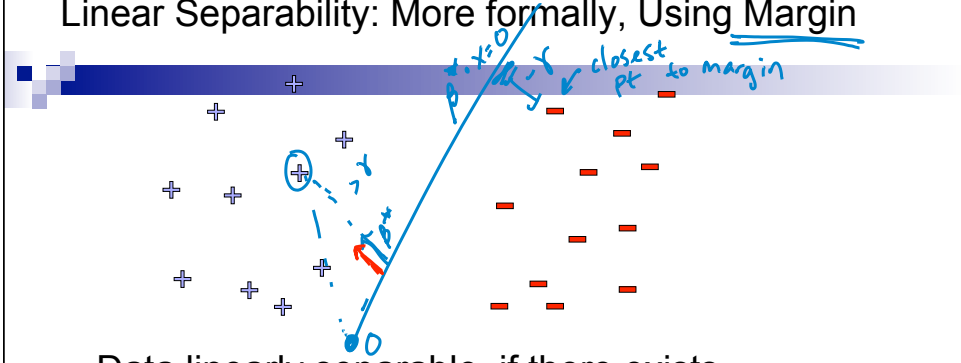
Module 5: Classification

Kernelized Perceptron: Kernels, again!

STAT/BIOSTAT 527, University of Washington
Emily Fox
June 4th, 2013

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector $\exists\, \beta^*$ $\|\beta^*\| = 1$
  - a margin $\gamma > 0$ ← margin of separability
- Such that all pts are γ far away or more from $\beta^* \cdot x = 0$

$\forall t$ if $y_t = +1$ $\beta^* \cdot x_t > \gamma$

$y_t = -1$ $\beta^* \cdot x_t < -\gamma$ $\Rightarrow$ $\boxed{y_t(\beta^* \cdot x_t) > \gamma}$

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $(x_1, y_1), \ldots, (x_n, y_n)$
  - Each covariate vector has bounded norm: $\forall t \quad \|x_t\| \le R$
  - If dataset is linearly separable: $\exists \beta^* \quad \|\beta^*\| = 1 \quad \text{s.t.} \quad y_t(\beta^* \cdot x_t) \ge \gamma \quad \forall t$ for some $\gamma > 0$
- Then the number of mistakes made by the online perceptron on this sequence is bounded by $\left(\frac{R}{\gamma}\right)^2$

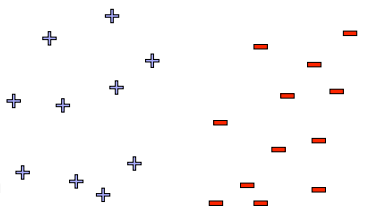
If mistake $\beta^{(k+1)} \leftarrow \beta^{(k)} + y_t x_t$; else $\beta^{(k+1)} \leftarrow \beta^{(k)}$

examples need not be iid nor random

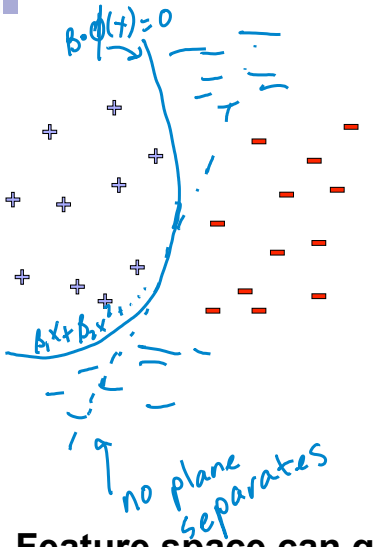
crazy! constant... doesn't depend on T or dim X

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!
    - 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)



What if the data are not linearly separable?



Use features of features of features….

$$\phi(x) : \mathbb{R}^d \rightarrow F$$

$\phi(x) =$ could even be ∞-dim

$\begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x e^{\sin x} \\ e^{-\frac{x^2}{\sigma^2}} \\ \vdots \end{pmatrix}$, $\begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_2^2 \\ \vdots \end{pmatrix}$ space here

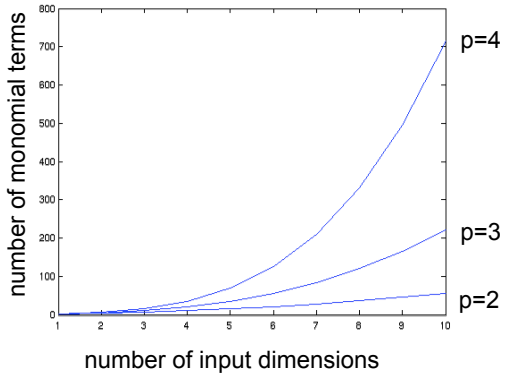
Feature space can get really large really quickly!

Higher Order Polynomials

as basis expansion

$$\#\text{ terms} = \binom{p+d-1}{p} = \frac{(p+d-1)!}{p!(d-1)!}$$

d – covariate dimension
p – degree of polynomial



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

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

Perceptron Revisited

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

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

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

$$\beta^{(t)} = \sum_{j \in M^{(t)}} y_j x_j \quad \text{(assuming } \beta^{(0)} = 0\text{)}$$

- Prediction rule now:

$$\text{sign}\left(\beta^{(t)} \cdot x\right) = \text{sign}\left(\left(\sum_{j \in M^{(t)}} y_j x_j\right) \cdot x\right) = \text{sign}\left(\sum_{j \in M^{(t)}} y_j (x_j \cdot x)\right)$$

classification only depends on inner prod of x

- When using high dimensional features:

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

when can I compute this efficiently?

Dot-Product of Polynomials

$\phi(u) \cdot \phi(v) =$ polynomials of degree exactly *p*

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

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

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

pf by induction:

for degree p: $\phi(u) \cdot \phi(v) = (u \cdot v)^p$ ← can compute very efficiently

"kernel trick"

The Kernel Trick Again: Kernelized Perceptron

- Every time you make a mistake, remember (x_t, y_t)

Keep track of indices $M^{(t)}$ where we make mistakes up to t

- Kernelized perceptron prediction for **x**:

$$\text{sign}(\beta^{(t)} \cdot \phi(x)) = \sum_{i \in M^{(t)}} y_i (\phi(x_i) \cdot \phi(x)) = \sum_{i \in M^{(t)}} y_i k(x_i, x)$$

Kernel fcn $k(u,v) = \phi(u) \cdot \phi(v)$

Going Infinite…

Change of notation: $h_j(x) \to \phi_j(x)$

- Nonparametric Gaussian regression: Would like to let the number of "features" (basis fcns) $M \to \infty$
- *Prior:* $p(\beta \mid 0, \alpha^{-1} I_M)$ (dim M)
- ~~*Predictions:*~~ $f = \Phi\beta$ (Dist. on f; $(f(x_1), \dots, f(x_n))^T$; $(\beta_1, \dots, \beta_n)^T$; linear comb. of Gaussians β_j $\Rightarrow$ f Gauss.)

$E[f] = \Phi E[\beta] = 0$

$\text{cov}(f) = \Phi E[\beta\beta^T]\Phi^T = \frac{1}{\alpha}\Phi\Phi^T$

$p(f) = N(f \mid 0, \alpha^{-1}\Phi\Phi^T)$

$\begin{bmatrix} \\ \\ \end{bmatrix}_{n\times 1} = \begin{bmatrix} \\ \\ \end{bmatrix}_{n\times M} \begin{bmatrix} \\ \\ \end{bmatrix}_{M\times 1}$ $[\phi_1(x_i) \dots \phi_M(x_i)]$

- Gaussian process models replace explicit basis function representation with a direct specification in terms of a ***positive definite kernel function***

Mercer Kernel Functions

- Predictions are of the form

$$p(f) = N(f \mid 0, \alpha^{-1}\Phi\Phi^T)$$

where the ***Gram matrix*** *K* is defined as

$$K_{ij} = k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

- *K* is a ***Mercer kernel*** if the Gram matrix is positive definite for any *n* and any $x_1, \ldots, x_n$

Polynomial kernels

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

$\phi(u) \cdot \phi(v) = (u \cdot v)^p =$ polynomials of degree exactly *p*

- How about all monomials of degree up to *p*?
  - Solution 0: $\phi(u) \cdot \phi(v) = \sum_{i=0}^{p} \binom{p}{i} (u \cdot v)^i$
  - Better solution: $(u \cdot v)^1 + (u \cdot v)^2 + (v \cdot u)^1 + (u \cdot v)^0 = (u \cdot v + 1)^2$

For degree p: $\phi(u) \cdot \phi(v) = K(u, v) = (u \cdot v + 1)^p$

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

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

- Sigmoid

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

eq. to $\phi(u) \cdot \phi(v)$

dim of $\phi(u)$ ∞

← inner prod in an ∞-dim space

MANY, MANY MORE...

- strings
- graph

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

Module 5: Classification

≈ perceptron + regularization

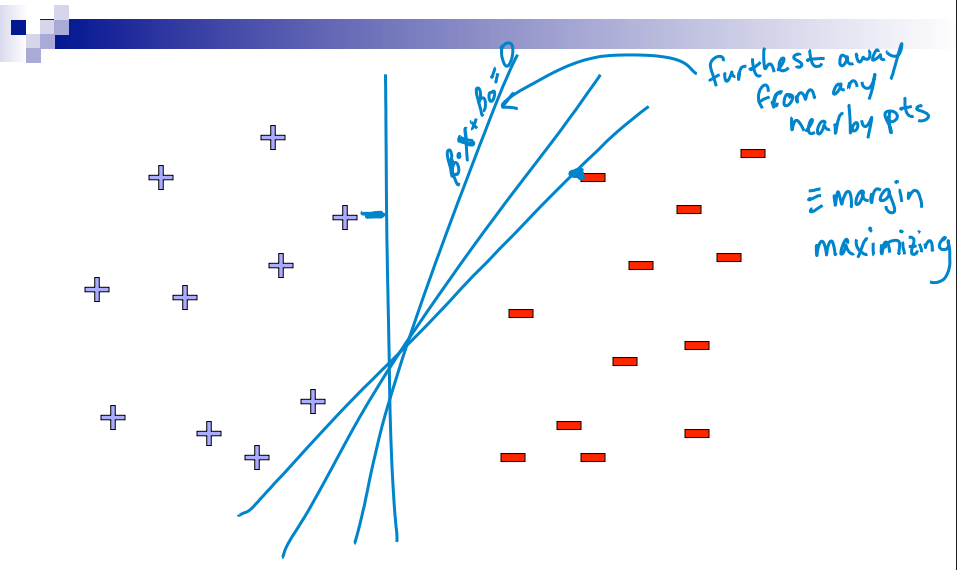
Support Vector Machines

STAT/BIOSTAT 527, University of Washington

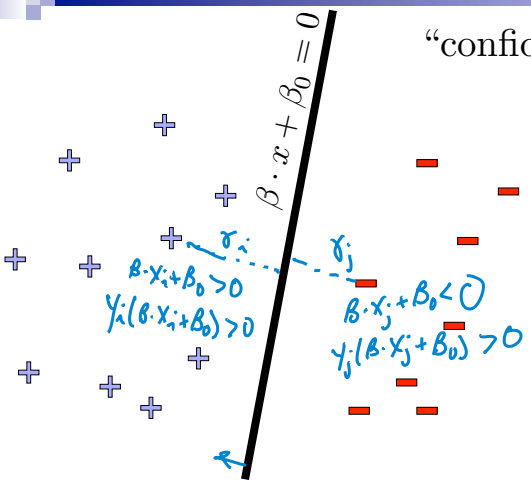
Emily Fox

June 4th, 2013

Linear classifiers – Which line is better?



Pick the one with the largest margin!

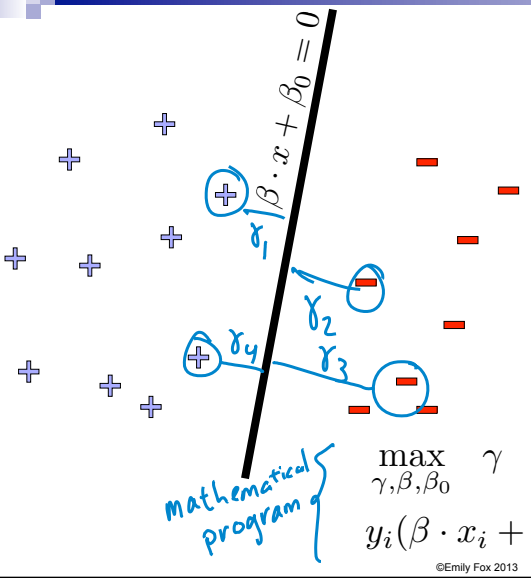


"confidence" $= y_i(\beta \cdot x_i + \beta_0)$

pos. if no mistake

Maximize the margin

maximize worst case margin



$\max_{\beta,\beta_0} \min_i \gamma_i$

$y_i(\beta \cdot x_i + \beta_0) = \gamma_i$

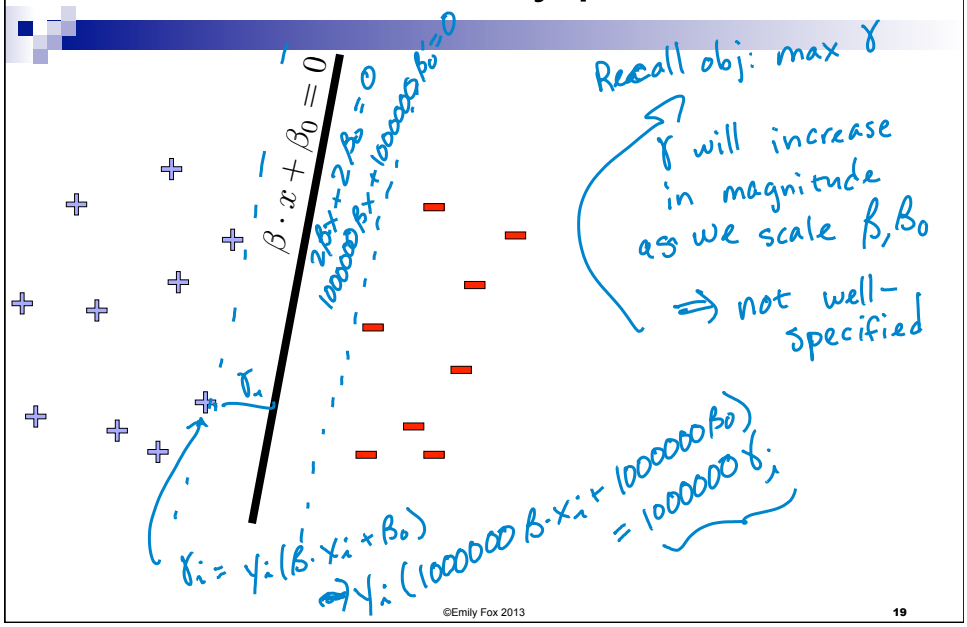
Find β, β_0 that gives max margin γ $\forall$ pts i

mathematical program

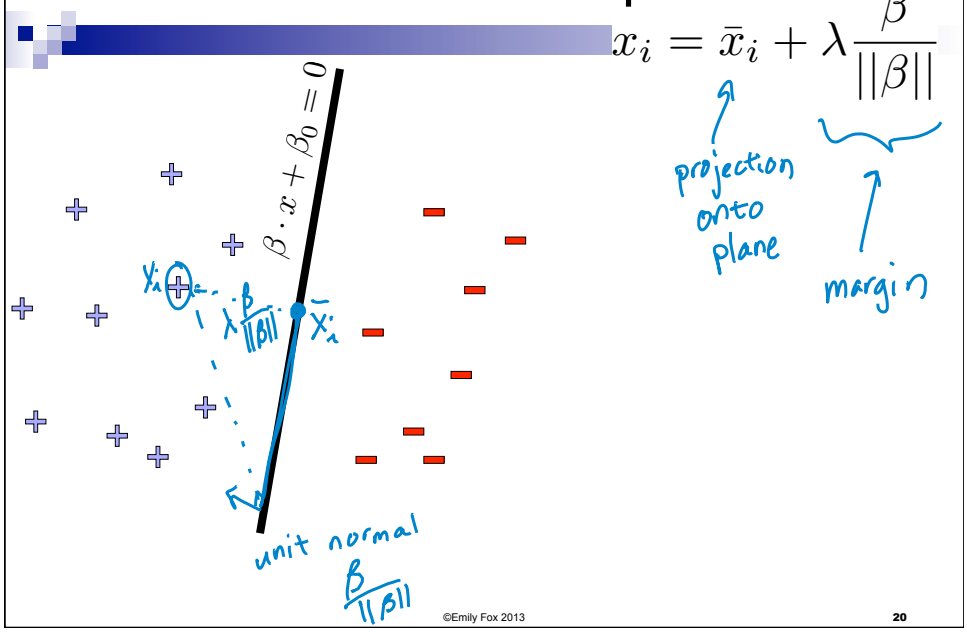
$$\max_{\gamma,\beta,\beta_0} \gamma$$

$$y_i(\beta \cdot x_i + \beta_0) \geq \gamma, \forall i \in \{1, \ldots, n\}$$

But there are many planes…

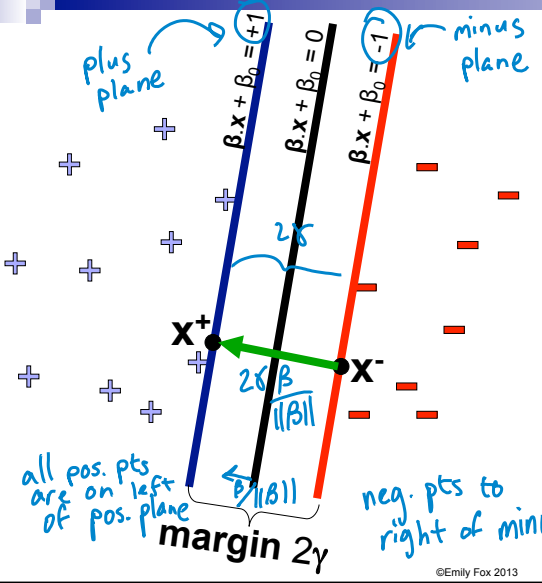


Review: Normal to a plane



A Convention: Normalized margin – Canonical hyperplanes

$$x_i = \bar{x}_i + \lambda \frac{\beta}{||\beta||}$$

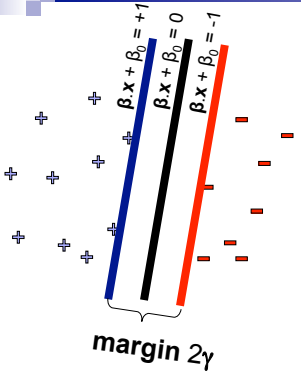


pts on planes:
$\beta \cdot x^+ + \beta_0 = +1$
$\beta \cdot x^- + \beta_0 = -1$
$x^+ = x^- + 2\gamma \frac{\beta}{||\beta||}$

$\beta \cdot (x^- + 2\gamma \frac{\beta}{||\beta||}) + \beta_0 = +1$
$\beta \cdot x^- + \beta_0 + 2\gamma \frac{\beta \cdot \beta}{||\beta||} = +1$
-1
$-1 + 2\gamma ||\beta|| = +1$
$\gamma = \frac{1}{||\beta||}$ margin inv. prop. to β

Margin maximization using canonical hyperplanes

$\max_x \frac{1}{f(x)} = \min_x f(x)$
$= \min_x f(x)^2$



Unnormalized problem: $\max_{\gamma,\beta,\beta_0} \gamma$

$$y_i(\beta \cdot x_i + \beta_0) \geq \gamma, \forall i \in \{1, \ldots, n\}$$

Normalized Problem: if $y_i(\beta \cdot x_i + \beta_0) \geq 1$ ← canonical hyperplanes

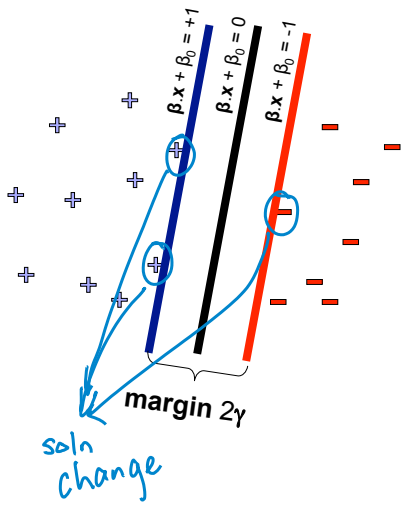
$\max \gamma = \max \frac{1}{||\beta||}$
$= \min ||\beta|| = \min ||\beta||^2$
derivatives will be easier

standard SVM problem
aka norm minimization or regularization!

$$\min_{\beta,\beta_0} ||\beta||_2^2$$

$$y_i(\beta \cdot x_i + \beta_0) \geq 1, \forall i \in \{1, \ldots, n\}$$

Support vector machines (SVMs)



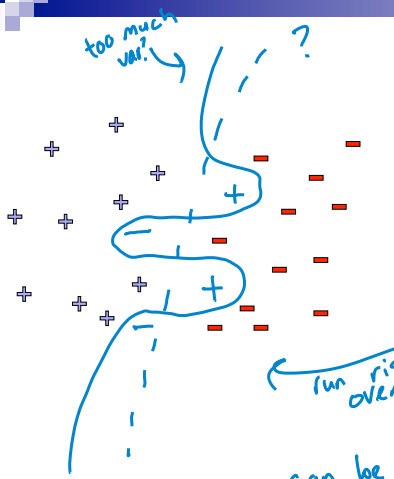
$$\min_{\beta,\beta_0} \; ||\beta||_2^2$$

$$y_i(\beta \cdot x_i + \beta_0) \geq 1, \forall i \in \{1, \ldots, n\}$$

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

* only perturbations of these pts will change the soln
(contrast w/ LDA, QDA)

What if the data are not linearly separable?



Use features of features of features….

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

SVMs popularized kernels in ML

can be done efficiently w/ kernels (just like in perceptron case)

What if the data are still not linearly separable?

$$\min_{\beta,\beta_0} \; ||\beta||_2^2$$

$$y_i(\beta \cdot x_i + \beta_0) \geq 1, \forall i \in \{1, \ldots, n\}$$

violates

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

$\exists i$ s.t. $y_i(\beta \cdot x_i + \beta_0) < 1$

or $1 - y_i(\beta \cdot x_i + \beta_0) > 0$ like hinge loss

- How bad is the violation?

constraint violation $= \begin{cases} 0 & \text{if } 1 - y_i(\beta \cdot x_i + \beta_0) < 0 \\ 1 - y_i(\beta \cdot x_i + \beta_0) & \text{otherwise} \end{cases}$

- Tradeoff margin violation with $||\boldsymbol{\beta}||$:

$$\min_{\beta,\beta_0} \; ||\beta||_2^2 + C \sum_{i=1} (1 - y_i(\beta \cdot x_i + \beta_0))_+$$

SVM objective

Now support vectors are
- on margin
or
- misclassified pts

$(1 - y_i(\beta \cdot x_i + \beta_0))_+$

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

- Perceptron was minimizing the hinge loss:

$$\sum_{i=1}^{n} (-y_i(\beta \cdot x_i + \beta_0))_+$$

both hinge losses

- SVMs minimizes the regularized hinge loss!!

$$||\beta||_2^2 + C \sum_{i=1}^{n} (1 - y_i(\beta \cdot x_i + \beta_0))_+$$

reg. — measure of fit — just convention

C=0 → β=0 → ignore data | C=∞ → want to separate data

Stochastic Gradient Descent for SVMs

- Perceptron minimization:

$$\sum_{i=1}^{n} (-y_i(\beta \cdot x_i + \beta_0))_+$$

- SGD for Perceptron:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \mathbb{I}\left[y_t(\beta^{(t)} \cdot x_t) \leq 0\right] y_t x_t$$

update weights

make mistake

stepsize $\eta = 1$

- SVMs minimization:

$$||\beta||_2^2 + C \sum_{i=1}^{n} (1 - y_i(\beta \cdot x_i + \beta_0))_+$$

- SGD for SVMs:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \eta\left(C\,\mathbb{I}(1 - y_t(\beta \cdot x_t + \beta_0) > 0) \cdot y_t x_t - 2\beta^{(t)}\right)$$

need to play w/ stepsize

- kernelize like perceptron

- pick C, η ← other ways too by CV

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

SVM:

only diff is loss fcn

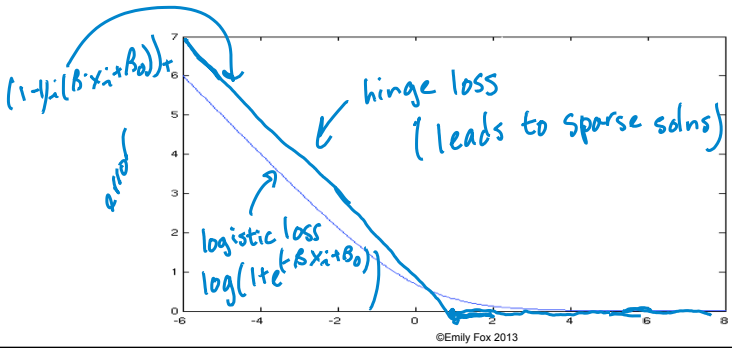
$$\min_{\beta,\beta_0} ||\beta||_2^2 + C \sum_{i=1}^{n} (1 - y_i(\beta \cdot x_i + \beta_0))_+$$

Logistic regression:

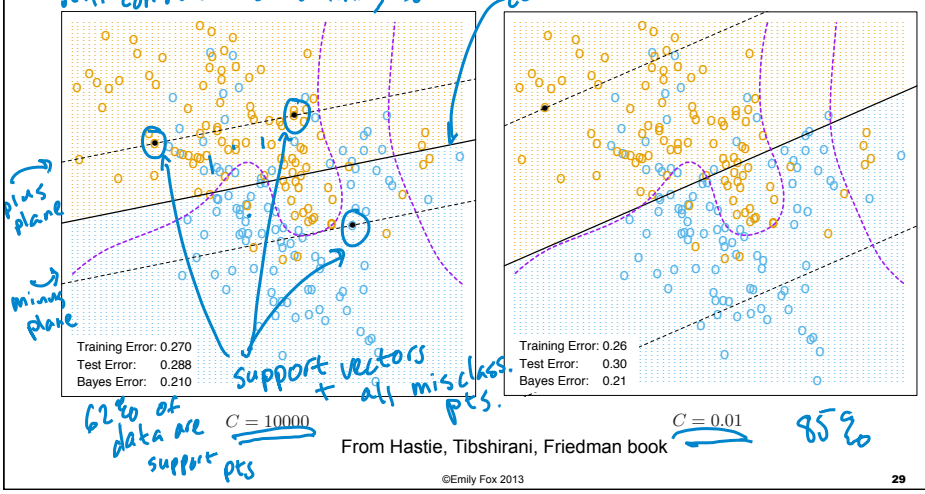
$$p(Y = 1 \mid x, \beta) = \frac{1}{1 + e^{-(\beta \cdot x + \beta_0)}}$$

Log loss:

$$\sum_i -\log p(Y = 1 \mid x_i, \beta) = \sum_i \log\left(1 + e^{-(\beta \cdot x_i + \beta_0)}\right)$$

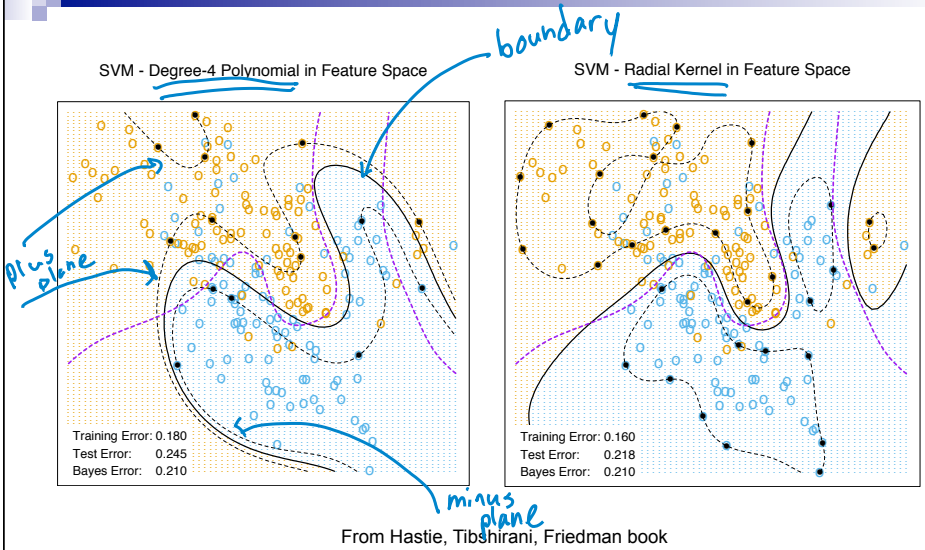


Mixture Model Example



From Hastie, Tibshirani, Friedman book

Mixture Model Example - Kernels



From Hastie, Tibshirani, Friedman book

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
- Handling multiple classes

Reading

- Hastie, Tibshirani, Friedman – 12.1-12.3