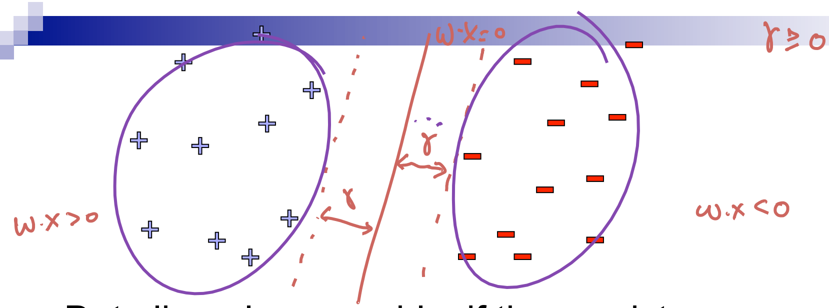
Kernels

Machine Learning – CSE546
Carlos Guestrin
University of Washington

October 28, 2014

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 $w^* x$

$\forall t$ if $y^t = +1$ $w^* x^t \geq \gamma$ | Linear Sep: $\forall t$ $y^t w^* x^t \geq \gamma$

$y^t = -1$ $w^* x^t \leq -\gamma$

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $\left[(x^1, y^1) \ldots (x^T, y^T)\right]$
  - Each feature vector has bounded norm: $\forall t \quad \|x^t\| \le R$
  - If dataset is linearly separable:

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

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

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

Doesn't depend on T

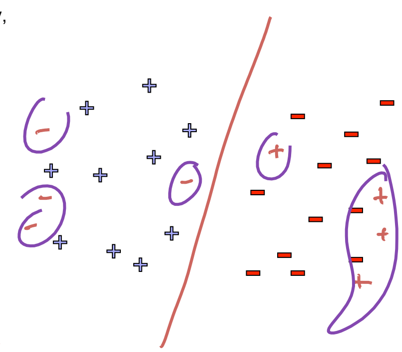
Constant number of mistakes

Independent of data size

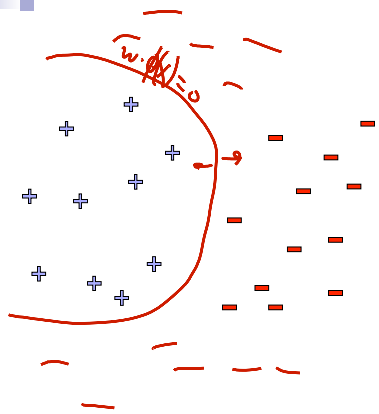
Beyond Linearly Separable Case

- 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)

(degree of non-linearity



What if the data is not linearly separable?



Use features of features of features of features….

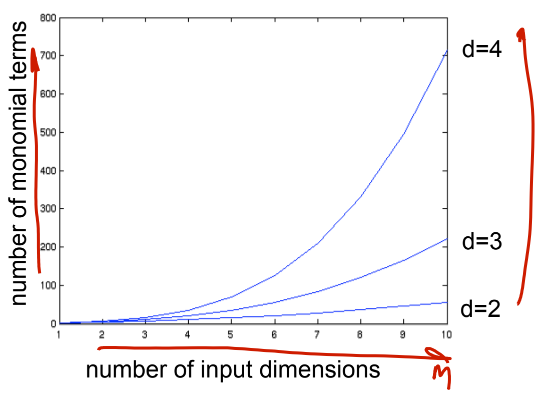
$$\Phi(\mathbf{x}) : R^m \mapsto F$$

$$\phi(x) = \begin{pmatrix} x \\ x^2 \\ x^3 \\ e^x \\ e^{\log x} \\ \vdots \end{pmatrix}$$

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: deal with high dim very efficiently

a lot

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

instead x, use high dim features $\phi(x)$

Perceptron Revisited

$x \cdot x^{(j)} = \sum_{i=1}^{m} x_i x_i^{(j)}$

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

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

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

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

- Prediction rule now:

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

- When using high dimensional features:

$$\text{Sign}\left(\phi(x) \cdot w^{(t)}\right) = \text{Sign}\left(\sum_{j \in M(t)} y^{(j)} \underbrace{\phi(x) \cdot \phi(x^{(j)})}_{\text{prediction only depends on dot products}}\right)$$

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

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

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

proof by single step of induction
for poly of degree exactly d

$$\underbrace{\phi(u) \cdot \phi(v) = (u \cdot v)^d}_{\text{kernel trick !!}} = K(u, v)$$

Finally the Kernel Trick!!! (Kernelized Perceptron)

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

  ↳ keep list of all mistakes ever made

- Kernelized Perceptron prediction for **x**:

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

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

Polynomial kernels

- All monomials of degree d in O(log 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:

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

proof by "induction"

For polynomials of degree d: $\quad \phi(u) \cdot \phi(v) = k(u,v) = (u \cdot v + 1)^d$!!

wow

Common kernels

- 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

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

Radial basis functions
RBF kernel
↓
projecting into infinite dim space.

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 Tuesday (nearest neighbors/ decision trees)…
- Only 80mins, so arrive early and settle down quickly, we'll start and end on time
- "Open book"
  - Textbook, Books, Course notes, Personal notes
- 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

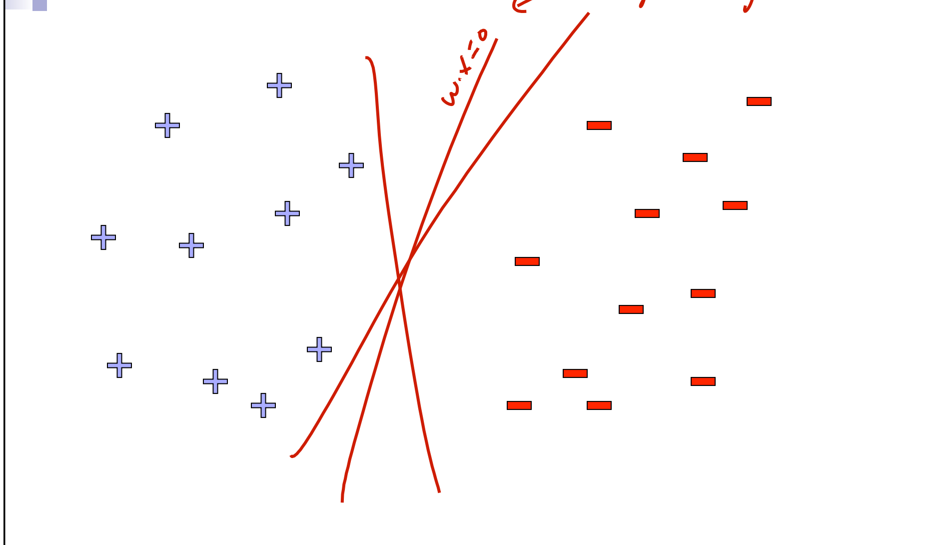
Support Vector Machines

Machine Learning – CSE546
Carlos Guestrin
University of Washington

October 28, 2014

Linear classifiers – Which line is better?

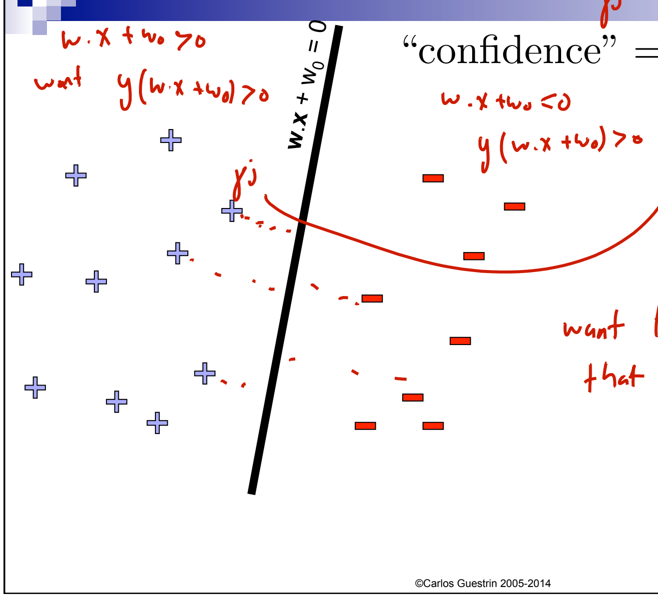
"largest" margin



Pick the one with the largest margin!

$y \in \{+1, -1\}$

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



want to pick w that maximizes γj

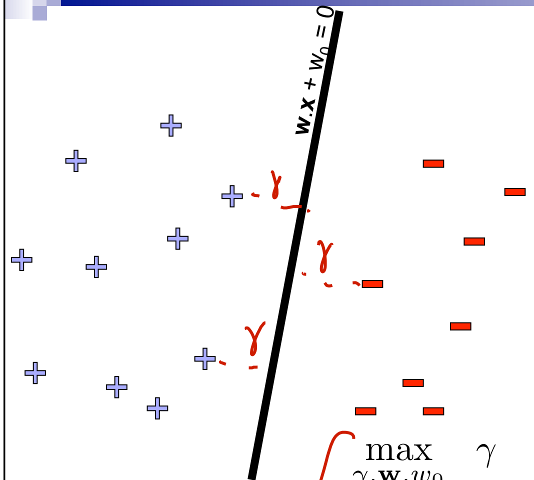
Maximize the margin

SVMs.

max worst-case margin

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

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



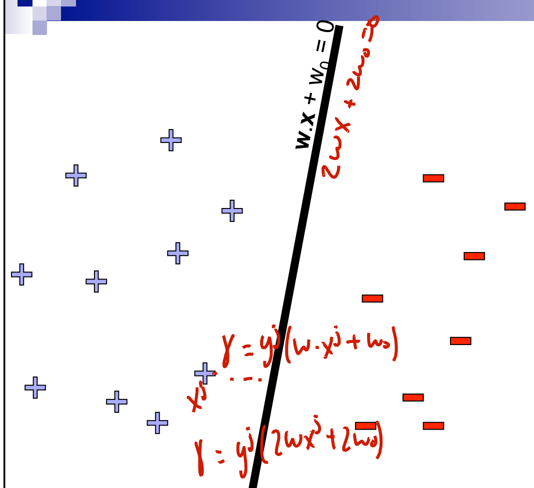
equivalently

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

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

But there are many planes…

γ will increase unboundedly



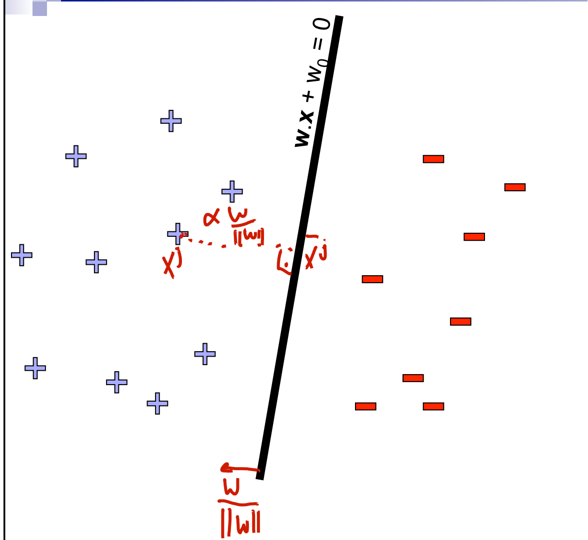
$\gamma = y^j(w \cdot x^j + w_0)$

$\gamma = y^j(2wx^j + 2w_0)$

Review: Normal to a plane

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

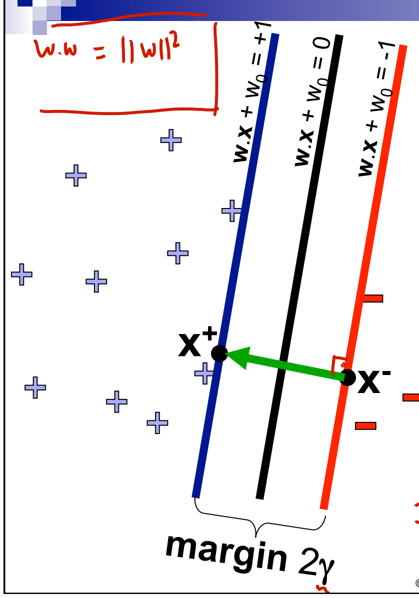
↑ projection   ↑ normal direction



A Convention: Normalized margin – Canonical hyperplanes

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

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



From canonical plane:

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

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

plug in

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

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

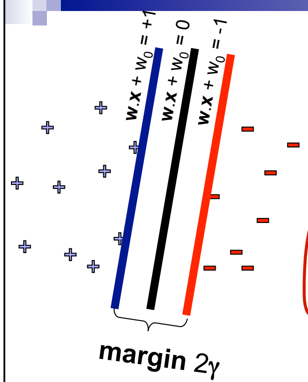
$\underbrace{w \cdot x^- + w_0}_{-1} + 2\gamma ||w|| = 1$

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

$\Rightarrow$ maximizing margin $\gamma \Leftrightarrow$ maximizing $\frac{1}{||w||} \Leftrightarrow$ minimizing $||w||$

regularization

Margin maximization using canonical hyperplanes



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:

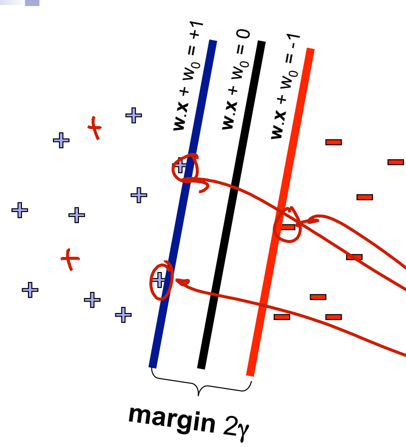
$\max \gamma \equiv \max \frac{1}{||w||} \equiv \min ||w|| \equiv \min ||w||^2$

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

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

Support vector machine

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

What if I forget all data but the support vectors $\Rightarrow$ same solution!