ISTA 421 + INFO 521
Introduction to Machine Learning

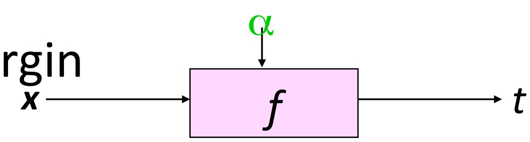
Lecture 23: Support Vector Machines - II and The Kernel Trick

Clay Morrison
claytonm@email.arizona.edu
Harvill 437A
Phone 621-6609

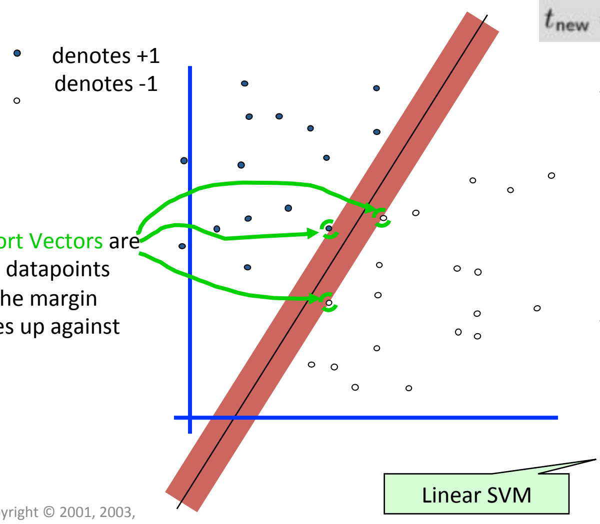
15 November 2017

1

Maximum Margin



$t_{\mathsf{new}} = \mathrm{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_{\mathsf{new}} + b)$

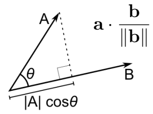


Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.
This is the simplest kind of SVM (Called an LSVM)

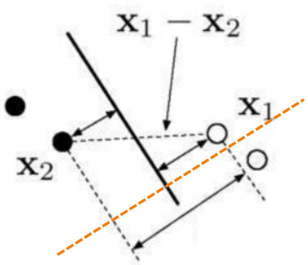
Copyright © 2001, 2003, Andrew W. Moore

2



$\mathbf{a} \cdot \frac{\mathbf{b}}{\|\mathbf{b}\|} = \|\mathbf{a}\| \cos\theta \qquad \|\boldsymbol{x}\| := \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}.$

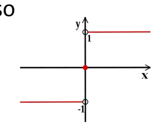
Defining the Margin



w is in the direction perpendicular to the boundary.
Normalize it to get the "unit vector": $\frac{\mathbf{w}}{\|\mathbf{w}\|}$

Our "*decision function*", which we will also refer to as *D*(*x*):

$$t_{\text{new}} = \text{sign}\ (\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{new}} + b)$$



Another complication:

We note that the argument in $D(x)$ is invariant under a rescaling: $\mathbf{w} \to \lambda\mathbf{w}$, $b \to \lambda b$.

We will implicitly fix a scale with:

$$\mathbf{w} \cdot \mathbf{x_1} + b = 1$$
$$\mathbf{w} \cdot \mathbf{x_2} + b = -1$$

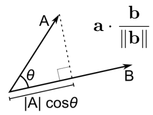
for the support vectors (*canonical hyperplanes*).

Combine both constraints by subtracting one from the other, to get the following:

$$\mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 2$$

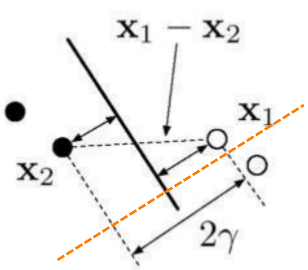
The margin (the length of the distance between the support vector canonical hyperplanes) will be given by the ***projection*** of the vector ($x_1 - x_2$) onto the normal vector to the hyperplane!
The projection is accomplished by taking the inner product of these two quantities

SISTA 3



$\mathbf{a} \cdot \frac{\mathbf{b}}{\|\mathbf{b}\|} = \|\mathbf{a}\| \cos\theta \qquad \|\boldsymbol{x}\| := \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}.$

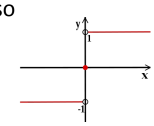
Defining the Margin



w is in the direction perpendicular to the boundary.
Normalize it to get the "unit vector": $\frac{\mathbf{w}}{\|\mathbf{w}\|}$

Our "*decision function*", which we will also refer to as *D*(*x*):

$$t_{\text{new}} = \text{sign}\ (\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{new}} + b)$$



Another complication:

We note that the argument in $D(x)$ is invariant under a rescaling: $\mathbf{w} \to \lambda\mathbf{w}$, $b \to \lambda b$.

We will implicitly fix a scale with:

$$\mathbf{w} \cdot \mathbf{x_1} + b = 1$$
$$\mathbf{w} \cdot \mathbf{x_2} + b = -1$$

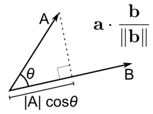
for the support vectors (*canonical hyperplanes*).

Combine both constraints by subtracting one from the other, to get the following:

$$\mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 2$$

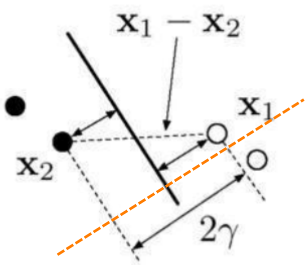
The margin (the length of the distance between the support vector canonical hyperplanes) will be given by the ***projection*** of the vector ($x_1 - x_2$) onto the normal vector to the hyperplane!
The projection is accomplished by taking the inner product of these two quantities

SISTA 4



$\mathbf{a} \cdot \frac{\mathbf{b}}{\|\mathbf{b}\|} = \|\mathbf{a}\| \cos\theta \qquad \|\boldsymbol{x}\| := \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}.$

Defining the Margin



$$
\begin{aligned}
2\gamma &= \frac{1}{||\mathbf{w}||}\mathbf{w}^{\mathsf{T}}(\mathbf{x}_1 - \mathbf{x}_2) \\
&= \frac{1}{||\mathbf{w}||}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_1 - \mathbf{w}^{\mathsf{T}}\mathbf{x}_2) \\
&= \frac{1}{||\mathbf{w}||}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_1 + b - \mathbf{w}^{\mathsf{T}}\mathbf{x}_2 - b) \\
&= \frac{1}{||\mathbf{w}||}(1+1) \\
\gamma &= \frac{1}{||\mathbf{w}||}.
\end{aligned}
$$

w is in the direction **perpendicular** to the boundary.
Normalize it to get the "unit vector": $\frac{\mathbf{w}}{||\mathbf{w}||}$

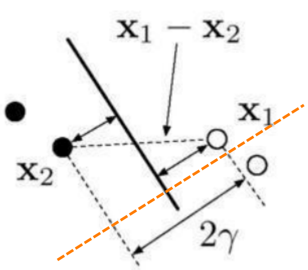
Constraints:
$\mathbf{w} \cdot \mathbf{x_1} + b = 1$
$\mathbf{w} \cdot \mathbf{x_2} + b = -1$

$\mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 2$

The margin (the length of the distance between the support vector canonical hyperplanes) will be given by the ***projection*** of the vector ($x_1 - x_2$) onto the normal vector to the hyperplane!
The projection is accomplished by taking the inner product of these two quantities

Maximizing the Margin

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



Recall, we set a scale for the closest points to the margin:

$\mathbf{w} \cdot \mathbf{x_1} + b = 1$ For the s.v. class 1
$\mathbf{w} \cdot \mathbf{x_2} + b = -1$ For the s.v. class 2

Therefore, **w** must be chosen such that:

$$
\begin{aligned}
&\mathbf{w} \cdot \mathbf{x_n} + b \geq 1 && \text{for all } \mathbf{x_n} \text{ in class 1} \\
&\mathbf{w} \cdot \mathbf{x_n} + b \leq -1 && \text{for all } \mathbf{x_n} \text{ in class 2}
\end{aligned}
$$

Combining both constraints is easy:

$t_n(\mathbf{w}^{\top}\mathbf{x}_n + b) \geq 1$ For all $\mathbf{x}_n$

There are a total of N constraints

Easier to *minimize* $\frac{1}{2}||\mathbf{w}||^2$

Constrained Optimization with Lagrange (KKT) Multipliers

- Find values of a set of parameters that maximize (or minimize) an objective function, but also satisfy some constraints.
- Create new objective function that includes the original plus an additional term for each constraint.

For example, minimize $f(x)$ subject to the constraint $g(w) \leq a$

$$\begin{array}{ll} \underset{w}{\text{argmin}} & f(w) \\ \text{subject to} & g(w) \leq a \end{array}$$

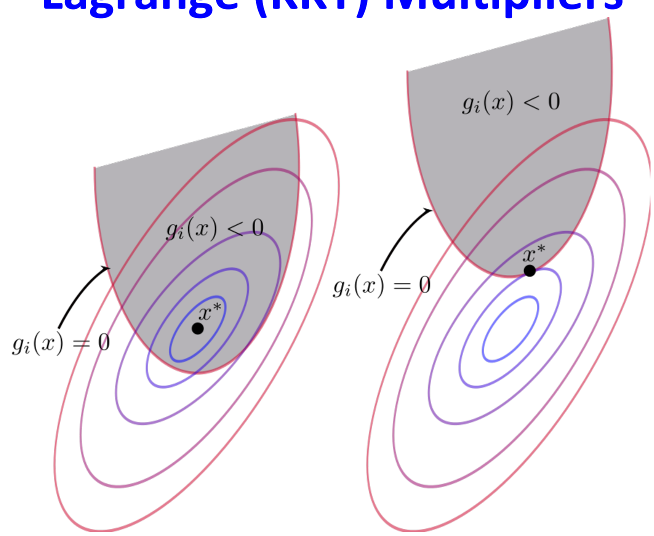
Add Lagrange term of the form $\lambda(a - g(w))$ and optimize for w and λ

$$\begin{array}{ll} \underset{w,\lambda}{\text{argmin}} & f(w) - \lambda(a - g(w)) \\ \text{subject to} & \lambda > 0 \end{array}$$

Bug in book!
(p. 188, comment 5.1)

(Strictly speaking, Lagrange multipliers are just for equality constraints, whereas constrained optimization problems with inequalities involve KKT (Karush-Kuhn-Tucker) conditions; so you may see the lambda above referred to as a “KKT” multiplier…) 7

Constrained Optimization with Lagrange (KKT) Multipliers



(Shamelessly cribbed from Wikipedia…)

8

Maximizing the Margin $\gamma = \frac{1}{\|\mathbf{w}\|}$

$\frac{1}{2}\|\mathbf{w}\|^2 \qquad t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1$

Maximizing the margin, γ, becomes a ***constrained optimization*** problem, namely, to minimize the following:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \ \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to} \quad t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1, \ \text{for all } n$$

We can incorporate the inequalities into the minimization by introducing ***Lagrange multipliers***, resulting in the following:

note: $||\mathbf{w}||^2 = \mathbf{w}^\top \mathbf{w}$

$$\underset{\mathbf{w},\alpha}{\operatorname{argmin}} \ \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{n=1}^{N} \alpha_n (t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1)$$
$$\text{subject to} \quad \alpha_n \geq 0, \ \text{for all } n,$$

Recall how we maximize/minimize!

$$\frac{\partial}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$$
$$\frac{\partial}{\partial b} = -\sum_{n=1}^{N} \alpha_n t_n.$$

Set to 0!

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$$
$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

These two identities must be satisfied at the optimum

Maximizing the Margin $\gamma = \frac{1}{\|\mathbf{w}\|}$

$\frac{1}{2}\|\mathbf{w}\|^2 \qquad t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1$

$$\underset{\mathbf{w},\alpha}{\operatorname{argmin}} \ \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{n=1}^{N} \alpha_n (t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1)$$
$$\text{subject to} \quad \alpha_n \geq 0, \ \text{for all } n,$$

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$$
$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

Plug constraint for **w** back into the objective function, to get:

$$\frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{n=1}^{N} \alpha_n (t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1)$$
$$= \frac{1}{2}\left(\sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m^\mathsf{T}\right)\left(\sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n\right) - \sum_{n=1}^{N} \alpha_n \left(t_n \left(\sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m^\mathsf{T} \mathbf{x}_n + b\right) - 1\right)$$
$$= \frac{1}{2}\sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\mathsf{T} \mathbf{x}_n - \sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\mathsf{T} \mathbf{x}_n - \sum_{n=1}^{N} \alpha_n t_n b + \sum_{n=1}^{N} \alpha_n$$
$$= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\mathsf{T} \mathbf{x}_n$$

This goes away by this constraint

This is the **dual** optimization problem (we have eliminated **w**!)
It is a ***quadratic optimization*** problem due to the $\alpha_m \alpha_n$ term (matlab: quadprog)

10

Making Predictions

Our final constraint problem:

$$\sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\top \mathbf{x}_n$$

Subject to: $\alpha_n \geq 0, \ \sum_{n=1}^{N} \alpha_n t_n = 0.$

Give it to a quadratic programming solver!

To predict, we need our decision function D(x)

$$t_{\text{new}} = \text{sign}\ (\mathbf{w}^\top \mathbf{x}_{\text{new}} + b)$$

But we just optimized for α's!

Recall: $\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$

So rewrite the decision function as:

$$t_{\text{new}} = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n^\top \mathbf{x}_{\text{new}} + b\right)$$

To find b, we will use the fact that for the closest points, $t_n\ (\mathbf{w}^\top \mathbf{x}_n + b) = 1$

$$b = t_n - \sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m^\top \mathbf{x}_n$$

(note that $t_n = 1/t_n$ in this case because $t_n = \{ 1, -1 \}$)

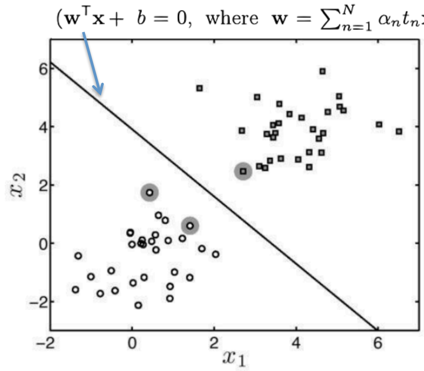
SISTA 11

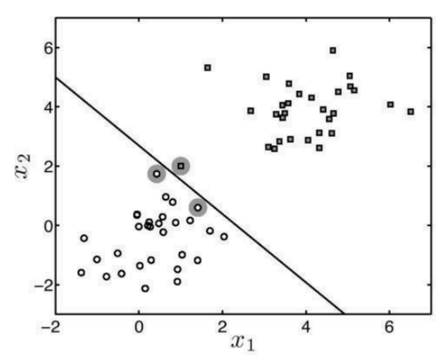
Hard Margin SMV

$$t_{new} = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n^\top \mathbf{x}_{new} + t_n - \sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m^\top \mathbf{x}_n\right)$$

After optimizing for all α_n's, the only α's that are **non-zero** are the **support vectors**!

$(\mathbf{w}^\top \mathbf{x} + \ b = 0, \ \text{where} \ \mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n)$





SISTA 12

Soft Margin SVM

- It allow points to lie on the wrong side of the boundary, need to “slacken” the constraints

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 - \xi_n \quad \text{where} \quad \xi_n \geq 0$$

- If $0 \leq \xi_n \leq 1$
  – Then the point lies on the correct side of the boundary, but within the boundary margin
- If $\xi_n > 1$
  – Then the point lies on the “wrong” side of the boundary

13

Soft Margin SVM

- It allow points to lie on the wrong side of the boundary, need to “slacken” the constraints

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 - \xi_n \quad \text{where} \quad \xi_n \geq 0$$

- The optimization task becomes:

Recall, the original constrained optimization was:

$$\underset{\mathbf{w}}{\text{argmin}} \quad \frac{1}{2}||\mathbf{w}||^2$$

subject to $t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1$, for all n

$$\underset{\mathbf{w}}{\text{argmin}} \quad \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

subject to $\xi_n \geq 0$ and $t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 - \xi_n$ for all n

- *C* controls to what extent we are willing to allow points to sit within the margin itself or on the wrong side of the decision boundary

14

Soft Margin SVM

- It allow points to lie on the wrong side of the boundary, need to "slacken" the constraints

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 - \xi_n \quad \text{where} \quad \xi_n \geq 0$$

- It turns out that incorporating the new constraint *C* does not change the overall optimization much!: Recall: $\sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\mathsf{T} \mathbf{x}_n \quad \alpha_n \geq 0, \ \sum_{n=1}^{N} \alpha_n t_n = 0.$

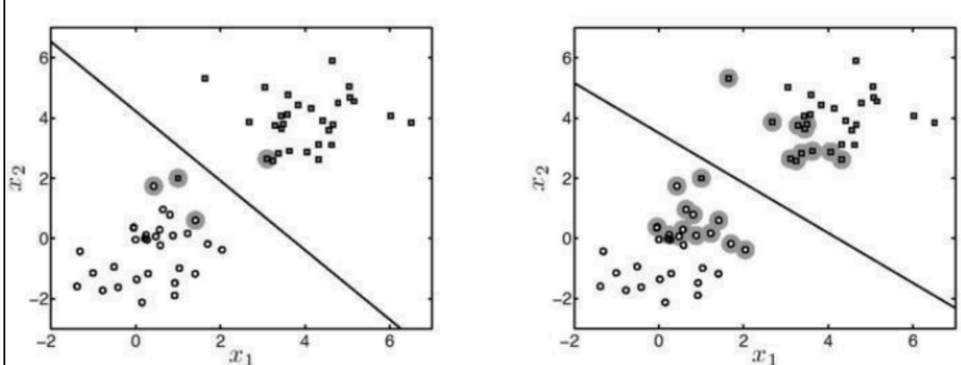
$$\underset{\mathbf{w}}{\operatorname{argmax}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^\mathsf{T} \mathbf{x}_m$$

Just adds an upper bound on the influence any training point can have

$$\text{subject to} \sum_{n=1}^{N} \alpha_n t_n = 0 \ \text{ and } \ 0 \leq \alpha_n \leq C, \ \text{for all } n.$$

15

Decision Boundary & Support Vectors for different *C*



(a) $C = 1$ (b) $C = 0.01$

Larger C = lower bias (fewer support vectors)
Smaller C = higher bias (more support vectors)

$$\underset{\mathbf{w}}{\operatorname{argmax}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^\mathsf{T} \mathbf{x}_m$$

$$\text{subject to} \sum_{n=1}^{N} \alpha_n t_n = 0 \ \text{ and } \ 0 \leq \alpha_n \leq C, \ \text{for all } n.$$

Kernels: Transforming Data

- So far, the boundary has been linear
- **Recall** in our treatment of Linear Regression, to get non-linear boundaries we added terms to **x** and extended **w** (… the weight space view)
  - In that case, **we explicitly projected the data**

$$\mathbf{x} \xrightarrow{\phi} \phi(\mathbf{x}) \qquad \text{e.g., } \phi(x) = 1 + x + x^2$$

$$f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$$

17

Kernels: Transforming Data

- Here we take a different approach: the model remains the same (linear decision boundary) but **the data are *implicitly* transformed into a new space**.
- **Kernel functions** are *similarity* functions.
  - They compute the similarity of pairs of input points **x** and **z**, as if they had been projected into some space ν by function ϕ (i.e., they compute the similarity between ϕ(**x**) and ϕ(**z**)), **but without actually doing the projection!**

$$\phi : \mathcal{X} \to \nu \qquad k(\mathbf{x}, \mathbf{z}) = \underbrace{\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle_\nu}_{\text{inner product}}$$

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

The word "kernel" is used in mathematics to denote a weighting function for a weighted sum or integral.

Similarity, here, is in terms of an inner product (on ν). An explicit representation for ϕ is not necessary **as long as ν is an inner product space**.

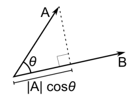
Formally, the real-valued function k(**x**,**z**) must satisfy Mercer's condition:

$$\iint g(\mathbf{x}) k(\mathbf{x}, \mathbf{z}) g(\mathbf{z}) \; d\mathbf{x} \; d\mathbf{z} \geq 0 \quad \text{for all square integrable fns } g(\mathbf{x}) \quad \int_{-\infty}^{\infty} |g(\mathbf{x})|^2 \, \mathrm{d}\mathbf{x} \neq \infty$$

18

Kernels: Transforming Data

- The most familiar (from this class so far) is the dot product, where the mapping function ϕ is the *identity* function:



$$\phi(\mathbf{x}) = \mathbf{x}$$
$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z} = x_1 z_1 + x_2 z_2 + \ldots + x_n z_n$$

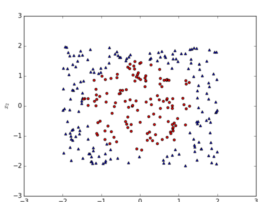
19

Kernels: Transforming Data

- Another example! $k(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^\top \mathbf{z}\right)^2$

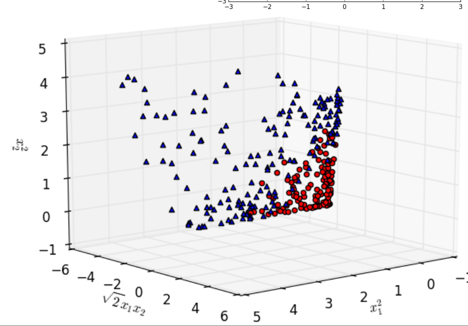
Find the explicit projection of the vectors

input space:
$\mathbf{x} = (x_1, x_2)^\top$



$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= \left(\mathbf{x}^\top \mathbf{z}\right)^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + \sqrt{2} x_1 z_1 \sqrt{2} x_2 z_2 + x_2^2 z_2^2 \\ &= \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right) \left(z_1^2, \sqrt{2} z_1 z_2, z_2^2\right)^\top \\ &= \phi(\mathbf{x})^\top \phi(\mathbf{z}) \end{aligned}$$

$$\phi(\mathbf{x}) = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right)^\top$$



The Kernel Trick

- SVMs are one of a **class** of algorithms that don't actually need to perform that actual transformation!
- The terms representing the data only appear within inner (i.e., dot) products: $\mathbf{x}_n^{\mathsf{T}}\mathbf{x}_m, \ \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_{\text{new}}$

$$\sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n,m=1}^{N}\alpha_m\alpha_n t_m t_n \mathbf{x}_m^{\mathsf{T}}\mathbf{x}_n \longleftarrow \text{objective}$$

$$\text{prediction} \longrightarrow t_{new} = \text{sign}\left(\sum_{n=1}^{N}\alpha_n t_n \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_{new} + t_n - \sum_{m=1}^{N}\alpha_m t_m \mathbf{x}_m^{\mathsf{T}}\mathbf{x}_n\right)$$

- We never see **x** on its own.
- Were we to do a transformation, we would need to calculate inner products in the new space
- BUT, if we can find a function $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^{\mathsf{T}}\phi(\mathbf{x}_m)$ then we can use *k* directly without performing the transform.

SISTA 21

The Kernelized Soft Margin SVM

Original soft margin SVM

$$\underset{\mathbf{w}}{\text{argmax}} \ \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n,m=1}^{N}\alpha_n\alpha_m t_n t_m \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_m$$

$$\text{subject to} \ \sum_{n=1}^{N}\alpha_n t_n = 0 \ \text{ and } \ 0 \le \alpha_n \le C, \ \text{for all } n.$$

Optimization to find α_n's

$$t_{\text{new}} = \text{sign}\left(\sum_{n=1}^{N}\alpha_n t_n \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_{\text{new}} + b\right) \longleftarrow \text{predictions}$$

Kernelized soft margin SVM:

$$\underset{\mathbf{w}}{\text{argmax}} \ \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n=1}^{N}\alpha_n\alpha_m t_n t_m \underbrace{k(\mathbf{x}_n, \mathbf{x}_m)}$$

$$\text{subject to} \ \sum_{n=1}^{N}\alpha_n t_n = 0 \ \text{ and } \ 0 \le \alpha_n \le C, \ \text{for all } n.$$

$$t_{\text{new}} = \text{sign}\left(\sum_{n=1}^{N}\alpha_n t_n \underbrace{k(\mathbf{x}_n, \mathbf{x}_{\text{new}})} + b\right).$$

SISTA 22

Some Kernels

linear $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$

polynomial $k(\mathbf{x}, \mathbf{z}) = \left(\alpha \mathbf{x}^\top \mathbf{z} + \beta\right)^\gamma$

Gaussian (squared exponential) $k(\mathbf{x}, \mathbf{z}) = \exp\left\{-\gamma \left(\mathbf{x} - \mathbf{z}\right)^\top \left(\mathbf{x} - \mathbf{z}\right)\right\}$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

23

Some Kernels

linear $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$

polynomial $k(\mathbf{x}, \mathbf{z}) = \left(\alpha \mathbf{x}^\top \mathbf{z} + \beta\right)$

Gaussian (squared exponential) $k(\mathbf{x}, \mathbf{z}) = \exp\left\{-\gamma \left(\mathbf{x}\right.\right.$

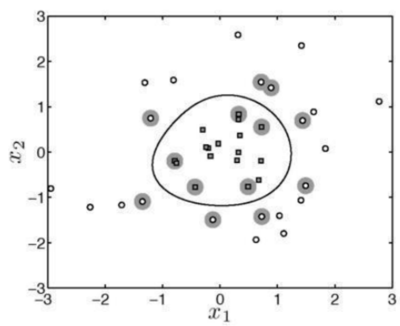
1. Linear Kernel
2. Polynomial Kernel
3. Gaussian Kernel
4. Exponential Kernel
5. Laplacian Kernel
6. ANOVA Kernel
7. Hyperbolic Tangent (Sigmoid) Kernel
8. Rational Quadratic Kernel
9. Multiquadric Kernel
10. Inverse Multiquadric Kernel
11. Circular Kernel
12. Spherical Kernel
13. Wave Kernel
14. Power Kernel
15. Log Kernel
16. Spline Kernel
17. B-Spline Kernel
18. Bessel Kernel
19. Cauchy Kernel
20. Chi-Square Kernel
21. Histogram Intersection Kernel
22. Generalized Histogram Intersection Kernel
23. Generalized T-Student Kernel
24. Bayesian Kernel
25. Wavelet Kernel

Kernel functions for machine learning

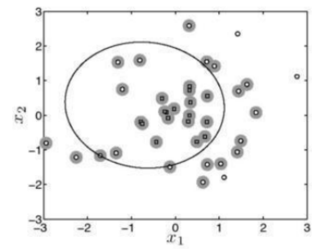
http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html

SVM Classification with a Gaussian Kernel

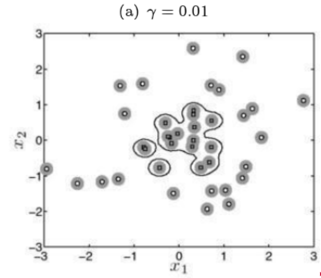
$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\gamma(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right\}$$



$\gamma = 10$, $C = 10$



(a) $\gamma = 0.01$



(b) $\gamma = 50$

γ and *C* are unfortunately both free parameters

25

SVM Odds-n-Ends

- SVM training is sensitive to **feature value ranges**.
  - This is because the view that SVM optimization has of the data is through the inner product! Greater magnitude values (positive or negative) dominate.
  - Scale your data! Common to linearly scale all feature (aka attribute) values to range [-1, 1] or [0,1].
  - Determine scaling for each feature based on your ***training data*** and **save** your scaling ranges; use the same scaling for any other data you use with the learned model.
- SVM training can be sensitive to "unbalanced" classes (one class is represented much more than another)
  - Implementations like **libsvm** allow you to weight class labels in order to "balance" the contribution of classes
- Parameter Selection:
  - Grid search: systematic combinations of parameters values and narrowing (usually doing CV at each point)

26

Multi-class Classification

- **1-against-the-rest** :
  - Binary classifiers: class c_i vs. {c_j, $\forall j \neq i$}.
  - Total of C classifiers created (note that that each classifier is trained on *all* data)
- **1-against-1**:
  - Binary classifiers: class c_i vs. c_j, $j \neq i$
  - Total of $C(C\text{-}1)/2$ classifiers created (can be much faster to train if training time is super-linear in data)
- Finally, NOTE: Multi-class classification is NOT the same thing as ***multi-label*** **classification**: the latter involves assigning more than one label to each instance.

SISTA 27

Kernel Nearest Neighbors

- We can Kernelize other methods
- E.g., **Nearest Neighbors**: the core of the algorithm that involves the data is a **distance metric**.
- We can turn the NN distance function into a form involving only inner products of **x**:

A common form of distance function (e.g., Euclidean distance) $(\mathbf{x}_{\text{new}} - \mathbf{x}_n)^{\mathsf{T}}(\mathbf{x}_{\text{new}} - \mathbf{x}_n)$

Multiply through: $\mathbf{x}_{\text{new}}^{\mathsf{T}}\mathbf{x}_{\text{new}} - 2\mathbf{x}_{\text{new}}^{\mathsf{T}}\mathbf{x}_n + \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_n$

Kernelize: $k(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{new}}) - 2k(\mathbf{x}_{\text{new}}, \mathbf{x}_n) + k(\mathbf{x}_n, \mathbf{x}_n)$

SISTA 28