Support Vector Machines

Louis Oliphant
oliphant@cs.wisc.edu
Cs540 section 2

Announcements

- Projects Due Today
  - I'll put links on course website tomorrow.
  - Check out projects Courses before next week
- Presentations next Week
  - 15 minutes total
    - Leave a few minutes for questions
  - 5 teams each day
  - Presentations In order as they appear on website
  - Email me any slides you want to use (or bring your own laptop)
  - Questions on final may be taken from presentations or project web-sites

Annoucements

- Things left in the course:
  - Presentations next week
  - Evaluate each-others projects (week after presentations)
  - 2 more lectures after presentations
- Reading:
  - Chapter 20 section 6 and 7 on Support Vector Machines

Things You Should Know

- In Depth
  - K-NN, Decision Trees, Perceptron, Neural Network, Ensembles, Naïve Bayes, Bayesian Network, K-Means clustering
  - Induction
  - Inference
  - How they Divide up feature space
  - Important aspects of each model

Things You Should Know

- Overview
  - Inductive Logic Programming
    - FOIL
    - PROGOL
    - GOLEM
  - Support Vector Machines
  - Re-enforcement Learning
    - Q-Learning
- Understand the important points of each model
  - When are they used, how are the models more or less expressive
  - Important terms
  - General Idea of how the algorithm works

What is a Support Vector Machine?

- An optimally defined surface
- Typically nonlinear in the input space
- Linear in a higher dimensional space
- Implicitly defined by a kernel function

Acknowledgments: These slides combine and modify ones provided by Andrew Moore (CMU), Glenn Fung (Wisconsin), and Olvi Mangasarian (Wisconsin), and Chuck Dyer (Wisconsin)

What are Support Vector Machines Used For?

- Classification
- Regression and data-fitting
- Supervised and unsupervised learning

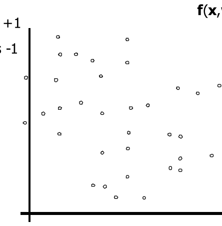
Linear Classifiers

x → f → y

f(**x**,**w**,b) = sign(**w** · **x** + b)

° denotes +1

° denotes -1



How would you classify this data?

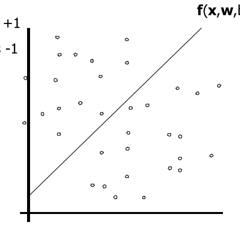
Linear Classifiers

x → f → y

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

° denotes +1

° denotes -1



How would you classify this data?

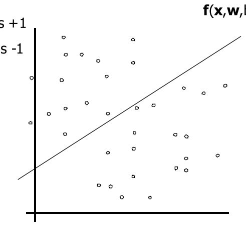
Linear Classifiers

x → f → y

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

° denotes +1

° denotes -1



How would you classify this data?

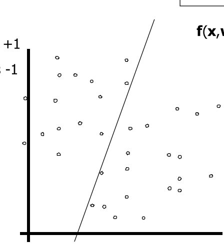
Linear Classifiers

x → f → y

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

° denotes +1

° denotes -1



How would you classify this data?

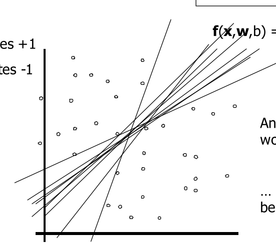
Linear Classifiers

x → f → y

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

° denotes +1

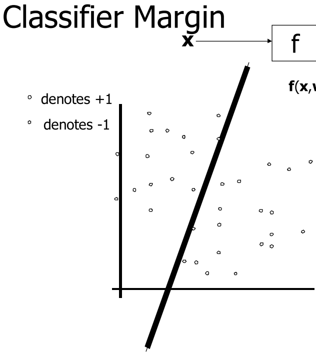
° denotes -1



Any of these would be fine ...

... but which is best?

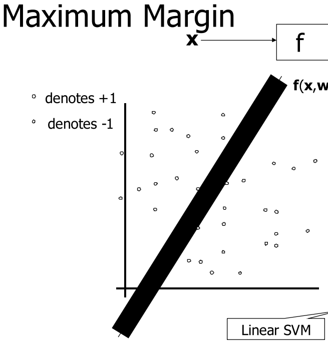
Classifier Margin



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 data point

Maximum Margin

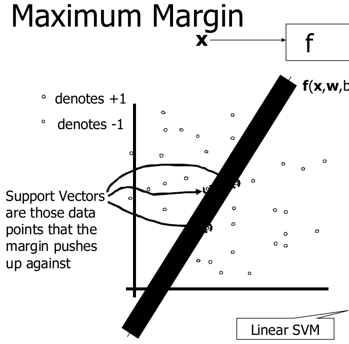


f(**x**,**w**,b) = sign(**w** · **x** + b)

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

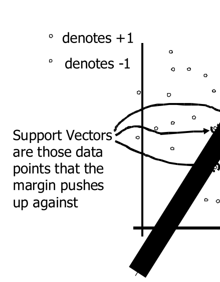


f(**x**,**w**,b) = sign(**w** · **x** + b)

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

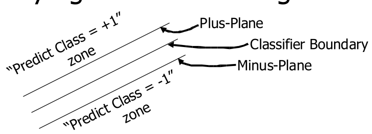
This is the simplest kind of SVM (Called an LSVM)

Why Maximum Margin?



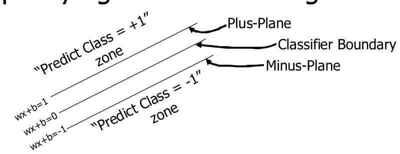
1. Intuitively this feels safest
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification
3. Robust to outliers since the model is immune to change/removal of any non-support-vector data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

Specifying a Line and Margin



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

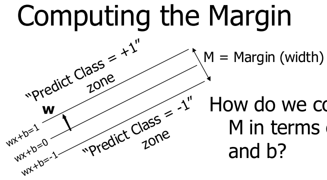
Specifying a Line and Margin



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

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

Computing the Margin

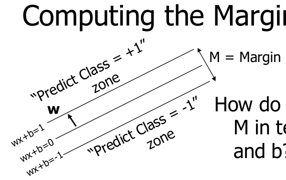


How do we compute M in terms of **w** and b?

- Plus-plane = { **w x** + b = +1 }
- Minus-plane = { **w x** + b = -1 }

Claim: The vector **w** is perpendicular to the Plus-Plane

Computing the Margin



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

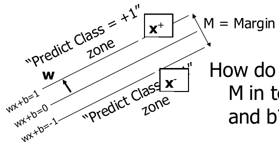
- Plus-plane = { **w x** + b = +1 }
- Minus-plane = { **w x** + b = -1 }

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

Let **u** and **v** be two vectors on the Plus Plane. What is **w** . (**u** – **v**) ?

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

Computing the Margin

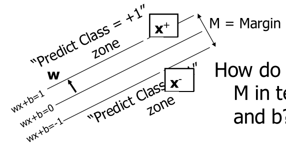


How do we compute M in terms of **w** and b?

- Plus-plane = { **w x** + b = +1 }
- Minus-plane = { **w 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

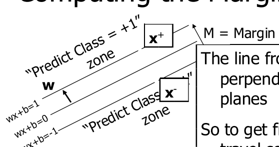
Computing the Margin



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

- Plus-plane = { **w x** + b = +1 }
- Minus-plane = { **w 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?

Computing the Margin

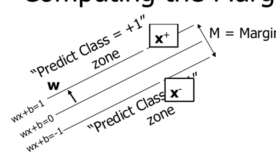


The line from $\mathbf{x}^-$ to $\mathbf{x}^+$ is perpendicular to the planes

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

- Plus-plane = { **w x** + b = +1 }
- Minus-plane = { **w 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?

Computing the Margin

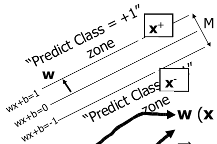


What we know:

- $\mathbf{w}\,\mathbf{x}^+ + b = +1$
- $\mathbf{w}\,\mathbf{x}^- + b = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda\, \mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$

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

Computing the Margin



What we know:

- $\mathbf{w}\,\mathbf{x}^+ + b = +1$
- $\mathbf{w}\,\mathbf{x}^- + b = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda\,\mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$

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

$\mathbf{w}\,(\mathbf{x}^- + \lambda\,\mathbf{w}) + b = 1$

$\Rightarrow$

$\mathbf{w}\,\mathbf{x}^- + b + \lambda\,\mathbf{w}\mathbf{w} = 1$

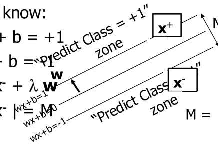
$\Rightarrow$

$-1 + \lambda\,\mathbf{w}\mathbf{w} = 1$

$\Rightarrow$

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

Computing the Margin



What we know:

- $\mathbf{w}\,\mathbf{x}^+ + b = +1$
- $\mathbf{w}\,\mathbf{x}^- + b = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda\,\mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$
-

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

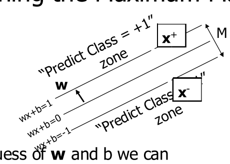
$M = |\mathbf{x}^+ - \mathbf{x}^-| = |\lambda\,\mathbf{w}| =$

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

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

$$= \frac{2}{\|w\|}$$

Learning the Maximum Margin Classifier



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

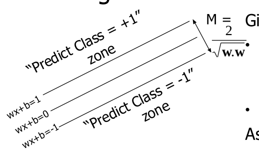
- Compute whether all data points in the correct half-planes
- Compute the width of the margin

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 data points. How?

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

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

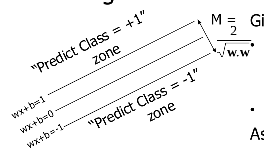
Assume R datapoints, each ($\mathbf{x}_k$, y_k) where y_k = +/- 1

What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

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

What should our quadratic optimization criterion be?

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

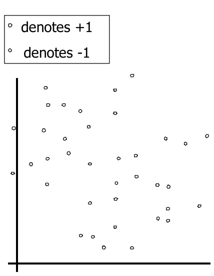
How many constraints will we have? R

What should they be?

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

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

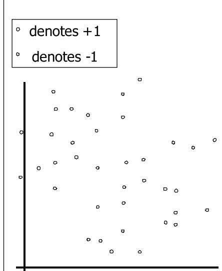
Uh-oh!



This is going to be a problem!

What should we do?

Uh-oh!



This is going to be a problem!

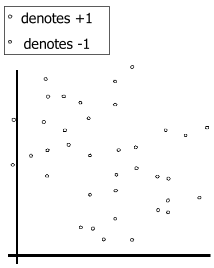
What should we do?

Idea 1:

Find minimum $||\mathbf{w}||^2$, while minimizing number of training set errors

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

Uh-oh!



This is going to be a problem!

What should we do?

Idea 1.1:

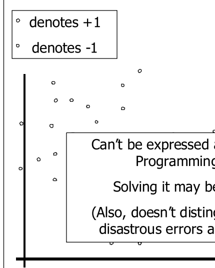
Minimize

$||\mathbf{w}||^2 + C$ (#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

$||\mathbf{w}||^2 + C$ (#train errors)

Tradeoff parameter

Can't be expressed as a Quadratic Programming problem.

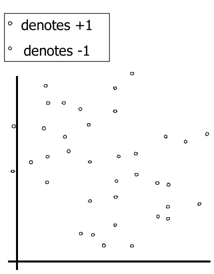
Solving it may be too slow.

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

There's a serio… us reject thi… you guess wh…

So… any other ideas?

Uh-oh!



This is going to be a problem!

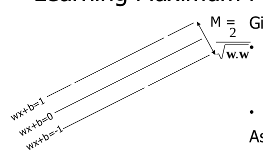
What should we do?

Idea 2.0:

Minimize

$||\mathbf{w}||^2 + C$ (distance of error points to their correct place)

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

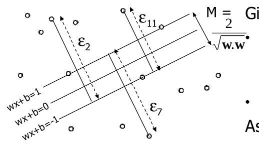
Assume R datapoints, each ($\mathbf{x}_k$, y_k) where y_k = +/- 1

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

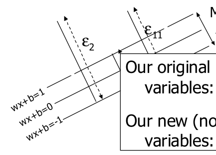
What should they be?

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

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

Learning Maximum Margi… …oise

m = # input dimensions



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

Given gu… …we can

- Compute sum of distances

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

Our new (noisy data) QP has m+1+R variables: w_1, w_2, … w_m, b, ε_k , ε_1 ,… ε_R

R= # records

What should our quadratic optimization criterion be?

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

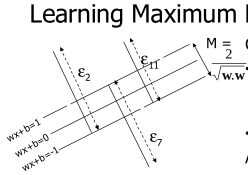
How many constraints will we have? R

What should they be?

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

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

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

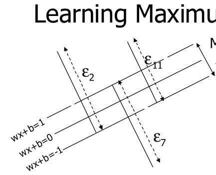
What should they be?

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

w . $\mathbf{x}_k$ + b <= -1+ε_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 **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?

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

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

ε_k >= 0 for all k

An Equivalent QP

Maximize $\sum_{k=1}^{R} \alpha_k + \sum_{k=1}^{R}\sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ 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(**x**,**w**,b) = sign(**w. 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 $Q_{kl} = y_k y_l (\mathbf{x}_k . \mathbf{x}_l)$

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

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

Then define:

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

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

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

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

Then classify with:

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

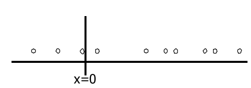
An Equivalent QP

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.

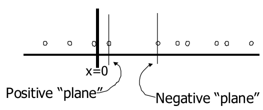
Suppose we're in 1 Dimension

What would SVMs do with this data?



Suppose we're in 1 Dimension

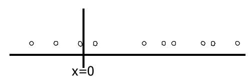
Not a big surprise



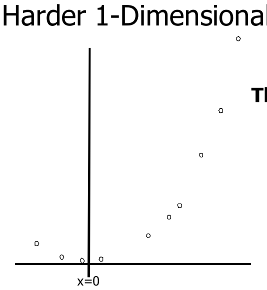
Harder 1-Dimensional Dataset

That's wiped the smirk off SVM's face

What can be done about this?



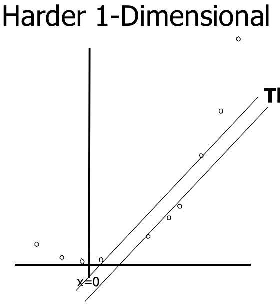
Harder 1-Dimensional Dataset



The Kernel Trick: Preprocess the data, mapping **x** into higher dimensional space **F**(**x**)

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

Harder 1-Dimensional Dataset

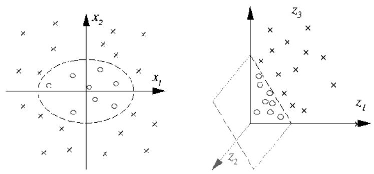


The Kernel Trick: Preprocess the data, mapping **x** into higher dimensional space **F**(**x**)

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

Example: All Degree 2 Monomials

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



B. Schölkopf, NIPS, 3 December 2001

- Project examples into some higher dimensional space where the data is linearly separable, defined by $\mathbf{z} = \mathbf{F}(\mathbf{x})$
- Training depends only on dot products of the form $\mathbf{F}(\mathbf{x}_i) \cdot \mathbf{F}(\mathbf{x}_j)$
- Example:

$$F(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{F}(\mathbf{x}_i) \cdot \mathbf{F}(\mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

Common SVM Basis Functions

$\mathbf{z}_k$ = (polynomial terms of $\mathbf{x}_k$ of degree 1 to q)

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

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

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

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

The Federalist Papers

- Written in 1787-1788 by Alexander Hamilton, John Jay, and James Madison to persuade the citizens of New York to ratify the constitution
- Papers consisted of short essays, 900 to 3500 words in length
- Authorship of 12 of those papers have been in dispute (Madison or Hamilton); these papers are referred to as the disputed Federalist papers

Description of the Data

- For every paper:
  - Machine readable text was created using a scanner
  - Computed relative frequencies of 70 words that Mosteller-Wallace identified as good candidates for author-attribution studies
  - Each document is represented as a vector containing the 70 real numbers corresponding to the 70 word frequencies
- The dataset consists of 118 papers:
  - 50 Madison papers
  - 56 Hamilton papers
  - 12 disputed papers

Function Words Based on Relative Frequencies

1	*a*	15	*do*	29	*is*	43	*or*	57	*this*
2	*all*	16	*down*	30	*it*	44	*our*	58	*to*
3	*also*	17	*even*	31	*its*	45	*shall*	59	*up*
4	*an*	18	*every*	32	*may*	46	*should*	60	*upon*
5	*and*	19	*for*	33	*more*	47	*so*	61	*was*
6	*any*	20	*from*	34	*must*	48	*some*	62	*were*
7	*are*	21	*had*	35	*my*	49	*such*	63	*what*
8	*as*	22	*has*	36	*no*	50	*than*	64	*when*
9	*at*	23	*have*	37	*not*	51	*that*	65	*which*
10	*be*	24	*her*	38	*now*	52	*the*	66	*who*
11	*been*	25	*his*	39	*of*	53	*their*	67	*will*
12	*but*	26	*if*	40	*on*	54	*then*	68	*with*
13	*by*	27	*in*	41	*one*	55	*there*	69	*would*
14	*can*	28	*into*	42	*only*	56	*things*	70	*your*

SLA Feature Selection for Classifying the Disputed Federalist Papers

- Apply the SVM Successive Linearization Algorithm for feature selection to:
  - Train on the 106 Federalist papers with known authors
  - Find a classification hyperplane that uses as few words as possible
- Use the hyperplane to classify the 12 disputed papers

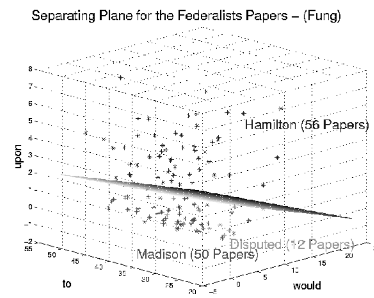
Hyperplane Classifier Using 3 Words

- A hyperplane depending on three words was found:

$$0.537to + 24.663upon + 2.953would = 66.616$$

- All disputed papers ended up on the Madison side of the plane

Results: 3D Plot of Hyperplane



Multi-Class Classification

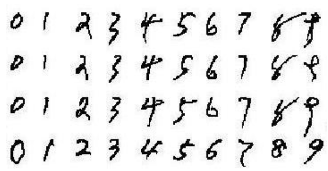
- SVMs can only handle two-class outputs
- What can be done?
- Answer: for N-class problems, learn N SVM's:
  - SVM 1 learns "Output=1" vs "Output ≠ 1"
  - SVM 2 learns "Output=2" vs "Output ≠ 2"
  - :
  - SVM N learns "Output=N" vs "Output ≠ N"
- To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region

Summary

- Learning linear functions
  - Pick separating plane that maximizes margin
  - Separating plane defined in terms of support vectors only
- Learning non-linear functions
  - Project examples into higher dimensional space
  - Use kernel functions for efficiency
- Generally avoids over-fitting problem
- Global optimization method; no local optima
- Can be expensive to apply, especially for multi-class problems

Case Study

- Handwritten digits important domain
  - Automated sorting of mail (zip code recognition)
- NIST dataset of handwritten digits
  - 60,000 labeled digits, 20x20=400 pixels in 8-bit greyscale values



Case Study -- Models

- 3 Nearest Neighbor
- ANN with 300 hidden units
- ANN specially crafted for the domain (LeNet)
  - Lots of work went into crafting this
- LeNet with Ensembles (Boosted LeNet)
- SVM (almost no effort in creating the model)
- Virtual SVM (specially crafted for the domain)
- Shape Matching instead of standard distance in 3NN
- Human (somewhere in the range of 0.2% error rate and 2.5% error rate)

Case Study Results

	3 NN	300 Hidden	LeNet	Boosted LeNet	SVM	Virtual SVM	Shape Match
Error Rate	2.4%	1.6%	0.9%	0.7%	1.1%	0.56%	0.63%
Runtime (ms/digit)	1000	10	30	50	2000	200	
Memory (MB)	12	.49	.012	.21	11		
Training time (days)	0	7	14	30	10		
% rejected to reach 0.5% accuracy	8.1%	3.2%	1.8%	0.5%	1.8%		