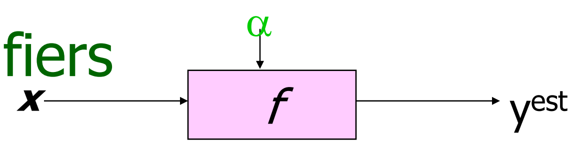
Support Vector Machines

Ronald J. Williams
CSG220
Spring 2007

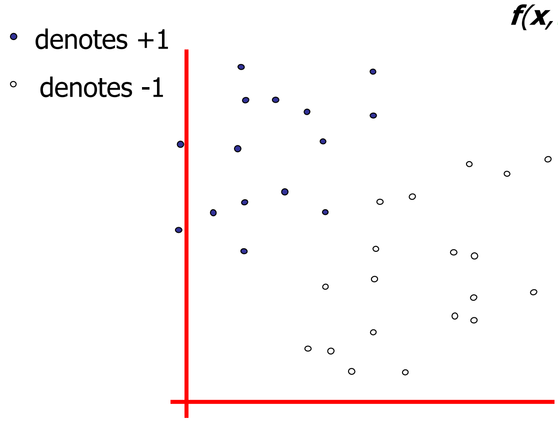
Adapted from the Andrew Moore tutorial of the same name

Note to other teachers and users of these slides. Andrew would be delighted if you found this source material useful in giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. PowerPoint originals are available. If you make use of a significant portion of these slides in your own lecture, please include this message, or the following link to the source repository of Andrew's tutorials: http://www.cs.cmu.edu/~awm/tutorials . Comments and corrections gratefully received.

Linear Classifiers

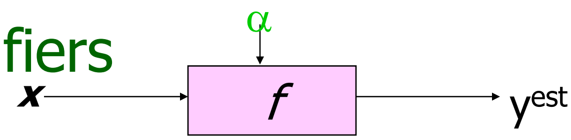


$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w}\cdot\mathbf{x} + b)$

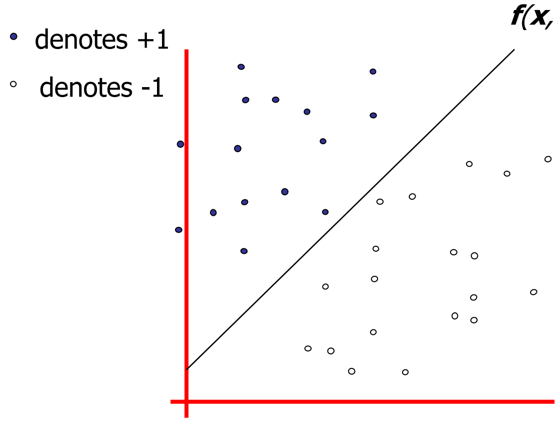


How would you classify this data?

Linear Classifiers

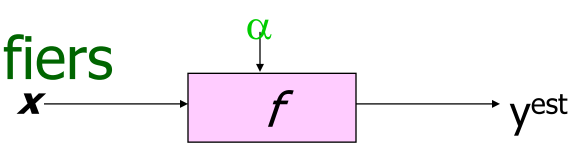


$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w}\cdot\mathbf{x} + b)$

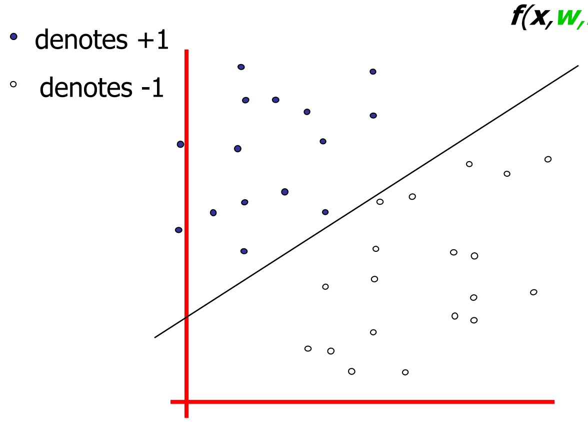


How would you classify this data?

Linear Classifiers

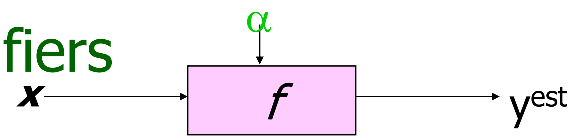


$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w}\cdot\mathbf{x} + b)$



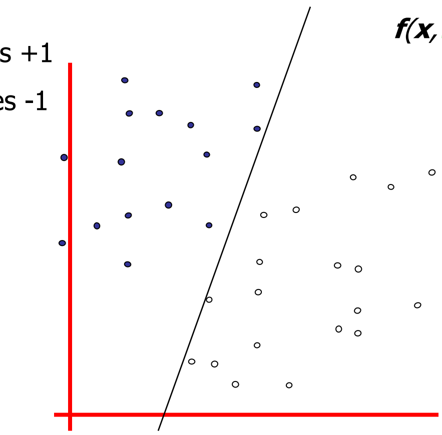
How would you classify this data?

Linear Classifiers



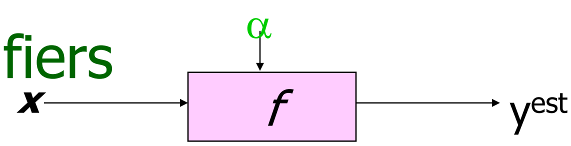
$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w} \cdot \mathbf{x} + b)$

- denotes +1
- denotes -1



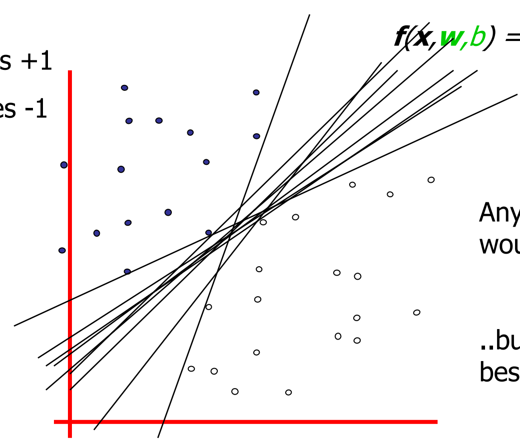
How would you classify this data?

Linear Classifiers



$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w} \cdot \mathbf{x} + b)$

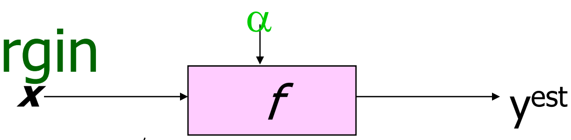
- denotes +1
- denotes -1



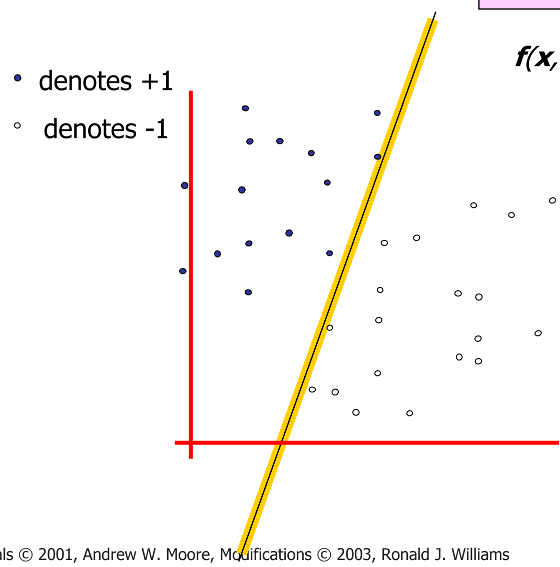
Any of these would be fine..

..but which is best?

Classifier Margin

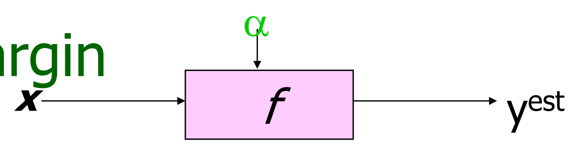


$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w}\cdot\mathbf{x} + b)$

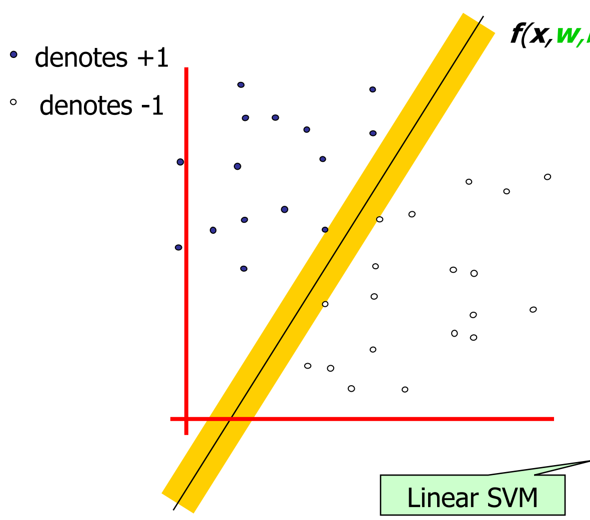


Define the margin of a linear classifier as the perpendicular distance from the decision boundary to the nearest datapoint.

Maximum Margin



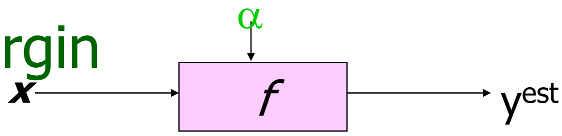
$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w}\cdot\mathbf{x} + b)$



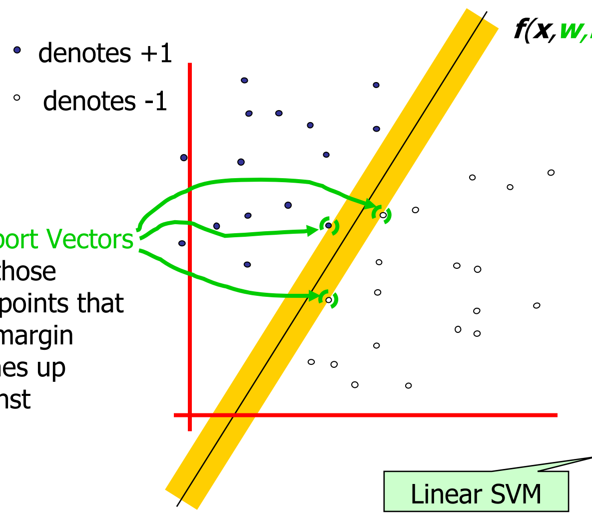
The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w}\cdot\mathbf{x} + b)$

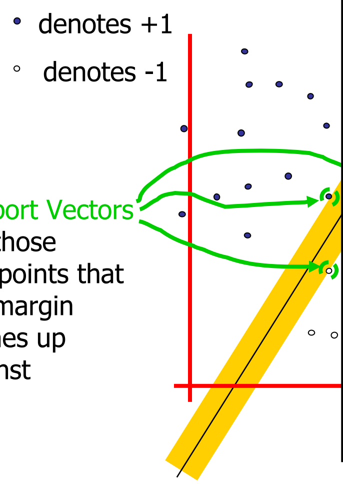


Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

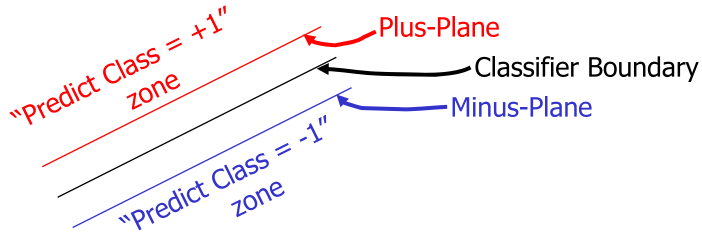
Why Maximum Margin?



Support Vectors are those datapoints that the margin pushes up against

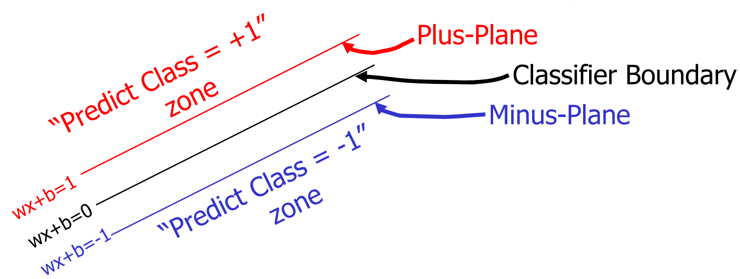
1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4. There's some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
5. Empirically it works very very well.

Specifying a line and margin



- How do we represent this mathematically?
- …in *m* input dimensions?

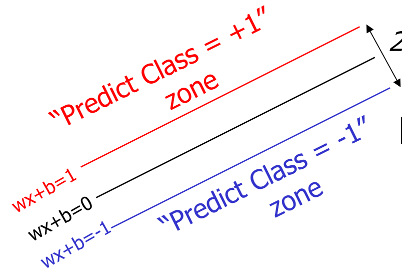
Specifying a line and margin



- Plus-plane = $\{\, \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \,\}$
- Minus-plane = $\{\, \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \,\}$

Classify as..	+1	if	$\mathbf{w} \cdot \mathbf{x} + b >= 1$
	-1	if	$\mathbf{w} \cdot \mathbf{x} + b <= -1$
	Universe explodes	if	$-1 < \mathbf{w} \cdot \mathbf{x} + b < 1$

Computing the margin width



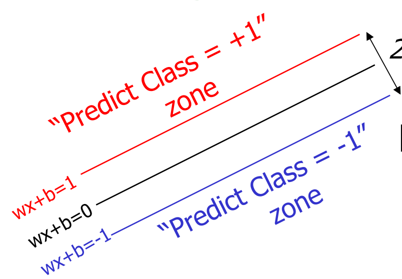
2M where *M* = Margin Width

How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

Claim: The vector **w** is perpendicular to the Plus Plane. Why?

Computing the margin width



2M where *M* = Margin Width

How do we compute *M* in terms of ***w*** and *b*?

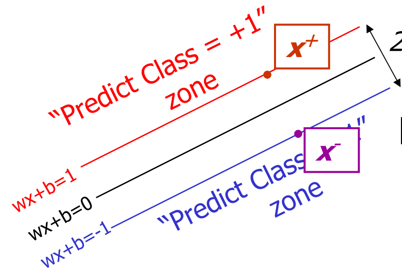
- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

Claim: The vector **w** is perpendicular to the Plus Plane. Why?

Let **u** and **v** be two vectors on the Plus Plane. What is $\mathbf{w} \cdot (\mathbf{u} - \mathbf{v})$?

And so of course the vector **w** is also perpendicular to the Minus Plane

Computing the margin width



2M where *M* = Margin Width

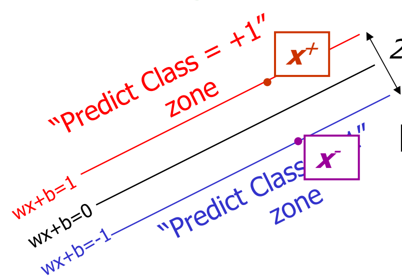
How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\mathbf{x}^-$ be any point on the minus plane
- Let $\mathbf{x}^+$ be the closest plus-plane-point to $\mathbf{x}^-$.

Any location in R^m: not necessarily a datapoint

 Support Vector Machines: Slide 15

Computing the margin width



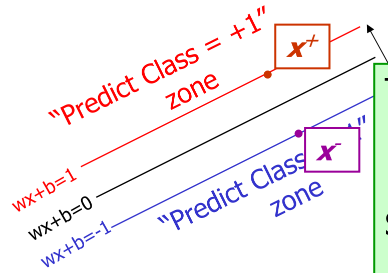
2M where *M* = Margin Width

How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\mathbf{x}^-$ be any point on the minus plane
- Let $\mathbf{x}^+$ be the closest plus-plane-point to $\mathbf{x}^-$.
- Claim: $\mathbf{x}^+ = \mathbf{x}^- + \lambda\, \mathbf{w}$ for some value of λ. Why?

 Support Vector Machines: Slide 16

Computing the margin width



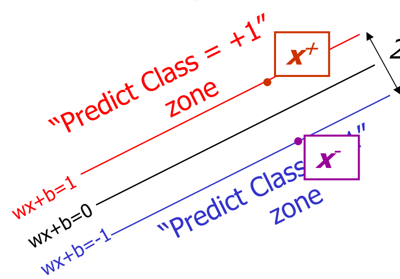
2M where *M* = Margin Width

The line from $\boldsymbol{x}^-$ to $\boldsymbol{x}^+$ is perpendicular to the planes.

So to get from $\boldsymbol{x}^-$ to $\boldsymbol{x}^+$ travel some distance in direction $\boldsymbol{w}$.

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\boldsymbol{x}^-$ be any point on the minus plane
- Let $\boldsymbol{x}^+$ be the closest plus-plane-point to $\boldsymbol{x}^-$.
- Claim: $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$ for some value of λ. Why?

Computing the margin width



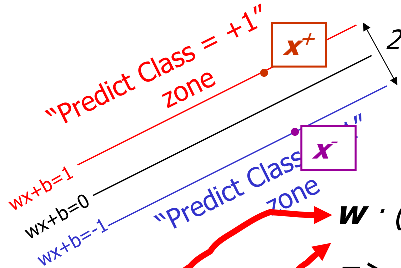
2M where *M* = Margin Width

What we know:

- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $||\boldsymbol{x}^+ - \boldsymbol{x}^- || = 2M$

It's now easy to get *M* in terms of $\boldsymbol{w}$ and *b*

Computing the margin width



2M where *M* = Margin Width

What we know:

- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = 2M$

It's now easy to get *M* in terms of ***w*** and *b*

$\boldsymbol{w} \cdot (\boldsymbol{x}^- + \lambda \boldsymbol{w}) + b = 1$

=>

$\boldsymbol{w} \cdot \boldsymbol{x}^- + b + \lambda \boldsymbol{w} \cdot \boldsymbol{w} = 1$

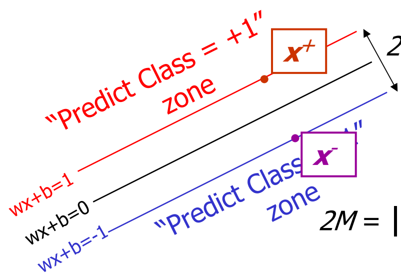
=>

$-1 + \lambda \boldsymbol{w} \cdot \boldsymbol{w} = 1$

=>

$$\lambda = \frac{2}{\|\mathbf{w}\|^2}$$

Computing the margin width



$2M = \frac{2}{\|\mathbf{w}\|}$ so $M = \frac{1}{\|\mathbf{w}\|}$

What we know:

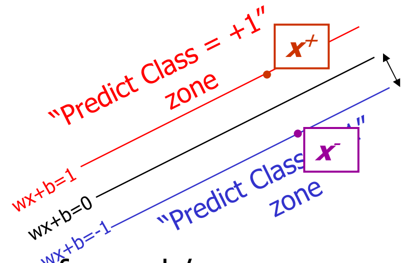
- $\boldsymbol{w} . \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} . \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = 2M$
- $\lambda = \frac{2}{\|\mathbf{w}\|^2}$

$$2M = |\boldsymbol{x}^+ - \boldsymbol{x}^-| = |\lambda \boldsymbol{w}|$$

$$= \lambda \|\mathbf{w}\| = \frac{2}{\|\mathbf{w}\|^2}\|\mathbf{w}\| = \frac{2}{\|\mathbf{w}\|}$$

$$\text{Therefore } M = \frac{1}{\|\mathbf{w}\|}$$

Learning the Maximum Margin Classifier



M = Margin Width = $\frac{1}{\|\mathbf{w}\|}$

Given a guess of ***w*** and *b* we can

- Compute whether all data points in the correct half-planes
- Compute the width of the margin

So now we just need to write a program to search the space of **w**'s and *b*'s to find the widest margin that matches all the datapoints. *How?*

Gradient descent? Simulated Annealing? Matrix Inversion? EM? Newton's Method?

Learning via Quadratic Programming

- QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.

Quadratic Programming

Find $\arg\max_{\mathbf{u}} \quad c + \mathbf{d}^T\mathbf{u} + \dfrac{\mathbf{u}^T R\mathbf{u}}{2}$ ⟵ Quadratic criterion

Subject to

$$\left.\begin{aligned} a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m &\le b_1 \\ a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m &\le b_2 \\ &\vdots \\ a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m &\le b_n \end{aligned}\right\} \text{n additional linear inequality constraints}$$

And subject to

$$\left.\begin{aligned} a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m &= b_{(n+1)} \\ a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m &= b_{(n+2)} \\ &\vdots \\ a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m &= b_{(n+e)} \end{aligned}\right\} \text{e additional linear equality constraints}$$

Quadratic Programming

Find $\arg\max_{\mathbf{u}} \quad c + \mathbf{d}^T\mathbf{u} + \dfrac{\mathbf{u}^T R\mathbf{u}}{2}$ ⟵ Quadratic criterion

Subject to

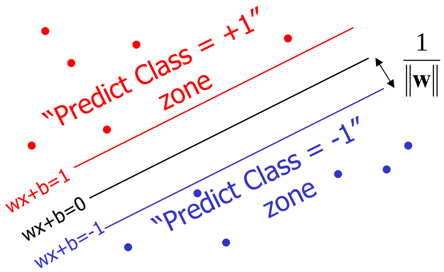
There exist algorithms for finding such constrained quadratic optima much more efficiently and reliably than gradient ascent. Really good news: There are no non-global optima in QP problems.

(But these algorithms are very fiddly…you probably don't want to write one yourself)

And subject to

e additional linear equality constraints

Learning the Maximum Margin Classifier



Given guess of ***w*** , *b* we can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

Assume *R* datapoints $(\mathbf{x}_k, y_k)$ where $y_k = \pm 1$

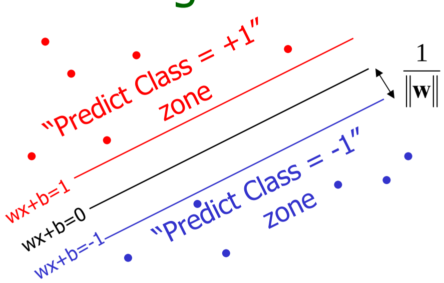
What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

 Support Vector Machines: Slide 25

Learning the Maximum Margin Classifier



Given guess of ***w*** , *b* we can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

Assume *R* datapoints $(\mathbf{x}_k, y_k)$ where $y_k = \pm 1$

What should our quadratic optimization criterion be?

Minimize $\|\mathbf{w}\|^2 = \mathbf{w} \cdot \mathbf{w}$

How many constraints will we have? *R*

What should they be?

$\mathbf{w} \cdot \mathbf{x}_k + b \geq 1 \text{ if } y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq -1 \text{ if } y_k = -1$

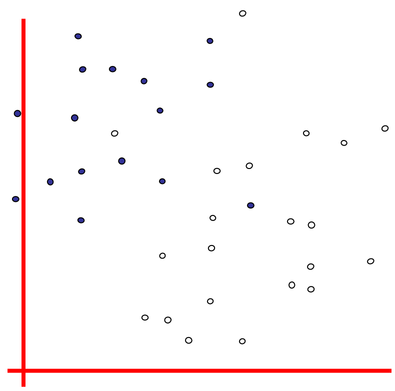
 Support Vector Machines: Slide 26

Uh-oh!

This is going to be a problem!

What should we do?

- denotes +1
- denotes -1



Uh-oh!

This is going to be a problem!

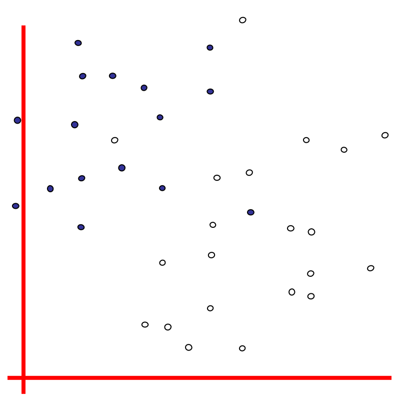
What should we do?

Idea 1:

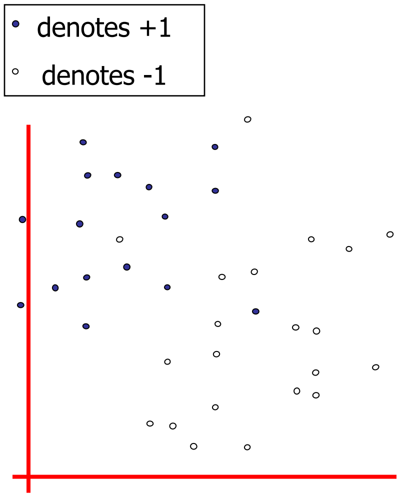
Find minimum $\mathbf{w'w}$, while minimizing number of training set errors.

Problemette: Two things to minimize makes for an ill-defined optimization

- denotes +1
- denotes -1



Uh-oh!



This is going to be a problem!

What should we do?

Idea 1.1:

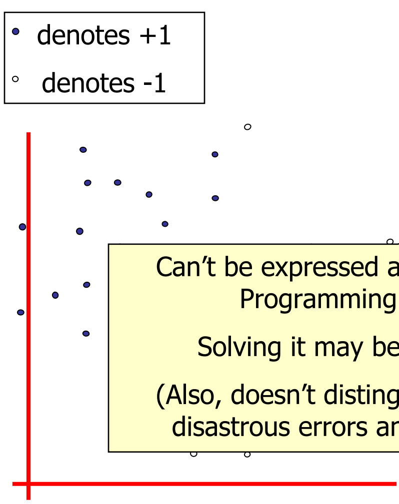
Minimize

$\boldsymbol{w \cdot w} + C\,(\#\text{train errors})$

Tradeoff parameter

There's a serious practical problem that's about to make us reject this approach. Can you guess what it is?

Uh-oh!



This is going to be a problem!

What should we do?

Idea 1.1:

Minimize

$\boldsymbol{w \cdot w} + C\,(\#\text{train errors})$

Tradeoff parameter

Can't be expressed as a Quadratic Programming problem.

Solving it may be too slow.

(Also, doesn't distinguish between disastrous errors and near misses)

So… any other ideas?

Uh-oh!

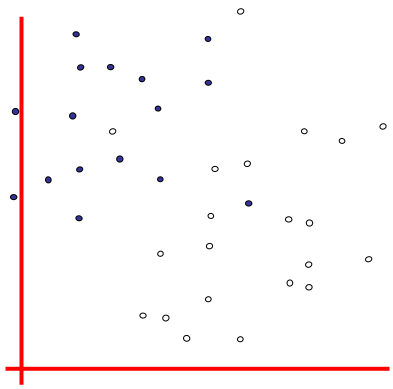
This is going to be a problem!

What should we do?

Idea 2.0:

Minimize
w·w *+ C (distance of error points to their correct place)*

• denotes +1
◦ denotes -1



Uh-oh!

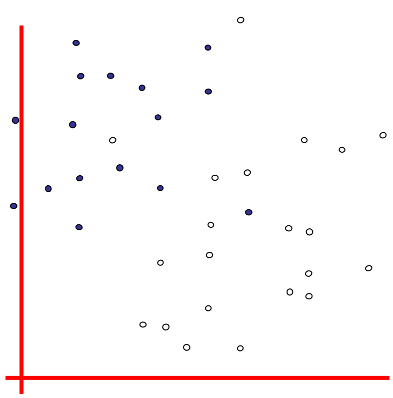
This is going to be a problem!

What should we do?

Idea 2.0:

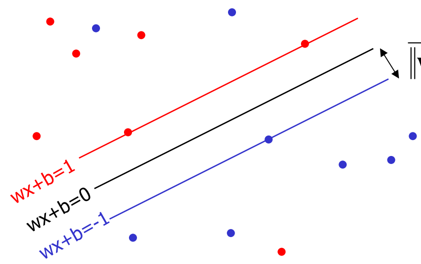
Minimize
w·w *+ C (distance of error points to their correct place)*

• denotes +1
◦ denotes -1



Called soft-margin SVM

Learning Maximum Margin with Noise



$\frac{1}{\|\mathbf{w}\|}$

Given guess of $\boldsymbol{w}$, b we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

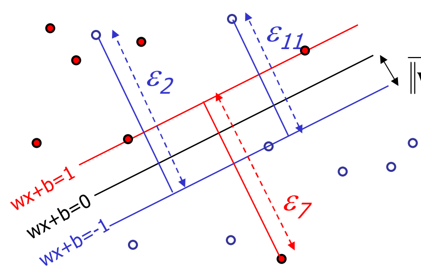
Assume R datapoints $(\boldsymbol{x}_k, y_k)$ where $y_k = \pm 1$

What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Learning Maximum Margin with Noise



$\frac{1}{\|\mathbf{w}\|}$

Given guess of $\boldsymbol{w}$, b we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume R datapoints $(\boldsymbol{x}_k, y_k)$ where $y_k = \pm 1$

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

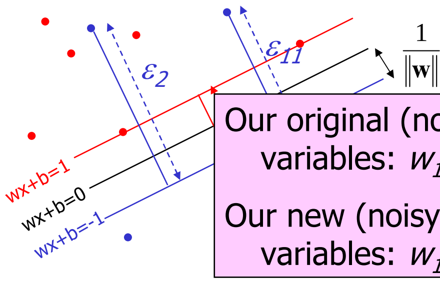
How many constraints will we have? R

What should they be?

$\boldsymbol{w} \cdot \boldsymbol{x}_k + b \geq 1-\varepsilon_k$ if $y_k = 1$

$\boldsymbol{w} \cdot \boldsymbol{x}_k + b \leq -1+\varepsilon_k$ if $y_k = -1$

Learning Maximum Margin with Noise



m = # input dimensions

Given gu… we can

- Compute sum of distances …

Our original (noiseless data) QP had $m+1$ variables: $w_1, w_2, \ldots w_m$, and b.

Our new (noisy data) QP has $m+1+R$ variables: $w_1, w_2, \ldots w_m, b, \varepsilon_k, \varepsilon_1, \ldots \varepsilon_R$

R= # records

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

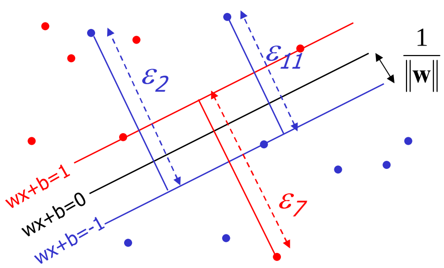
How many constraints will we have? *R*

What should they be?

$\mathbf{w} \cdot \mathbf{x}_k + b \geq 1-\varepsilon_k$ if $y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq -1+\varepsilon_k$ if $y_k = -1$

Learning Maximum Margin with Noise



Given guess of $\mathbf{w}$, b we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints $(\mathbf{x}_k, y_k)$ where $y_k = \pm 1$

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

How many constraints will we have? *R*

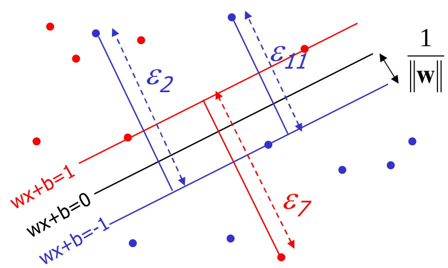
What should they be?

$\mathbf{w} \cdot \mathbf{x}_k + b \geq 1-\varepsilon_k$ if $y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq -1+\varepsilon_k$ if $y_k = -1$

There's a bug in this QP. Can you spot it?

Learning Maximum Margin with Noise



Given guess of ***w*** , *b* we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints $(\mathbf{x}_k, y_k)$ where $y_k = \pm 1$

What should our quadratic optimization criterion be?

Minimize $$\frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

How many constraints will we have? *2R*

What should they be?

$\mathbf{w} \cdot \mathbf{x}_k + b \geq 1-\varepsilon_k$ if $y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq -1+\varepsilon_k$ if $y_k = -1$

$\varepsilon_k \geq 0$ for all k

An Equivalent QP

Maximize $$\sum_{k=1}^{R}\alpha_k + \sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl} \quad \text{where} \quad Q_{kl} = y_k y_l(\mathbf{x}_k \cdot \mathbf{x}_l)$$

Subject to these constraints: $$0 \leq \alpha_k \leq C \quad \forall k \qquad \sum_{k=1}^{R}\alpha_k y_k = 0$$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R}\alpha_k y_k \mathbf{x}_k$$

$$b = y_i - \mathbf{x}_i \cdot \mathbf{w}$$

where $0 < \alpha_i < C$

Then classify with:

$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w} \cdot \mathbf{x} + b)$

An Equivalent QP

Maximize $\sum_{k=1}^{R} \alpha_k + \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k \qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

$$b = y_i - \mathbf{x}_i \cdot \mathbf{w}$$

where $0 < \alpha_i < C$

Datapoints with $\alpha_k > 0$ will be the support vectors

..so this sum only needs to be over the support vectors.

$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w} \cdot \mathbf{x} + b)$

An Equivalent QP

Why did I tell you about this equivalent QP? (Dual QP)

- It's a formulation that QP packages can optimize more quickly
- And especially because of further jaw-dropping developments you're about to learn.

But first ...

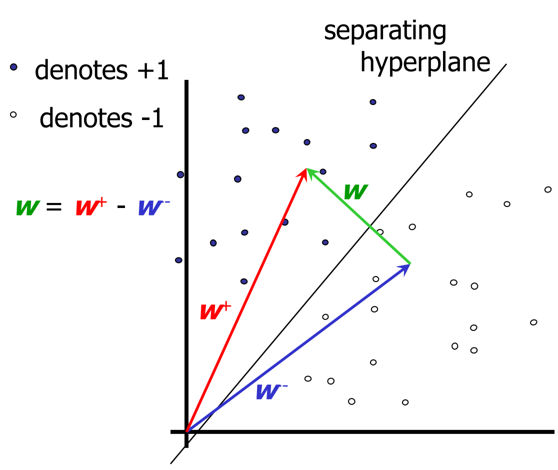
- Let's compare the two QP formulations
  - Original solves for *m+1+R* numbers
    - weights ***w*** and bias *b*
    - slack variables ε_k (one per data point)
  - Dual solves for *R+1* numbers
    - bias b and one α_k per data point
- Consider

m = dimension of ***w***

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

  - Expresses ***w*** in terms of a new basis consisting of the instance vectors $\boldsymbol{x}_k$
  - What's this all about?

Using the instances as a basis



$$I^+ = \{i \mid y_i = +1\}$$

$$I^- = \{i \mid y_i = -1\}$$

$$\mathbf{w}^+ = \sum_{i \varepsilon I^+} \alpha_i \mathbf{x}_i \,, \quad \alpha_i \geq 0$$

$$\mathbf{w}^- = \sum_{i \varepsilon I^-} \alpha_i \mathbf{x}_i \,, \quad \alpha_i \geq 0$$

$$\begin{aligned} \mathbf{w} &= \mathbf{w}^+ - \mathbf{w}^- \\ &= \sum_{i \varepsilon I^+} \alpha_i \mathbf{x}_i - \sum_{i \varepsilon I^-} \alpha_i \mathbf{x}_i \\ &= \sum_i \alpha_i y_i \mathbf{x}_i \end{aligned}$$

Obviously, coefficients not unique in general

Perceptron algorithm: dual version

- Even our old familiar perceptron algorithm can be expressed using the instances as a basis
- Let's review the original version ...

Perceptron algorithm revisited

$\mathbf{w} \leftarrow \mathbf{0}$

$b \leftarrow 0$

Repeat

For $i = 1$ to R

If $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 0$

$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$

$b \leftarrow b + \eta y_i$

Until no errors

A concise way to test for incorrect classification

Classify arbitrary $\boldsymbol{x}$ according to $\mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$

Perceptron algorithm: dual version

- Notice that ***w*** can be written as

$$\eta y_{i_1}\mathbf{x}_{i_1} + \eta y_{i_2}\mathbf{x}_{i_2} + \cdots + \eta y_{i_m}\mathbf{x}_{i_m}$$

where i_1, i_2, ..., i_m are the indices of the succession of training data for which errors occur.

- Written more concisely,

$$\mathbf{w} = \sum_{i=i_1}^{i_m} \eta y_i \mathbf{x}_i$$

- Similarly,

$$b = \sum_{i=i_1}^{i_m} \eta y_i$$

Perceptron algorithm: dual form

- Note that the same *i* may occur multiple times among the indices i_1, i_2, ..., i_m representing the succession of training data causing weight changes (i.e., one training pattern may lead to multiple weight changes as the algorithm proceeds).
- Regrouping the sums according to the indexing of the training data, we get

$$\mathbf{w} = \sum_{i=1}^{R} \alpha_i \eta y_i \mathbf{x}_i \quad \text{and} \quad b = \sum_{i=1}^{R} \alpha_i \eta y_i$$

where α_i is a nonnegative integer counting the number of times the training pattern $(\mathbf{x}_i, y_i)$ contributed to a weight correction

Perceptron algorithm: dual form

Classification of an arbitrary ***x*** is then given by

$$\operatorname{sgn}(\mathbf{w}\cdot\mathbf{x}+b) = \operatorname{sgn}\left(\left(\sum_{i=1}^{R}\alpha_i\eta y_i\mathbf{x}_i\right)\cdot\mathbf{x}+\sum_{i=1}^{R}\alpha_i\eta y_i\right)$$

$$= \operatorname{sgn}\left(\sum_{i=1}^{R}\alpha_i\eta y_i\mathbf{x}_i\cdot\mathbf{x}+\sum_{i=1}^{R}\alpha_i\eta y_i\right)$$

$$= \operatorname{sgn}\left(\sum_{i=1}^{R}\alpha_i y_i\mathbf{x}_i\cdot\mathbf{x}+\sum_{i=1}^{R}\alpha_i y_i\right)$$

Learning the dual form directly

- To learn the quantities $\alpha_1, \alpha_2, \ldots, \alpha_R$ and b directly, replace the body of the inner loop by

$$\alpha_i \leftarrow \alpha_i + 1$$
$$b \leftarrow b + y_i$$

- If the data are not linearly separable, the α_i values grow without bound for misclassified points
- Each α_i is a measure of how much the instance $\boldsymbol{x}_i$ contributes to the classification
- Thus the familiar perceptron algorithm provides one example of how linear classifiers can be expressed in terms of the data vectors themselves

Classification using the dual form

- Given any $\alpha_1, \alpha_2, \ldots, \alpha_R$ and *b* (however obtained), the classification of arbitrary ***x*** is given by

$$\mathrm{sgn}\left(\sum_{i=1}^{R} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \right)$$

- Has the flavor of an instance-based method
    - classify by comparing (in this case computing the dot product of) the new instance with all the training instances
    - but training instances with $\alpha_i = 0$ (e.g., non-support vectors in SVMs) can be ignored

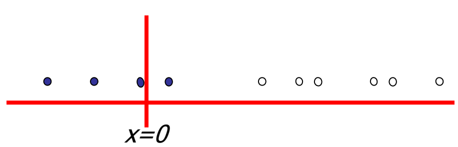
- Now back to SVMs ...

Nonlinear SVMs: The "kernel trick"

- So far we've covered Linear SVMs only
  - Hard-margin (requires linear separability)
  - Soft-margin (tolerates nonlinear separability)
- What if we want to form nonlinear separations?
- Simple solution
  - Map data into a higher-dimensional feature space
  - Find linear separations in this new space
- Magically, all the bad things that would ordinarily happen with this approach don't   That's why it's a trick!
- Let's start with some simple examples ...

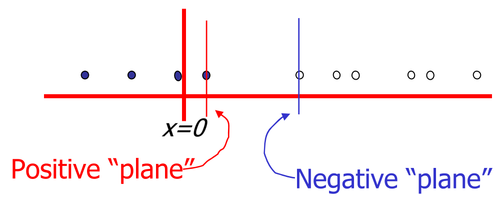
Suppose we're in 1-dimension

What would SVMs do with this data?

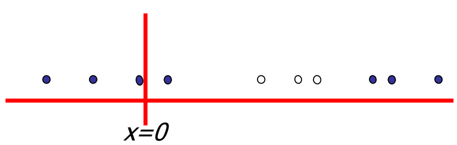


Suppose we're in 1-dimension

Not a big surprise



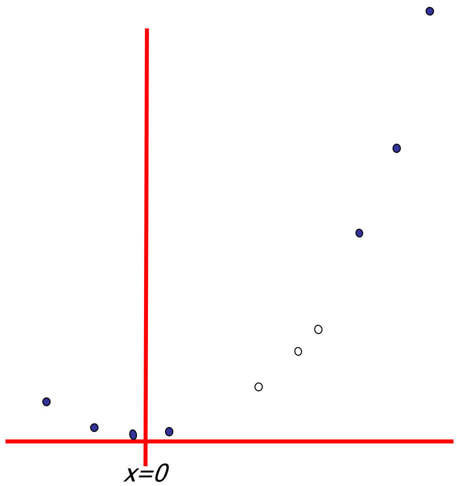
Harder 1-dimensional dataset



That's wiped the smirk off SVM's face.

What can be done about this?

Harder 1-dimensional dataset

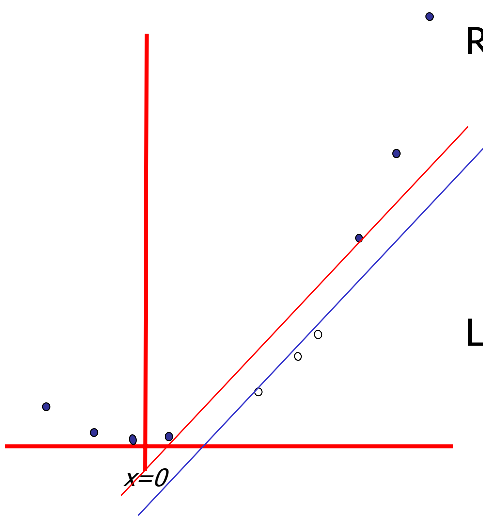


Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Harder 1-dimensional dataset



Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 0 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \text{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\text{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

This is sensible.

Is that the end of the story?

No…here's where the trick comes in!

Quadratic Basis Functions

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ : \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ : \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ : \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ : \\ \sqrt{2}x_1x_m \\ : \\ \sqrt{2}x_{m-1}x_m \end{pmatrix}$$

Constant Term

Linear Terms

Pure Quadratic Terms

Quadratic Cross-Terms

Number of terms (assuming m input dimensions) = (m+2)-choose-2

= (m+2)(m+1)/2

= (as near as makes no difference) $m^2/2$

You may be wondering what those $\sqrt{2}$'s are doing.

- You should be happy that they do no harm
- You'll find out why they're there soon.

QP with basis functions

Maximize $\sum_{k=1}^{R} \alpha_k + \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x}_l))$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\quad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_i - \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{w}$$

$$\text{where } 0 < \alpha_i < C$$

Then classify with:

f(x,w,b) *= sgn(**w** · Φ(**x**) + b)*

All we've done here is to replace every vector ***x*** in the instance space in the earlier description with its corresponding feature vector ***Φ(x)***

QP with basis functions

Maximize $\sum_{k=1}^{R} \alpha_k + \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x}_l))$

We must do $R^2/2$ dot products to get this matrix ready.

Each dot product requires $m^2/2$ additions and mutiplications

The whole thing costs $R^2 m^2 /4$. Yeeks!

...or does it?

Subject to these constraints: $0 \le \alpha_k \le$

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_i - \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{w}$$

$$\text{where } 0 < \alpha_i < C$$

f(x,w,b) *= sgn(**w** · Φ(**x**) + b)*

All we've done here is to replace every vector ***x*** in the instance space in the earlier description with its corresponding feature vector ***Φ(x)***

Quadratic Dot Products

$$
\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =
\begin{pmatrix}
1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ : \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ : \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ : \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ : \\ \sqrt{2}a_1a_m \\ : \\ \sqrt{2}a_{m-1}a_m
\end{pmatrix}
\bullet
\begin{pmatrix}
1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ : \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ : \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ : \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ : \\ \sqrt{2}b_1b_m \\ : \\ \sqrt{2}b_{m-1}b_m
\end{pmatrix}
$$

$$1$$

$$+$$

$$\sum_{i=1}^{m} 2a_ib_i$$

$$+$$

$$\sum_{i=1}^{m} a_i^2b_i^2$$

$$+$$

$$\sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j$$

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =$$

$$1 + 2\sum_{i=1}^{m} a_ib_i + \sum_{i=1}^{m} a_i^2b_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j$$

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

$$(\mathbf{a}\cdot\mathbf{b}+1)^2$$

$$= (\mathbf{a}\cdot\mathbf{b})^2 + 2\mathbf{a}\cdot\mathbf{b} + 1$$

$$= \left(\sum_{i=1}^{m} a_ib_i\right)^2 + 2\sum_{i=1}^{m} a_ib_i + 1$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{m} a_ib_ia_jb_j + 2\sum_{i=1}^{m} a_ib_i + 1$$

$$= \sum_{i=1}^{m} (a_ib_i)^2 + 2\sum_{i=1}^{m}\sum_{j=i+1}^{m} a_ib_ia_jb_j + 2\sum_{i=1}^{m} a_ib_i + 1$$

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =$$
$$1 + 2\sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{m} a_i^2 b_i^2 + \sum_{i=1}^{m} \sum_{j=i+1}^{m} 2 a_i a_j b_i b_j$$

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

$$(\mathbf{a} \cdot \mathbf{b} + 1)^2$$
$$= (\mathbf{a} \cdot \mathbf{b})^2 + 2\mathbf{a} \cdot \mathbf{b} + 1$$
$$= \left(\sum_{i=1}^{m} a_i b_i \right)^2 + 2\sum_{i=1}^{m} a_i b_i + 1$$
$$= \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$
$$= \sum_{i=1}^{m} (a_i b_i)^2 + 2\sum_{i=1}^{m} \sum_{j=i+1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

They're the same!

And this is only O(m) to compute!

QP with basis functions

Maximize $\sum_{k=1}^{R} \alpha_k + \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_l))$

We must do $R^2/2$ dot products to get this matrix ready.

Each dot product now only requires *m* additions and multiplications

Subject to these constraints: $0 \leq \alpha_k \leq$

Then define:

$$\mathbf{w} = \sum_{k} \alpha_k y_k \Phi(\mathbf{x}_k)$$

$$b = y_i - \Phi(\mathbf{x}_i) \cdot \mathbf{w}$$
$$\text{where } 0 < \alpha_i < C$$

Then classify with:

f(x,w,b) *= sgn(**w** · Φ(**x**) + b)*

All we've done here is to replace every vector ***x*** in the instance space in the earlier description with its corresponding feature vector ***Φ(x)***

Higher Order Polynomials

Poly-nomial	$\Phi(x)$	Cost to build Q_{kl} matrix traditionally	Cost if 100 inputs	$\Phi(a) \cdot \Phi(b)$	Cost to build Q_{kl} matrix sneakily	Cost if 100 inputs
Quadratic	All $m^2/2$ terms up to degree 2	$m^2 R^2 /4$	2,500 R^2	$(\mathbf{a} \cdot \mathbf{b}+1)^2$	$m R^2 / 2$	50 R^2
Cubic	All $m^3/6$ terms up to degree 3	$m^3 R^2 /12$	83,000 R^2	$(\mathbf{a} \cdot \mathbf{b}+1)^3$	$m R^2 / 2$	50 R^2
Quartic	All $m^4/24$ terms up to degree 4	$m^4 R^2 /48$	1,960,000 R^2	$(\mathbf{a} \cdot \mathbf{b}+1)^4$	$m R^2 / 2$	50 R^2

QP with basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. *What are they?*

$$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x}_l))$$

constraints:

$$\forall k \qquad \sum_{k=1}^{R} \alpha_k y_k = 0$$

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_i - \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{w}$$

$$\text{where } \; 0 < \alpha_i < C$$

Then classify with:

$f(\mathbf{x},\mathbf{w},b) = sgn(\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}) + b)$

QP with basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. *What are they?*

$$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x}_l))$$

constraints:

$$\forall k \qquad \sum_{k}^{R} \alpha_k y_k = 0$$

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive *(why?)*

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_i - \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{w}$$

$$\text{where } 0 < \alpha_i < C$$

Then classify with:

***f*(*x*, *w*, *b*) = *sgn*(*w* · Φ(*x*) + *b*)**

QP with basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. *What are they?*

$$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x}_l))$$

The use of Maximum Margin magically makes this not a problem

constraints:

$$\forall k \qquad \sum \alpha_k y_k = 0$$

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive *(why?)*

Because each ***w*** · Φ(***x***) (see below) needs 75 million operations. *What can be done?*

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_i - \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{w}$$

$$\text{where } 0 < \alpha_i < C$$

Then classify with:

***f*(*x*, *w*, *b*) = *sgn*(*w* · Φ(*x*) + *b*)**

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

$$Q_{kl} = y_k y_l(\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$$

The use of Maximum Margin magically makes this not a problem

constraints: $\forall k$ $\quad \sum_k \alpha_k y_k = 0$

•The fear of overfitting with this enormous number of terms

•The evaluation phase (doing a set of predictions on a test set) will be very expensive (why?)

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

Because each $\mathbf{w}\cdot\Phi(\mathbf{x})$ (see below) needs 75 million operations. What can be done?

$$\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)\cdot\mathbf{\Phi}(\mathbf{x})$$
$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only *Sm* operations (*S* = # support vectors)

Then classify with:

***f*(*x*,*w*,*b*) = *sgn*(*w* · Φ(*x*) + *b*)**

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

$$Q_{kl} = y_k y_l(\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$$

The use of Maximum Margin magically makes this not a problem

constraints: $\forall k$ $\quad \sum_k \alpha_k y_k = 0$

•The fear of overfitting with this enormous number of terms

•The evaluation phase (doing a set of predictions on a test set) will be very expensive (why?)

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

Because each $\mathbf{w}\cdot\Phi(\mathbf{x})$ (see below) needs 75 million operations. What can be done?

$$\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)\cdot\mathbf{\Phi}(\mathbf{x})$$
$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only *Sm* operations (*S* = # support vectors)

When you see this many callout bubbles on a slide it's time to wrap the author in a blanket, gently take him away and murmur "someone's been at the PowerPoint for too long."

Originals © 2001, Andrew W. Moore, Modifications © 2003, Ronald J. Williams

QP with Quintic basis functions

Maximize $\sum_{k=1}^{R} \alpha_k + \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ wh

Andrew's opinion of why SVMs don't overfit as much as you'd think:

No matter what the basis function, there are really only up to R parameters: α_1, α_2 .. α_R, and usually most are set to zero by the Maximum Margin.

Asking for small **w·w** is like "weight decay" in Neural Nets and like Ridge Regression parameters in Linear regression and like the use of Priors in Bayesian Regression---all designed to smooth the function and reduce overfitting.

Subject to these constraints: $0 \le \alpha_k \le C$

Then define:

$$\mathbf{w} = \sum_k \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x})$$
$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only *Sm* operations (*S* = # support vectors)

Then classify with:

f(***x***,***w***,*b*) = *sgn*(***w*** · Φ(***x***) + *b*)

The kernel trick revealed

- A nonlinear SVM is just a straightforward generalization of the approach in which standard dot products $\mathbf{x} \cdot \mathbf{z}$ are replaced by more general kernel functions

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

  representing dot products in a higher-dimensional space.
- The beauty of the "kernel trick" is that everything can be computed in terms of this kernel function *K.* The nonlinear embedding $\mathbf{\Phi}$ itself never has to be computed or represented.
- Nor is the weight vector ***w*** explicitly represented or used in any computations.

The kernel trick revealed (cont.)

- The bias weight *b* is computed as follows, where *i* is any index for which $0 < \alpha_i < C$:

$$\begin{aligned} b &= y_i - \Phi(\mathbf{x}_i) \cdot \mathbf{w} \\ &= y_i - \Phi(\mathbf{x}_i) \cdot \sum_k \alpha_k y_k \Phi(\mathbf{x}_k) \\ &= y_i - \sum_k \alpha_k y_k \Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_i) \\ &= y_i - \sum_k \alpha_k y_k K(\mathbf{x}_k, \mathbf{x}_i) \end{aligned}$$

- In practice, for numerical stability, this computation is done for all such *i* and *b* is then taken to be the average.

SVM Kernel Functions

- *K(**a**,**b**)=(**a** · **b** +1)^d^* is one example of an SVM Kernel Function
- Beyond polynomials there are other very high dimensional basis functions that can be made practical by finding the right Kernel Function
  - Radial-Basis-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$$

  - Neural-net-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \tanh(\kappa \mathbf{a}.\mathbf{b} - \delta)$$

σ, κ and δ are magic parameters that must be chosen by a model selection method such as CV or VCSRM*

*see VC-dim lecture

- Choosing kernel function = main design decision

VC-dimension of an SVM

- Very very very loosely speaking there is some theory which under some different assumptions puts an upper bound on the VC dimension as

$$\left\lceil \frac{\text{Diameter}}{\text{Margin}} \right\rceil$$

- where
  - *Diameter* is the diameter of the smallest sphere that can enclose all the high-dimensional term-vectors derived from the training set.
  - *Margin* is the smallest margin we'll let the SVM use
- This can be used in SRM (Structural Risk Minimization) for choosing the polynomial degree, RBF σ, etc.
  - But most people just use Cross-Validation

SVM Performance

- Anecdotally they work very very well indeed.
- Example: They are currently the best-known classifier on a well-studied hand-written-character recognition benchmark
- Another Example: Andrew knows several reliable people doing practical real-world work who claim that SVMs have saved them when their other favorite classifiers did poorly.
- There is a lot of excitement and religious fervor about SVMs as of 2001.

Doing multi-class classification

- SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).
- What can be done?
- Answer: with output arity N, learn N SVM's
  - SVM 1 learns "Output==1" vs "Output != 1"
  - SVM 2 learns "Output==2" vs "Output != 2"
  - :
  - SVM N learns "Output==N" vs "Output != N"
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

References

- An excellent tutorial on VC-dimension and Support Vector Machines:

C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998. http://citeseer.nj.nec.com/burges98tutorial.html

- The VC/SRM/SVM Bible:

Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998

What You Should Know

- Linear SVMs
- The definition of a maximum margin classifier
- What QP can do for you (but not how QP solutions are computed)
- How Maximum Margin can be turned into a QP problem
- How we deal with noisy (non-separable) data
- How we permit non-linear boundaries
- How SVM Kernel functions permit us to pretend we're working with ultra-high-dimensional basis-function terms

