Support Vector Machines
CPS 271

Material from:
Lise Getoor,
Andrew Moore http://www.cs.cmu.edu/~awm/tutorials
Tom Dietterich, Andrew Ng, Michael Littman, Rich Maclin

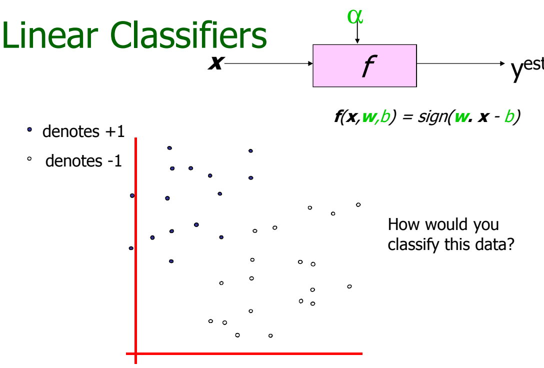
3 Views

- Geometric
  - Maximizing Margin
- Kernel Methods
  - Making nonlinear decision boundaries linear
  - Efficiently!
- Capacity
  - Structural Risk Minimization

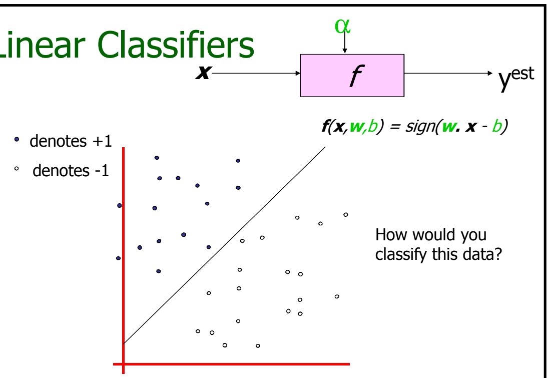
SVM History

- SVM is a classifier derived from statistical learning theory by Vapnik and Chervonenkis
- SVM was first introduced by Boser, Guyon and Vapnik in COLT-92
- SVM became famous when, using pixel maps as input, it gave accuracy comparable to NNs with hand-designed features in a handwriting recognition task
- SVM is closely related to:
  - Kernel machines (a generalization of SVMs), large margin classifiers, reproducing kernel Hilbert space, Gaussian process, Boosting

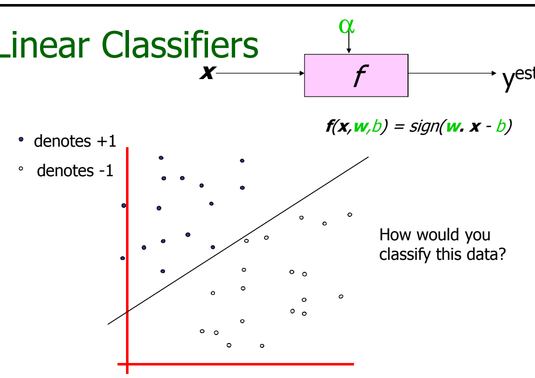
Linear Classifiers



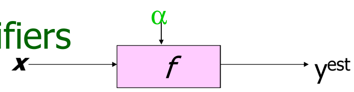
Linear Classifiers



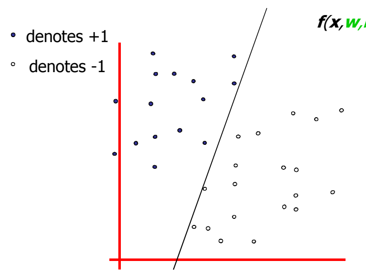
Linear Classifiers



Linear Classifiers

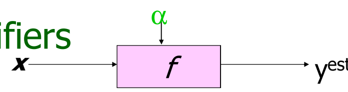


f(**x**,**w**,b) = *sign*(**w**. **x** - b)

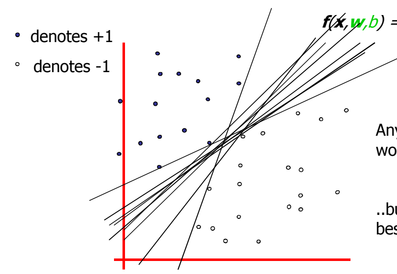


How would you classify this data?

Linear Classifiers



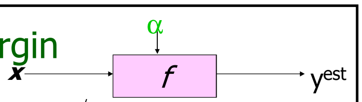
f(**x**,**w**,b) = *sign*(**w**. **x** - b)



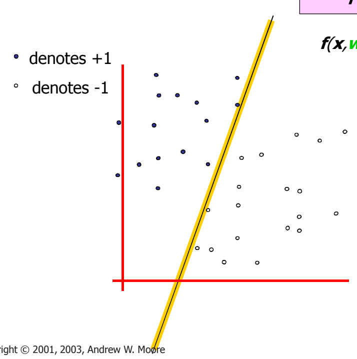
Any of these would be fine..

..but which is best?

Classifier Margin

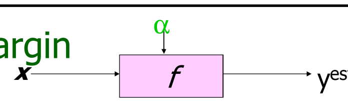


f(**x**,**w**,b) = *sign*(**w**. **x** - b)

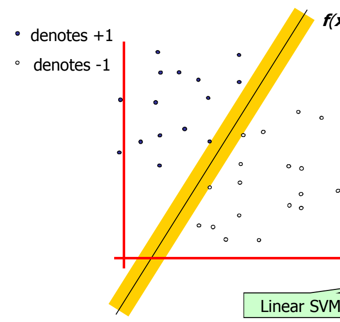


Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



f(**x**,**w**,b) = *sign*(**w**. **x** - b)

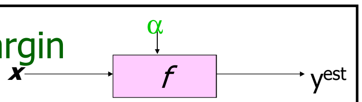


The maximum margin linear classifier is the linear classifier with the maximum margin.

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

Linear SVM

Maximum Margin

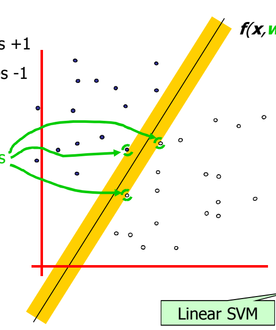


f(**x**,**w**,b) = *sign*(**w**. **x** - b)

denotes +1

denotes -1

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)

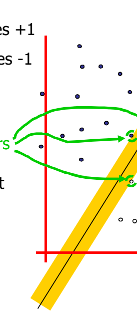
Linear SVM

Why Maximum Margin?

denotes +1

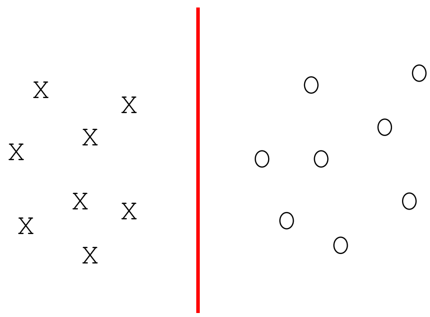
denotes -1

Support Vectors are those datapoints that the margin pushes up against

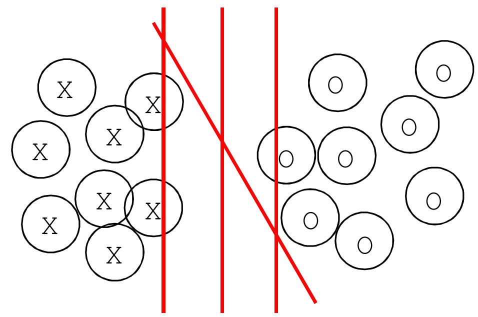


1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4. 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.

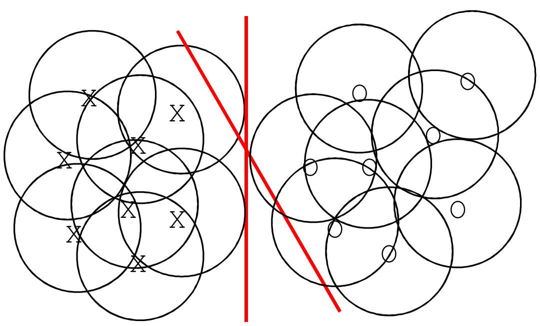
A “Good” Separator



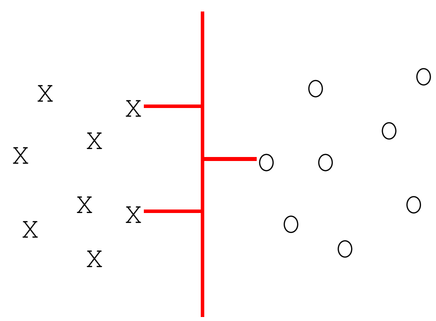
Ruling Out Some Separators



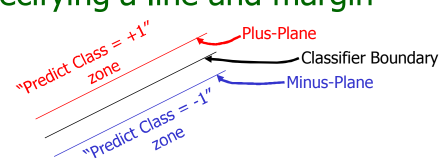
Lots of Noise



Maximizing the Margin

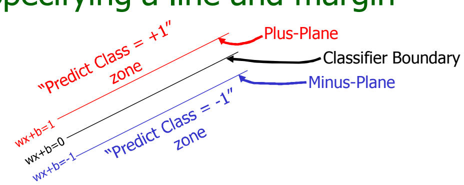


Specifying a line and margin



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

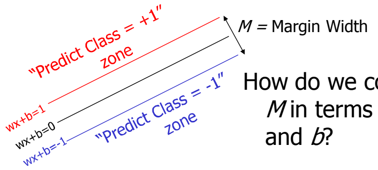
Specifying a line and margin



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

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

Computing the margin width

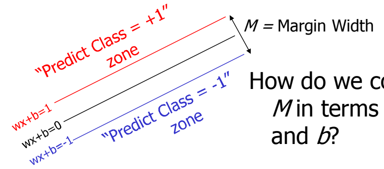


How do we compute M in terms of $\boldsymbol{w}$ and b?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = -1 \}$

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

Computing the margin width



How do we compute M in terms of $\boldsymbol{w}$ and b?

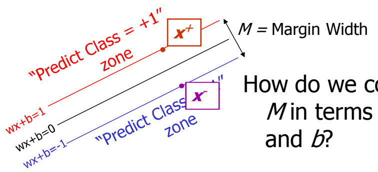
- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = -1 \}$

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

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

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

Computing the margin width

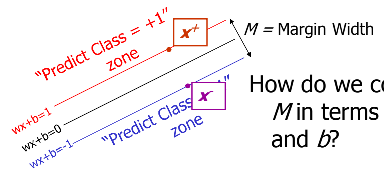


How do we compute M in terms of $\boldsymbol{w}$ and b?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = -1 \}$
- 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 $\boldsymbol{x}^-$.

Any location in R^m: not necessarily a datapoint

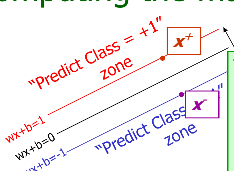
Computing the margin width



How do we compute M in terms of $\boldsymbol{w}$ and b?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = -1 \}$
- 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

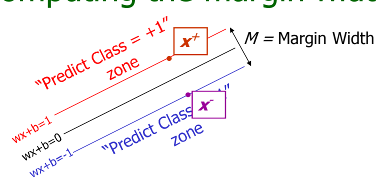


The line from $\boldsymbol{x}^-$ to $\boldsymbol{x}^+$ is perpendicular to the planes.

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

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = -1 \}$
- 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

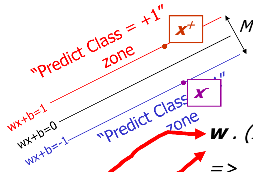


What we know:

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

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

Computing the margin width



M = Margin Width

What we know:

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

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

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

=>

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

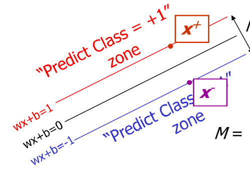
=>

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

=>

$$\lambda = \frac{2}{\mathbf{w}.\mathbf{w}}$$

Computing the margin width



M = Margin Width = $\frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

What we know:

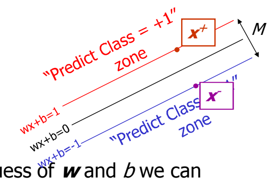
- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = M$
- $\lambda = \frac{2}{\mathbf{w}.\mathbf{w}}$

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

$$= \lambda |\mathbf{w}| = \lambda \sqrt{\mathbf{w}.\mathbf{w}}$$

$$= \frac{2\sqrt{\mathbf{w}.\mathbf{w}}}{\mathbf{w}.\mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

Learning the Maximum Margin Classifier



M = Margin Width = $\frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

Given a guess of $\boldsymbol{w}$ 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?

Useful Stuff

Don't worry... it's **good** for you...

- Linear Programming

find **w**
argmax c·**w**
subject to
$\mathbf{w} \cdot a_i \le b_i$, for i = 1, …, m
$w_j \ge 0$ for j = 1, …, n

There are fast algorithms for solving linear programs including the simplex algorithm and Karmarkar's algorithm

Duality

- Primal
  Find argmax c·**w**
  subject to
  $A\mathbf{w} \le b$
  $w_j \ge 0$ for j = 1, …, n
- Equivalent Dual
  Find argmin b·**y**
  subject to
  $A^T\mathbf{y} \ge c$
  $y_j \ge 0$ for j = 1, …, n

Strong duality result:
If **w*** is an optimal solution for the primal, then the dual has optimal solution **y*** such that:
c·**w*=** b·**y***

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 $\underset{\mathbf{u}}{\arg\max}\ c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$ ⟵ Quadratic criterion

Subject to

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

And subject to

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

Quadratic Programming

Find $\underset{\mathbf{u}}{\arg\max}\ c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$ ⟵ Quadratic criterion

Subject to

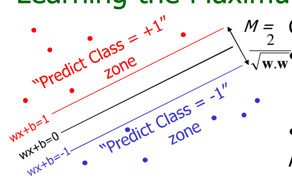
And subject to

$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}$

There exist algorithms for finding such constrained quadratic optima much more efficiently and reliably than gradient ascent.

(But they are very fiddly...you probably don't want to write one yourself)

Learning the Maximum Margin Classifier



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

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

Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\boldsymbol{w.w}$

How many constraints will we have? R

What should they be?

$\boldsymbol{w} \,.\, \boldsymbol{x}_k + b >= 1$ if $y_k = 1$

$\boldsymbol{w} \,.\, \boldsymbol{x}_k + b <= -1$ if $y_k = -1$

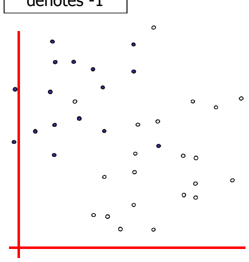
Yay, we're done!!

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!

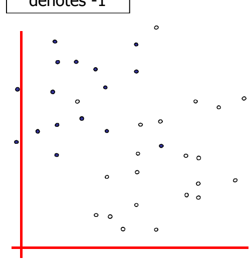
What should we do?

Idea 1:

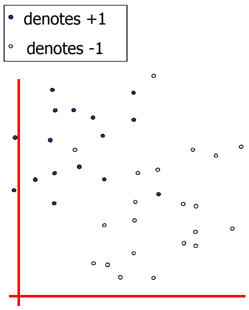
Find minimum $\boldsymbol{w.w}$, while minimizing number of training set errors.

Problem: 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:

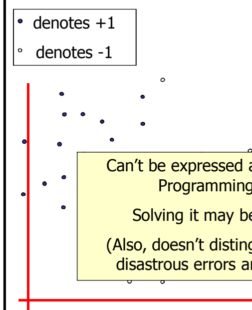
Minimize

w.w *+ C (#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

w.w *+ C (#train errors)*

Tradeoff parameter

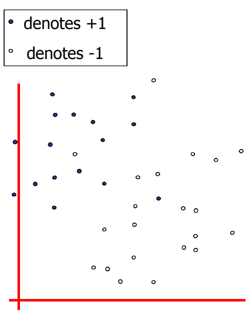
Can't be expressed as a Quadratic Programming problem.

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!

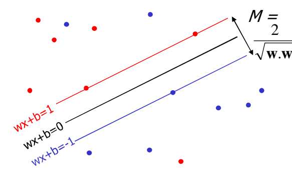
What should we do?

Idea 2.0:

Minimize

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

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

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

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

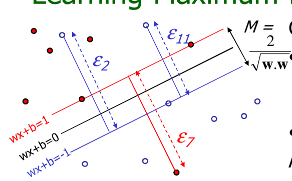
Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

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

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

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

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

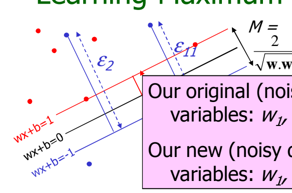
How many constraints will we have? *R*

What should they be?

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

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

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

m = # input dimensions

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

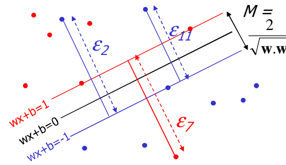
How many constraints will we have? *R*

What should they be?

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

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

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

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, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

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

How many constraints will we have? R

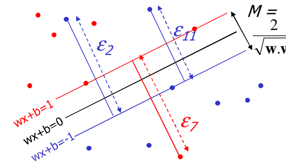
What should they be?

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

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

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

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

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, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

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

How many constraints will we have? $2R$

What should they be?

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

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

$\varepsilon_k >= 0$ for all k

Yay, we're done!!

An Equivalent Dual QP

Maximize $\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l(\mathbf{x}_k.\mathbf{x}_l)$

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:

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

$$b = y_K(1-\varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

An Equivalent Dual QP

Maximize $\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l(\mathbf{x}_k.\mathbf{x}_l)$

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:

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

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

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

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

$$b = y_K(1-\varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

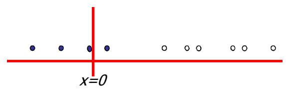
$$\text{where } K = \arg\max_k \alpha_k$$

An Equivalent Dual QP

Why did I tell you about this equivalent QP?
- It's a formulation that QP packages can optimize more quickly
- Because of further developments you're about to learn.

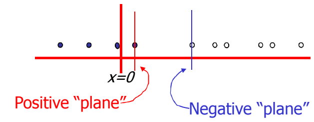
Suppose we're in 1-dimension

What would SVMs do with this data?



Suppose we're in 1-dimension

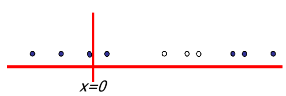
Not a big surprise



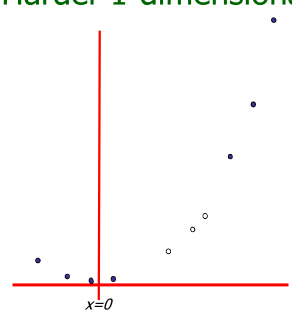
Harder 1-dimensional dataset

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

What can be done about this?



Harder 1-dimensional dataset

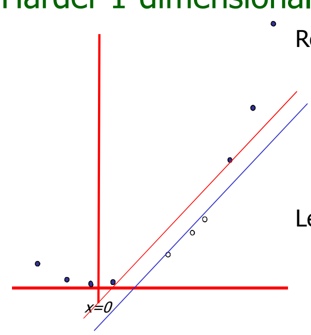


Remember how permitting non-linear basis functions made linear regression so much nicer?

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 1 to q)

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

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

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

This is sensible.

Is that the end of the story?

No…there's one more trick!

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 $\sum_{k=1}^{R} \alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l(\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

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:

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

$$b = y_K(1-\varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where} \quad K = \arg\max_k \alpha_k$$

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w*. ϕ(*x*) - *b*)**

QP with basis functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l(\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

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

Then define:

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

$$b = y_K(1-\varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where} \quad K = \arg\max_k \alpha_k$$

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

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

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

...or does it?

***f*(*x*,*w*,*b*) = *sign*(*w*. ϕ(*x*) - *b*)**

Quadratic Dot Products

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

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

Quadratic Dot Products

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

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

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

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

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

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

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

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

Quadratic Dot Products

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

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

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

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

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

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

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

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

They're the same!

And this is only O(m) to compute!

QP with Quadratic basis functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

We must do R²/2 dot products to get this matrix ready.

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

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

Then define:

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

$$b = y_K(1 - \varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w***. ϕ(***x***) - *b*)

Higher Order Polynomials

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

QP with Quintic basis functions

We must do R²/2 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?*

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

constraints: $\forall k \quad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

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

$$b = y_K(1 - \varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w***. ϕ(***x***) - *b*)

QP with Quintic basis functions

We must do R²/2 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?

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

constraints: $\forall k \quad \sum_{k=1}^{R} \alpha_k y_k = 0$

- •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:

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

$$b = y_K(1 - \varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w***. ϕ(***x***) - *b*)

QP with Quintic basis functions

We must do R²/2 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: $\forall k \quad \sum \alpha_k y_k = 0$

- •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:

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

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

$$b = y_K(1 - \varepsilon_K) - \mathbf{x}_K.\mathbf{w}_K$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w***. ϕ(***x***) - *b*)

QP with Quintic basis functions

We must do R²/2 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: $\forall k \quad \sum \alpha_k y_k = 0$

- •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:

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

Because each **w.** ϕ(**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*) = *sign*(***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: $\forall k$ $\sum \alpha_k y_k = 0$

- 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?)

Then define:

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

Because each $\mathbf{w}.\ \phi(\mathbf{x})$ (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:

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \phi(\mathbf{x}) - b)$

QP with Quintic basis functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ wh

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

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: $\alpha_1, \alpha_2 .. \alpha_R$, 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.

Then define:

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

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

Only Sm operations (S=#support vectors)

Then classify with:

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \phi(\mathbf{x}) - b)$

SVM Kernel Functions

- $K(\mathbf{a},\mathbf{b})=(\mathbf{a} \, . \, \mathbf{b} + 1)^d$ is an 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

Review

Primal Equations

Separating Plane

$\vec{w} \cdot \vec{x} + b = 0$

$\vec{w}$ - weights, $\vec{x}$ - input features,

b - threshold

For all positive examples

$\vec{w} \cdot \vec{x}_{pos} + b = 1$

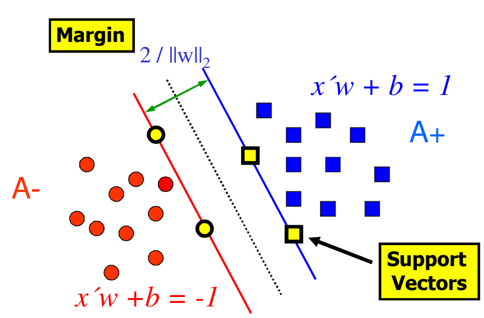
For all negative examples

$\vec{w} \cdot \vec{x}_{neg} + b = -1$

Distance between blue and red planes (the margin)

$\text{margin} = \frac{2}{\|\vec{w}\|}$ ← Euclidean length ("2 norm") of the weight vector

What the Equations Mean



The Primal QP

$$\min_{\vec{w},b} \|\vec{w}\|^2$$

such that

$$\vec{w}\cdot\vec{x}_{pos} + b \geq 1 \quad \text{(for + examples)}$$

$$\vec{w}\cdot\vec{x}_{neg} + b \leq -1 \quad \text{(for – examples)}$$

Note : $\vec{w}, b$ are our adjustable parameters

We can now use existing optimization packages to find a solution to the above (a global optimal soln)

Dealing with Non-Separable Data

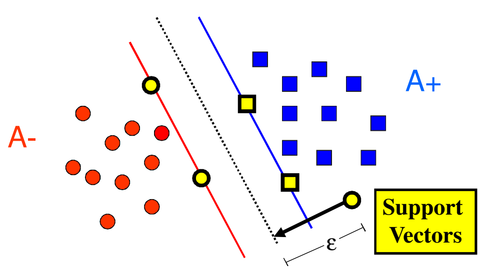
We can add what is called a "slack" variable to each example

This variable can be viewed as:

0 if the example is correctly separated

ε "distance" we need to move example to make it correct (i.e., the distance from its surface)

"Slack" Variables



The Math Program with Slack Variables

$$\min_{\vec{w},\vec{s},\gamma} \|\vec{w}\|^2 + C\|\vec{\varepsilon}\|_1$$

$\vec{w}$ – one for each input feature

$\vec{\varepsilon}$ – one for each example

C – scaling constant

$\|\vec{\varepsilon}\|_1$ – "one norm" - sum of components (all positive)

such that

$$\vec{w}\cdot\vec{x}_{pos_i} + b \geq 1 - \varepsilon_i$$

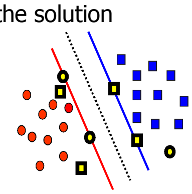
$$\vec{w}\cdot\vec{x}_{neg_j} + b_j \leq -1 + \varepsilon_j$$

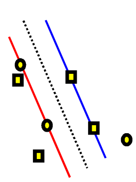
$$\forall_k \varepsilon_k \geq 0$$

This is the "traditional" Support Vector Machine

Why the word "Support"?

- All those examples **on** or on the **wrong side** of the two separating planes are the support vectors
  - We'd get the same answer if we deleted all the non-support vectors!
  - i.e., the "support vectors [examples]" support the solution





But what does a support vector mean?

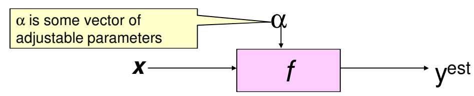
- Support vectors are either:
  - Misclassifications
  - Data points that are just barely within the class (Correct points that could most easily be misclassified)
- In high dimensions, support vectors determine the capacity of the classifier
- Large margins typically involve fewer support vectors
- Intuition (and intuition *only*):
  - Wide margin = lots of room to maneuver
  - Lots of room to maneuver = fewer bends
  - Fewer bends = fewer support vectors

How do we characterize "power"?

- Different machines have different amounts of "power".
- Tradeoff between:
  - More power: Can model more complex classifiers but might overfit.
  - Less power: Not going to overfit, but restricted in what it can model.
- How do we characterize the amount of power?

A learning machine

- A learning machine ***f*** takes an input ***x*** and transforms it, somehow using weights α, into a predicted output y^{est} = +/- 1



Some definitions

- Given some machine **f**
- And under the assumption that all training points (x_k, y_k) were drawn i.i.d from some distribution.
- And under the assumption that future test points will be drawn from the same distribution
- Define

$$R(\alpha) = \text{TESTERR}(\alpha) = E\left[\frac{1}{2}|y - f(x,\alpha)|\right] = \begin{array}{c}\text{Probability of}\\ \text{Misclassification}\end{array}$$

Official terminology

Some definitions

- Given some machine **f**
- And under the assumption that all training points (x_k, y_k) were drawn i.i.d from some distribution.
- And under the assumption that future test points will be drawn from the same distribution
- Define

$$R(\alpha) = \text{TESTERR}(\alpha) = E\left[\frac{1}{2}|y - f(x,\alpha)|\right] = \begin{array}{c}\text{Probability of}\\ \text{Misclassification}\end{array}$$

Official terminology

$$R^{emp}(\alpha) = \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}|y_k - f(x_k,\alpha)| = \begin{array}{c}\text{Fraction Training}\\ \text{Set misclassified}\end{array}$$

R = #training set data points

Vapnik-Chervonenkis dimension

$$\text{TESTERR}(\alpha) = E\left[\frac{1}{2}|y - f(x,\alpha)|\right] \qquad \text{TRAINERR}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}|y_k - f(x_k,\alpha)|$$

- Given some machine **f**, let *h* be its VC dimension.
- *h* is a measure of **f**'s power (*h* does not depend on the choice of training set)
- Vapnik showed that with probability 1-η

$$\text{TESTERR}(\alpha) \le \text{TRAINERR}(\alpha) + \sqrt{\frac{h(\log(2R/h)+1) - \log(\eta/4)}{R}}$$

This gives us a way to estimate the error on future data based only on the training error and the VC-dimension of ***f***

Structural Risk Minimization

- Let $\phi(f)$ = the set of functions representable by f.
- Suppose $\varphi(f_1) \subseteq \varphi(f_2) \subseteq \cdots \varphi(f_n)$
- Then $h(f_1) \le h(f_2) \le \cdots h(f_n)$
- We're trying to decide which machine to use.
- We train each machine and make a table…

$$\text{TESTERR}(\alpha) \le \text{TRAINERR}(\alpha) + \sqrt{\frac{h(\log(2R/h)+1) - \log(\eta/4)}{R}}$$

i	f_i	TRAINERR	VC-Conf	Probable upper bound on TESTERR	Choice
1	f_1				
2	f_2				
3	f_3				☒
4	f_4				
5	f_5				
6	f_6				

SVMs and PAC Learning

- Theorems connect PAC theory to the size of the ***margin***
- Basically, the ***larger*** the margin, the better the expected accuracy
- See, for example, Chapter 4 of *Support Vector Machines* by Christianini and Shawe-Taylor, Cambridge University Press, 2002

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

PAC and the Number of Support Vectors

- The fewer the support vectors, the better the generalization will be
- Recall, non-support vectors are
  - Correctly classified
  - Don't change the learned model if left out of the training set
- So

$$\text{leave-one-out error rate} \leq \frac{\#\,\text{support vectors}}{\#\,\text{training examples}}$$

Understanding LOO

- LOO estimates probability that a classifier trained on n-1 points gets the nth point right
- For largish n, LOO is (sort of) an average of n such draws
- For SVM with k support vectors, n training points
  - At least n-k draws will produce the same classifier
  - At least this many will get the next point right
- Suggests empirical error of our SVM should be at least as low as k/n

Finding Non-Linear Separating Surfaces

- Map inputs into new space

Example: features x_1 x_2
5 4

Example: features x_1 x_2 x_1^2 x_2^2 x_1*x_2
5 4 25 16 20

- Solve SVM program in this new space
  - Computationally complex if many features
  - But a clever trick exists

The Kernel Trick

- Optimization problems often/always have a "primal" and a "dual" representation
  - We just saw the primal formulation
  - The dual formulation is better for the case of a non-linear separating surface

Generalizing the Dot Product

We can generalize

$\text{Dot\_Product}(\vec{x}_i, \vec{x}_j) \equiv \vec{x}_i \cdot \vec{x}_j$

to other "kernel functions"

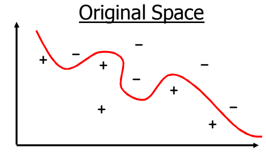
e.g., $K(\vec{x}_i, \vec{x}_j) \equiv (\vec{x}_i \cdot \vec{x}_j)^{\delta}$

An acceptable kernel (usually non-linear) maps the original features into a new space implicitly

- in this new space we're computing a dot product
- we don't need to explicitly know the features in the new space
- usually more efficient than directly converting to new space

Visualizing the Kernel

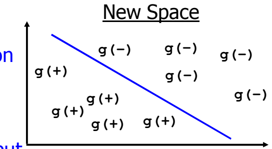
Original Space



Input Space

Separating plane (non-linear here but linear in derived space)

New Space



Derived Feature Space

g() is feature transformation function

process is similar to what hidden units do in ANNs but kernel is user chosen

QP with kernel

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l k(\mathbf{x}_k, \mathbf{x}_l)$

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

Then classify with:

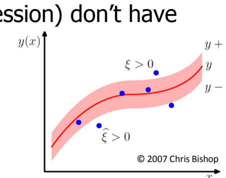
$$f(x, \mathbf{w}, b) = sign\left(\sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k k(x, x_n) + b \right)$$

$$b = y_j (1 - \varepsilon_j) - \sum_{k \text{ s.t. } \alpha_k > 0} \mathrm{k}(x_k, x_j)$$

$$\text{where} \quad j = \arg \max_j \alpha_j$$

Extensions

- Class probabilities
  - Use distance from boundary
  - Fit a logistic sigmoid to output of SVM (icky)
  - Logistic regression variants of SVM exist, but (as with ordinary logistic regression) don't have direct solutions



- Support vector regression
  - Similar to SVM
  - Instead of >1, <-1, add constraints for true target values

Relevance Vector Machine

- Bayesian Version of SVM
- Provides probabilities on outputs
- Tends to produce sparser solutions
- Requires non-linear optimization
- Can be slow

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.

Key SVM Ideas

- Maximize the margin between positive and negative examples (connects to PAC theory)
- Penalize errors in non-separable case
- Only the support vectors contribute to the solution
- Kernels map examples into a new, usually non-linear space
  - We implicitly do dot products in this new space (in the "dual" form of the SVM program)
  - Kernels are a separate idea from SVMs (remember we introduced them for GP), but they combine very nicely with SVMs

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: AWM knows several reliable people doing practical real-world work who claim that SVMs have saved them when their other favorite classifiers did poorly. (REP too)
- There was a lot of excitement and religious fervor about SVMs and Kernel machines in 2004. In 2007, SVMs have cooled off, but they're still pretty neat and useful!
- Despite this, some practitioners are a little skeptical.

SVM Implementations

- Sequential Minimal Optimization, SMO, efficient implementation of SVMs, Platt
  - in Weka
- SVMlight
  - http://svmlight.joachims.org/
- Good implementations will tend to have quadratic run time in the number of data points (may be less of number of support vectors is small)

References

- Tutorial on VC-dimension and Support Vector Machines:

  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

LTUs/Perceptrons Re-Visited

In perceptrons, if classification +1 and -1,

$$\vec{w}_{k+1} = \vec{w}_k + \eta \; y_i \vec{x}_i$$

if the example x_i is currently misclassified

So

$$\vec{w}_{final} = \sum_{i=1}^{\#examples} a_i y_i \vec{x}_i$$

where a_i is some number of times we get $\vec{x}_i$ wrong and change weights

This assumes $\vec{w}_{initial} = \vec{0}$ (all zero)

Dual Form of the Perceptron Learning Rule

output of perceptron $\equiv h(\vec{x}) = \operatorname{sgn}(\vec{w} \cdot \vec{x})$

$$\operatorname{sgn}(z) = \left\langle \begin{array}{ll} +1 & \text{if } z \ge 0 \\ -1 & \text{otherwise} \end{array} \right.$$

$$\text{So } h(\vec{x}) = \operatorname{sgn}\left(\left[\sum_{i=1}^{\#examples} a_i y_i \vec{x}_i\right] \cdot \vec{x}\right)$$

$$= \operatorname{sgn}\left(\sum a_i y_i [\vec{x}_i \cdot \vec{x}]\right)$$

New (i.e., dual) perceptron algorithm:

For each example i

$$\text{if } y_i * \left(\sum_{j=1}^{\#examples} a_j y_j [\vec{x}_j \cdot \vec{x}_i]\right) \le 0 \quad (\text{i.e., predicted}_i \ne \text{actual}_i)$$

$$\text{then } a_i = a_i + 1 \quad (\text{counts errors})$$

Primal versus Dual Space

- Primal – "weight space"
  - Weight features to make output decision

$$h(\vec{x}_{new}) = \text{sgn}(\vec{w} \cdot \vec{x}_{new})$$

- Dual – "training-examples space"
  - Weight distance (which is based on the features) to training examples

$$h(\vec{x}_{new}) = \text{sgn}\left(\sum_{j=1}^{\#examples} \alpha_j y_j [\vec{x}_j \cdot \vec{x}_{new}]\right)$$

Why not use dual perceptrons?

- Perceptrons don't maximize the margin
- No regularization
- Less pressure to produce sparse classifiers
- More risk of overfitting