EECS 545: Machine Learning

Lecture 6. Kernel methods

Honglak Lee

1/26/2011

Outline

- Recap: Exponential Family distribution
- Recap: Probabilisitic Generative models
  - Gaussian Discriminant Analysis
  - Naive Bayes
- Kernel methods

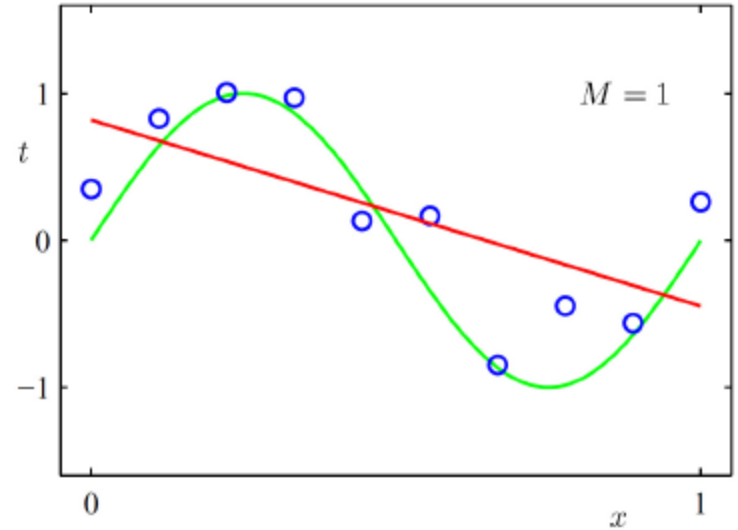
Kernel methods: Motivation

Linear regression

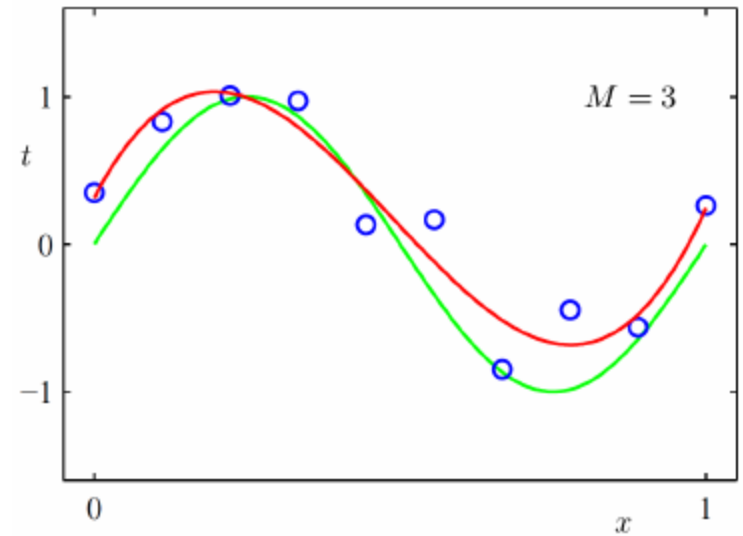
- Example: 1D regression, one input x, one output $y(x)$
- Linear model $y(x) = w^T x$ can only produce straight lines through origin
- Not very flexible/powerful
- Can we do better?

Feature Transformations

Replace x → (1,x) →



Replace x → $(1,x,x^2,x^3)$ →



Linear models with transformations

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}'\boldsymbol{\phi}(\mathbf{x}) = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x})$$

- Use sum of squares error function (idea from Gauss) plus regularization

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}'\mathbf{w}$$

$w \in \mathbb{R}^M$ $\quad \in \mathbb{R}^{M \times M}$ $\quad O(M^3)$

- Solution is simple

$\nabla_w J = 0$

$$\Longleftrightarrow \quad \mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}(\underbrace{\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_n)}_{\phi^T(x_n)\,w} - t_n)\boldsymbol{\phi}(\mathbf{x}_n) \rightarrow \mathbf{w} = \boxed{(\lambda\mathbf{I} + \boldsymbol{\Phi}'\boldsymbol{\Phi})}^{-1}\boldsymbol{\Phi}'\mathbf{t}$$

This is nice, but:

- What transformations to use?
- Computational complexity increases if transformation vector gets longer (*n* data points of dimension *d* are stored row-wise in Φ, and we must invert $\Phi'\Phi$ which is a *d* x *d* matrix, complexity scales with d^3)

Kernel Functions

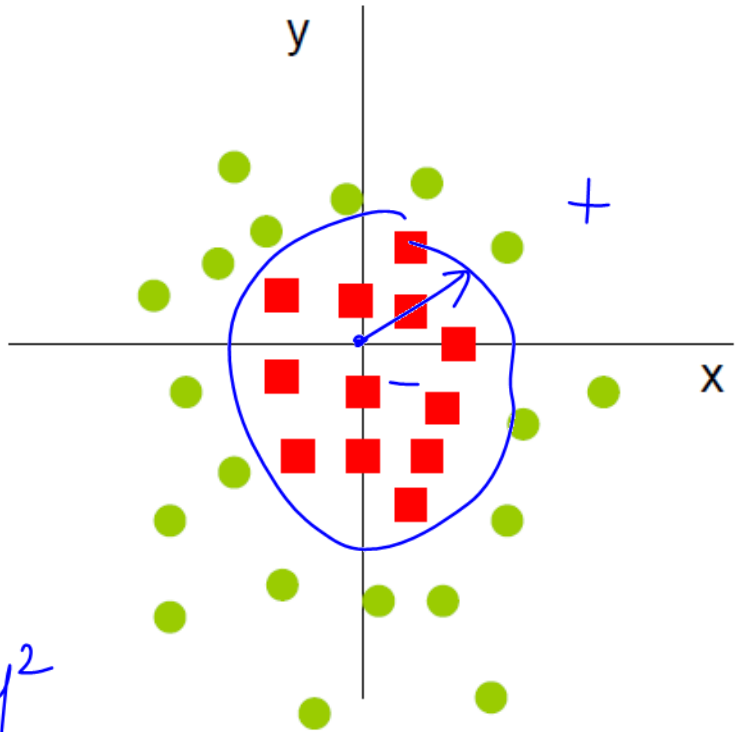
- We have been mapping each data point x through a fixed non-linear mapping to get a feature vector $\phi(\mathbf{x})$ $\quad \underset{\in \mathbb{R}^d}{x} \longmapsto \phi(x)$
  - The feature vector extracts important properties from the surface representation of **x**.
  - It may make many inferences easier.

- Unfortunately, the feature vector may be high-dimensional, even infinite-dimensional.

Linear classifiers

- No linear separating plane
- Linear classifiers not very flexible/powerful
- Can we do better?

(x, y) or $(1, x, y)$



$r^2 = x^2 + y^2$

or $r = \sqrt{x^2 + y^2}$

Linear classifiers

$w^T \phi(x) = -1 + \sqrt{x^2+y^2}$

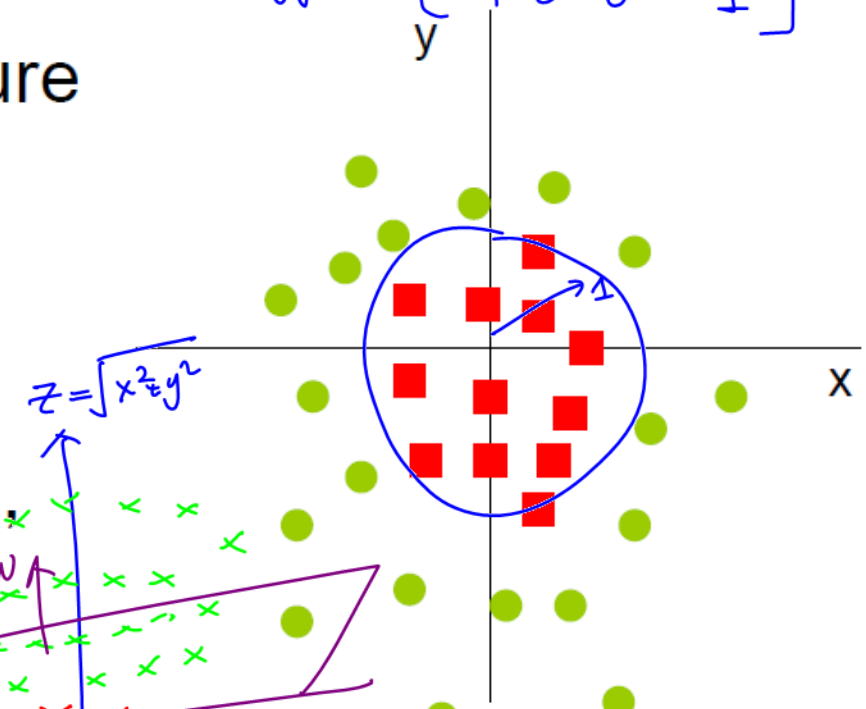
$\geq 0 \rightarrow$ positive

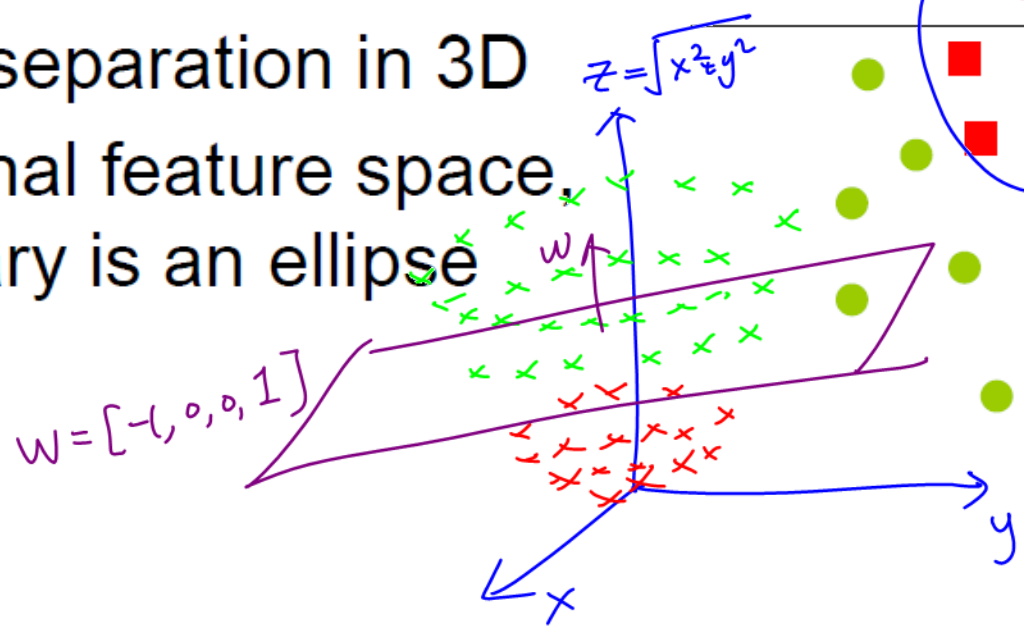
$< 0 \rightarrow$ negative

$\phi(x) = (1, x, y, \sqrt{x^2+y^2})$

$w = [-1 \;\; 0 \;\; 0 \;\; 1]$

- Add distance to origin $(x^2+y^2)^{1/2}$ as a third feature
- Data now lives on a parabolic surface in 3D
- Linear separation in 3D
- In original feature space, boundary is an ellipse





Linear classifiers

- Data has been mapped to a new, higher dimensional space (where it lives on a curved manifold)
- Alternative way to think about this: data still lives in original space but the definition of *distance* or *inner product* has been changed (PRMLex6.3)
- Suddenly *linear* methods become *non-linear*

Polynomial expansion

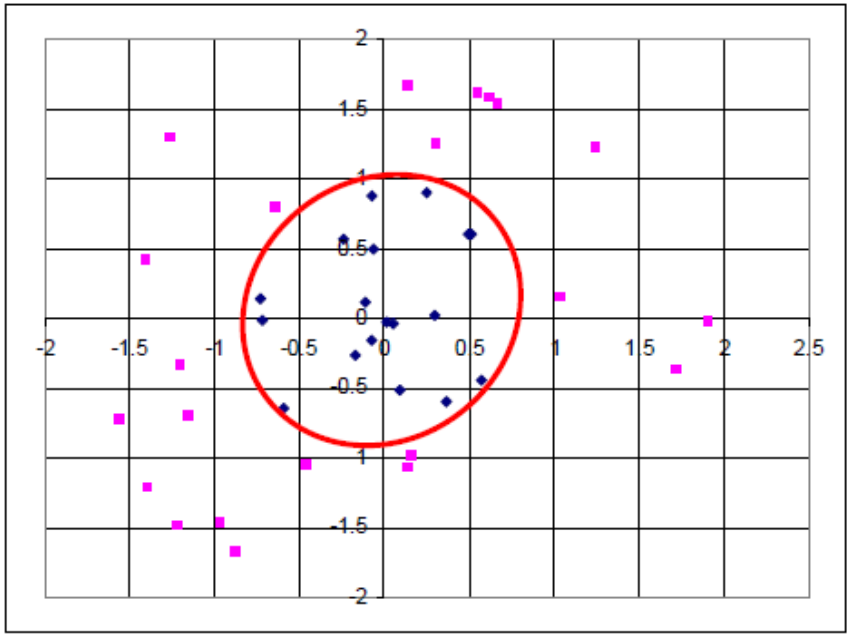
$w^T \phi(x)$ $\quad + w_3 xy$

- Replace $(x,y) \rightarrow (x^2,y^2)$

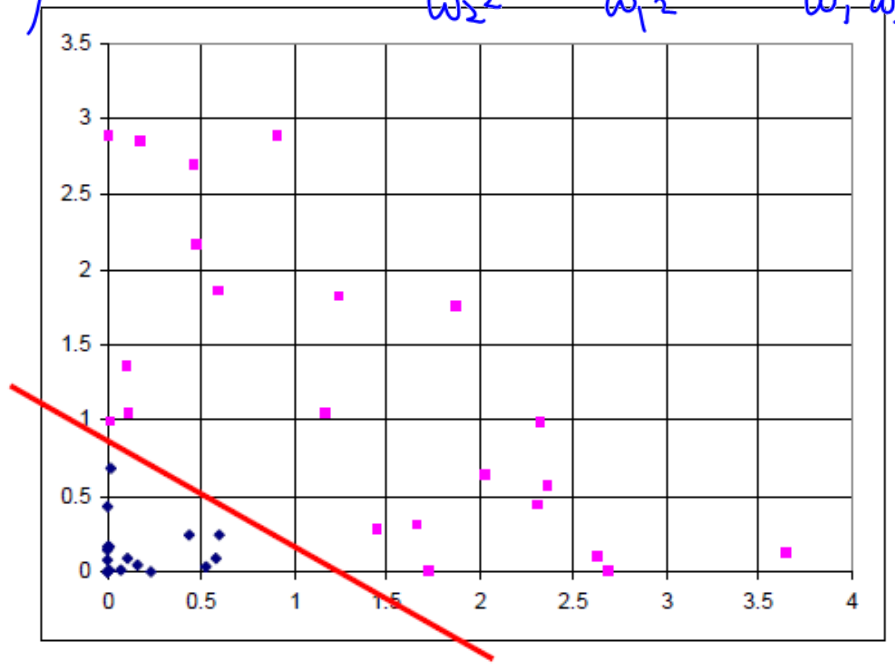
$= w_0 + w_1 x^2 + w_2 y^2 = 0$

$\Longleftrightarrow \frac{x^2}{w_2^2} + \frac{y^2}{w_1^2} = -\frac{w_0}{w_1^2 w_2^2}$

xy



Not linearly separable



Linearly separable

So different expansions make the problem solvable with linear methods!

Classifiers on steroids

- Create polynomial combinations of the original features, up to some order
- Put these in the classifier instead of the original ones
- Problem: curse of dimensionality

Curse of dimensionality

$x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n}$ $\quad d_1 + d_2 + \cdots + d_n = d$

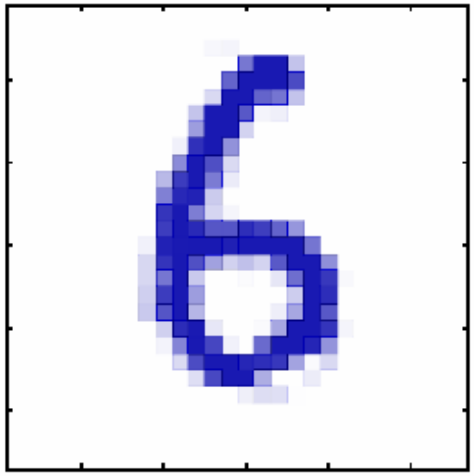
- Dimensionality n, polynomials up to order d

$$n_f = \frac{(n+d-1)!}{d!(n-1)!}$$

sampling r times from n types of elements (with repetition).

$${}_nH_r = {}_{n+r-1}C_r$$

- Suppose we use pixel values of small images, e.g. 28 x 28 = 784 pixels
- d=1 n_f=784
- d=2 n_f=307k
- d=3 n_f=80M
- d=4 n_f=16G



Kernels to the rescue!

1. Embed data in a high dimensional space
2. Use simple models (linear relations) in this space
3. Use algorithms that do *not need the coordinates of the embedded points, but only pairwise inner products*
4. Compute these inner products efficiently using a kernel

The kernel trick

- It would be nice if we could work in a higher dimensional space *without increasing computational complexity*
- Solution: make mapping *implicit*, instead of *explicit*

Kernel Functions

- A kernel function *k*(x,x') is intended to represent the similarity between x and x'.

- A popular way to express similarity is as the inner product of feature vectors:

# of training examples.

$N \times N$

$$k(\mathbf{x}, \mathbf{x}') = \boxed{\phi(\mathbf{x})^T \phi(\mathbf{x}')} \rightarrow K \in \mathbb{R}^{N \times N}$$

example 1 example 2

- We *define* a kernel function *k*(x,x') as one that can be expressed as an inner product, but we may not need to compute it that way.

2D Example

- Normal inner product between two vectors

$\phi(x) = \underline{(x_1, x_2)}$ and $\underline{(y_1, y_2)} = \phi(y)$

example 1 example 2

$$k(x, y) = x_1 y_1 + x_2 y_2 = \phi^T(x)\,\phi(y)$$

$\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$

- Let's replace this by its square

$$k(x, y) = (x_1 y_1 + x_2 y_2)^2 = x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 y_1 x_2 y_2$$

- This is the same as the regular inner product between $(x_1, x_2, \sqrt{2}x_1x_2)$ and $(y_1, y_2, \sqrt{2}y_1y_2)$
- Or between $(x_1, x_2, x_1x_2, x_2x_1)$ and $(y_1, y_2, y_1y_2, y_2y_1)$

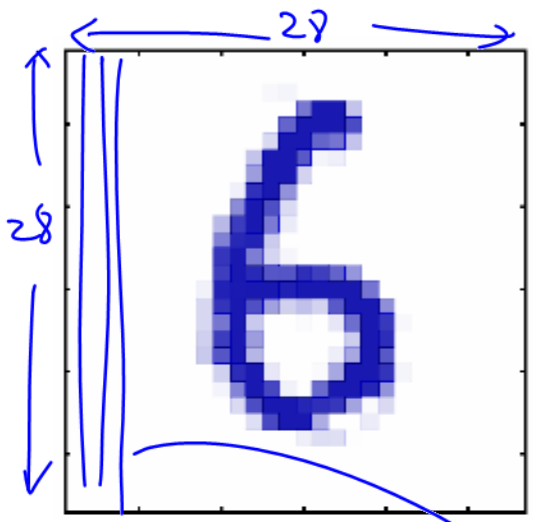
Solution not unique!

Example

- Take the pixel values and compute

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d \in \mathbb{R}$$

$x \in \mathbb{R}^{784}$



$x = \boxed{6}$ $x' = \boxed{9}$

and you compute the inner product in the space of all polynomials (for dim(**x**)=784 and *d*=4 a 16G dimensional space!)

28
28
784

Kernel trick

- So by using different definitions for inner product, we can compute inner products in a high dimensional space, with only the computational complexity of a low dimensional space
- Many algorithms can be expressed completely in terms of kernels $k(x,x')$, rather than other operations on x.
- In this case, you can replace one kernel with another, and get a new algorithm that works over a different domain.

Kernel trick

- The dual representation, and its solutions, are entirely written in terms of kernels.
- The elements of the Gram matrix finite dim $\mathbf{K} = \underline{\mathbf{\Phi}\mathbf{\Phi}^T}$
  - are $$K_{nm} = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$$
- These represent the pairwise similarities among all the observed feature vectors.
  - We may be able to compute the kernels more efficiently than the feature vectors.

Kernel substitution

- To use the kernel trick, we must formulate (training and test) algorithms purely in terms of inner products between data points
- We can *not* access the coordinates of points in the high-dimensional feature space
- This seems a huge limitation, but it turns out that quite a lot can be done

Example: distance

- Distance between samples can be expressed in inner products:

$$(\phi(x) - \phi(z))^T (\phi(x) - \phi(z))$$

$$\begin{aligned}
||\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\mathbf{z})||^2 &= \langle \boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\mathbf{z}), \boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\mathbf{z}) \rangle \\
&= \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}) \rangle - 2\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle + \langle \boldsymbol{\phi}(\mathbf{z}), \boldsymbol{\phi}(\mathbf{z}) \rangle \\
&= \kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}, \mathbf{z}) + \kappa(\mathbf{z}, \mathbf{z})
\end{aligned}$$

- So nothing stops you from doing k-nearest neighbor searches in high dimensional spaces

Example: the mean

- Can you determine the mean of data in the mapped feature space through kernel operations only → no, you cannot compute any point explicitly

Example: distance to mean

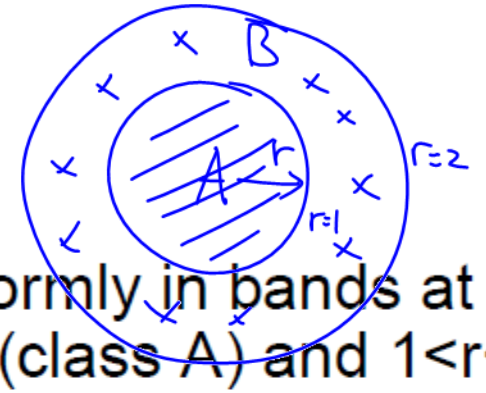
- Mean of a set given by $\phi_S = \frac{1}{l}\sum_{i=1}^{l}\phi(\mathbf{x}_i)$
- Distance to mean:

$$||\phi(\mathbf{x}) - \phi_s||^2 = \langle\phi(\mathbf{x}), \phi(\mathbf{x})\rangle + \langle\phi_S, \phi_S\rangle - 2\langle\phi(\mathbf{x}), \phi_S\rangle$$

$$= \kappa(\mathbf{x}, \mathbf{x}) + \frac{1}{l^2}\underbrace{\sum_{i,j=1}^{l}\kappa(\mathbf{x}_i, \mathbf{x}_j)}_{} - \frac{2}{l}\sum_{i=1}^{l}\kappa(\mathbf{x}, \mathbf{x}_i)$$

Matrix of all pairwise inner products **K** called the Gram matrix. This term is average of the Gram matrix

Exercise



- Create 2-class, nD data (vary n) uniformly in bands at distance r from the origin with 0<r<1 (class A) and 1<r<2 (class B). 100 points per class
- Method 1: Try the nearest mean classifier (NMC). Will it give good results?
- Method 2: Try an explicit feature transformation that results in good classification performance with NMC
- Method 3: Try an implicit feature transform (kernel trick) with a polynomial kernel and a kernel formulation of the nearest mean
- Compare performance and computational complexity of the three methods.

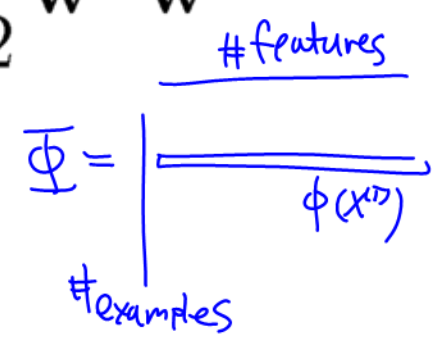
Dual Representations

- Recall regression problems with error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\phi(x_n) - t_n\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- $J(\mathbf{w})$ is minimized at

$$\mathbf{w}_{ML} = (\underbrace{\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi}}_{M \times M})^{-1}\mathbf{\Phi}^T\mathbf{t}$$



- Recall the N x M design matrix that is central to this solution.
- We can approach the solution a different way

The Design Matrix

- The design matrix is an NxM matrix, applying
  - the M basis functions (across)
  - to N data points (down)

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \phi_0(x_i) & \phi_1(x_i) & \cdots & \phi_{M-1}(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

feature mapping of $x^{(i)}$

$$\Phi \mathbf{w} \approx \mathbf{t}$$

The Gram Matrix

$(\Phi^T\Phi)_{ij}$ = covariance of feature i & feature j

- For regression, a key term is the *M*x*M* matrix

$$\mathbf{\Phi}^T\mathbf{\Phi}$$

"Covariance"

- Here, we will use the NxN *Gram matrix*

K_{nm} = inner product of $\phi(x^{(n)}), \phi(x^{(m)})$ similarity bet. n-th, m-th examples

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$$

"pairwise similarity"

- Note that $K_{nm} = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$
  - The pairwise similarities of all the data points in the training set.
- Note that kernel methods use only K, not Φ.

Dual Representations

$$\Phi^T = \begin{bmatrix} | & | & & | \\ \phi(x^{(1)}) & \phi(x^{(2)}) & \cdots & \phi(x^{(N)}) \\ | & | & & | \end{bmatrix}$$

- Another way to minimize $J(\mathbf{w})$ is

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\underbrace{\mathbf{w}^T\phi(\mathbf{x}_n)}_{\text{scalar}} - t_n\}\phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\phi(\mathbf{x}_n) = \mathbf{\Phi}^T\mathbf{a}$$

scalar $\in \mathbb{R}^M$

a_n

- where

$$a_n = -\frac{1}{\lambda}\{\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\}$$

$w = \Phi^T a$

unknown variable that we want to solve.

- Let a be the parameter, instead of w.
- Transform $J(\mathbf{w})$ to $J(\mathbf{a})$ by substituting

$$\mathbf{w} = \mathbf{\Phi}^T\mathbf{a}$$

$$\underline{\Phi}^T a = \begin{bmatrix} | & | & & | \\ \phi(x^{(1)}) & \phi(x^{(2)}) & \cdots & \phi(x^{(N)}) \\ | & | & & | \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix}$$

$$= \sum_i \phi(x^{(i)}) a_i$$

Dual Representations

- Transform $J(\mathbf{w})$ to $J(\mathbf{a})$ by using $\mathbf{w} = \mathbf{\Phi}^T \mathbf{a}$
- and the *Gram* matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$ $\cancel{\Phi^T \Phi}$
- Find a to minimize $J(\mathbf{a})$: $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{t}$
- For predictions:

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{a}^T \mathbf{\Phi} \phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$$

- where

$$\mathbf{k}(\mathbf{x}) \text{ has elements } k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$$

Regularized linear regression

Training

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}'\boldsymbol{\phi}(\mathbf{x}) = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x})$$

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\overset{\Phi^T a}{\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_n)} - t_n)^2 + \frac{\lambda}{2}\overset{\Phi^T a}{\mathbf{w}'\mathbf{w}}$$

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}(\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_n) - t_n)\boldsymbol{\phi}(\mathbf{x}_n) = \sum_{n=1}^{N} a_n \boldsymbol{\phi}(\mathbf{x}_n) = \boldsymbol{\Phi}'\mathbf{a}$$

$w = \Phi^T a$

$$\text{with } a_n = -\frac{1}{\lambda}(\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_n - t_n)$$

$$\Rightarrow J(\mathbf{a}) = \frac{1}{2}\mathbf{a}'\underbrace{\boxed{\boldsymbol{\Phi}\boldsymbol{\Phi}'}}_{K}\underbrace{\boxed{\boldsymbol{\Phi}\boldsymbol{\Phi}'}}_{K}\mathbf{a} - \mathbf{a}'\underbrace{\boxed{\boldsymbol{\Phi}\boldsymbol{\Phi}'}}_{K}\mathbf{t} + \frac{1}{2}\mathbf{t}'\mathbf{t} + \frac{\lambda}{2}\mathbf{a}'\underbrace{\boxed{\boldsymbol{\Phi}\boldsymbol{\Phi}'}}_{K}\mathbf{a}$$

$$\Rightarrow J(\mathbf{a}) = \cancel{2}\frac{1}{\cancel{2}}\cancel{\mathbf{a}'}\mathbf{K}\mathbf{K}\underline{\mathbf{a}} - \cancel{\mathbf{a}'}\mathbf{K}\mathbf{t} + \frac{1}{\cancel{2}}\cancel{\mathbf{t}'}\mathbf{t} + \cancel{2}\frac{\lambda}{\cancel{2}}\cancel{\mathbf{a}'}\mathbf{K}\underline{\mathbf{a}}$$

$$\Rightarrow \nabla_a J = KKa - Kt + \lambda Ka = K\left[(K+\lambda I)a - t\right] = 0$$

$$\Rightarrow \mathbf{a} = (\underbrace{\mathbf{K} + \lambda\mathbf{I}_N}_{N \times N})^{-1}\mathbf{t}$$

$N \ll M$

Testing

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})'\boxed{(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}} = a$$

$\mathbf{k}(\mathbf{x})$ has elements $\kappa_n(\mathbf{x}_n, \mathbf{x})$

Testing : given test example x

$$w^T \phi(x) \geq 0$$

$$\boxed{w = \Phi^T a}$$

$$= \Phi^T (K + \lambda I)^{-1} t$$

$$\Rightarrow (a^T \Phi)\, \phi(x)$$

$$= a^T \begin{bmatrix} - \phi(x^{(1)}) - \\ - \phi(x^{(2)}) - \\ \vdots \\ - \phi(x^{(N)}) - \end{bmatrix} \phi(x) = \sum_i \underset{\substack{\| \\ a^{(i)}}}{a_i} \underbrace{\phi(x^{(i)})^T \phi(x)}_{\underbrace{K(x^{(i)}, x)}_{\equiv k_{(i)}}}$$

$[a_1\ a_2 \cdots a_N]$

$$= a^T k$$

$$= \sum_i a^{(i)} K(x^{(i)}, x)$$

Primal versus dual

- Primal: $\mathbf{w} = (\mathbf{\Phi}'\mathbf{\Phi} + \lambda \mathbf{I}_M)^{-1}\mathbf{\Phi}'\mathbf{t}$
- Dual: $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{t}$
- Primal: invert *M* by *M* matrix (*M* = dim feature space), *w* vector of length *M* $O(M^3)$
- Dual: invert *N* by *N* matrix (*N* = nr data points), *a* vector of length *N* $O(N^3)$
- Primal: cheaper because usually *N* > *M*
- Dual: can use the kernel trick!!!

So: What is the biggest problem of kernel trick methods?

Memory-Based Methods

- Store many instances x in their surface representation.

$\{ K(x^{(i)}, x) \}_{i=1,\cdots,N}$

test examples

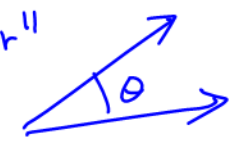
- Use kernels $k(x,x')$ to represent similarity.

$= \phi(x)^T \phi(x')$

$\|\phi(x)\| = \|\phi(x')\| = 1$

"similar"

$K_{xx'}$: large



$\phi^T(x)\phi(x') = \cos\theta$

- Kernels can be defined over vectors, images, sequences, graphs, text, etc.

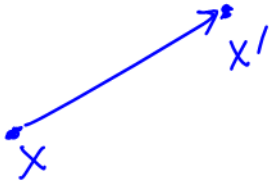
"dissimilar"

$K_{xx'}$: small

Simple Types of Kernels

- Inner product in Euclidean spaces
  - $k(\mathbf{x},\mathbf{x}') = \mathbf{x}^T\mathbf{x}'$ $\Leftrightarrow$ primal.

- *Stationary kernels* depend only on difference
  - $\underbrace{k(\mathbf{x},\mathbf{x}')}_{\text{scalar}} = k(\mathbf{x}-\mathbf{x}')$



- *Radial basis functions* depend only on the magnitude of the difference
  - $k(\mathbf{x},\mathbf{x}') = k(\underbrace{||\mathbf{x}-\mathbf{x}'||}_{r})$

Constructing Kernels 1

- One can do *kernel engineering* to create kernels for particular purposes, expressing different kinds of similarity. $x \longmapsto \phi(x)$

- Method 1: One way is to define the feature space mapping $\phi(\mathbf{x})$ and then define the kernel

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

Constructing Kernels 1

- Define a kernel function directly, such as

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$$

- In 2D, we can explicitly identify the feature map $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$
- such that $\underbrace{k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})}$

$$\underset{(x_1, x_2)}{\overset{\shortparallel}{x}} \rightarrow \phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

- But these can be very complex.
  - Kernels help us avoid that complexity.

Constructing Kernels 2

- A simpler way to test without having to construct $\Phi(x)$:
- Use the necessary and sufficient condition that for a function $k(x,x')$ to be a valid kernel:
  - the Gram matrix K, whose elements are given by $k(x_n,x_m)$, should be positive semidefinite for all possible choices of the set $\{x_n\}$
  - I.e., K is positive semidefinite:

$$a^T K a \equiv \sum_{ij} a_i K_{ij} a_j \geq 0, \forall a \in R^N$$

Constructing Kernels 3

- There are a number of axioms that help us construct new, more complex kernels, from simpler known kernels. For example,

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$$

$x \mapsto \phi_1(x)$

$x' \mapsto \phi_1(x')$

$k_1(x, x') = \phi_1(x)^T \phi_1(x')$

- Prove that the Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2)$$

- is a valid kernel.

vector

$x \longmapsto f(x)\, \phi_1(x)$

$x' \longmapsto f(x')\, \phi_1(x')$

Constructing Kernels 3

- Building kernels out of simpler kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Most popular kernels

- Simple Polynomial Kernel (terms of degree 2)

$$\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}'\mathbf{z})^2$$

- Generalized Polynomial kernel – degree M

$$\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}'\mathbf{z} + \overset{1}{c})^M, c > 0$$

- Gaussian Kernels

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp(-\frac{||\mathbf{x} - \mathbf{z}||^2}{2\sigma^2})$$

- Sigmoidal Kernels (Gram Matrix not p.d.)

$$\kappa(\mathbf{x}, \mathbf{z}) = \tanh(a\mathbf{x}'\mathbf{z} + b)$$

Gaussian kernel

- Not related to Gaussian pdf $k(x,x') = f(\|x - x'\|)$
- Translation invariant (depends only on distance between points)
- Corresponds to an infinitely dimensional space! (PRMLex6.11)

Kernel regression

Radial Basis Functions

- Basis functions can be chosen that depend only on distance from selected centers:

$$\phi_j(\mathbf{x}) = h(||\mathbf{x} - \mu_j||)$$

- A function *f*(x) can be approximated as a linear combination of the basis functions

$$f(\mathbf{x}) = \sum_{n=1}^{N} w_n h(||\mathbf{x} - \mu_n||)$$

- With a basis function at each training data point, the approximation is exact on the training data.

Kernel Regression

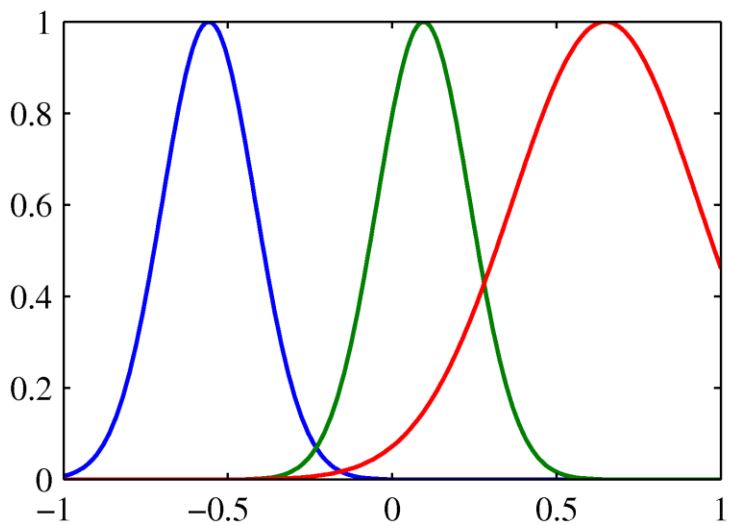
- Using radial basis functions around the training data points, predict a value $y(\mathrm{x})$ as the average of target values t_n, weighted by similarities $k(\mathrm{x},\mathrm{x}_n)$:

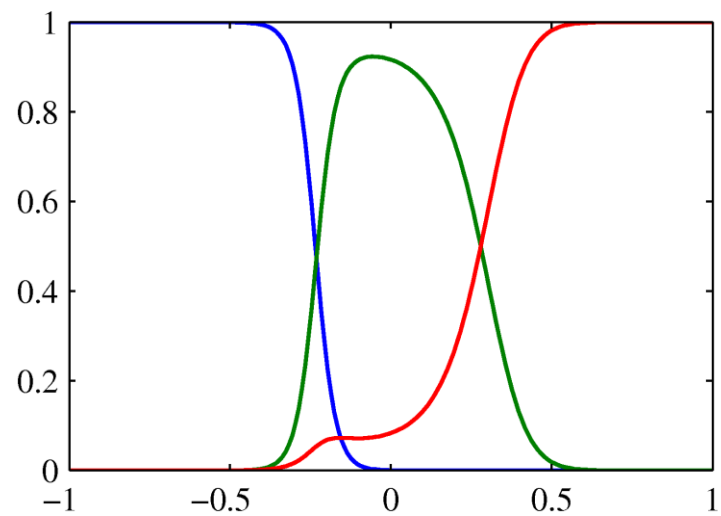
$$y(\mathbf{x}) = \sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) t_n$$

Kernel Normalization

- The weighted average approach assumes

$$\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) = 1$$





Narayada-Watson model

- From kernel density estimation:

$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^{N} \underline{f(\mathbf{x} - \mathbf{x}_n, t - t_n)}$$

Joint density function centered at x_n, t_n

  – where f(x,t) is the component density function and there is one such component centred on each data point

- We now find an expression for the regression function y(x), corresponding to the conditional average of the target variable conditioned on the input variable

Narayada-Watson model

$$
\begin{aligned}
y(\mathbf{x}) &= \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{\infty} t p(t|\mathbf{x})\,\mathrm{d}t \\
&= \frac{\int t p(\mathbf{x}, t)\,\mathrm{d}t}{\int p(\mathbf{x}, t)\,\mathrm{d}t} \\
&= \frac{\sum_n \int t f(\mathbf{x} - \mathbf{x}_n, t - t_n)\,\mathrm{d}t}{\sum_m \int f(\mathbf{x} - \mathbf{x}_m, t - t_m)\,\mathrm{d}t}. \qquad (6.43)
\end{aligned}
$$

We now assume for simplicity that the component density functions have zero mean so that

$$
\int_{-\infty}^{\infty} f(\mathbf{x}, t) t\,\mathrm{d}t = 0 \qquad (6.44)
$$

for all values of $\mathbf{x}$. Using a simple change of variable, we then obtain

$$
\begin{aligned}
y(\mathbf{x}) &= \frac{\sum_n g(\mathbf{x} - \mathbf{x}_n) t_n}{\sum_m g(\mathbf{x} - \mathbf{x}_m)} \\
&= \sum_n k(\mathbf{x}, \mathbf{x}_n) t_n \qquad (6.45)
\end{aligned}
$$

Narayada-Watson model

- Prediction function:

$$
\begin{aligned}
y(\mathbf{x}) &= \frac{\sum_n g(\mathbf{x}-\mathbf{x}_n)t_n}{\sum_m g(\mathbf{x}-\mathbf{x}_m)} \\
&= \sum_n k(\mathbf{x},\mathbf{x}_n)t_n
\end{aligned}
$$

  – where

$$
k(\mathbf{x},\mathbf{x}_n) = \frac{g(\mathbf{x}-\mathbf{x}_n)}{\sum_m g(\mathbf{x}-\mathbf{x}_m)}
$$

$$
g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x},t)\,\mathrm{d}t.
$$

Narayada-Watson model

- This model is also known as kernel regression.
- For a localized kernel function, it has the property of giving more weight to data points that a close to x

Kernel Regression Example

- On the familiar sinusoidal data set: