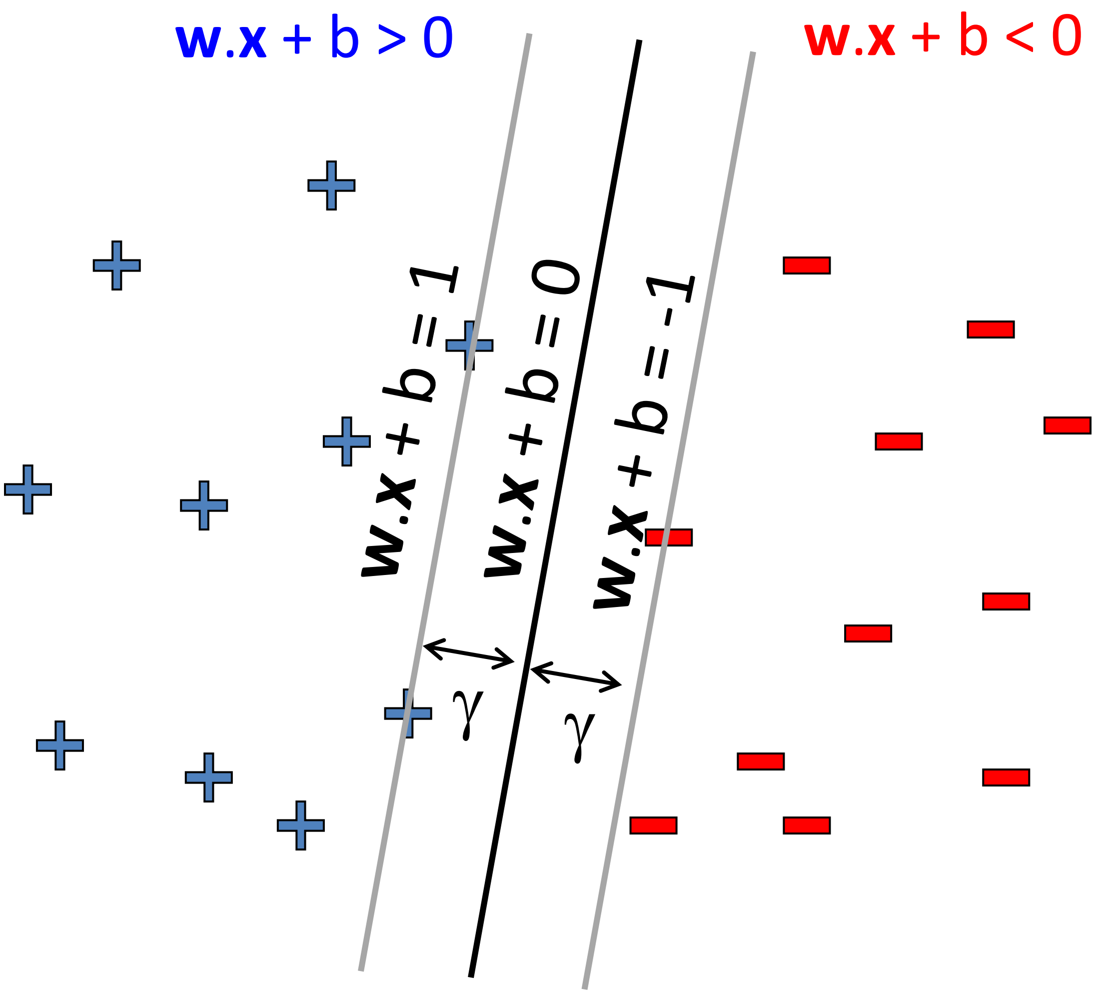
Support Vector Machines

Aarti Singh

Machine Learning 10-701/15-781
Oct 18, 2010

Support Vector Machines



Linearly separable case

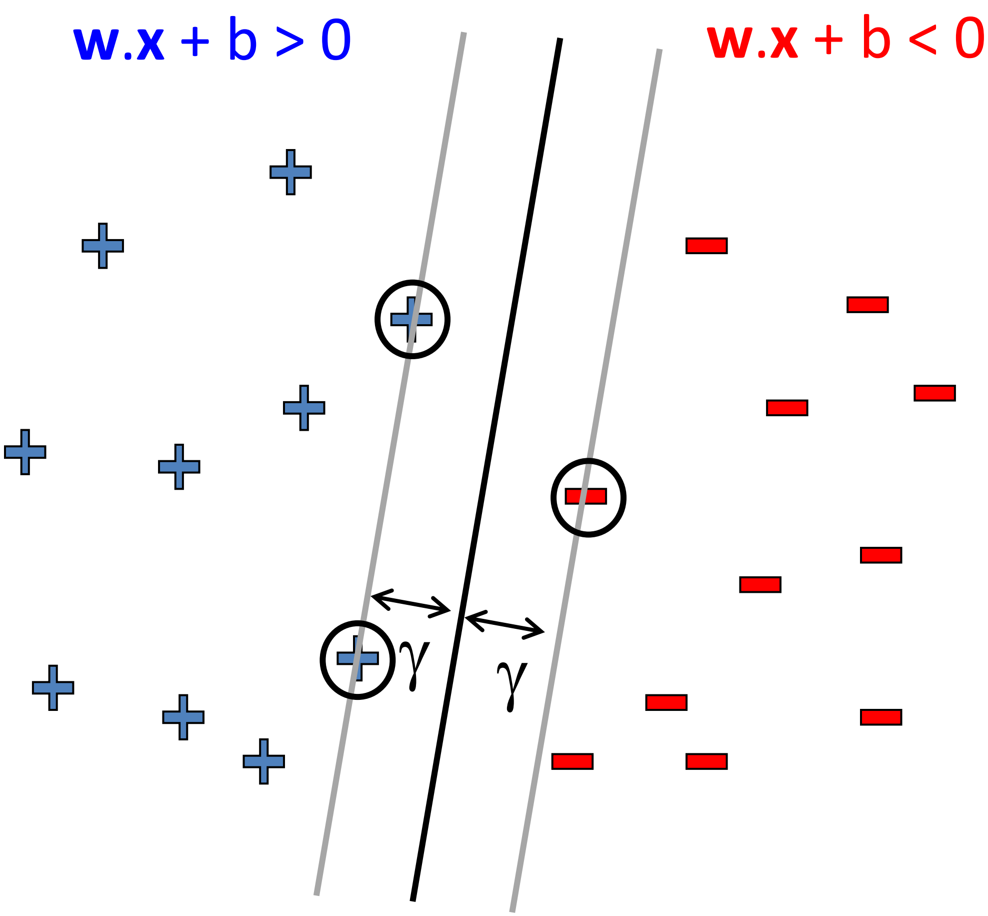
$$\min_{\mathbf{w},b} \ \mathbf{w}.\mathbf{w}$$

$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 \ \ \forall j$$

Solve efficiently by quadratic programming (QP)

- Well-studied solution algorithms

Support Vectors



Linear hyperplane defined by "support vectors"

$$j: (\mathbf{w}.\mathbf{x}_j + b)\, y_j = 1$$

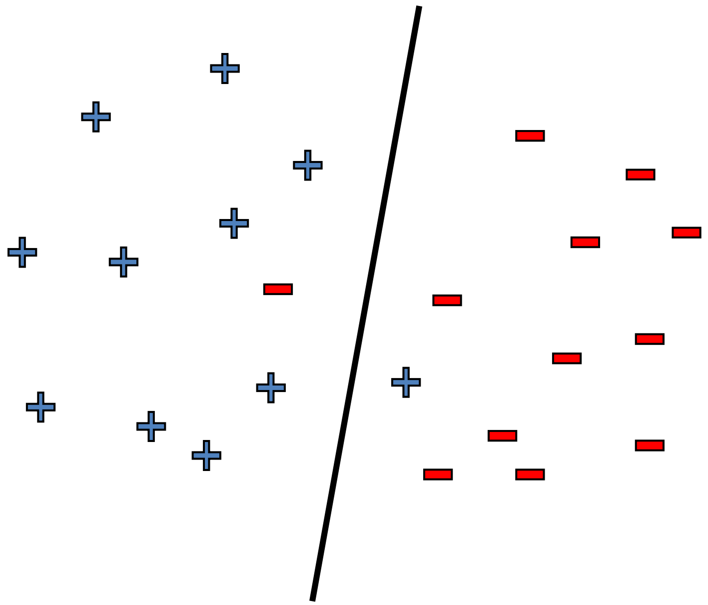
Moving other points a little doesn't effect the decision boundary

only need to store the support vectors to predict labels of new points

How many support vectors in linearly separable case?

≤ m+1

What if data is not linearly separable?



Use features of features of features of features....

$x_1^2, x_2^2, x_1x_2, \ldots, \exp(x_1)$

But run risk of overfitting!

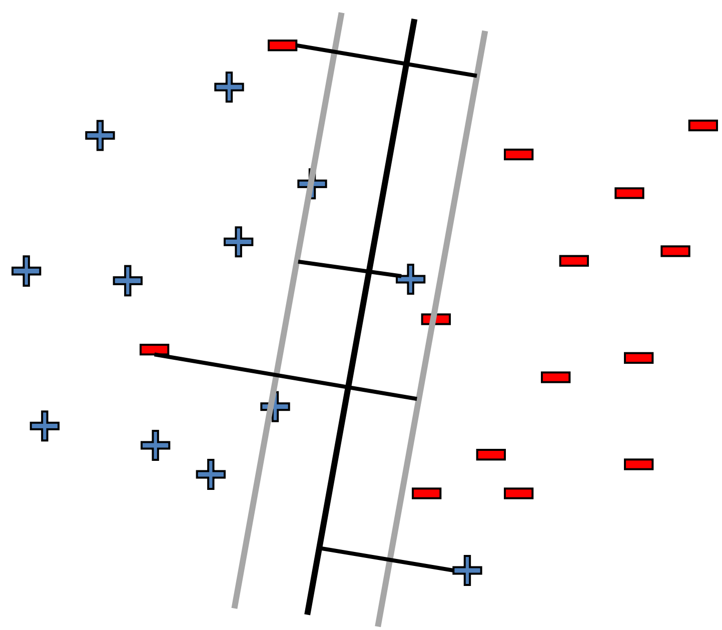
What if data is still not linearly separable?

Allow "error" in classification

$$\min_{\mathbf{w},b} \; \mathbf{w}.\mathbf{w} + C \sum_j \xi_j$$

$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 - \xi_j \quad \forall j$$

$$\xi_j \geq 0 \quad \forall j$$



Soft margin approach

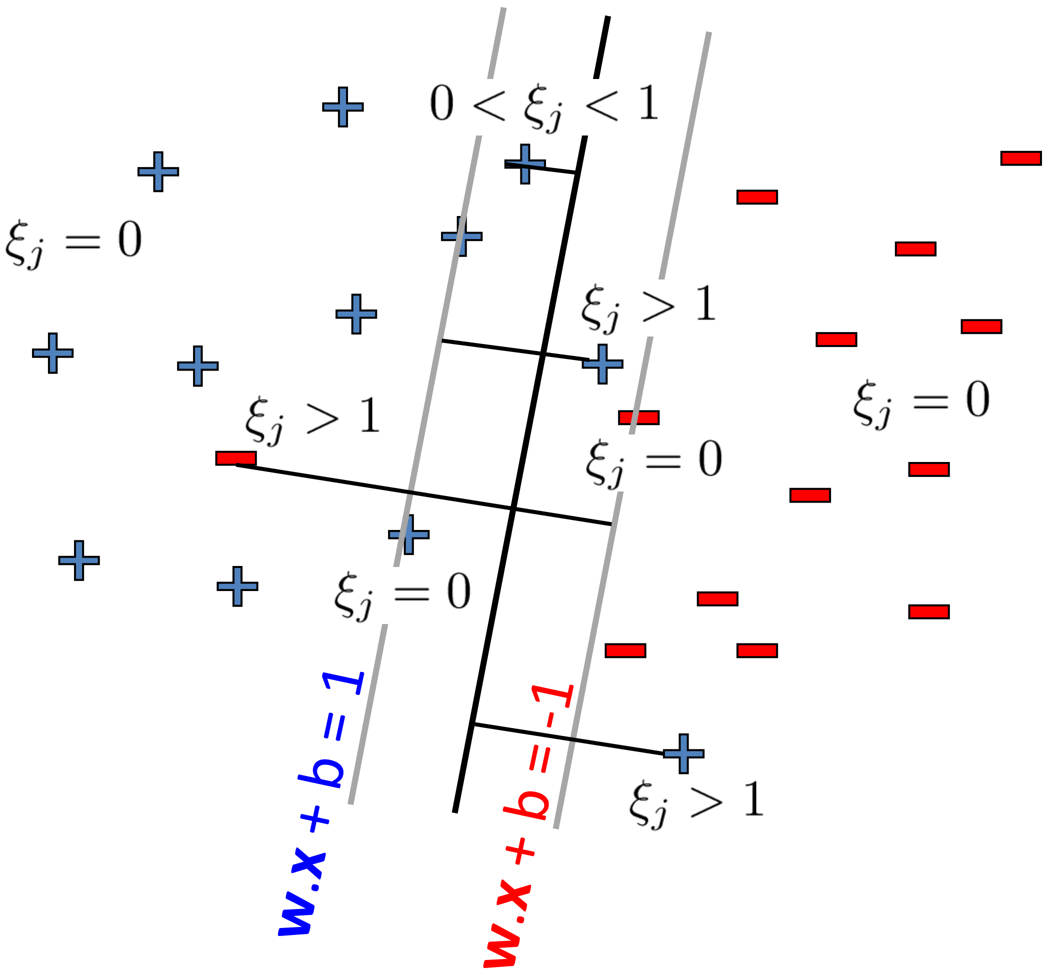
ξ_j - "slack" variables = (>1 if x_j misclassifed)

pay linear penalty if mistake

C - tradeoff parameter (chosen by cross-validation)

Still QP ☺

Soft-margin SVM



Soften the constraints:

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

$\xi_j \geq 0 \quad \forall j$

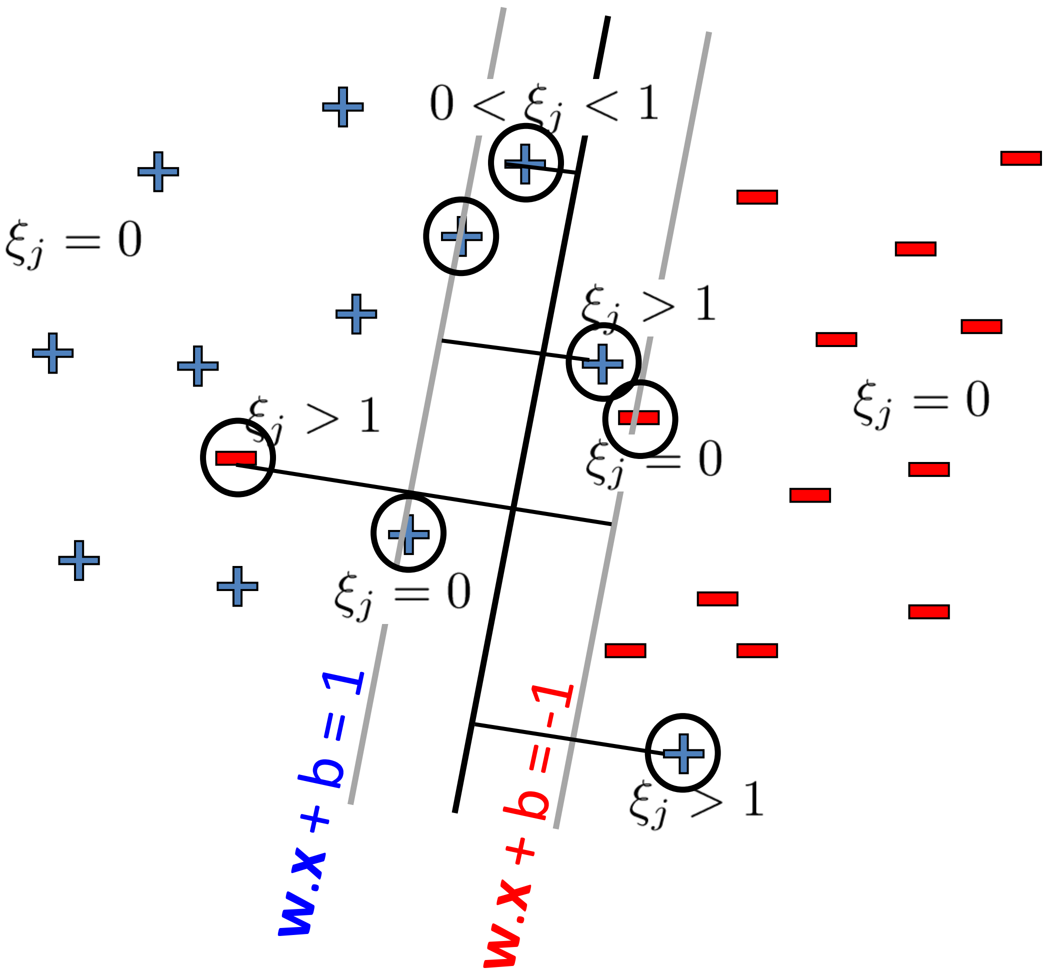
Penalty for misclassifying:

$C\, \xi_j$

How do we recover hard margin SVM?

Set $C = \infty$

Support Vectors



Soften the constraints:

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

$$\xi_j \geq 0 \quad \forall j$$

Penalty for misclassifying:

$$C\,\xi_j$$

How do we recover hard margin SVM?

Set $C = \infty$

Slack variables – Hinge loss

Complexity penalization

$$\xi_j = \text{loss}(f(x_j), y_j)$$

$$\min_{\mathbf{w},b} \; \mathbf{w}.\mathbf{w} + C \sum_j \xi_j$$

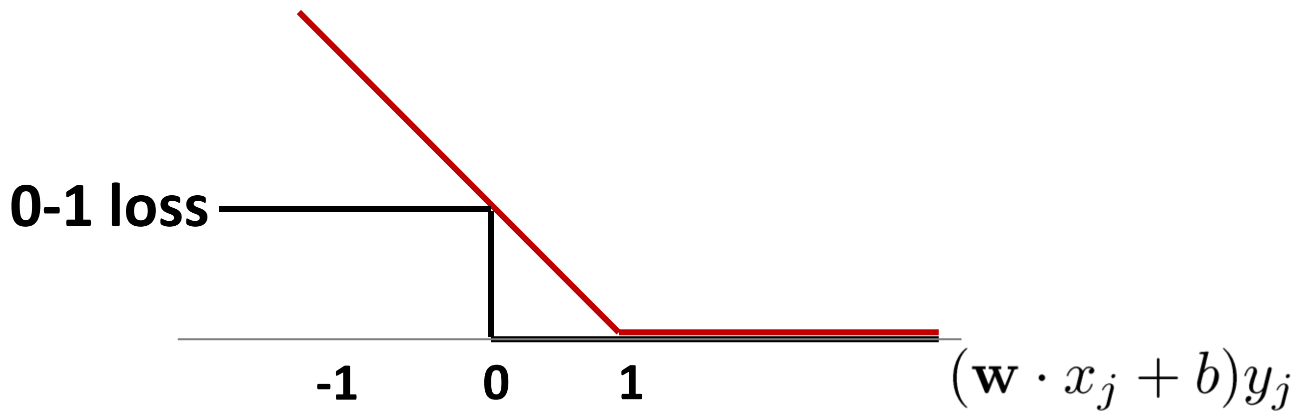
$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 - \xi_j \quad \forall j$$

$$\xi_j \geq 0 \quad \forall j$$

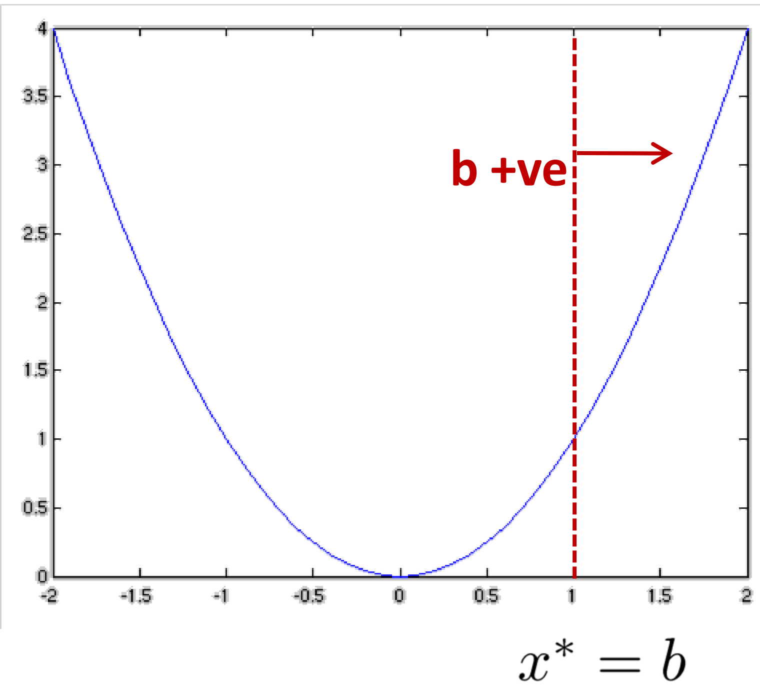
$$f(x_j) = \text{sgn}(\mathbf{w} \cdot \text{x}_\text{j} + \text{b})$$

$$\xi_j = (1 - (\mathbf{w} \cdot x_j + b)y_j))_+$$

Hinge loss



Constrained Optimization



Primal problem:

$$\min_x \; x^2$$
$$\text{s.t.} \quad x \geq b$$

Moving the constraint to objective function

Lagrangian:

$$L(x, \alpha) = x^2 - \alpha(x - b)$$
$$\text{s.t.} \quad \alpha \geq 0$$

Dual problem:

$$\max_\alpha \; d(\alpha) \rightarrow \min_x L(x, \alpha)$$
$$\text{s.t.} \quad \alpha \geq 0$$

α = 0 constraint is ineffective

α > 0 constraint is effective

Dual SVM – linearly separable case

- Primal problem: $\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w}$

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

w – weights on features

- Dual problem:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$

$$\alpha_j \geq 0, \ \forall j$$

α – weights on training pts

Dual SVM – linearly separable case

- Dual problem:

$$\max_{\alpha} \min_{\mathbf{w},b} L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$

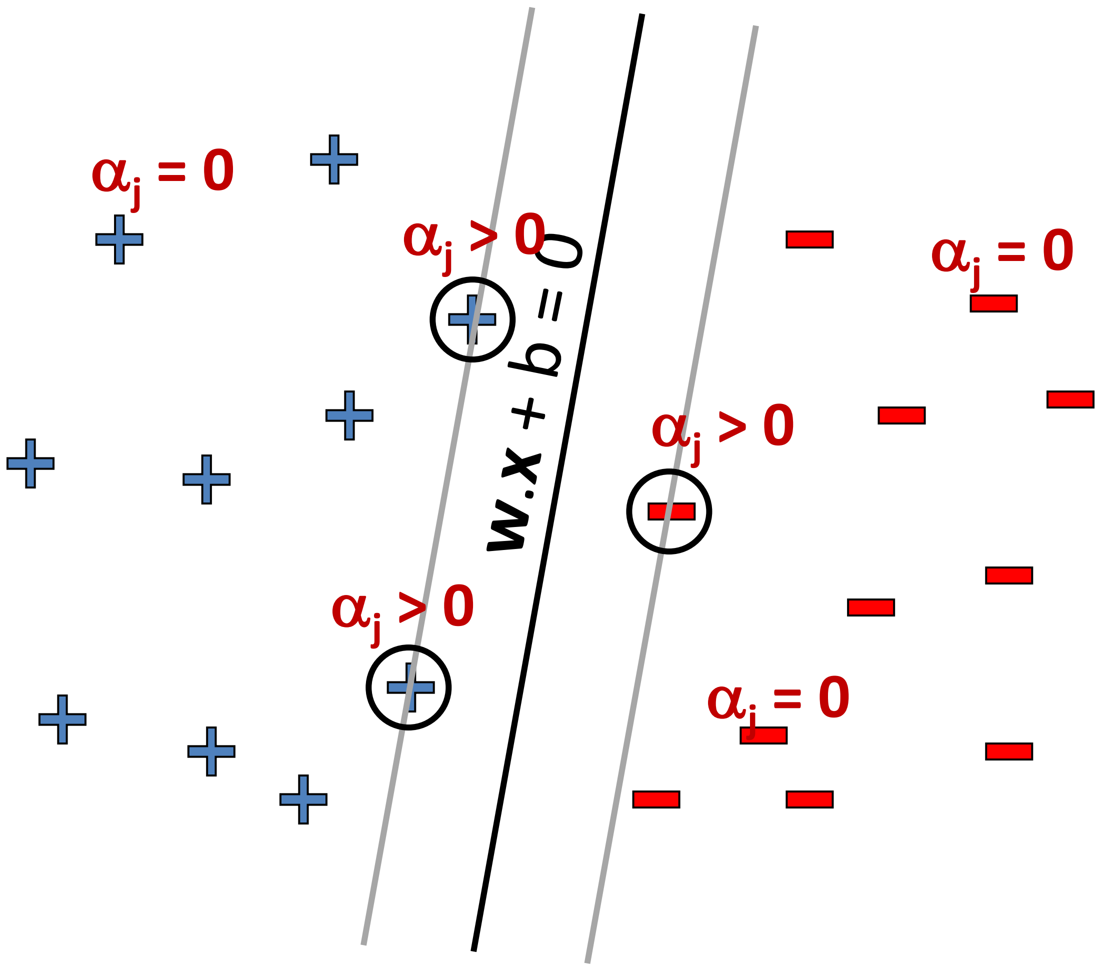
$$\alpha_j \geq 0, \ \forall j$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \qquad \Rightarrow \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = 0 \qquad \Rightarrow \sum_j \alpha_j y_j = 0$$

If we can solve for αs (dual problem), then we have a solution for **w**,b (primal problem)

Dual SVM Interpretation: Sparsity



$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

Only few α_js can be non-zero : where constraint is tight

$$(\mathbf{w}.\mathbf{x}_j + b)y_j = 1$$

Support vectors – training points j whose α_js are non-zero

Dual SVM – linearly separable case

maximize$_\alpha$ $\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \mathbf{x}_i.\mathbf{x}_j$

$\sum_i \alpha_i y_i = 0$

$\alpha_i \geq 0$

Dual problem is also QP

Solution gives α_js ⟶

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $\alpha_k > 0$

Use support vectors to compute b

Dual SVM – non-separable case

- Primal problem:

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \quad \forall j$$
$$\xi_j \geq 0, \quad \forall j$$

α_j

μ_j

Lagrange Multipliers

- Dual problem:

$$\max_{\alpha,\mu} \min_{\mathbf{w},b} L(\mathbf{w}, b, \alpha, \mu)$$
$$s.t. \alpha_j \geq 0 \quad \forall j$$
$$\mu_j \geq 0 \quad \forall j$$

Dual SVM – non-separable case

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i . \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

comes from $\dfrac{\partial L}{\partial \mu} = 0$

Intuition:

Earlier - If constraint violated, $\alpha_i \rightarrow \infty$

Now - If constraint violated, $\alpha_i \leq C$

Dual problem is also QP

Solution gives α_js

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

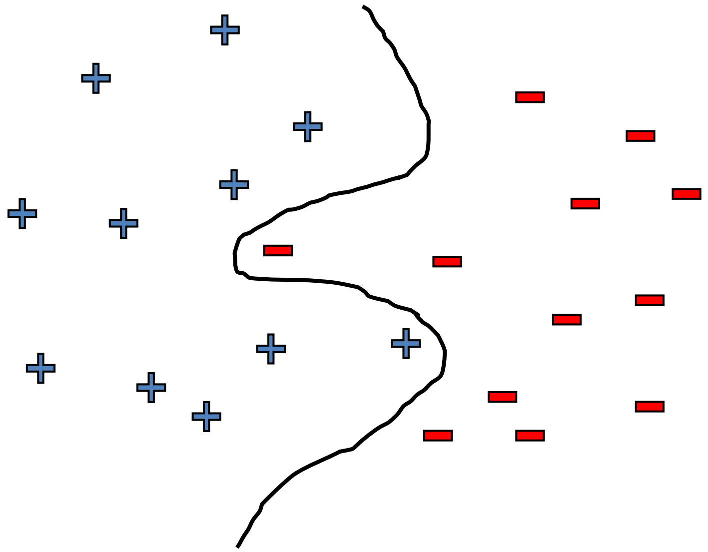
for any k where $C > \alpha_k > 0$

So why solve the dual SVM?

- There are some quadratic programming algorithms that can solve the dual faster than the primal, (specially in high dimensions m>>n)

- But, more importantly, the "**kernel trick**"!!!

What if data is not linearly separable?



Use features of features of features of features….

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, \ldots., \exp(x_1))$$

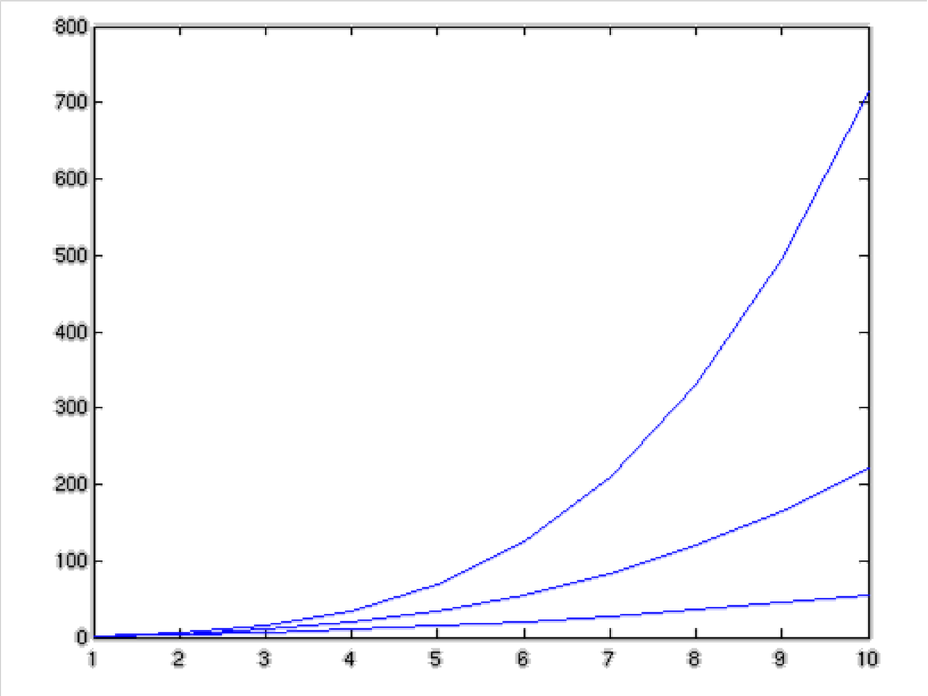
Feature space becomes really large very quickly!

Higher Order Polynomials

m – input features

d – degree of polynomial

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



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

Dual formulation only depends on dot-products, not on w!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \underbrace{\mathbf{x}_i . \mathbf{x}_j}$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

$$\Downarrow$$

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \underbrace{K(\mathbf{x}_i, \mathbf{x}_j)}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

$\Phi(\mathbf{x})$ – High-dimensional feature space, but never need it explicitly as long as we can compute the dot product fast using some Kernel K

Dot Product of Polynomials

$$\Phi(\mathbf{x}) = \text{polynomials of degree exactly d}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

d=1 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = x_1 z_1 + x_2 z_2 = \mathbf{x} \cdot \mathbf{z}$

d=2 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 \\ \sqrt{2} z_1 z_2 \\ z_2^2 \end{bmatrix} = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2$

$$= (x_1 z_1 + x_2 z_2)^2$$

$$= (\mathbf{x} \cdot \mathbf{z})^2$$

d $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$

Finally: The Kernel Trick!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

- Never represent features explicitly
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features

$$\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

- Very interesting theory – Reproducing Kernel Hilbert Spaces
  - Not covered in detail in 10701/15781, more in 10702

Common Kernels

- Polynomials of degree 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/Radial kernels (polynomials of all orders – recall series expansion of exp)

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

- Sigmoid

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

Overfitting

- Huge feature space with kernels, what about overfitting???
  - Maximizing margin leads to sparse set of support vectors
  - Some interesting theory says that SVMs search for simple hypothesis with large margin
  - Often robust to overfitting

What about classification time?

$$\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

- For a new input **x**, if we need to represent Φ(**x**), we are in trouble!
- Recall classifier: sign(**w**.Φ(**x**)+b)
- Using kernels we are cool!

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

SVMs with Kernels

- Choose a set of features and kernel function
- Solve dual problem to obtain support vectors α_i
- At classification time, compute:

$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

$$b = y_k - \sum_i \alpha_i y_i K(\mathbf{x}_k, \mathbf{x}_i)$$

for any k where $C > \alpha_k > 0$

Classify as

$$sign\left(\mathbf{w} \cdot \Phi(\mathbf{x}) + b\right)$$

SVMs vs. Kernel Regression

SVMs

$$sign\left(\mathbf{w}\cdot\Phi(\mathbf{x})+b\right)$$

or

$$sign\left(\sum_i \alpha_i y_i K(\mathbf{x},\mathbf{x}_i)+b\right)$$

Kernel Regression

$$sign\left(\frac{\sum_i y_i K(\mathbf{x},\mathbf{x}_i)}{\sum_j K(\mathbf{x},\mathbf{x}_j)}\right)$$

Differences:

- SVMs:
  - Learn weights α_i (and bandwidth)
  - Often sparse solution
- KR:
  - Fixed "weights", learn bandwidth
  - Solution may not be sparse
  - Much simpler to implement

SVMs vs. Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	Yes!	Yes!

Kernels in Logistic Regression

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

- Define weights in terms of features:

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$$

$$P(Y=1 \mid x, \mathbf{w}) \;=\; \frac{1}{1+e^{-(\sum_i \alpha_i \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})+b)}}$$

$$= \frac{1}{1+e^{-(\sum_i \alpha_i K(\mathbf{x},\mathbf{x}_i)+b)}}$$

- Derive simple gradient descent rule on α_i

SVMs vs. Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	Yes!	Yes!
Solution sparse	Often yes!	Almost always no!
Semantics of output	“Margin”	Real probabilities

What you need to know…

- Dual SVM formulation
  - How it’s derived
- The kernel trick
- Common kernels
- Differences between SVMs and kernel regression
- Differences between SVMs and logistic regression
- Kernelized logistic regression

Announcements - Midterm

- When: Wednesday, 10/20
- Where: In Class
- What: You, your pencil, your textbook, your notes, course slides, ~~your calculator~~, your good mood :)
- What NOT: No computers, iphones, or anything else that has an internet connection.
- Material: Everything from the beginning of the semester, until, and including SVMs and the Kernel trick

Midterm Review

- What is ML? loss functions
- Bayes optimal rules (classification, regression)
- Parametric approaches
  - Learning distributions: MLE, MAP
  - Classification: Naïve Bayes, Logistic Regression
  - Regression: Linear
- Non-parametric approaches
  - Density estimation: Histogram, Kernel density estimation
  - Classification: kNN, Decision Trees
  - Regression: Kernel regression
- Model Selection, Overfitting, Bias-variance tradeoff, estimating the generalization error
- Boosting, SVM