ADEC 7430:
Big Data Econometrics

Support Vector Machines

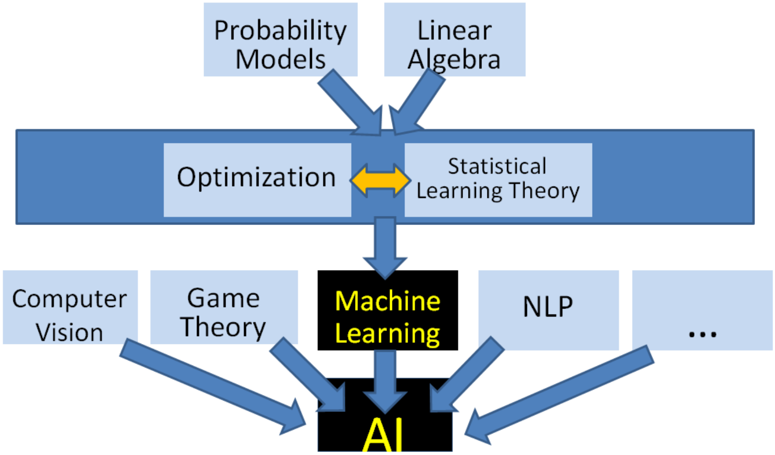
Dr. Nathan Bastian

Woods College of Advancing Studies

Boston College

Assignment

- **Reading:** Ch. 9
- **Study:** Lecture Slides, Lecture Videos
- **Activity:** Quiz 7, R Lab 7, Discussion #7



References

- *An Introduction to Statistical Learning, with Applications in R* (2013), by G. James, D. Witten, T. Hastie, and R. Tibshirani.

- *The Elements of Statistical Learning* (2009), by T. Hastie, R. Tibshirani, and J. Friedman.

- *Machine Learning: A Probabilistic Perspective* (2012), by K. Murphy

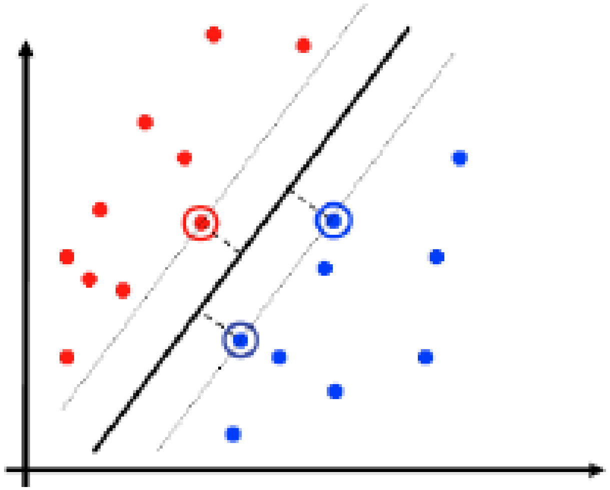
Lesson Goals

- Describe how the maximal margin classifier works for datasets in which two classes are separable by a linear boundary.
- Discuss the support vector classifier, which extends the maximal margin classifier to work with overlapping classes.
- Explain support vector machines, which extend support vector classifiers to accommodate non-linear class boundaries.
- Examine how kernel methods are used with support vector machines.

Support Vector Machines

- We approach the two-class classification problem in a direct way by trying and finding a plane that *separates the classes* in the feature space.

- Support vector machines (SVMs) choose the linear separator with the **largest margin**.



Support Vector Machines (cont'd)

- Imagine a situation where you have a two-class classification problem with two predictors X_1 and X_2.

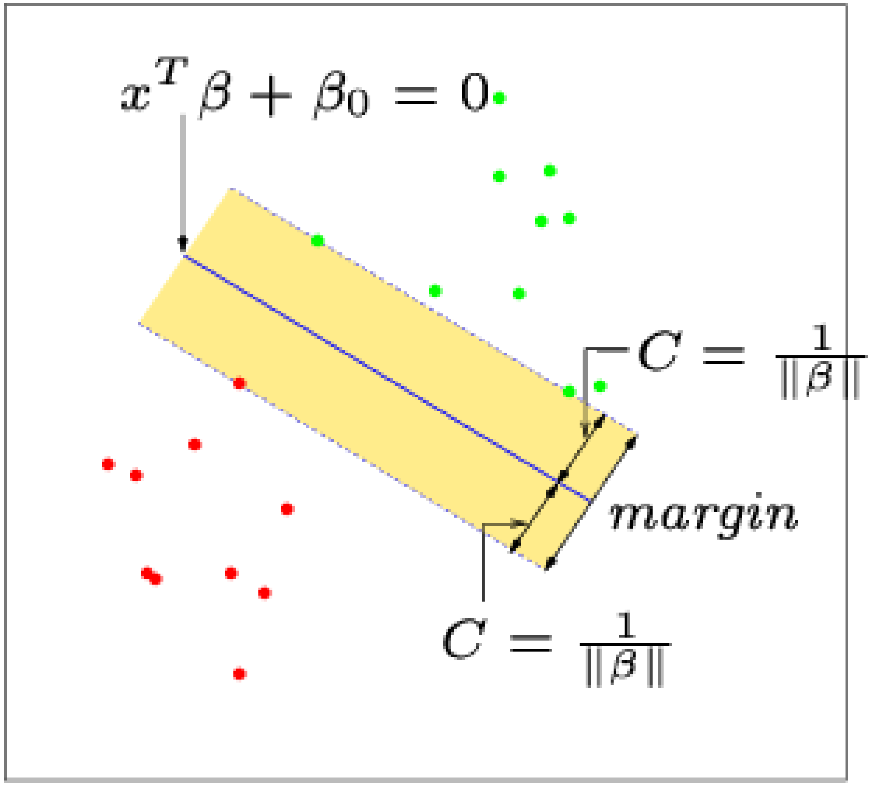
- Suppose that the two classes are *linearly separable* (i.e. one can draw a straight line in which all points on one side belong to the first class and points on the other side to the second class).

- Then a natural approach is to find the straight line that gives the largest margin (biggest separation) between the classes (i.e. the points are as far from the line as possible).

- This is the basic idea of a *support vector classifier*.

Support Vector Machines (cont'd)

- *C* is the minimum perpendicular distance between each point and the separating line.
- We find the line which maximizes *C*.
- This line is called the *optimal separating hyperplane*.
- The classification of a point depends on which side of the line it falls on.



Support Vector Machines (cont'd)

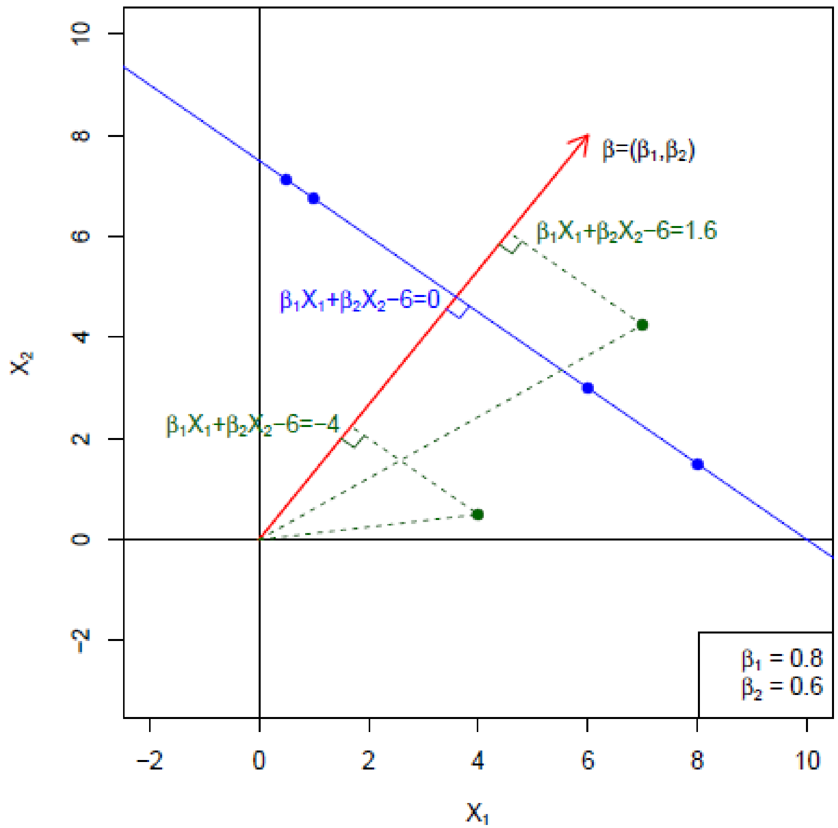
- **What is a hyperplane?**
  - A hyperplane in p dimensions is a flat affine subspace of dimension $p - 1$.
  - In general the equation for a hyperplane has the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

  - In $p = 2$ dimensions a hyperplane is a line.
  - If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
  - The vector $\beta = (\beta_1, \beta_2, \cdots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.

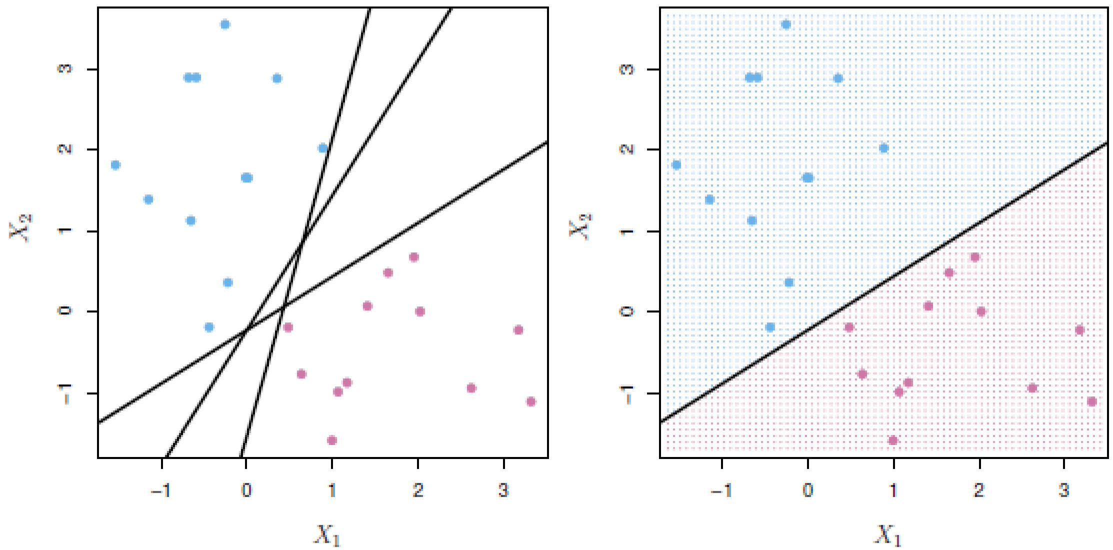
Support Vector Machines (cont'd)

- **Hyperplane in 2 Dimensions**



Support Vector Machines (cont'd)

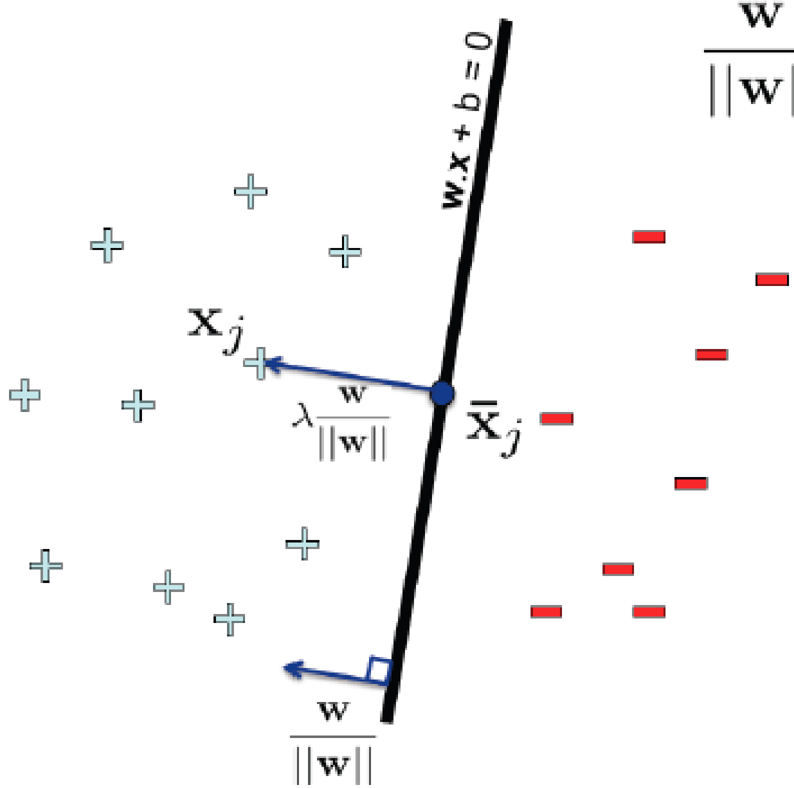
- **Separating Hyperplanes**



- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for mauve, then if $Y_i \cdot f(X_i) > 0$ for all i, $f(X) = 0$ defines a *separating hyperplane*.

Support Vector Machines (cont'd)

- **Review**: Normal to a plane



$\frac{\mathbf{w}}{||\mathbf{w}||}$ -- unit vector parallel to w

$\bar{\mathbf{x}}_j$ -- projection of x_j onto the plane

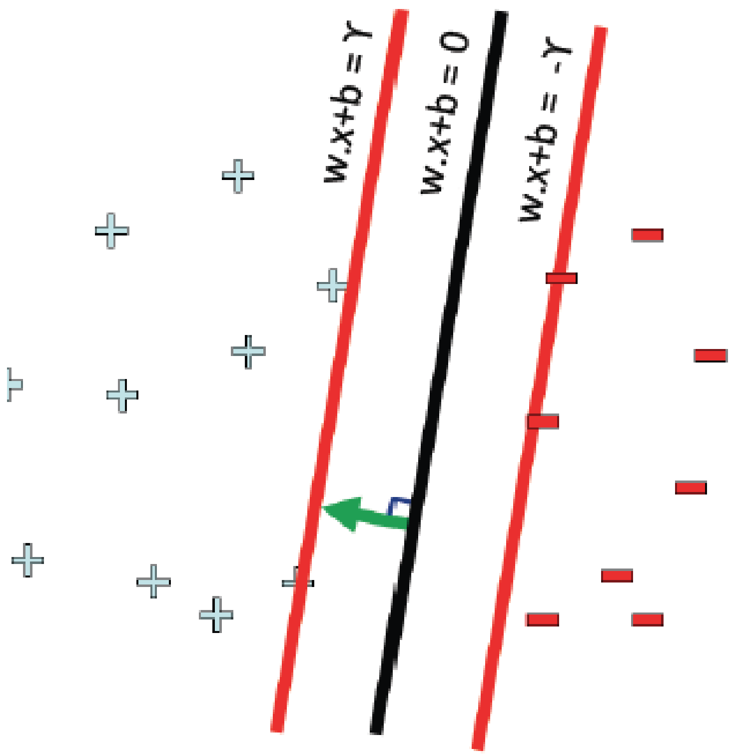
$$x_j - \bar{x}_j = \lambda \frac{w}{||w||}$$

λ is the length of the vector, i.e.

$$||x_j - \bar{x}_j|| = \frac{\lambda}{||w||}||w|| = \lambda$$

Support Vector Machines (cont'd)

- **<u>Our goal</u>**: Maximize the margin!



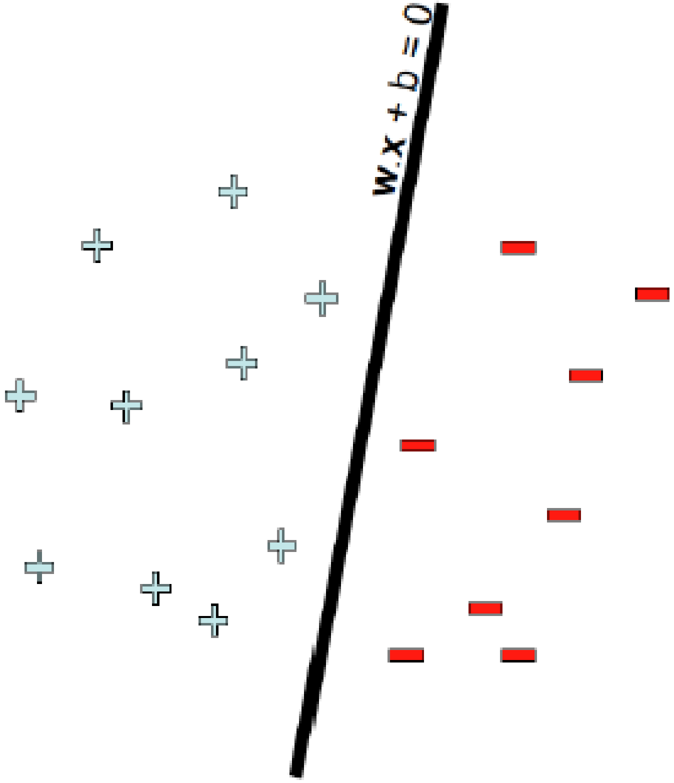
$$\max \gamma$$

$$\text{s.t. } y_j(w \cdot x_j + b) \geq \gamma, \forall j$$

$$y_j = +1, \quad w \cdot x_j + b \geq \gamma$$
$$y_j = -1, \quad w \cdot x_j + b \leq -\gamma$$

Support Vector Machines (cont'd)

- **Scale invariance**



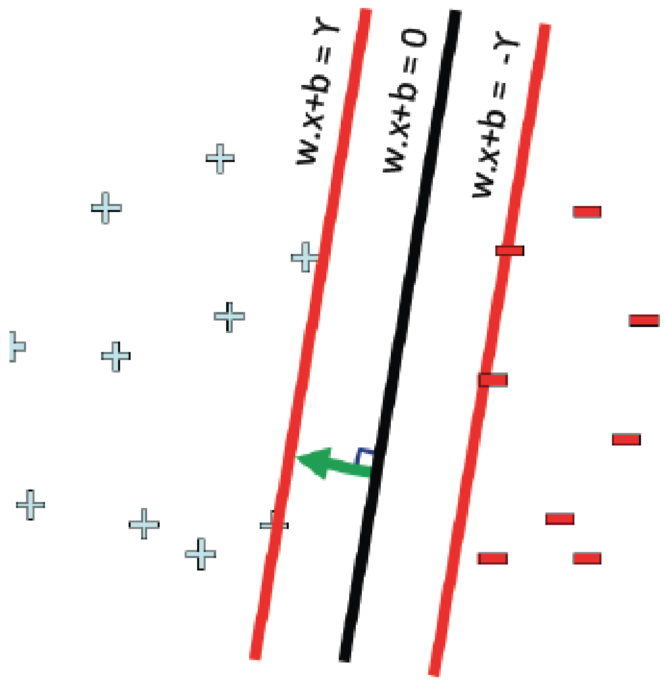
Any other ways of writing the same dividing line?

- **w.x** + b = 0
- 2**w.x** + 2b = 0
- 1000**w.x** + 1000b = 0
- ….

$$\|w\| = 1$$

Support Vector Machines (cont'd)

- **Our goal**: Maximize the margin!



$$\max \gamma$$
$$\text{s.t. } y_j(w \cdot x_j + b) \geq \gamma, \forall j$$
$$\|w\| = 1$$

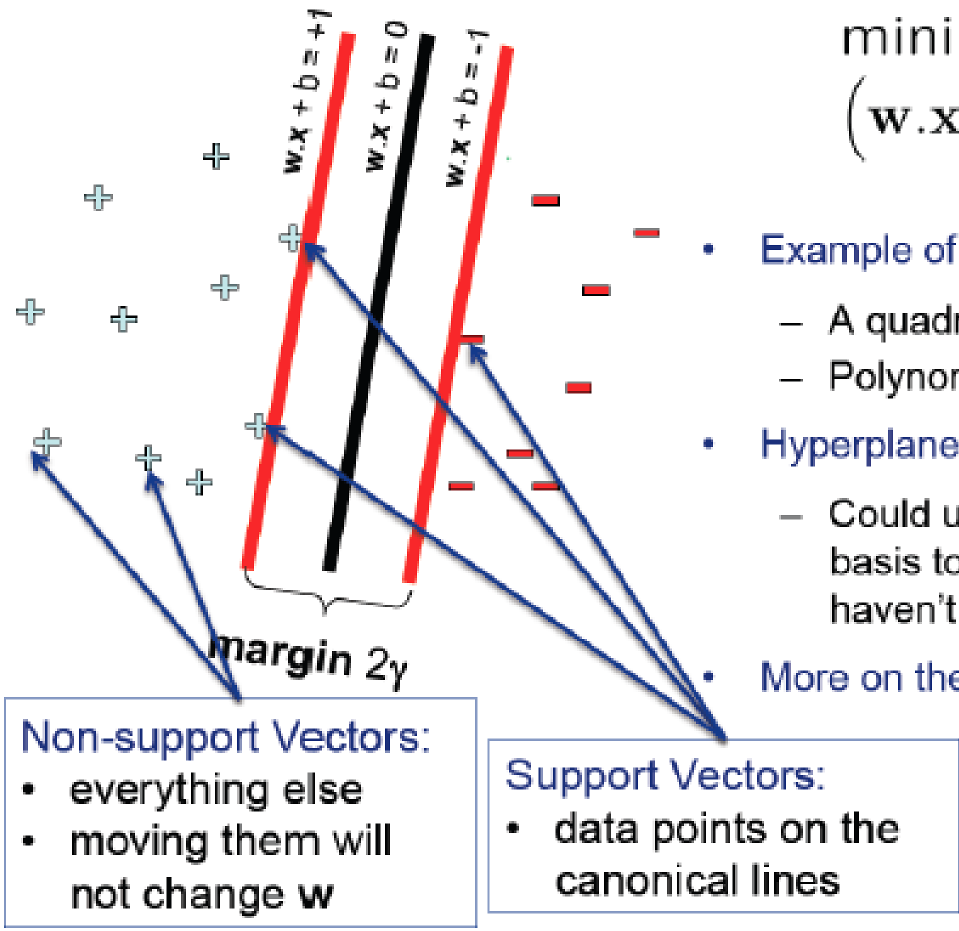
$$\text{Let } w' = \frac{w}{\gamma}$$

$$\Rightarrow \quad y_j(\frac{w}{\gamma} \cdot x_j + \frac{b}{\gamma}) \geq 1, \forall j$$

$$\text{In addition, we note } \|w'\| = \frac{\|w\|}{\gamma} = \frac{1}{\gamma}$$
$$\text{therefore, } \gamma = \frac{1}{\|w'\|}$$

$$\Rightarrow \quad \max \frac{1}{\|w'\|}$$
$$\text{s.t. } y_j(w \cdot x_j + b) \geq 1, \forall j$$

Support Vector Machines (cont'd)



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

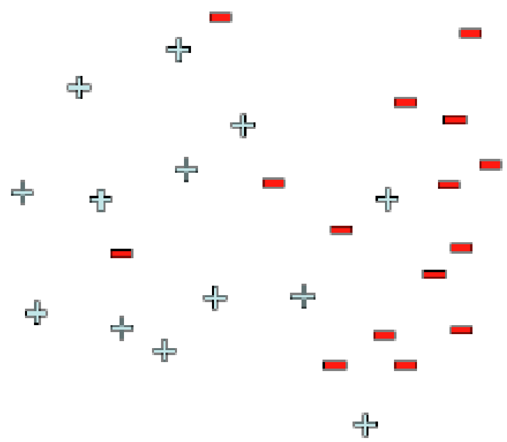
- Example of a **convex optimization** problem
  - A quadratic program
  - Polynomial-time algorithms to solve!
- Hyperplane defined by **support vectors**
  - Could use them as a lower-dimension basis to write down line, although we haven't seen how yet
- More on these later

Support Vector Machines (cont'd)

- This idea works just as well with more than two predictors, too!
- For example, with three predictors you want to find the plane that produces the largest separation between the classes.
- With more than three dimensions, it becomes hard to visualize a plane but it still exists. In general, they are called *hyperplanes*.
- In practice, it is not usually possible to find a hyperplane that perfectly separates two classes.

Support Vector Machines (cont'd)

- **What if the data is not linearly separable?**



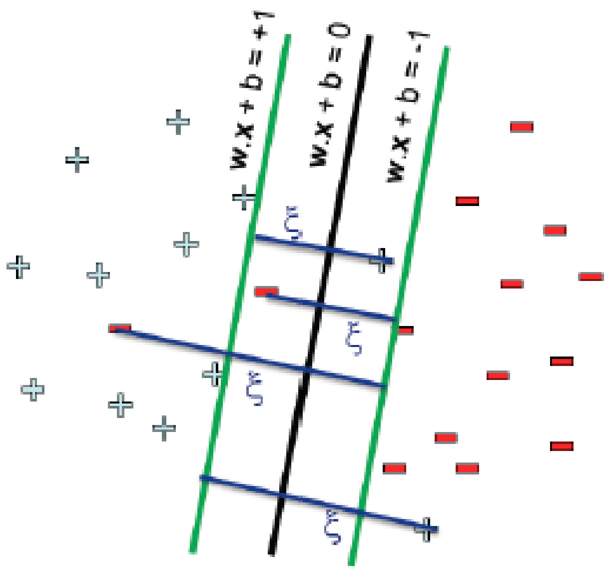
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\ \#(\text{mistakes})$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad ,\forall j$$

- First Idea: Jointly minimize **w.w** and number of training mistakes
  - How to tradeoff two criteria?
  - Pick C using held-out data

- Tradeoff #(mistakes) and **w.w**
  - 0/1 loss
  - Not QP anymore
  - Also doesn't distinguish near misses and really bad mistakes
  - NP hard to find optimal solution!!!

Support Vector Machines (cont'd)

- **Allowing for slack: "soft margin" SVM**



$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\,\Sigma_j\,\xi_j$$

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

"slack variables"

Slack penalty $C > 0$:

•$C=\infty$ → have to separate the data!

•$C=0$ → ignores the data entirely!

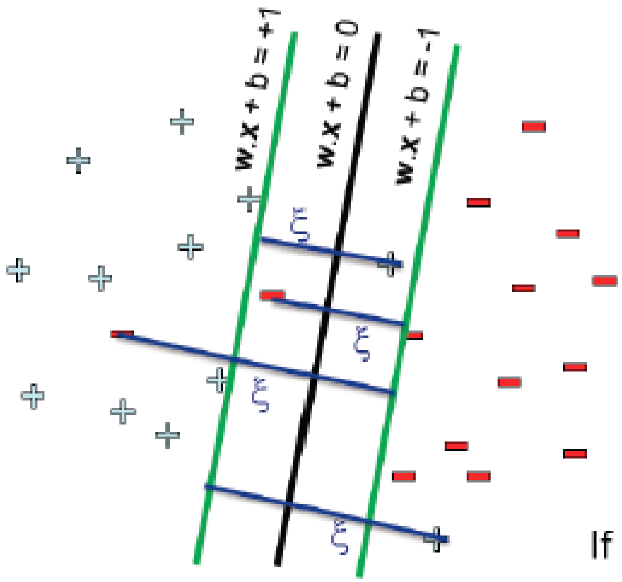
For each data point:

•If margin ≥ 1, don't care

•If margin < 1, pay linear penalty

Support Vector Machines (cont'd)

- **Allowing for slack: "soft margin" SVM**



$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\,\Sigma_j\,\xi_j$$

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

"slack variables"

What is the (optimal) value of ξ_j as a function of **w** and b?

If $(w \cdot x_j + b)\, y_j \geq 1$, then $\xi_j = 0$

If $(w \cdot x_j + b)\, y_j < 1$, then $\xi_j = 1 - (w \cdot x_j + b)\, y_j$

$$\xi_j = \max\left(0, 1 - (w \cdot x_j + b)\, y_j\right)$$

Sometimes written as

$$\left(1 - (w \cdot x_j + b)\, y_j\right)_+$$

Support Vector Machines (cont'd)

- **Equivalent hinge loss formulation (similar to ridge regression)**

$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\,\Sigma_j\,\xi_j$$

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

Substituting $\xi_j = \max\left(0, 1 - (w \cdot x_j + b)\, y_j\right)$ into the objective, we get:

$$\boxed{\min ||w||^2 + C \sum_j \max\left(0, 1 - (w \cdot x_j + b)\, y_j\right)}$$

The **hinge loss** is defined as $L(y, \hat{y}) = \max\Big(0, 1 - \hat{y}y\Big)$

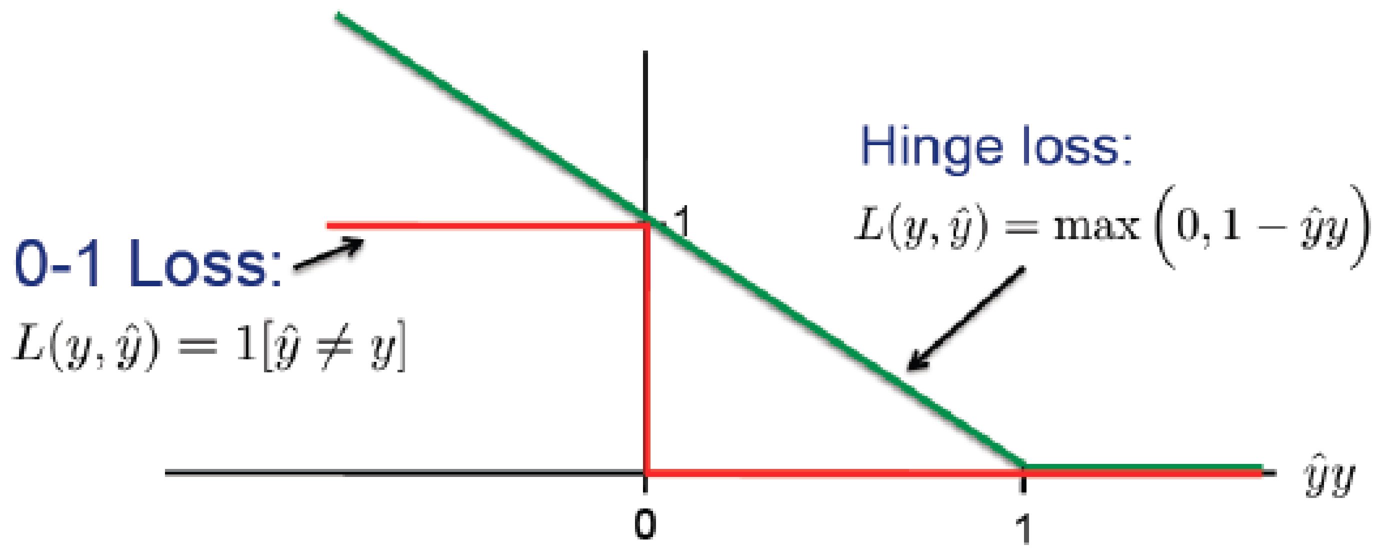
$$\min_{w,b} ||w||_2^2 + C \underline{\sum_j L(y_j, \mathbf{w} \cdot x_j + b)}$$

This is called **regularization**; used to prevent overfitting!

This part is empirical risk minimization, using the hinge loss

Support Vector Machines (cont'd)

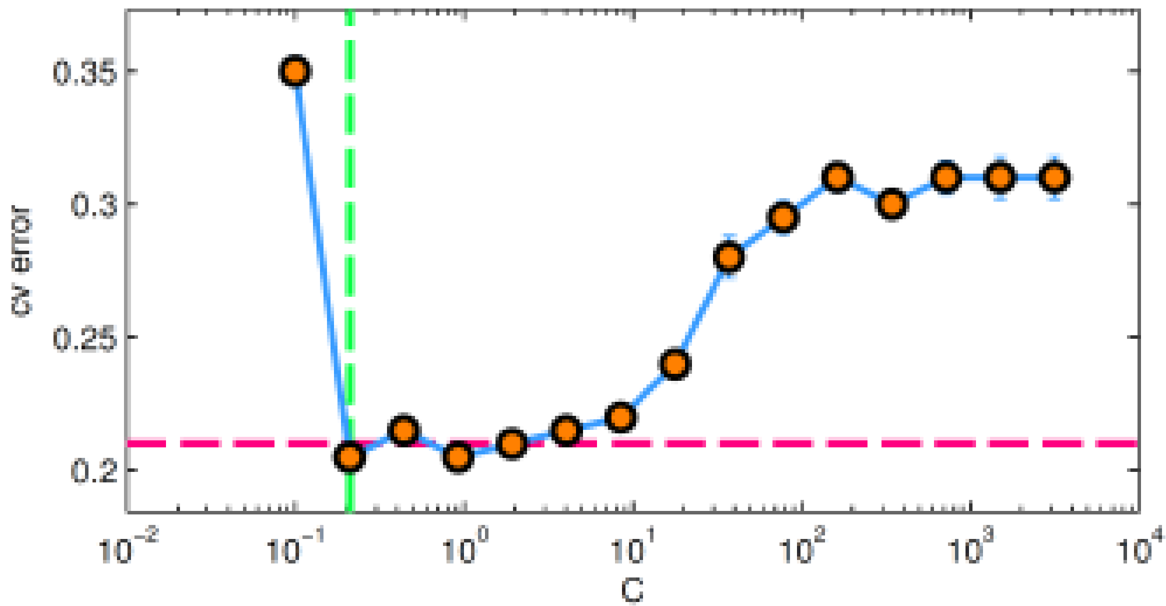
- **Hinge loss vs. 0/1 loss**



Hinge loss upper bounds 0/1 loss!

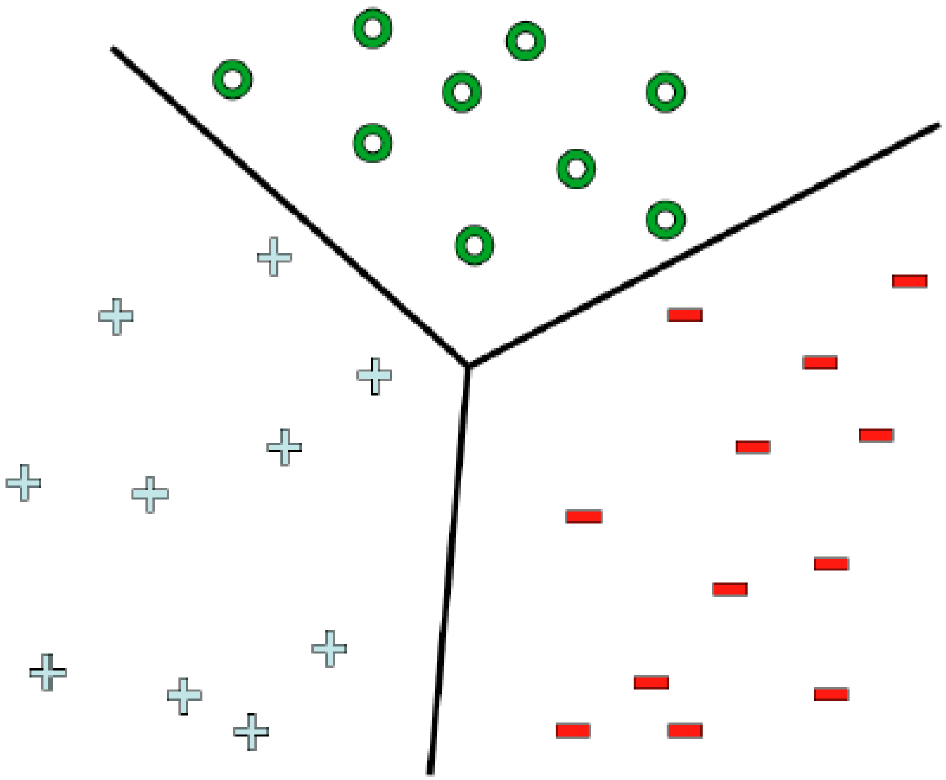
Support Vector Machines (cont'd)

- We can use cross-validation to choose the best *C*
- The larger the value of *C*, the slower the training is:



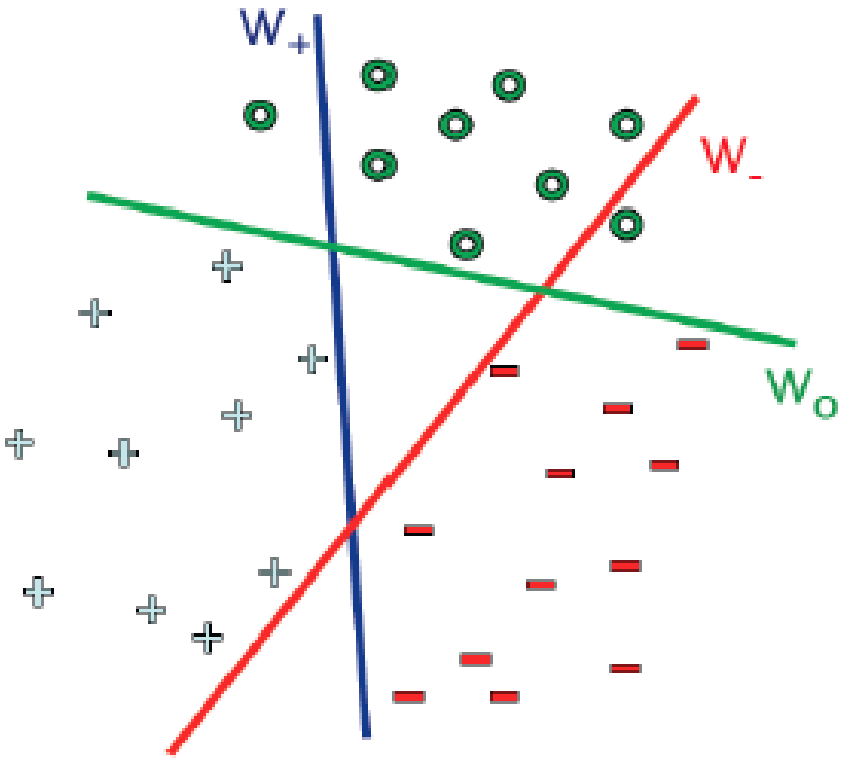
Support Vector Machines (cont'd)

- **How do we do multi-class classification?**



Support Vector Machines (cont'd)

- **One versus all classification**



Learn 3 classifiers:

- \- vs {o,+}, weights w_-
- \+ vs {o,-}, weights w_+
- o vs {+,-}, weights w_o

Predict label using:

$$\hat{y} \leftarrow \arg\max_k \; w_k \cdot x + b_k$$

Support Vector Machines (cont'd)

- **Multi-class SVM**

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.

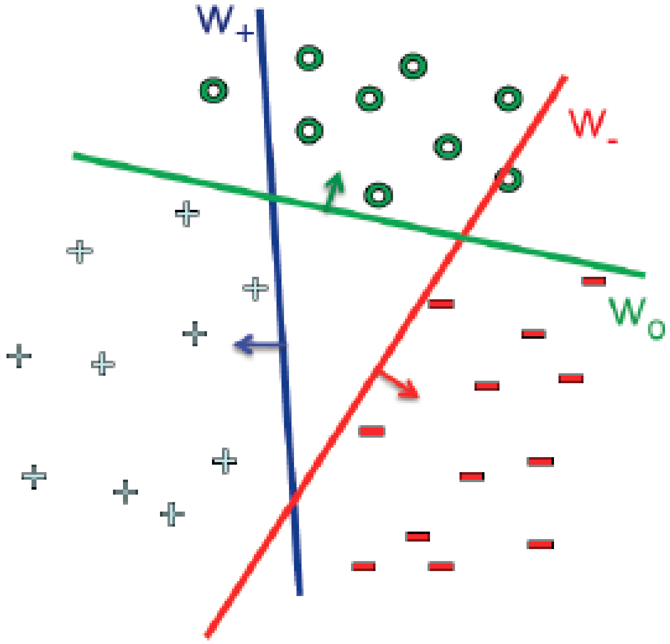
Support Vector Machines (cont'd)

- **Multi-class SVM**

Simultaneously learn 3 sets of weights:

•How do we guarantee the correct labels?

•Need new constraints!



The "score" of the correct class must be better than the "score" of wrong classes:

$$w^{(y_j)} \cdot x_j + b^{(y_j)} > w^{(y)} \cdot x_j + b^{(y)} \qquad \forall j,\ y \neq y_j$$

Support Vector Machines (cont'd)

- **Multi-class SVM**
  - As for the SVM, we introduce slack variables and maximize margin:

$$\begin{aligned} &\text{minimize}_{\mathbf{w},b} \quad \textstyle\sum_y \mathbf{w}^{(y)}.\mathbf{w}^{(y)} + C\sum_j \xi_j \\ &\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1 - \xi_j, \ \forall y' \neq y_j, \ \forall j \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \xi_j \geq 0, \ \forall j \end{aligned}$$

To predict, we use:

$$\hat{y} \leftarrow \arg\max_k \ w_k \cdot x + b_k$$

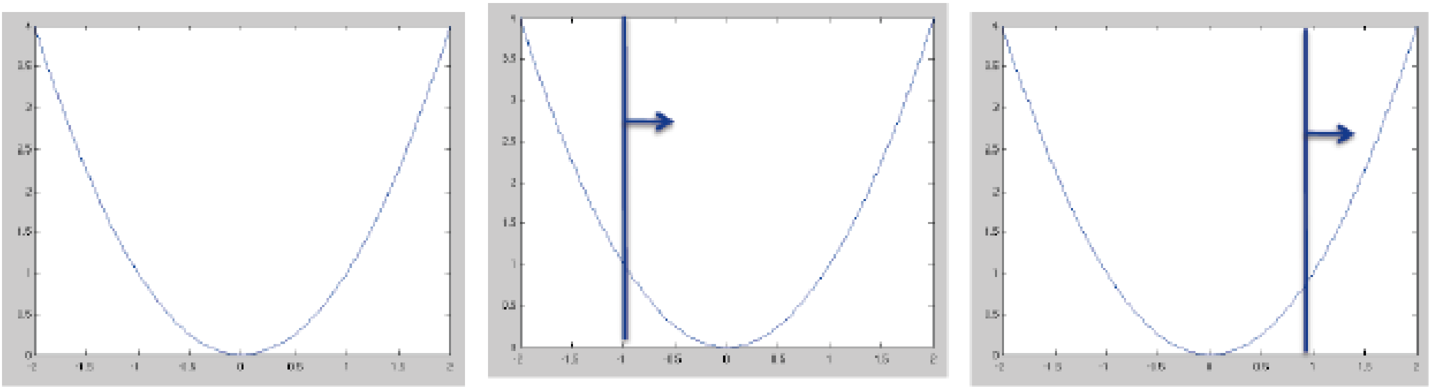
Support Vector Machines (cont'd)

- **How do we solve with constraints?**
    - Lagrange multipliers!

Constrained optimization

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

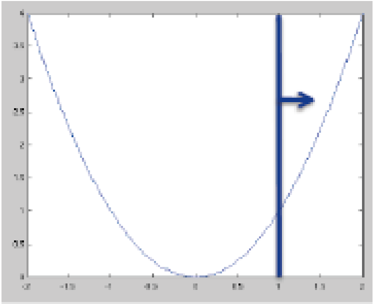
No Constraint | $x \geq -1$ | $x \geq 1$



$x^*=0$ | $x^*=0$ | $x^*=1$

Support Vector Machines (cont'd)

- **Lagrange multipliers – dual variables**



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

Add Lagrange multiplier

Rewrite Constraint

Introduce Lagrangian (objective):

$$L(x, \alpha) = x^2 - \alpha(x - b)$$

Why is this equivalent?

- min is fighting max!

$x<b \rightarrow (x-b)<0 \rightarrow \max_\alpha -\alpha(x-b) = \infty$

- min won't let this happen!

$x>b,\ \alpha \geq 0 \rightarrow (x-b)>0 \rightarrow \max_\alpha -\alpha(x-b) = 0,\ \alpha^*=0$

- min is cool with 0, and $L(x, \alpha)=x^2$ (original objective)

$x=b \rightarrow \alpha$ can be anything, and $L(x, \alpha)=x^2$ (original objective)

We will solve:

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

Add new constraint

The *min* on the outside forces *max* to behave, so constraints will be satisfied.

Support Vector Machines (cont'd)

- **Primal and Dual**

Primal

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

Dual

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

- **Why people are interested in the dual problem?**
  - It might be easier to optimize the dual.
  - The dual problem may have some nice properties (Kernel trick in SVM)

Support Vector Machines (cont'd)

- **Dual SVM derivation (1) – the linearly separable case**

Original optimization problem:

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

Rewrite constraints

One Lagrange multiplier per example

Lagrangian:

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

Our goal now is to solve: $\min_{\vec{w},b} \ \max_{\vec{\alpha}\geq 0} \ L(\vec{w}, \vec{\alpha})$

Support Vector Machines (cont'd)

- **Dual SVM derivation (2) – the linearly separable case**

(Primal) $$\min_{\vec{w},b}\ \max_{\vec{\alpha}\geq 0}\ \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Swap min and max

(Dual) $$\max_{\vec{\alpha}\geq 0}\ \min_{\vec{w},b}\ \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Slater's condition from convex optimization guarantees that these two optimization problems are equivalent!

Support Vector Machines (cont'd)

- **Dual SVM derivation (3) – the linearly separable case**

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1 \right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

$$\frac{1}{2}\|\vec{w}\|^2 = \frac{1}{2}\|\sum_j \alpha_j y_j \vec{x}_j\|^2 = \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j)$$

$$-\sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b) y_j - 1 \right] = -\sum_j \alpha_j \left[\sum_i \alpha_i y_i (\vec{x}_i \cdot \vec{x}_j) y_j \right] - \sum_j \alpha_j y_j b + \sum_j \alpha_j$$

$$= -\sum_{i,j} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j) + \sum_j \alpha_j$$

Support Vector Machines (cont'd)

- **Dual SVM derivation (3) – the linearly separable case**

$$\text{(Dual)} \quad \max_{\vec{\alpha}\geq 0} \; \min_{\vec{w},b} \; \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

$$\text{(Dual)} \quad \max_{\vec{\alpha}\geq 0,\, \sum_j \alpha_j y_j = 0} \; \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} \underbrace{y_i y_j \alpha_i \alpha_j}_{\text{scalars}} \underbrace{(\vec{x}_i \cdot \vec{x}_j)}_{\text{dot product}}$$

Sums over all training examples   scalars   dot product

Support Vector Machines (cont'd)

- **Dual SVM derivation (3) – the linearly separable case**
  - So, in the dual formulation we solve for α directly!
  - **w** and b are computed from α (if needed)

Lagrangian:

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

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

$\alpha_j > 0$ for some j implies constraint is tight. We use this to obtain b:

$$y_j \left(\vec{w} \cdot \vec{x}_j + b\right) = 1 \quad (1)$$

$$y_j y_j \left(\vec{w} \cdot \vec{x}_j + b\right) = y_j \quad (2)$$

$$\left(\vec{w} \cdot \vec{x}_j + b\right) = y_j \quad (3)$$

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

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

for any k where $\alpha_k > 0$

Support Vector Machines (cont'd)

- **SVM Primal and Dual – the linearly separable case**

(Primal)
$$\min_{\vec{w},b} \max_{\vec{\alpha}\geq 0} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Swap min and max

(Dual)
$$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

(Dual)
$$\max_{\vec{\alpha}\geq 0,\, \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j\right)$$

Kernel trick

Support Vector Machines (cont'd)

- **Dual for the non-separable case: same basic story**

Primal:

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

Solve for w,b, α:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

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

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

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

What changed?

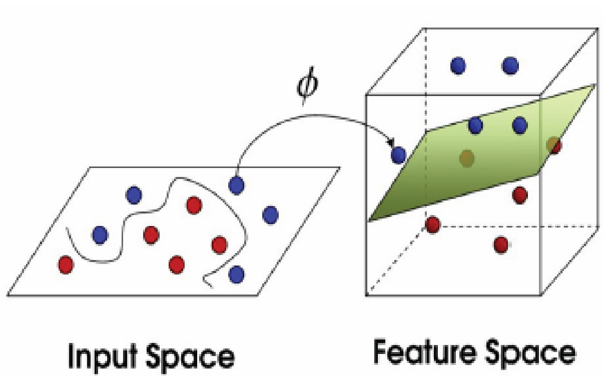
- Added upper bound of C on α_i!
- Intuitive explanation:
  - Without slack, $\alpha_i \rightarrow \infty$ when constraints are violated (points misclassified)
  - Upper bound of C limits the α_i, so misclassifications are allowed

Support Vector Machines (cont'd)

- Note that there are some quadratic programming algorithms that can solve the dual faster than the primal (at least for small data sets).

- So, what do we do if a linear boundary won't work?

Use features of features of features of features….



$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \dots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \dots \\ e^{x^{(1)}} \\ \dots \end{pmatrix}$$

Feature space can get really large really quickly!

Support Vector Machines (cont'd)

- **Feature Expansion**
  - Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,.... Hence go from a p-dimensional space to a $M > p$ dimensional space.
  - Fit a support-vector classifier in the enlarged space.
  - This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

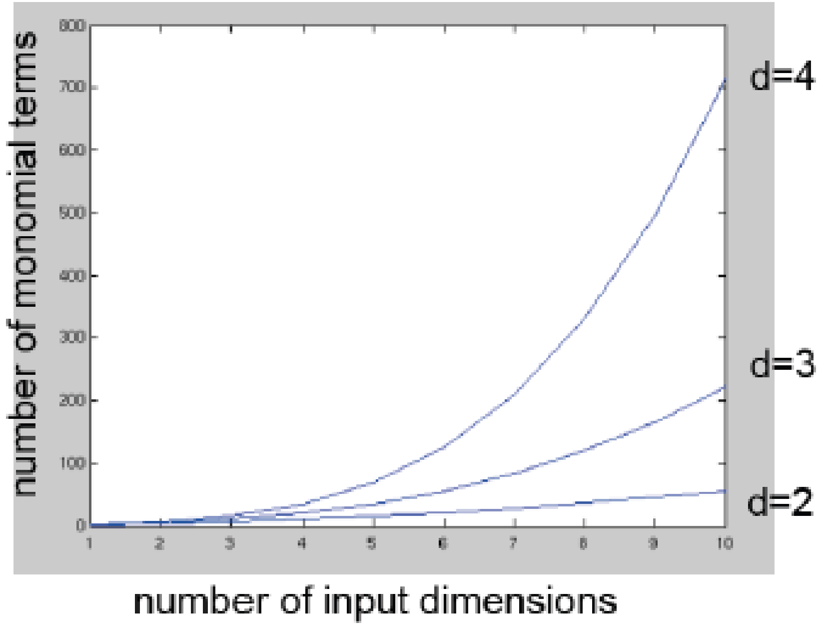
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Support Vector Machines (cont'd)

- **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

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

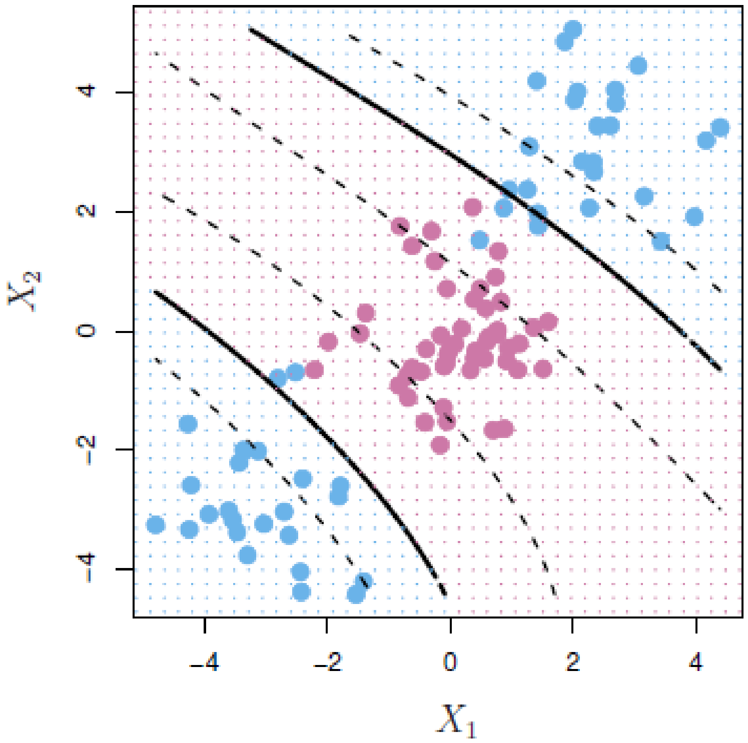
Support Vector Machines (cont'd)

- **Higher order polynomials**

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space



$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2=0$$

Kernel Methods

Nonlinearities and Kernels

- Polynomials (especially high-dimensional ones) get wild rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers – through the use of *kernels*.
- We must first understand the role of **dot products** in support-vector classifiers.

Kernel Methods (cont'd)

- **Dual formulation only depends on dot-products!**

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

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

Remember the examples x only appear in one dot product

First, we introduce features:

$$\mathbf{x}_i\mathbf{x}_j \quad \rightarrow \quad \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$$

Next, replace the dot product with a Kernel:

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

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Kernel Methods (cont'd)

- **Efficient dot-product of polynomials**

Polynomials of degree exactly *d*

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

d=2

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} \begin{aligned} &= u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2 \\ &= (u_1 v_1 + u_2 v_2)^2 \\ &= (u.v)^2 \end{aligned}$$

For any *d (we will skip proof):*

$$\phi(u).\phi(v) = (u.v)^d$$

- Taking a dot product and exponentiating gives the same results as mapping into high-dimensional space and then taking the dot product.

Kernel Methods (cont'd)

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

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

- Never compute features explicitly!!!
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features
- But, $O(n^2)$ time in size of dataset to compute objective
  - Naïve implements slow
  - much work on speeding up

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

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

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

Kernel Methods (cont'd)

- **Common Kernels**
  - Polynomials of degree exactly *d*

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

  - Polynomials of degree up to *d*

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

Squared Euclidean distance

  - Gaussian kernels

$$K(\vec{u},\vec{v}) = \exp\left(-\frac{||\vec{u}-\vec{v}||_2^2}{2\sigma^2}\right)$$

← RBF kernel
(Radial Basis Function)

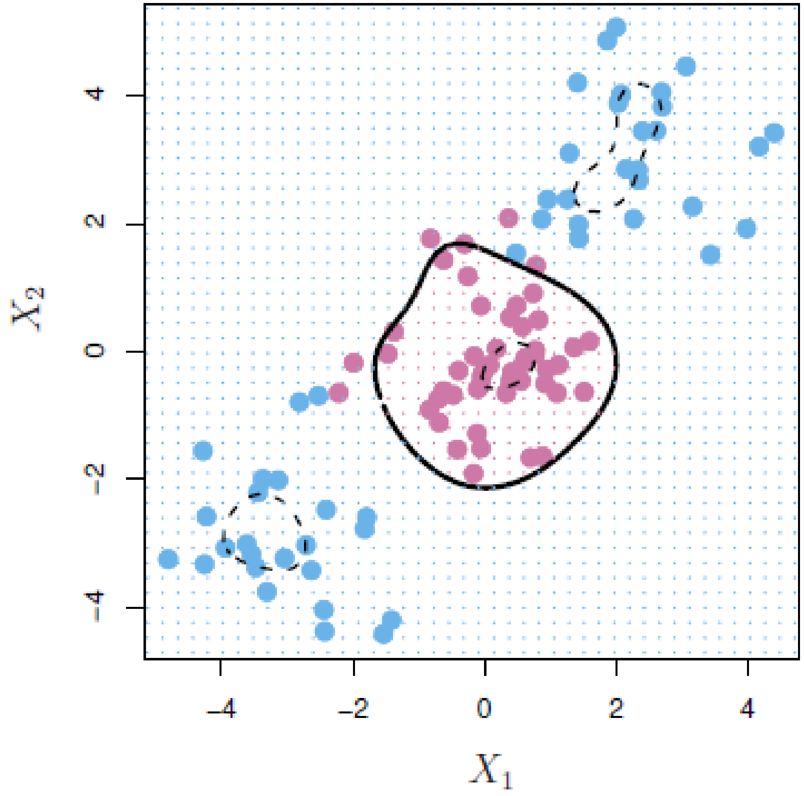
  - Sigmoid

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

  - And many others: very active area of research!

Kernel Methods (cont'd)

- **Radial Basis Function (RBF) Kernel (Gaussian)**



Kernel Methods (cont'd)

- **RBF Kernel: Standardize the Data**

$$K(\vec{u}, \vec{v}) = \exp\left(-\frac{||\vec{u} - \vec{v}||_2^2}{2\sigma^2}\right)$$

Example:
u = (1, 1000), v = (2, 2000)
the distance is dominated by the second feature

Normalize the feature

$$x(i,j) = (x(i,j) - \mu_j)/\sigma_j^2$$

$x(i,j)$: the value of j-th feature in i-th sample

μ_j : mean of j-th feature

σ_j : standard deviation of j-th feature

Kernel Methods (cont'd)

- **RBF Kernel: Parameter**

$$K(\vec{u}, \vec{v}) = \exp\left(-\frac{||\vec{u} - \vec{v}||_2^2}{2\sigma^2}\right)$$

$$\begin{aligned} K(u, v) &= exp(-\frac{\|u - v\|_2^2}{2\sigma^2}) \\ &= exp(\frac{1}{2\sigma^2} \cdot (-\|u - v\|_2^2)) \\ &= exp(\gamma \cdot (-\|u - v\|_2^2)) \end{aligned}$$

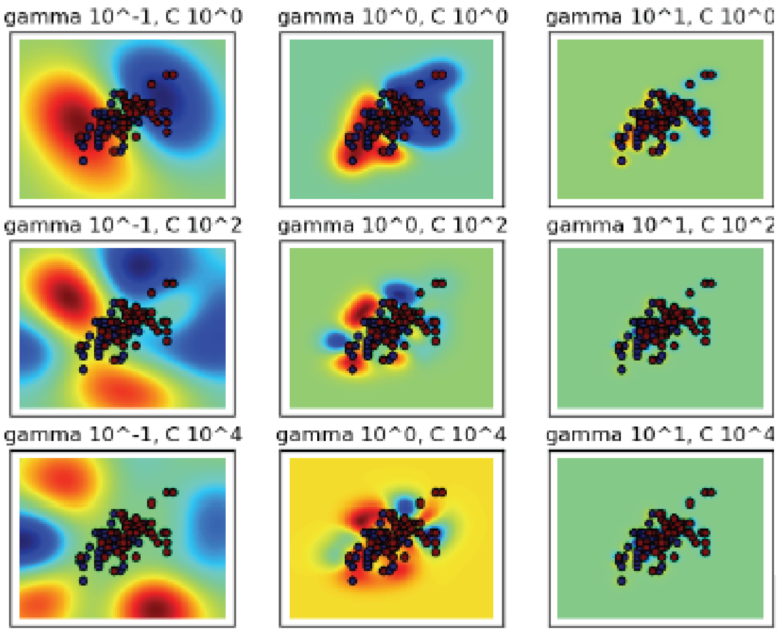
$$\gamma = \frac{1}{2\sigma^2}$$

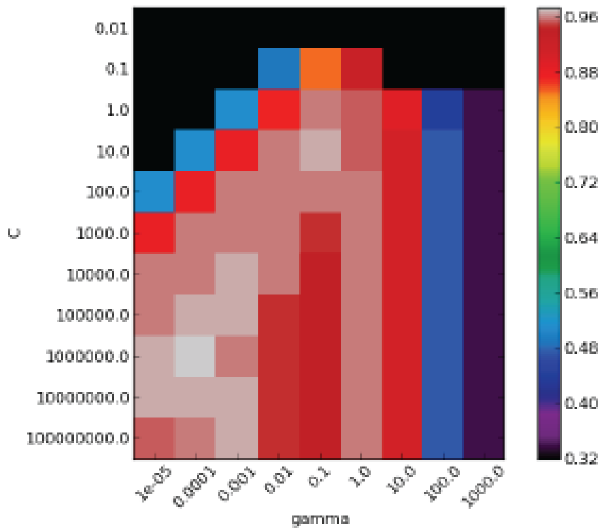
Kernel Methods (cont'd)

- **RBF Kernel: Parameter**

$$\gamma = \frac{1}{2\sigma^2}$$

Large gamma → small variance
→ need heavy regularization → need small C



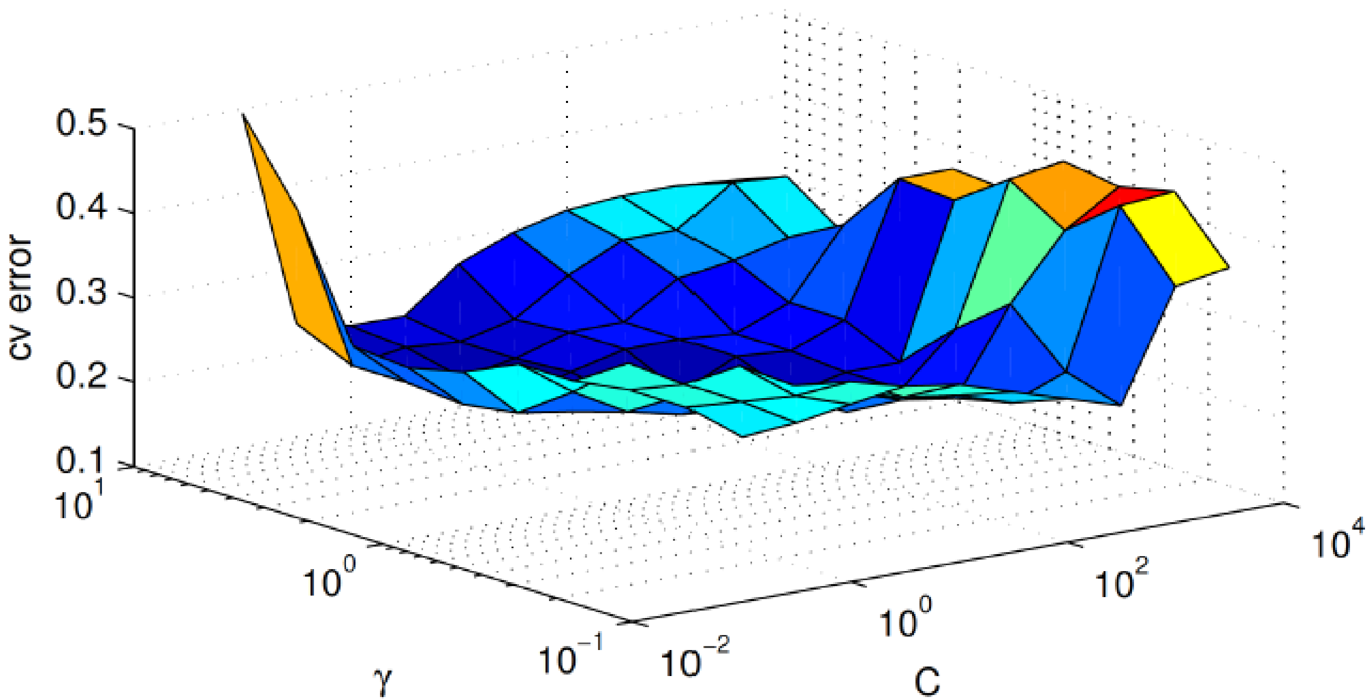


Kernel Methods (cont'd)

- **RBF Kernel: Parameter**

$$\gamma = \frac{1}{2\sigma^2}$$

Large gamma → small variance
→ need heavy regularization → need small C



Kernel Methods (cont'd)

- **RBF Kernel: Parameter**
  - Use cross-validation to find the best parameters for *C* and gamma
- **Kernel Algebra**

kernel composition	feature composition
a) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) + k_b(\mathbf{x},\mathbf{v})$	$\boldsymbol{\phi}(\mathbf{x}) = (\boldsymbol{\phi}_a(\mathbf{x}), \boldsymbol{\phi}_b(\mathbf{x})),$
b) $k(\mathbf{x},\mathbf{v}) = f k_a(\mathbf{x},\mathbf{v}),\ f > 0$	$\boldsymbol{\phi}(\mathbf{x}) = \sqrt{f}\boldsymbol{\phi}_a(\mathbf{x})$
c) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) k_b(\mathbf{x},\mathbf{v})$	$\phi_m(\mathbf{x}) = \phi_{ai}(\mathbf{x})\phi_{bj}(\mathbf{x})$
d) $k(\mathbf{x},\mathbf{v}) = \mathbf{x}^T A \mathbf{v}$, A positive semi-definite	$\boldsymbol{\phi}(\mathbf{x}) = L^T \mathbf{x}$, where $A = LL^T$.
e) $k(\mathbf{x},\mathbf{v}) = f(\mathbf{x}) f(\mathbf{v}) k_a(\mathbf{x},\mathbf{v})$	$\phi(\mathbf{x}) = f(\mathbf{x})\phi_a(\mathbf{x})$

Q: How would you prove that the "Gaussian kernel" is a valid kernel?

A: Expand the Euclidean norm as follows:

$$\exp\left(-\frac{||\vec{u}-\vec{v}||_2^2}{2\sigma^2}\right) = \exp\left(-\frac{||\vec{u}||_2^2}{2\sigma^2}\right)\exp\left(-\frac{||\vec{v}||_2^2}{2\sigma^2}\right)\exp\left(\frac{\vec{u}\cdot\vec{v}}{\sigma^2}\right)$$

Then, apply (e) from above

To see that this is a kernel, use the Taylor series expansion of the exponential, together with repeated application of (a), (b), and (c):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

The feature mapping is infinite dimensional!

Kernel Methods (cont'd)

- Given the huge feature space with kernels, should we worry about overfitting?
  - SVM objective seeks a solution with a large margin, as theory says this leads to good generalization.

- Everything overfits sometimes!!

- We can control by:
  - Setting *C*
  - Choosing a better Kernel
  - Varying parameters of the Kernel (e.g. width of Gaussian).

Summary

- How the maximal margin classifier works for datasets in which two classes are separable by a linear boundary.
- How the support vector classifier, which extends the maximal margin classifier, works with overlapping classes.
- Support vector machines, which extend support vector classifiers to accommodate non-linear class boundaries.
- How kernel methods are used with support vector machines.