Support Vector Machines

Aarti Singh

Machine Learning 10-601
Nov 22, 2011

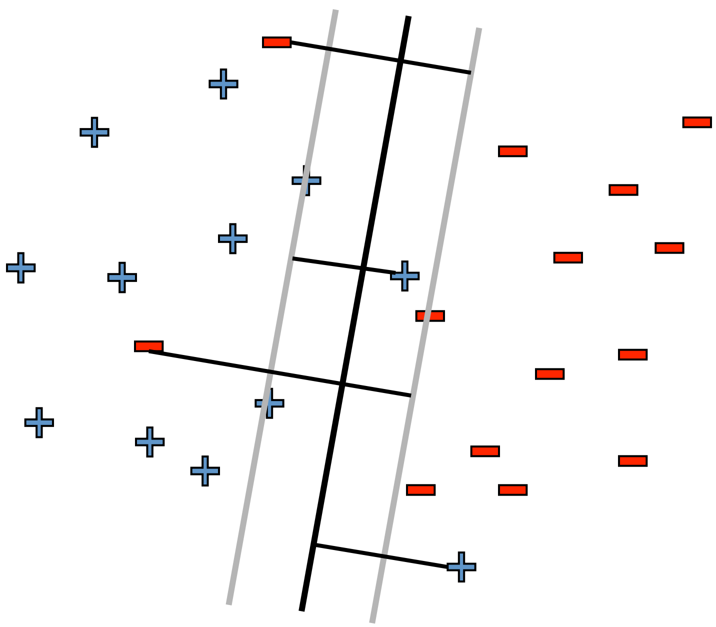
MACHINE LEARNING DEPARTMENT
Carnegie Mellon.
School of Computer Science

SVMs reminder

$$\min_{\mathbf{w},b,\xi} \; \overbrace{\mathbf{w}.\mathbf{w}}^{\text{Regularization}} + C \, \overbrace{\Sigma \xi_j}^{\text{Hinge loss}}$$

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

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



Soft margin approach

Essentially a constrained optimization problem!

Constrained Optimization

Primal problem:

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

$$\equiv \min_x \max_{\alpha \geq 0} \overbrace{L(x, \alpha)}^{\text{Lagrangian}}$$

Lagrange multiplier

Dual problem:

$$d^* = \max_\alpha \; d(\alpha) \quad \text{s.t.} \quad \alpha \geq 0$$

$$\equiv \max_{\alpha \geq 0} \min_x L(x, \alpha)$$

In general, $d^* \leq f^*$

When is $d^* = f^*$?

Constrained Optimization

Primal problem:

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

$$x^* = \text{primal solution}$$

Dual problem:

$$d^* = \max_\alpha \; d(\alpha) \quad \text{s.t.} \quad \alpha \geq 0$$

$$\alpha^* = \text{dual solution}$$

$d^* = f^*$ if f is convex and x^*, α^* satisfy the KKT conditions:

$\nabla L(x^*, \alpha^*) = 0$ Zero Gradient

$x^* \geq b$ Primal feasibility

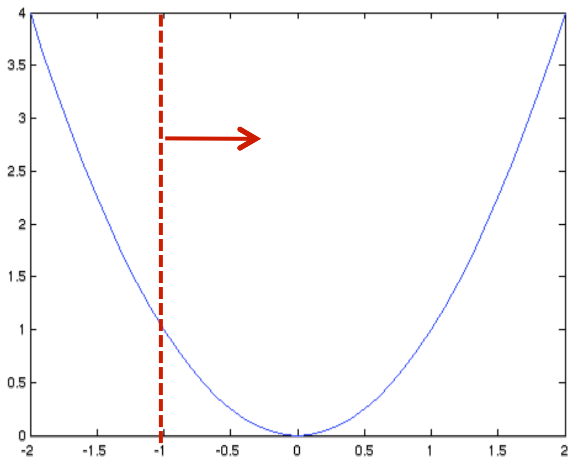
$\alpha^* \geq 0$ Dual feasibility

$\alpha^*(x^* - b) = 0$ Complementary slackness

➔ If $\alpha^* > 0$, then $x^* = b$ i.e. Constraint is effective

Constrained Optimization

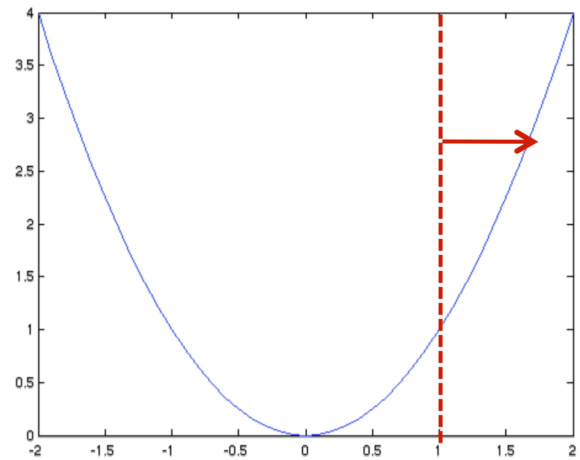
$$\min_x \; x^2$$
$$\text{s.t.} \quad x \geq -1$$



$$x^* = 0$$

α* = 0
Constraint is ineffective

$$\min_x \; x^2$$
$$\text{s.t.} \quad x \geq 1$$



$$x^* = 1$$

α* = $1/2$ > 0
Constraint is effective

Complementary slackness α*(x*-1) = 0

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

- Lagrangian:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$
$$\alpha_j \geq 0, \ \forall j$$

α - weights on training pts

$\alpha_j > 0$ constraint is effective	$(w.x_j+b)y_j = 1$ point j is a support vector!!

Dual SVM – linearly separable case

- Dual problem derivation:

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

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

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

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

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

Dual SVM – linearly separable case

- Dual problem derivation:

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

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

$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j \qquad \sum_j \alpha_j y_j = 0$$

- Dual problem:

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

$$\sum_i \alpha_i y_i = 0$$
$$\alpha_i \geq 0$$

Dual SVM – linearly separable case

- Dual problem:

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

$$\sum_i \alpha_i y_i = 0$$
$$\alpha_i \geq 0$$

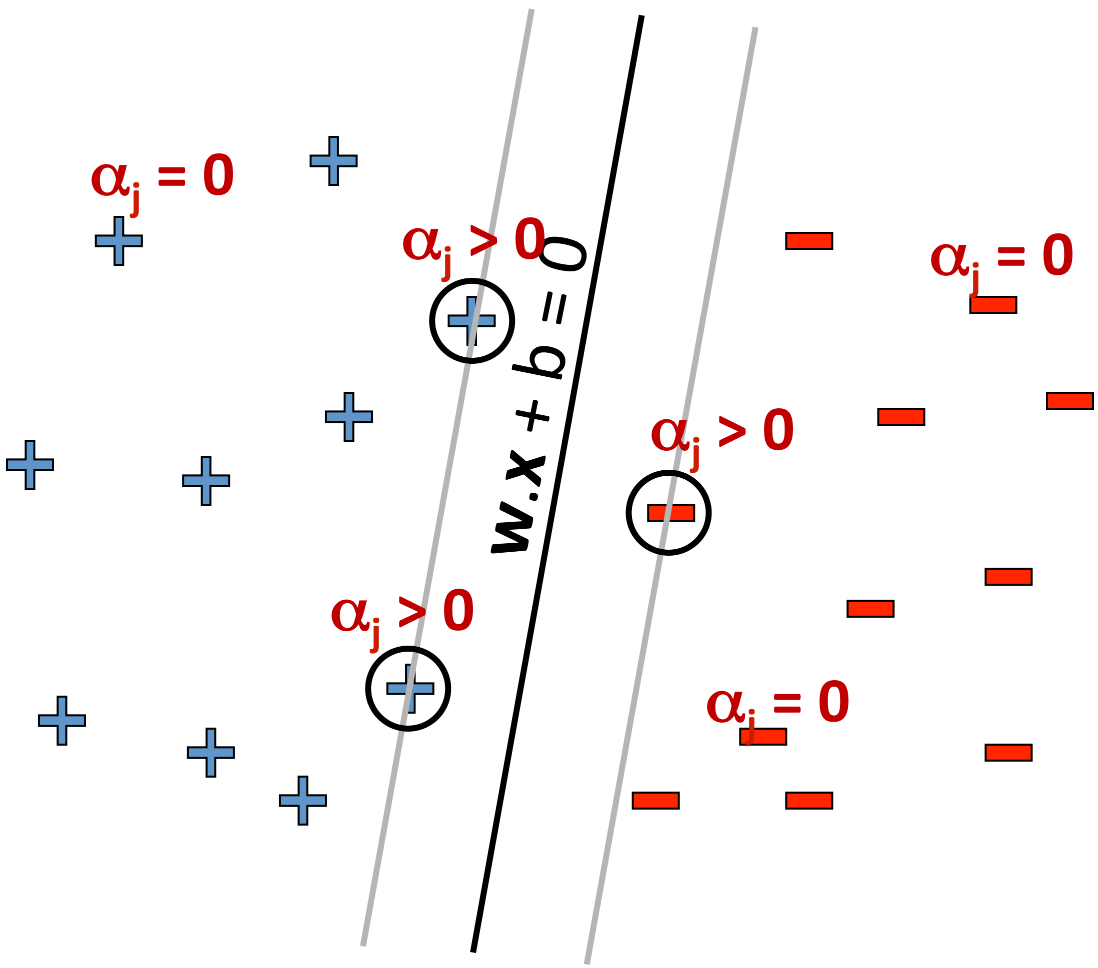
Dual problem is also QP

Solution gives α_js

$$\boxed{\begin{array}{l} \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \\ b = y_k - \mathbf{w}.\mathbf{x}_k \\ \text{for any } k \text{ where } \alpha_k > 0 \end{array}}$$

$w.x_k + b = y_k$ ⇦ $(w.x_k + b) y_k = 1$ ⇦ Use support vectors to compute b

Dual SVM Interpretation: Sparsity



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

j = Support vectors

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

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 d>>n

Recall: $(w_1, w_2, \ldots, w_d, b)$ – d+1 primal variables

$\alpha_1, \alpha_2, \ldots, \alpha_n$ – n dual variables

Dual SVM – non-separable case

- Primal problem:

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

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

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

$$\sum_i \alpha_i y_i = 0$$

$$\boxed{C \geq} \alpha_i \geq 0$$

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

Intuition:

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

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?

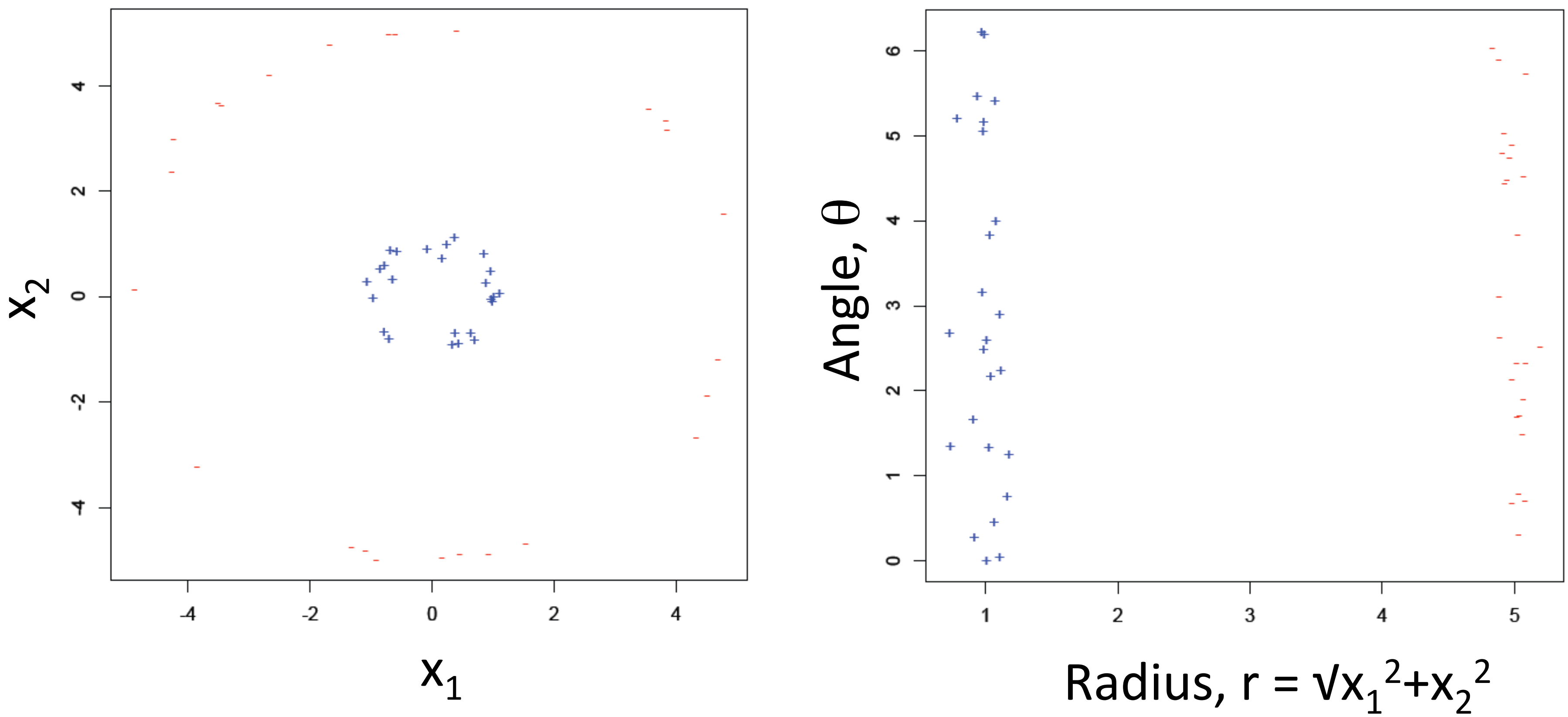
- There are some quadratic programming algorithms that can solve the dual faster than the primal, specially in high dimensions d>>n

Recall: $(w_1, w_2, \ldots, w_d, b)$ – d+1 primal variables
$\alpha_1, \alpha_2, \ldots, \alpha_n$ – n dual variables

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

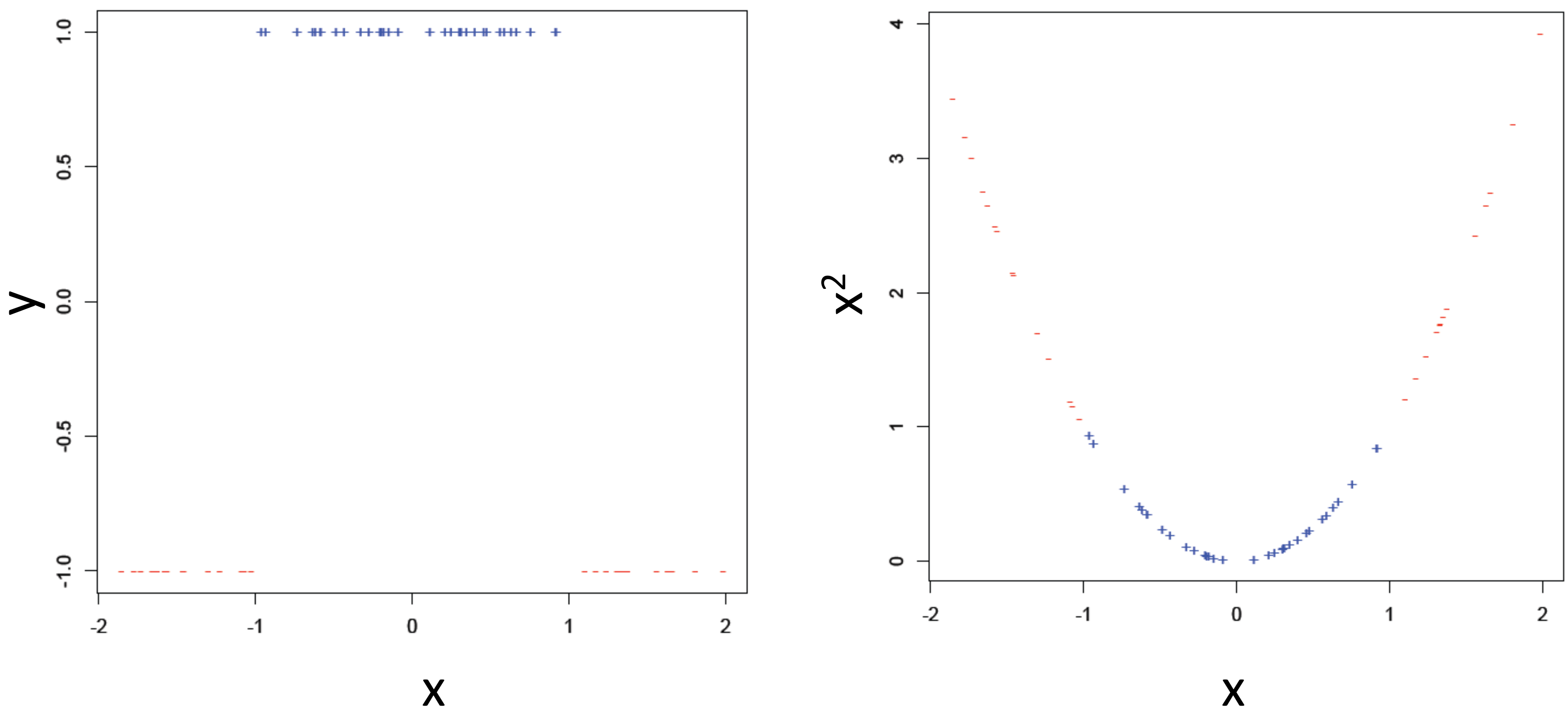
What if data is not linearly separable?

Using non-linear features to get linear separation

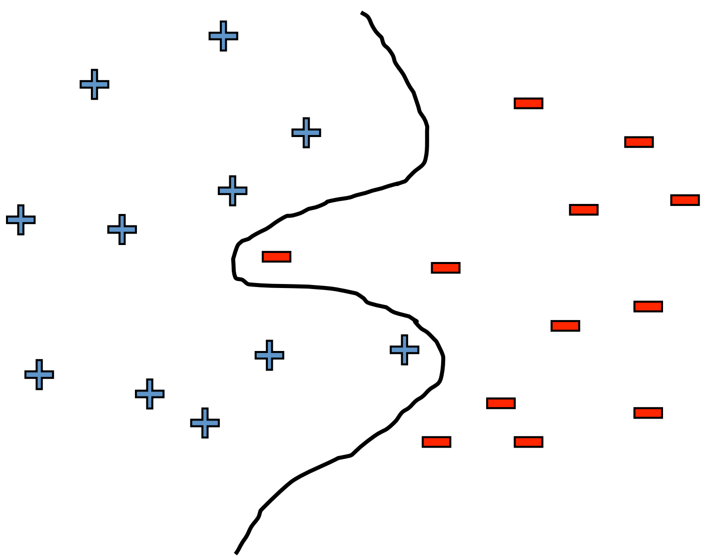


What if data is not linearly separable?

Using non-linear features to get linear separation



What if data is not linearly separable?



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

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

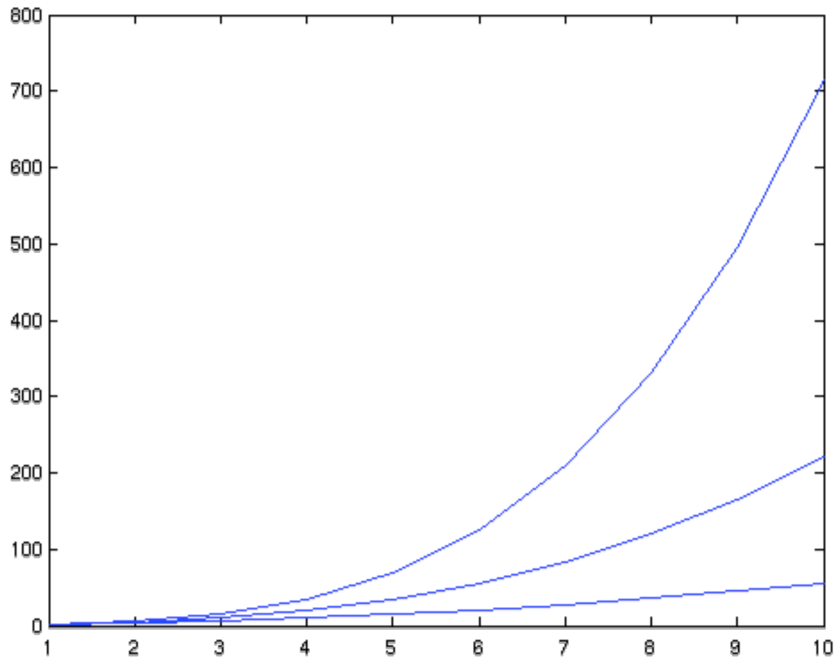
Feature space becomes really large very quickly!

Can we get by without having to write out the features explicitly?

Higher Order Polynomials

d – input features m – degree of polynomial

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



grows fast!
m = 6, d = 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

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \Phi(\mathbf{x}) = \text{polynomials of degree exactly } m$$

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

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

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

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

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

Don’t store high-dim features - Only evaluate dot-products with kernels

Finally: The Kernel Trick!

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

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

$$\sum_i \alpha_i y_i = 0$$

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

$$\boxed{\begin{array}{l} \mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i) \\ b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k) \\ \text{for any } k \text{ where } C > \alpha_k > 0 \end{array}}$$

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

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)

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

- Sigmoid

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

Which Functions Can Be Kernels?

- not all functions
- for some definitions of $K(x_1,x_2)$ there is no corresponding projection $\phi(x)$

- Nice theory on this, including how to construct new kernels from existing ones

- Initially kernels were defined over data points in Euclidean space, but more recently over strings, over trees, over graphs, …

Overfitting

- Huge feature space with kernels, what about overfitting???

  – Maximizing margin leads to sparse set of support vectors (decision boundary not too complicated)

  – 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:

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

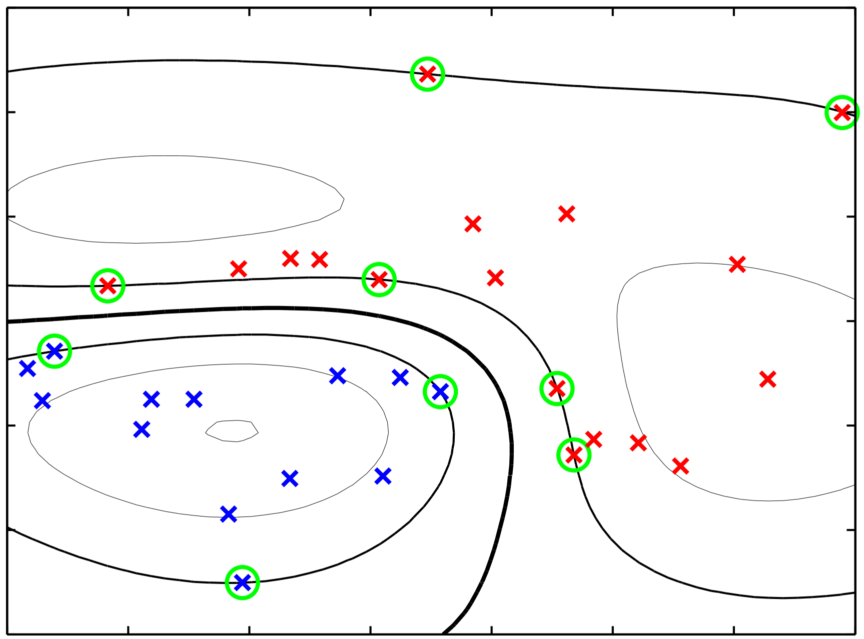
$$b = y_k - \sum_i \alpha_i y_i K(\mathbf{x}_k, \mathbf{x}_i)$$

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

Classify as

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

SVM Decision Surface using Gaussian Kernel



Bishop Fig 7.2

$$\hat{f}(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b$$

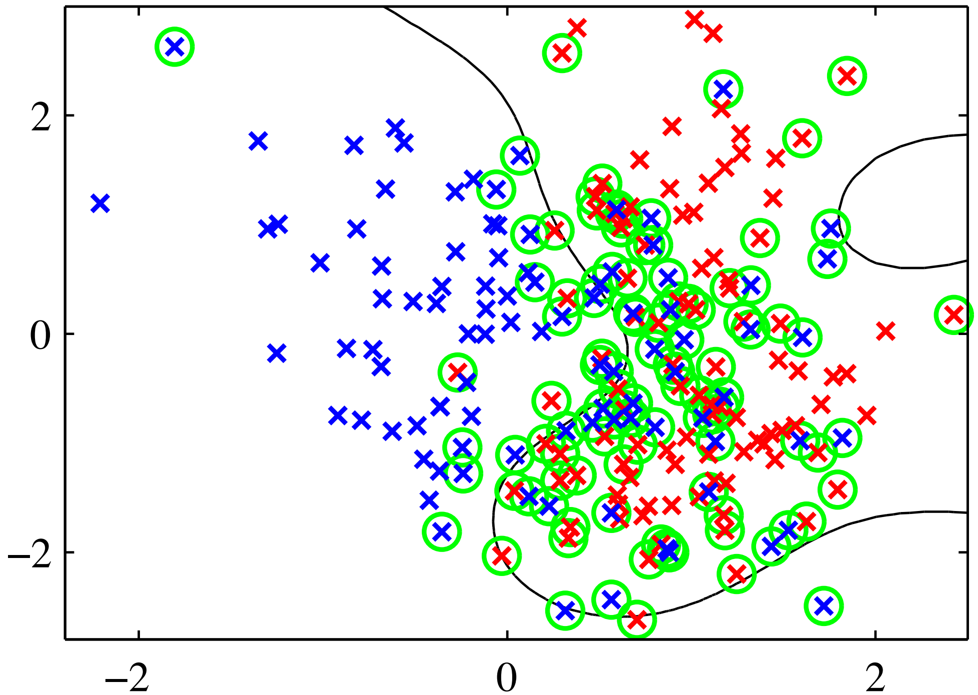
Circled points are the support vectors: training examples with non-zero α_j

Points plotted in original 2-D space.

Contour lines show constant $\hat{f}(\mathbf{x})$

$$\hat{f}(\mathbf{x}) = b + \sum_{l=1}^{M} \alpha_l \; y_l \; \kappa(\mathbf{x}, \mathbf{x}_l) = b + \sum_{l=1}^{M} \alpha_l \; y_l \exp(-\|\mathbf{x} - \mathbf{x}_l\|^2 / 2\sigma^2)$$

SVM Soft Margin Decision Surface using Gaussian Kernel



Bishop Fig 7.4

$$\hat{f}(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b$$

Circled points are the <u>support vectors</u>: training examples with non-zero α_j

Points plotted in <u>original</u> 2-D space.

Contour lines show constant $\hat{f}(\mathbf{x})$

$$\hat{f}(\mathbf{x}) = b + \sum_{l=1}^{M} \alpha_l \; y_l \; \kappa(\mathbf{x}, \mathbf{x}_l) = b + \sum_{l=1}^{M} \alpha_l \; y_l \exp(-\|\mathbf{x} - \mathbf{x}_l\|^2 / 2\sigma^2)$$

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:

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

$$\begin{aligned} P(Y=1 \mid x, \mathbf{w}) &= \frac{1}{1+e^{-\left(\sum_i \alpha_i \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})+b\right)}} \\ &= \frac{1}{1+e^{-\left(\sum_i \alpha_i K(\mathbf{x},\mathbf{x}_i)+b\right)}} \end{aligned}$$

- 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		
Semantics of output		

Kernels : Key Points

- Many learning tasks are framed as optimization problems
- Primal and Dual formulations of optimization problems
- Dual version framed in terms of dot products between x's
- Kernel functions $K(x,y)$ allow calculating dot products $<\Phi(x),\Phi(y)>$ without bothering to project x into $\Phi(x)$
- Leads to major efficiencies, and ability to use very high dimensional (virtual) feature spaces

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

- Also, Kernel PCA, Kernel ICA, ...