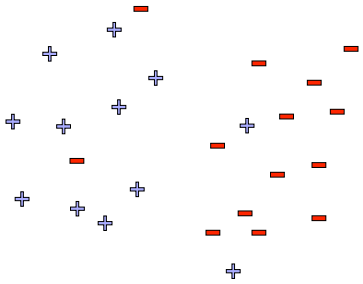


Use features of features of features of features….

What if the data is still not linearly separable?

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

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



- If data is not linearly separable, some points don't satisfy margin constraint:
- How bad is the violation?
- Tradeoff margin violation with ||**w**||:

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

- Perceptron was minimizing the hinge loss:

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

- SVMs minimizes the regularized hinge loss!!

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

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:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)}\cdot\mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- SVMs minimization:

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

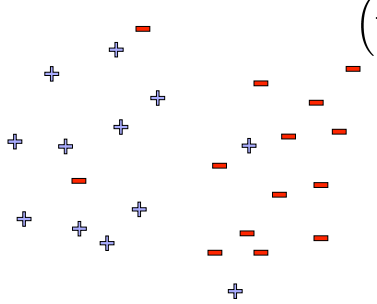
- SGD for SVMs:

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 optimize SVMs with SGD
  - Many other approaches possible

Slack variables – Hinge loss

$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad ,\forall j$$



- If margin ¸ 1, don't care
- If margin < 1, pay linear penalty

Side note: What's the difference between SVMs and logistic regression?

SVM:

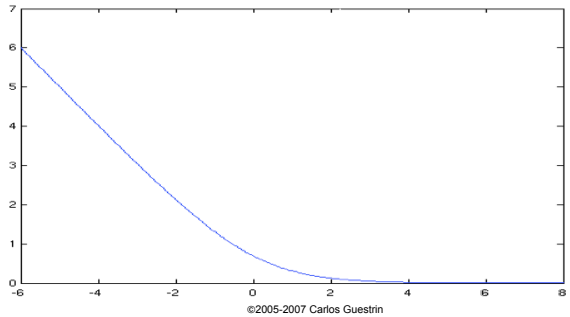
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

Logistic regression:

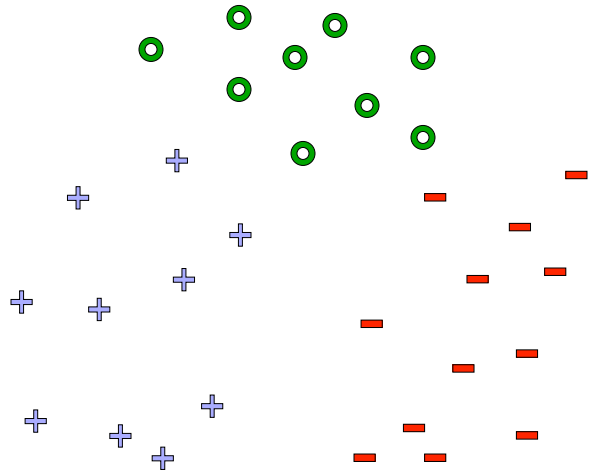
$$P(Y = 1 \mid x, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}.\mathbf{x}+b)}}$$

Log loss:

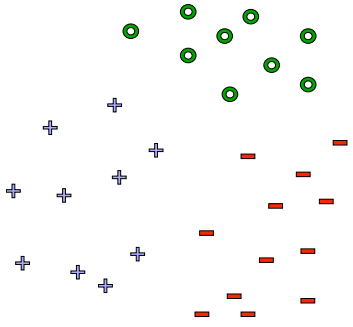
$$-\ln P(Y = 1 \mid x, \mathbf{w}) = \ln\left(1 + e^{-(\mathbf{w}.\mathbf{x}+b)}\right)$$



What about multiple classes?



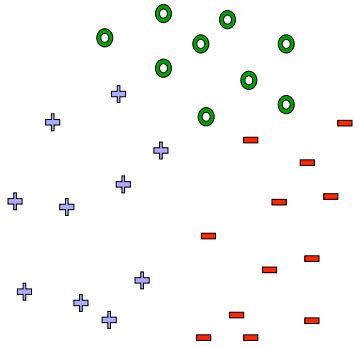
One against All



Learn 3 classifiers:

Learn 1 classifier: Multiclass SVM

Simultaneously learn 3 sets of weights



$$\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1, \ \forall y' \neq y_j, \ \forall j$$

Learn 1 classifier: Multiclass SVM

$$\text{minimize}_{\mathbf{w},b} \quad \sum_y \mathbf{w}^{(y)}.\mathbf{w}^{(y)} + C \sum_j \xi_j$$

$$\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1 - \xi_j, \ \forall y' \neq y_j, \ \forall j$$

$$\xi_j \geq 0, \ \forall j$$