Machine Learning 10-601

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

November 6, 2017

Today: Kernel methods, SVM

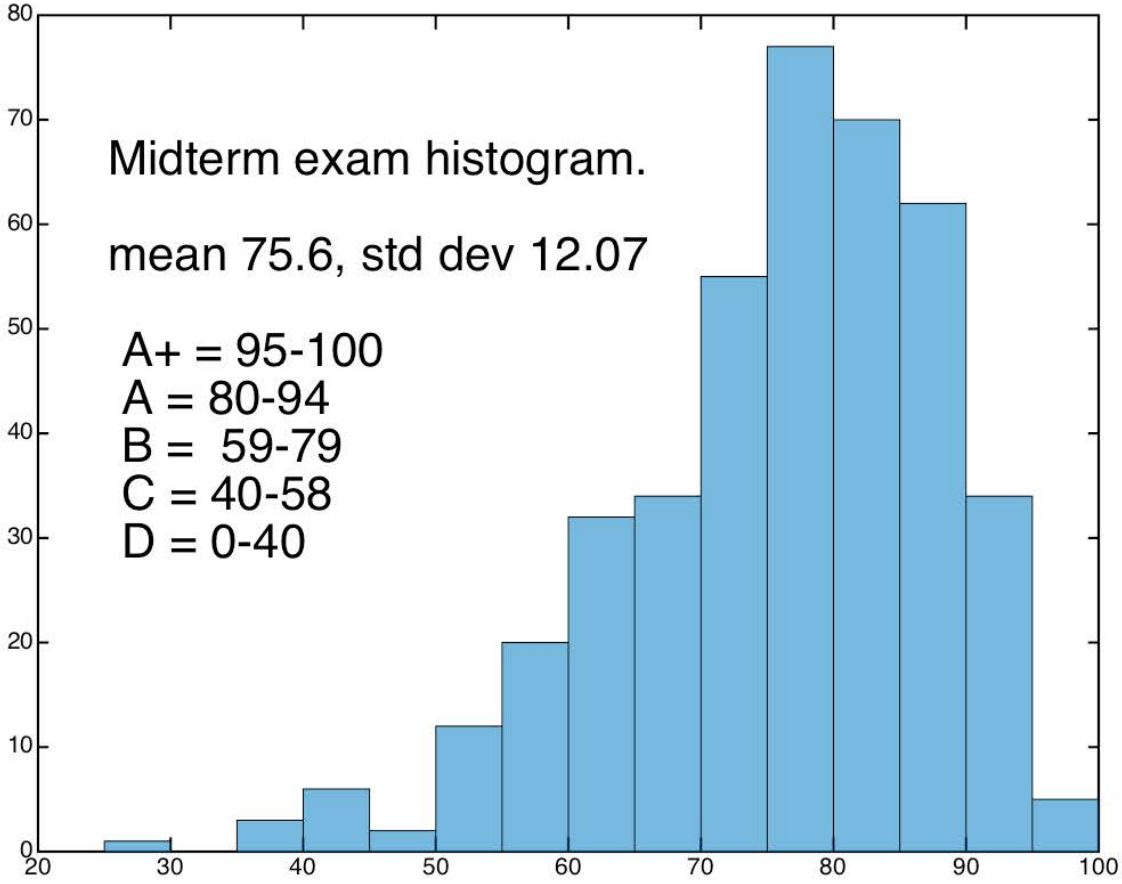
- Regression: Primal and dual forms
- Kernels for regression

Readings:

Recommended:
Kernels: Bishop Ch. 6.1
SVMs: Bishop Ch. 7, through 7.1.2

Optional:
Bishop Ch 6.2, 6.3

Thanks to Aarti Singh, Eric Xing, Maria Balcan, John Shawe-Taylor for several slides

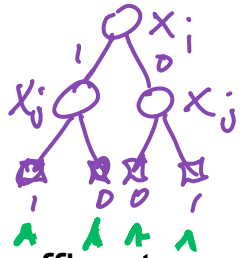

Midterm exam histogram.
mean 75.6, std dev 12.07
A+ = 95-100
A = 80-94
B = 59-79
C = 40-58
D = 0-40
0
10
20
30
40
50
60
70
80
20
30
40
50
60
70
80
90
100

Example: Depth 2 Decision Trees

$$m \geq \frac{1}{\epsilon}(\ln |H| + \ln(1/\delta))$$

Consider classification problem f:X→Y:

- instances: $X = \langle X_1 \ldots X_N \rangle$ where each X_i is boolean
- learned hypotheses are decision trees of depth 2, using only two variables



$$|H| = \binom{n}{2} \cdot 16 = \frac{N(N-1)}{2} 16$$

$$|H| = 8N(N-1)$$

How many training examples *m* suffice to assure that with probability at least 0.99, *any* learner that outputs a consistent depth 2 decision tree will have true error at most 0.05?

$$m \geq \frac{1}{0.05}\left(\ln(8N^2 - 8N) + \ln\frac{1}{0.01}\right)$$

$N=4 \rightarrow m \geq 184$

$N=10, m \geq 224$

$N=100, m \geq 318$

Sample Complexity based on VC dimension

How many randomly drawn examples suffice to ε-exhaust $VS_{H,D}$ with probability at least (1-δ)?

ie., to guarantee that any hypothesis that perfectly fits the training data is probably (1-δ) approximately (ε) correct

$$m \geq \frac{1}{\epsilon}(4 \log_2(2/\delta) + 8VC(H) \log_2(13/\epsilon))$$

Compare to our earlier results based on |*H*|:

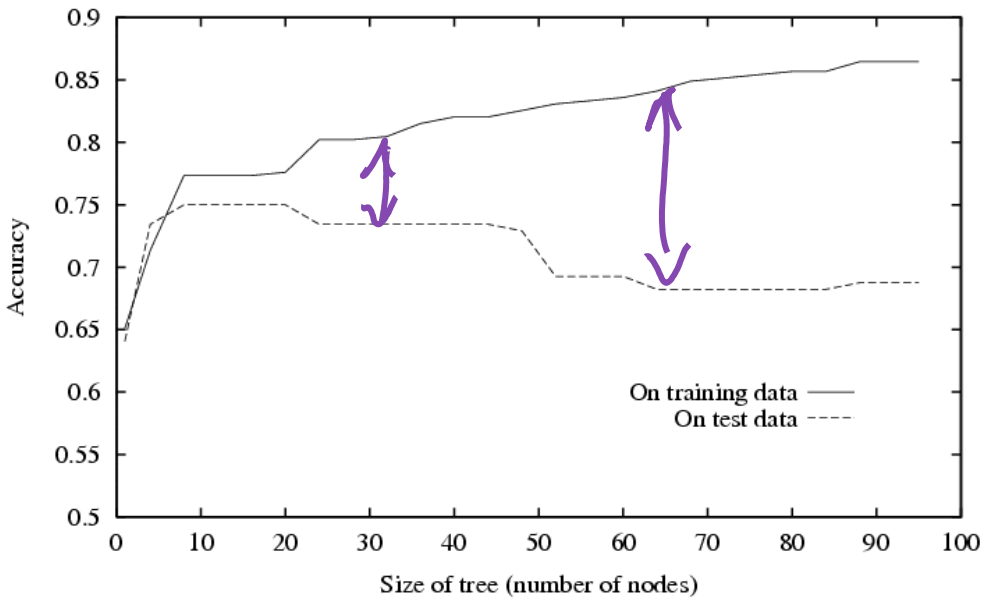
$$m \geq \frac{1}{\epsilon}(\ln(1/\delta) + \ln|H|)$$

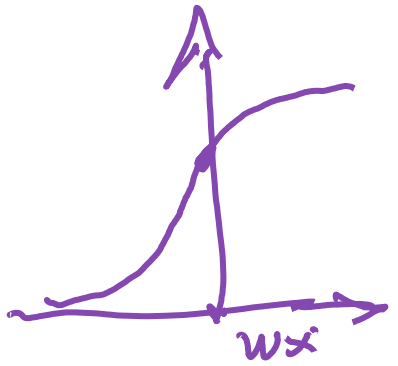
Agnostic Learning: VC Bounds

[Schölkopf and Smola, 2002]

With probability at least (1-δ) every $h \in H$ satisfies

$$error_{true}(h) < error_{train}(h) + \sqrt{\frac{VC(H)(\ln \frac{2m}{VC(H)} + 1) + \ln \frac{4}{\delta}}{m}}$$



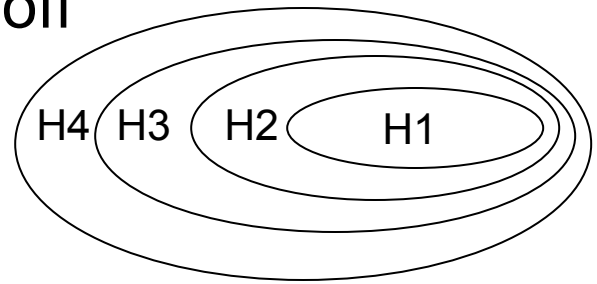


$$\frac{\partial \sigma(x)}{\partial x} = (1-0)0$$

Structural Risk Minimization [Vapnik]

Which hypothesis space should we choose?

- Bias / variance tradeoff



SRM: choose H to minimize bound on expected true error!

$$error_{true}(h) < error_{train}(h) + \sqrt{\frac{VC(H)(\ln \frac{2m}{VC(H)} + 1) + \ln \frac{4}{\delta}}{m}}$$

* nice idea, but unfortunately a somewhat loose bound...

What You Should Know

- Sample complexity varies with the learning setting
  - Learner actively queries trainer
  - Examples arrive at random
  - …

- Within the PAC learning setting, we can bound the probability that learner will output hypothesis with true error within ε of training error
  - For ANY consistent learner (case where $c \in H$, zero training error)
  - For ANY "best fit" hypothesis (agnostic learning, where perhaps c not in H)

- VC dimension as measure of complexity of H

- More results coming this semester

- Conference on Learning Theory: http://www.learningtheory.org
- ML Department course on Machine Learning Theory

Regression

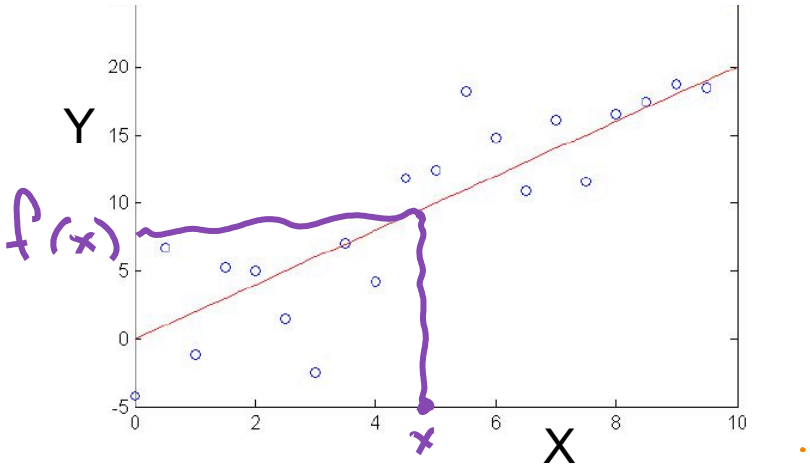
Wish to learn f:X→Y, where Y is real, given $\{<x^1,y^1>\ldots<x^n,y^n>\}$

Approach:

1. choose some parameterized form for P(Y|X; θ)
   (θ is the vector of parameters)

2. derive learning algorithm as MCLE or MAP estimate for θ

1. Choose parameterized form for P(Y|X; θ)



Let's assume Y is some deterministic f(X), plus random noise

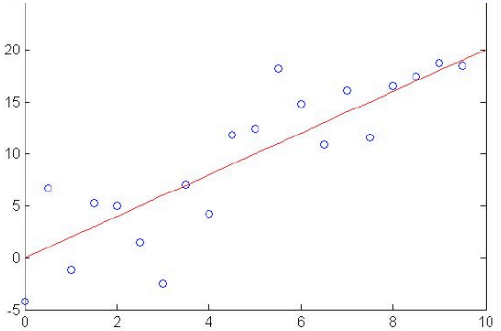
$$y = f(x) + \epsilon \qquad \text{where} \qquad \epsilon \sim N(0, \sigma)$$

Therefore Y is a random variable that follows the distribution

$$p(y|x) = N(f(x), \sigma)$$

and the expected value of y for any given x is f(x)

Consider Linear Regression



$$p(y|x; W) = N(w_0 + w_1 x, \sigma)$$

mean

How can we learn W from the training data?

Learn Maximum Conditional Likelihood Estimate!

$$W_{MCLE} = \arg\max_W \prod_l p(y^l|x^l, W)$$

$$W_{MCLE} = \arg\max_W \sum_l \ln p(y^l|x^l, W)$$

where

$$\ln p(y|x, W) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{y - (w_0 + w_1 x)}{\sigma}\right)^2\right)$$

ln( ) + ( )

therefore

$$W_{MCLE} = \arg\min_W \sum_l (y^l - (w_0 + w_1 x^l))^2$$

How about MAP instead of MLE estimate?

If we assume zero-mean Gaussian prior on each w_i

$$P(W = \langle w_1 \dots w_n \rangle | \mu = 0, \sigma) = \frac{1}{(\sqrt{2\pi\sigma^2})^n} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} w_i^2\right)$$

$$\ln P(W) \propto \sum_{i=1}^{n} w_i^2$$

Then:

$$W = \arg\max_W \ln P(W|\mu = 0, \sigma) + \sum_l \ln P(y^l|x^l; W)$$

$$= \arg\max_W -c\sum_i w_i^2 + \sum_l \ln P(y^l|x^l; W)$$

$$W_{MAP} = \arg\min_W c\sum_i w_i^2 + \sum_l (y^l - f(x^l; W))^2$$

Ridge regression

but what if y=f(x) is non-linear

Kernel Functions

- Kernel functions provide a way to manipulate data as though it were projected into a higher dimensional space, by operating on it in its original space

- This leads to efficient algorithms that learn
  - linear regression in implicit high dimension spaces, corresponding to non-linear regression in their actual input space
  - linear decision surfaces in implicit high dimension spaces, corresponding to nonlinear classifiers in their actual input space

- And is a key component of algorithms such as
  - Support Vector Machines
  - kernel regression
  - kernel PCA
  - kernel CCA
  -

Regularized Linear Regression

where $x_0 = 1$ → $y = \sum_{i=0}^{k} w_i x_i$

$y = w_0 + \sum_{i=1}^{k} w_i x_i$

Wish to learn f: X → Y, where X=<X_1, … X_n>, Y real-valued

Learn $\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} x_i w_i = \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}}\mathbf{w}$

inner product
dot product

$[x_1 \cdots x_n] \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$

where $\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{l=1}^{M} (y^l - \mathbf{x}^{\mathbf{T}l}\mathbf{w})^2 + \lambda \sum_{k}^{N} w_k^2$

$$\left\| \begin{bmatrix} y^1 \\ \vdots \\ y^M \end{bmatrix} - \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ & & \vdots & \\ x_1^M & & \cdots & x_n^M \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \right\|_2^2 + \left\| \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \right\|_2^2 >$$

Linear Regression

Wish to learn $f : X \rightarrow Y$, where $X = \langle X_1, X_2, \ldots X_N \rangle$, $Y \in \Re$

Learn $\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} x_i w_i = \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}} \mathbf{w}$

where $\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{l=1}^{M} (y^l - \mathbf{x}^{\mathbf{T}l} \mathbf{w})^2 + \lambda \sum_{k=1}^{N} w_k^2$

$$\mathbf{w} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2$$

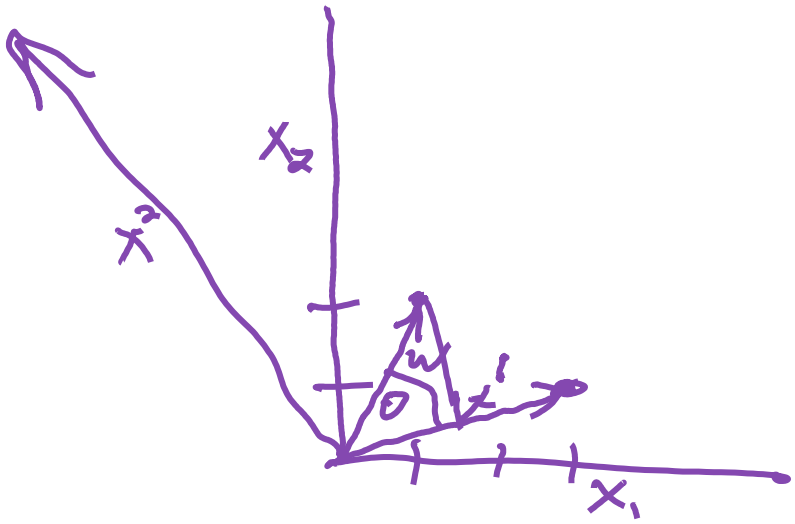
here the l^{th} row of $\mathbf{X}$ is the l^{th} training example $\mathbf{x}^{\mathbf{T}l}$

and $\|\mathbf{w}\|^2 = \sum_{k=1}^{N} w_k^2 = \|\mathbf{w}\|_2^2$

Vectors, Data Points, Inner Products

Consider $\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} x_i w_i = \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}} \mathbf{w}$

where $\mathbf{x}^T = [3\ 1]$ and $\mathbf{w}^T = [1\ 2]$, so $\hat{f}(\mathbf{x}) = 5$



$w = x'\alpha_1 + x^2\alpha_2$

$w = U_1 1 + U_2 2$

U_1 unit vector $[1, 0]$

$U_2 = [0\ 1]$

for any two vectors, their dot product (aka inner product) is equal to product of their lengths, times the cosine of angle between them.

$$\langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}} \mathbf{w} \equiv \sum_{i=1}^{N} x_i w_i = \|\mathbf{x}\| \|\mathbf{w}\| \cos(\theta)$$

Linear Regression: Primal Form

Learn $\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} x_i w_i = \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}}\mathbf{w}$

where $\mathbf{w} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2$

solve by taking derivative wrt $\mathbf{w}$, setting to zero…

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\ \mathbf{X}^T\mathbf{y}$$

so: $\hat{f}(\mathbf{x_{new}}) = \mathbf{x_{new}^T}\mathbf{w} = \mathbf{x_{new}^T}\ (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\ \mathbf{X}^T\mathbf{y}$

Aha!

Learn $\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} x_i w_i = \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}} \mathbf{w}$

where $\mathbf{w} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2$

solution: $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$

But notice w lies in the space spanned by training examples (why?)

Linear Regression: Dual Form

Primal form:

Learn $\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} x_i w_i = \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^{\mathbf{T}}\mathbf{w}$

$$\mathbf{w} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2$$

Solution: $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\,\mathbf{X}^T\mathbf{y}$

features × # features

Dual form: use fact that $\mathbf{w} = \sum_{l=1}^{M} \alpha_l\, \mathbf{x}^l$

Learn $\hat{f}(\mathbf{x}) = \sum_{l=1}^{M} \alpha_l \langle \mathbf{x}, \mathbf{x}^l \rangle$

$f(x) = x^T w$

$f(x) = x^T \sum_l^M \alpha_l x^l$

$= \sum_l \alpha_l\, x^T x^l$

Solution: $\alpha = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\,\mathbf{y}$

train examps × # train examps

Key ingredients of dual solution

Step 1: Compute

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ that is $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Step 2: Evaluate on new point $\mathbf{x}$ by

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Important observation: Both steps only involve inner products between input data points

[slide from John Shawe-Taylor]

Applying the 'kernel trick'

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ a kernel function κ that computes:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and we obtain an algorithm for ridge regression in the feature space F defined by the mapping

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in F$$

Note if ϕ is the identity this has no effect.

[slide from John Shawe-Taylor]

Kernel functions

Original space

Projected space
(higher dimensional)

Example: Quadratic Kernel

Suppose we have data originally in 2D, but project it into 3D using $\Phi(\mathbf{x})$

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

$f(x) = w_1 x_1 + w_2 x_2$ $\qquad$ $f(x) = w_1 x_1^2 + w_2 \sqrt{2} x_1 x_2 + w_3 x_2^2$

this converts our original linear regression into quadratic regression!

But we can use the following kernel function to calculate inner products in the projected 3D space, in terms of operations in the 2D space

$$\langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle = \langle \mathbf{x_i}, \mathbf{x_j} \rangle^2 \equiv \kappa_\Phi(\mathbf{x_i}, \mathbf{x_j})$$

And use it to train and apply our regression function, never leaving 2D space

$$\hat{f}(\mathbf{x}) = \sum_{l=1}^{M} \alpha_l \, \kappa(\mathbf{x}, \mathbf{x}^l) \qquad \alpha = (\mathbf{K} + \lambda \mathbf{I})^{-1} \, \mathbf{y} \qquad \mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Implications of the "Kernel Trick"

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$ pixels.

- By using the quadratic kernel we implement the regression function in a $1,000,000$ dimensional space

- but actually using less computation for the learning phase than we did in the original space – inverting a 1000×1000 matrix instead of a 1024×1024 matrix.

[slide from John Shawe-Taylor]