An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

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

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

Then define:

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

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

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

Then classify with:

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

An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

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

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

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.

(prolly << R)

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

where $K = \arg\max_k \alpha$ ...

$\mathbf{x} + b)$

An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

Maximize $\sum_{k}^{R} \alpha_k$ … $= y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$
α_k

Subject to … constraints … $= 0$

Then … $\mathbf{w} = \sum_{k=1}$ … $+ b)$

$b = y_K (1$ …

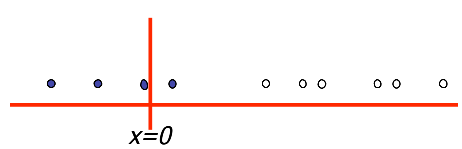
where $K = \arg$ …

Why did I tell you about this equivalent QP?

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

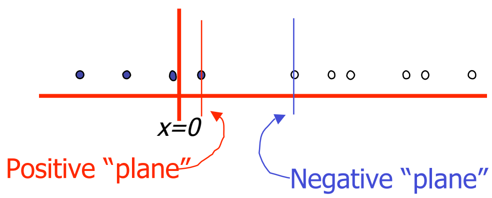
Suppose we're in 1-dimension

What would SVMs do with this data?



Suppose we're in 1-dimension

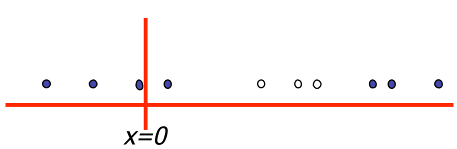
Not a big surprise



Harder 1-dimensional dataset

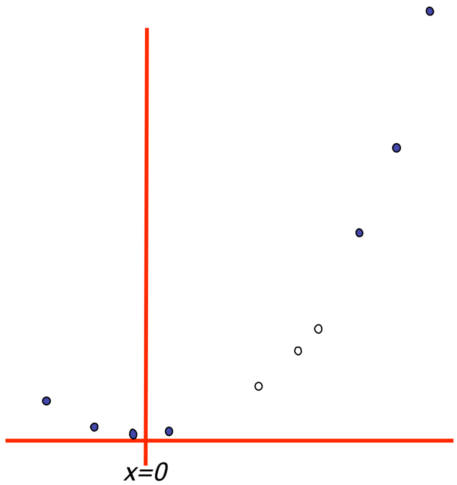
That's wiped the smirk off SVM's face.

What can be done about this?



Doesn't look like slack variables will save us this time…

Harder 1-dimensional dataset



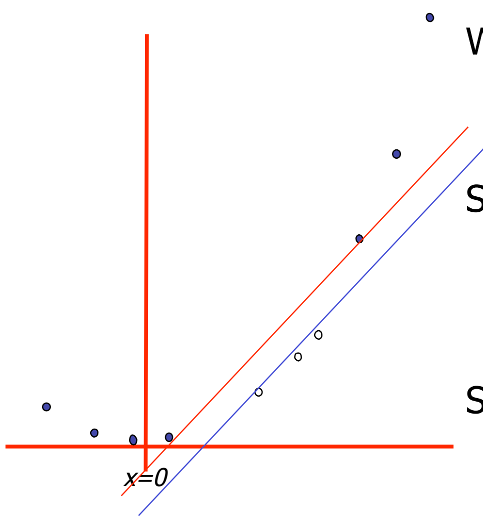
We're going to *make up a new feature.*

Sort of. We'll compute it from the feature(s) we have.

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

New features are sometimes called *basis functions.*

Harder 1-dimensional dataset



We're going to *make up a new feature.*

Sort of. We'll compute it from the feature(s) we have.

Separable! MAGIC!

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

Just put this "augmented" data into our linear SVM.

Common SVM "extra features"

$\boldsymbol{z}_k$ = (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.

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***:

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

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

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

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

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

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!

Higher Order Polynomials

Poly-nomial	$\phi(\boldsymbol{x})$	Cost to build Q_{kl} matrix traditionally	Cost if 100 inputs	$\phi(\boldsymbol{a})\cdot\phi(\boldsymbol{b})$	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 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?*

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

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

Then define:

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

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

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

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?*

- 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?)*

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

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

Then define:

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

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

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

***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?*

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

The use of Maximum Margin magically makes this not a problem

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

constraints:

- 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 $\mathbf{w} \cdot \phi(\mathbf{x})$ (see below) needs 75 million operations. *What can be done?*

Then define:

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

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

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

Then classify with:

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

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?*

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

The use of Maximum Margin magically makes this not a problem

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

constraints:

- 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 $\mathbf{w} \cdot \phi(\mathbf{x})$ (see below) needs 75 million operations. *What can be done?*

Then define:

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

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

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

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

Then classify with:

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

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*

*see last lecture

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.
- Despite this, some practitioners (including your lecturer) are a little skeptical.

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.

  BUT YOU SHOULD PROBABLY READ ALMOST ANYTHING ELSE ABOUT SVMS FIRST.

What You Should Know

- Linear SVMs
- The definition of a maximum margin classifier
- What QP can do for you (but, for this class, you don't need to know how it does it)
- 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 a zillion features

What really happens

- Johnny Machine Learning gets a dataset

- Wants to try SVMs
  - Linear: "Not bad, but I think it could be better."
  - Adjusts C to trade off margin vs. slack
  - Still not satisfied: Tries kernels, typically polynomial. Starts with quadratic, then goes up to about degree 5.

- Johnny goes to Machine Learning conference
  - Johnny: "Wow, a quartic kernel with C=2.375 works great!"
  - Audience member: "Why did you pick those, Johnny?"
  - Johnny: "Cross validation told me to!"