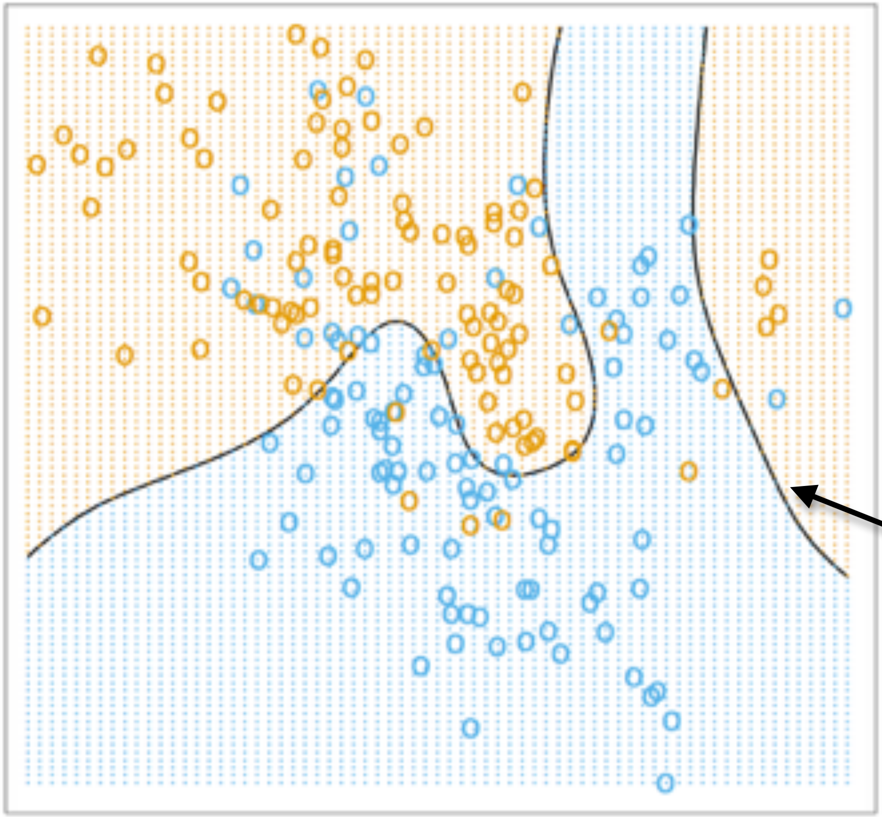
Nearest Neighbor

Machine Learning – CSE546

Kevin Jamieson
University of Washington

October 26, 2017

Some data, Bayes Classifier



Training data:

True label: +1

True label: -1

Optimal "Bayes" classifier:

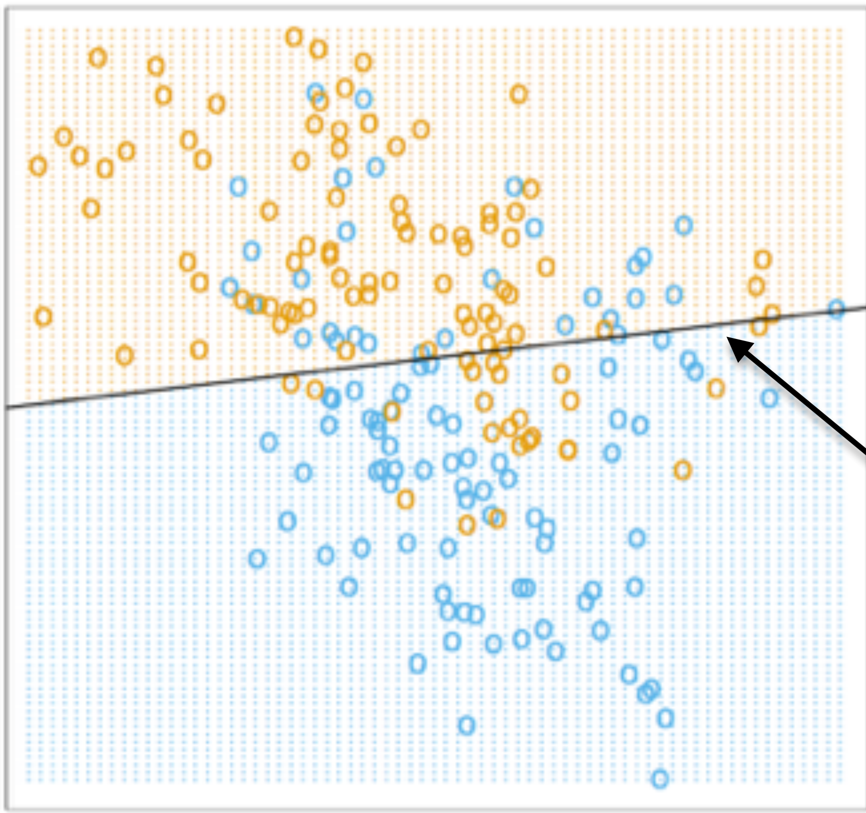
$$\mathbb{P}(Y = 1 | X = x) = \frac{1}{2}$$

Predicted label: +1

Predicted label: -1

Figures stolen from Hastie et al

Linear Decision Boundary



Training data:

True label: +1

True label: -1

Learned:

Linear Decision boundary

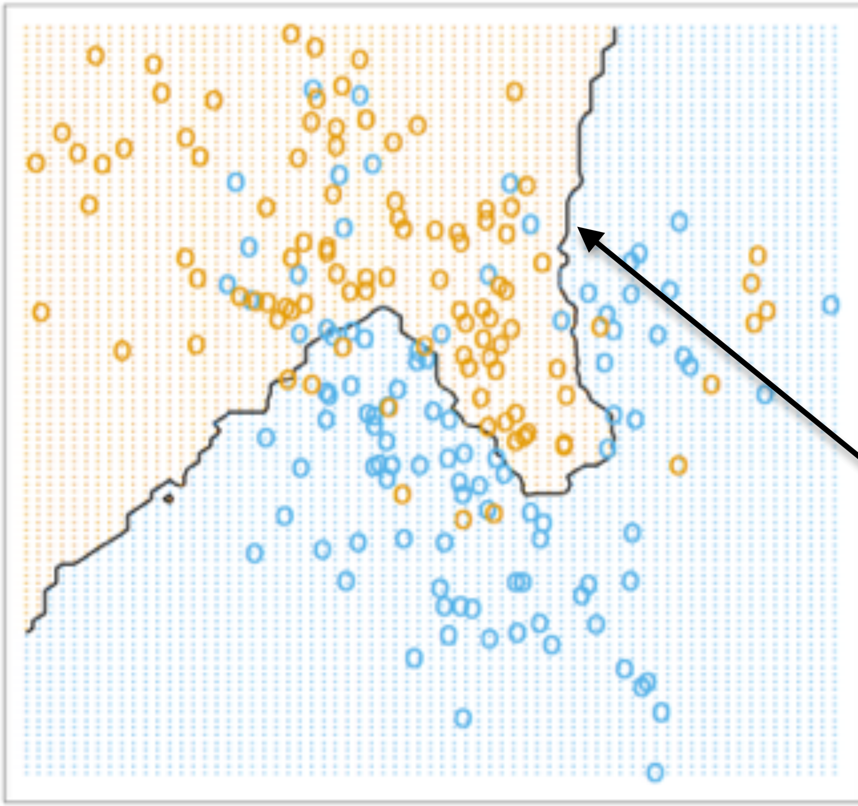
$$x^T w + b = 0$$

Predicted label: +1

Predicted label: -1

Figures stolen from Hastie et al

15 Nearest Neighbor Boundary



Training data:

True label: +1

True label: -1

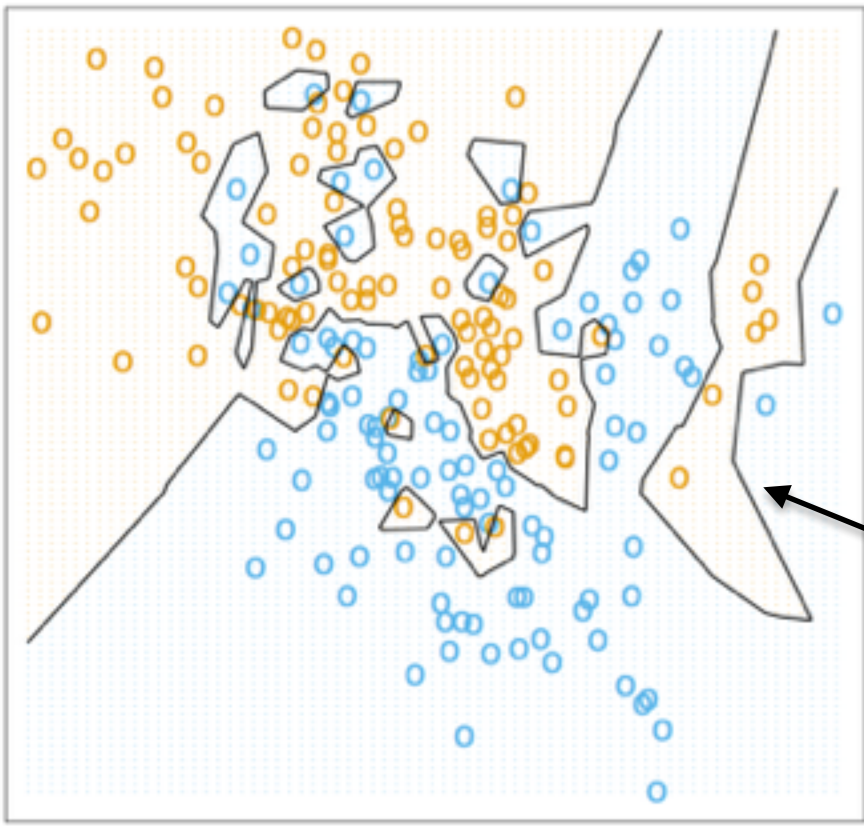
Learned:

15 nearest neighbor decision boundary (majority vote)

Predicted label: +1

Predicted label: -1

1 Nearest Neighbor Boundary



Training data:

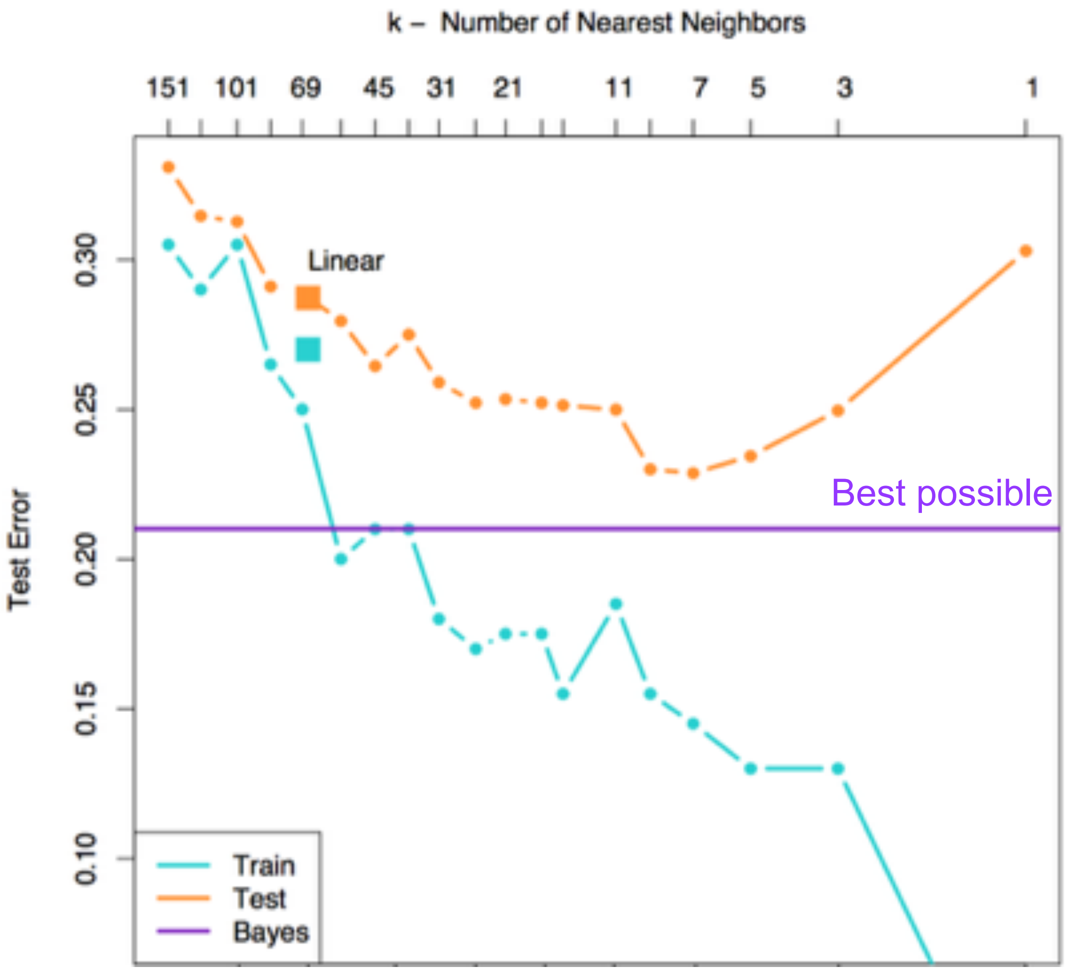
- True label: +1
- True label: -1

Learned:

1 nearest neighbor decision boundary (majority vote)

- Predicted label: +1
- Predicted label: -1

k-Nearest Neighbor Error



Bias-Variance tradeoff

As k->infinity?

Bias:

Variance:

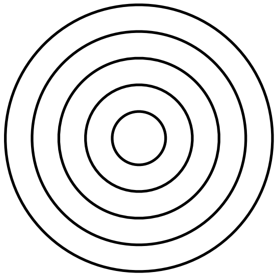
As k->1?

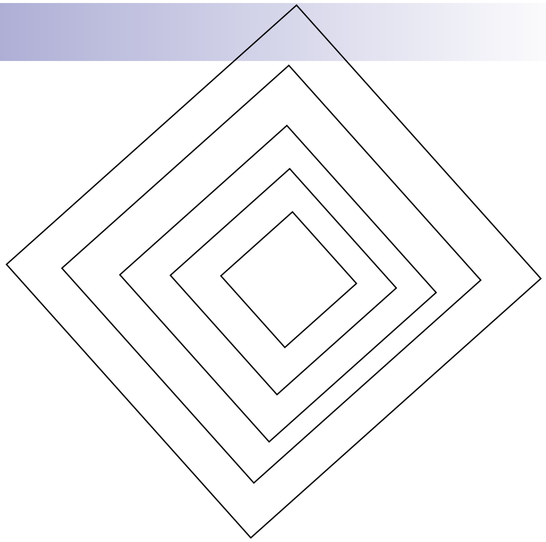
Bias:

Variance:

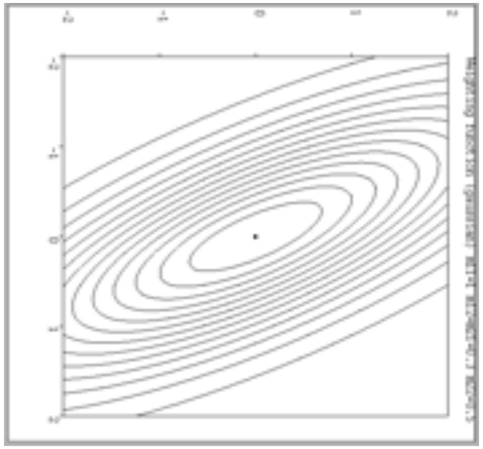
Notable distance metrics (and their level sets)

L_2 norm

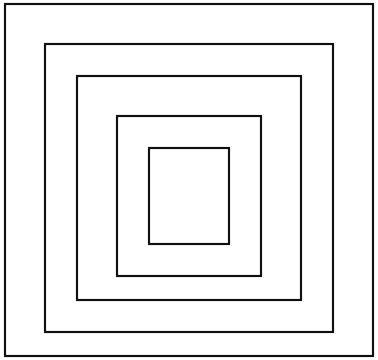




L_1 norm (taxi-cab)



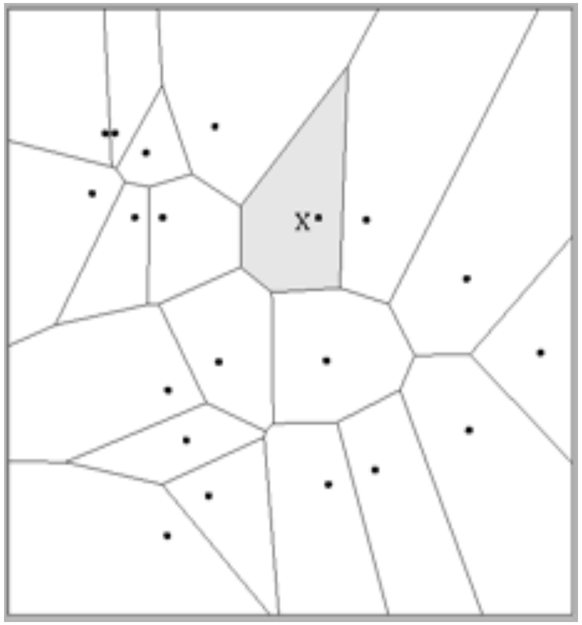
Mahalanobis (here, Σ on the previous slide is not necessarily diagonal, but is symmetric



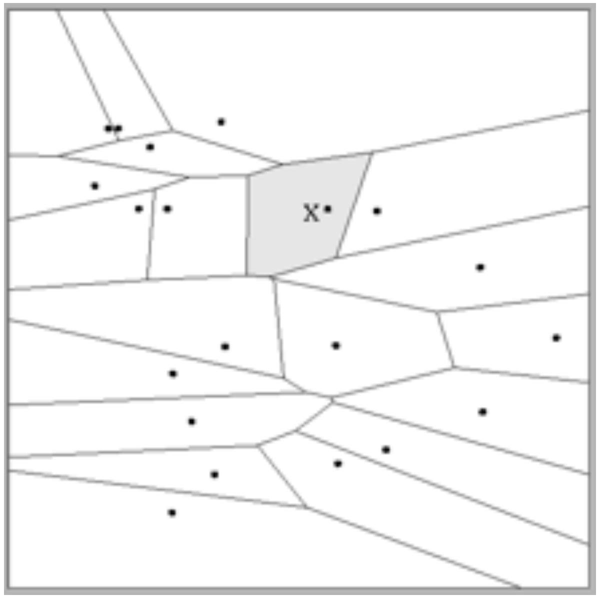
L1 *(max) norm*

1 nearest neighbor

One can draw the nearest-neighbor regions in input space.



$Dist(\mathbf{x}^i,\mathbf{x}^j) = (x^i_1 - x^j_1)^2 + (x^i_2 - x^j_2)^2$



$Dist(\mathbf{x}^i,\mathbf{x}^j) =(x^i_1 - x^j_1)^2+(3x^i_2 - 3x^j_2)^2$

The relative scalings in the distance metric affect region shapes

1 nearest neighbor guarantee

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{1, \dots, k\}$$

As $n \to \infty$, assume the x_i's become *dense* in $\mathbb{R}^d$

Note: any $x_a \in \mathbb{R}^d$ has the same label distribution as x_b with $b = 1NN(a)$

[Cover, Hart, 1967]

1 nearest neighbor guarantee

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{1, \dots, k\}$$

As $n \to \infty$, assume the x_i's become *dense* in $\mathbb{R}^d$

Note: any $x_a \in \mathbb{R}^d$ has the same label distribution as x_b with $b = 1NN(a)$

If $p_\ell = \mathbb{P}(Y_a = \ell) = \mathbb{P}(Y_b = \ell)$ and $\ell^* = \arg\max_{\ell=1,\dots,k} p_\ell$ then

$$\text{Bates error} = 1 - p_{\ell^*}$$

1 nearest neighbor guarantee

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{1, \dots, k\}$$

As $n \to \infty$, assume the x_i's become *dense* in $\mathbb{R}^d$

Note: any $x_a \in \mathbb{R}^d$ has the same label distribution as x_b with $b = 1NN(a)$

If $p_\ell = \mathbb{P}(Y_a = \ell) = \mathbb{P}(Y_b = \ell)$ and $\ell^* = \arg \max_{\ell=1,\dots,k} p_\ell$ then

$$\text{Bates error} = 1 - p_{\ell^*}$$

$$\text{1-nearest neighbor error} = \mathbb{P}(Y_a \neq Y_b) = \sum_{\ell=1}^k \mathbb{P}(Y_a = \ell, Y_b \neq \ell)$$

1 nearest neighbor guarantee

$$\{(x_i, y_i)\})_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{1, \dots, k\}$$

As $n \to \infty$, assume the x_i's become *dense* in $\mathbb{R}^d$

Note: any $x_a \in \mathbb{R}^d$ has the same label distribution as x_b with $b = 1NN(a)$

If $p_\ell = \mathbb{P}(Y_a = \ell) = \mathbb{P}(Y_b = \ell)$ and $\ell^* = \arg\max_{\ell=1,\dots,k} p_\ell$ then

$$\text{Bates error } = 1 - p_{\ell^*}$$

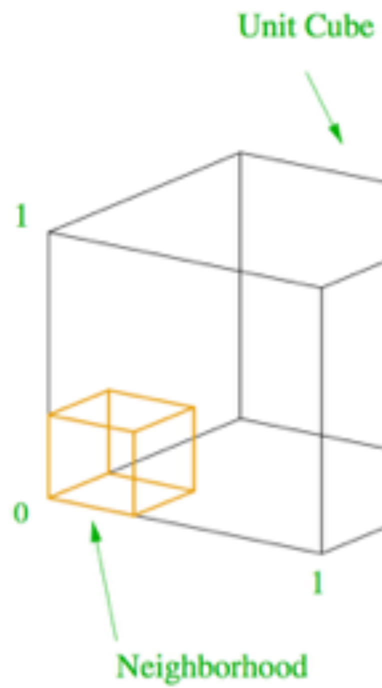
$$\text{1-nearest neighbor error } = \mathbb{P}(Y_a \neq Y_b) = \sum_{\ell=1}^k \mathbb{P}(Y_a = \ell, Y_b \neq \ell)$$

$$= \sum_{\ell=1}^k p_\ell (1 - p_\ell) \leq 2(1 - p_{\ell^*}) - \frac{k}{k-1}(1 - p_{\ell^*})^2$$

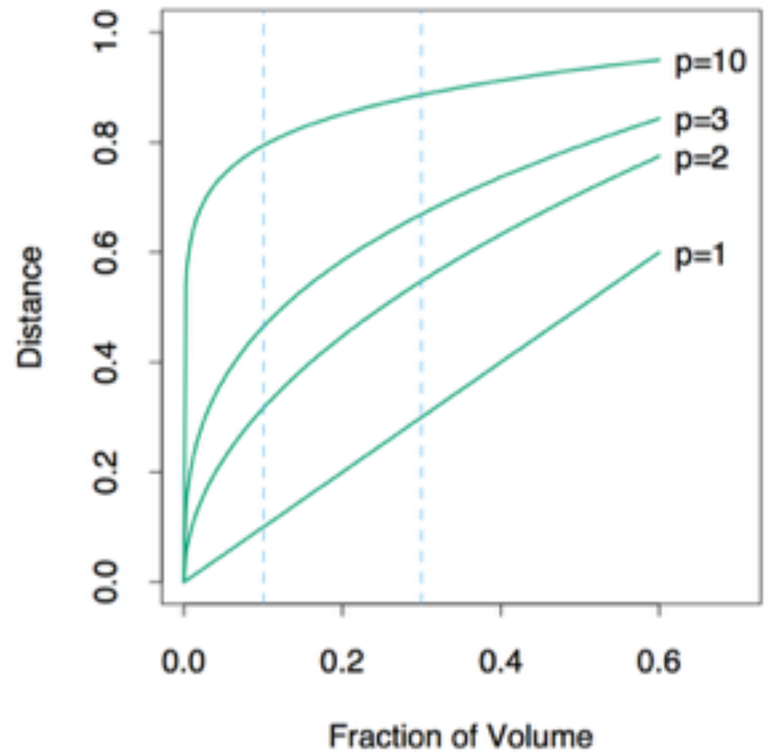
As x->infinity, then 1-NN rule error is at most twice the Bayes error!

[Cover, Hart, 1967]

Curse of dimensionality Ex. 1



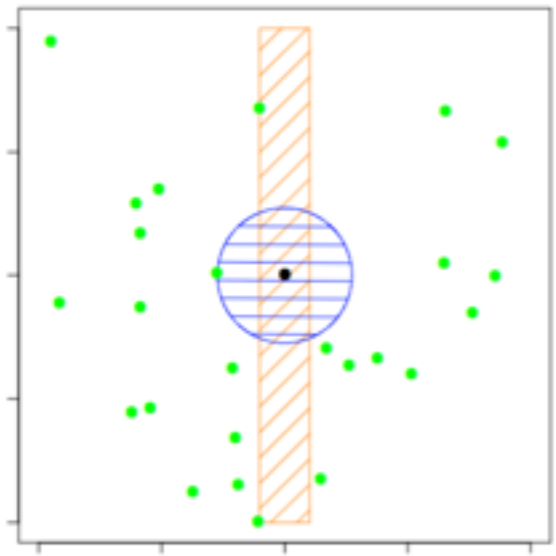
side length r

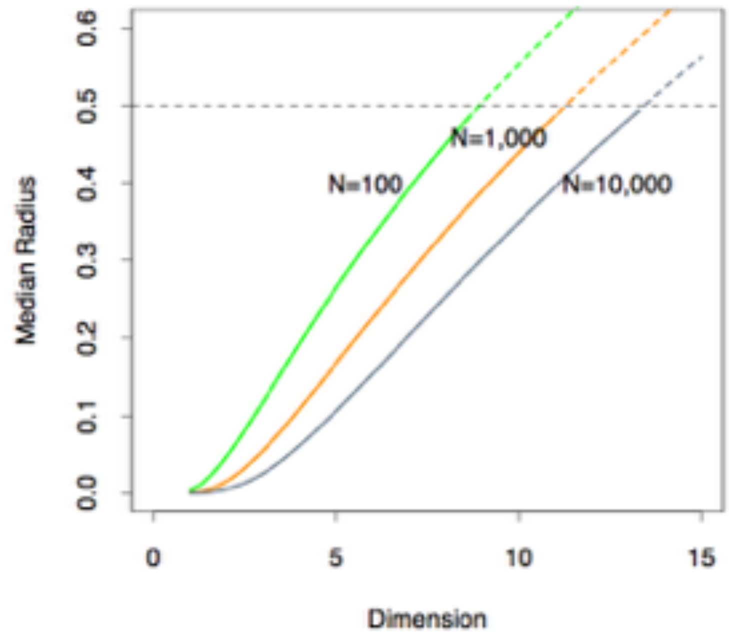


X is uniformly distributed over $[0,1]^p$. What is $\mathbb{P}(X \in [0,r]^p)$?

Curse of dimensionality Ex. 2

$\{X_i\}_{i=1}^n$ are uniformly distributed over $[-.5, .5]^p$.

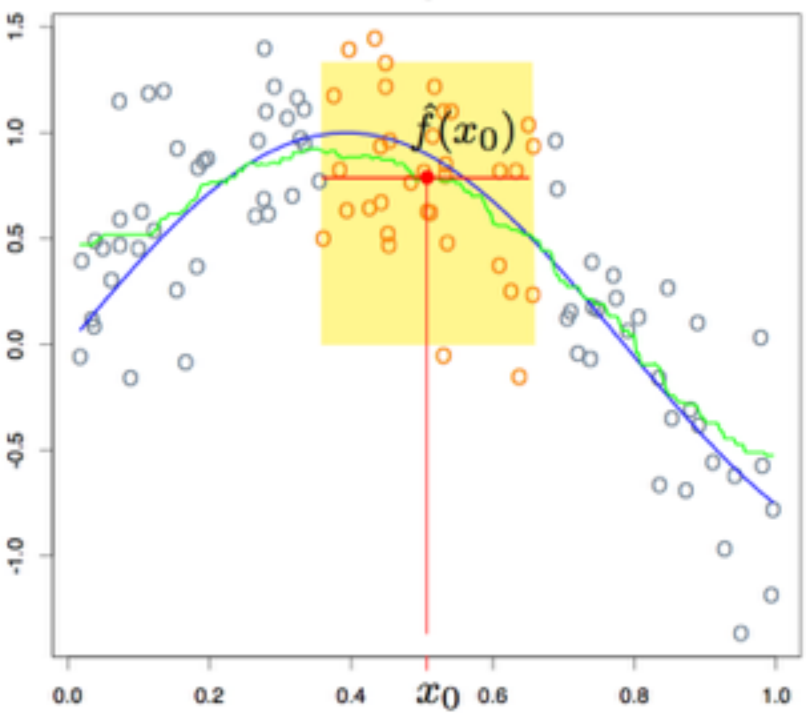




What is the median distance from a point at origin to its 1NN?

Nearest neighbor regression

$$\{(x_i, y_i)\}_{i=1}^n$$

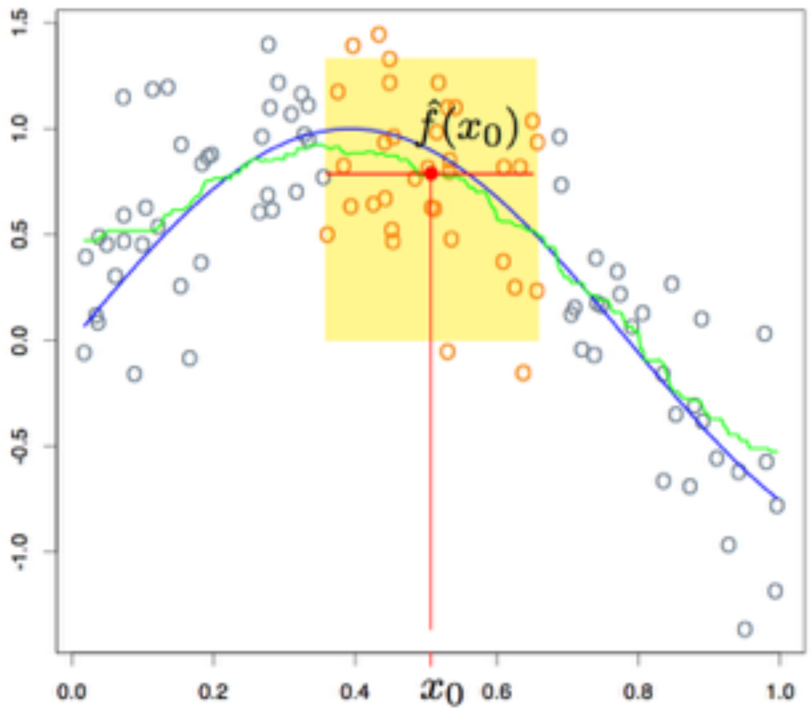


$\mathcal{N}_k(x_0) = k$-nearest neighbors of x_0

$$\widehat{f}(x_0) = \sum_{x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} \, y_i$$

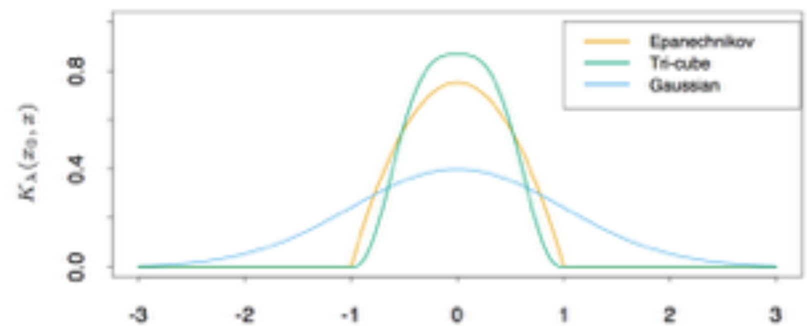
Nearest neighbor regression

$$\{(x_i, y_i)\}_{i=1}^n$$



Why are far-away neighbors weighted same as close neighbors!

Kernel smoothing: $K(x, y)$



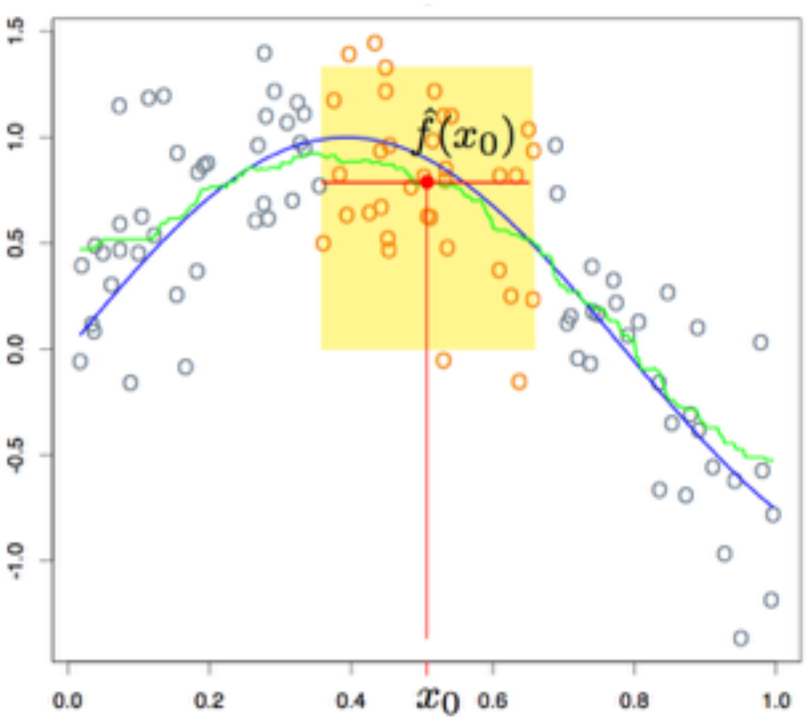
$\mathcal{N}_k(x_0) = k$-nearest neighbors of x_0

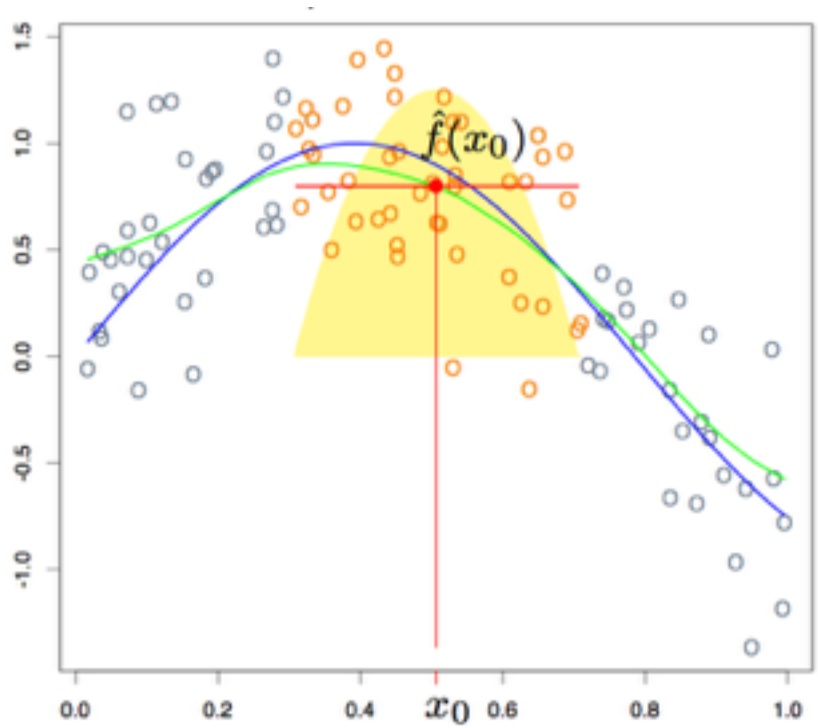
$$\widehat{f}(x_0) = \sum_{x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} y_i$$

$$\widehat{f}(x_0) = \frac{\sum_{i=1}^n K(x_0, x_i) y_i}{\sum_{i=1}^n K(x_0, x_i)}$$

Nearest neighbor regression

$$\{(x_i, y_i)\}_{i=1}^n$$





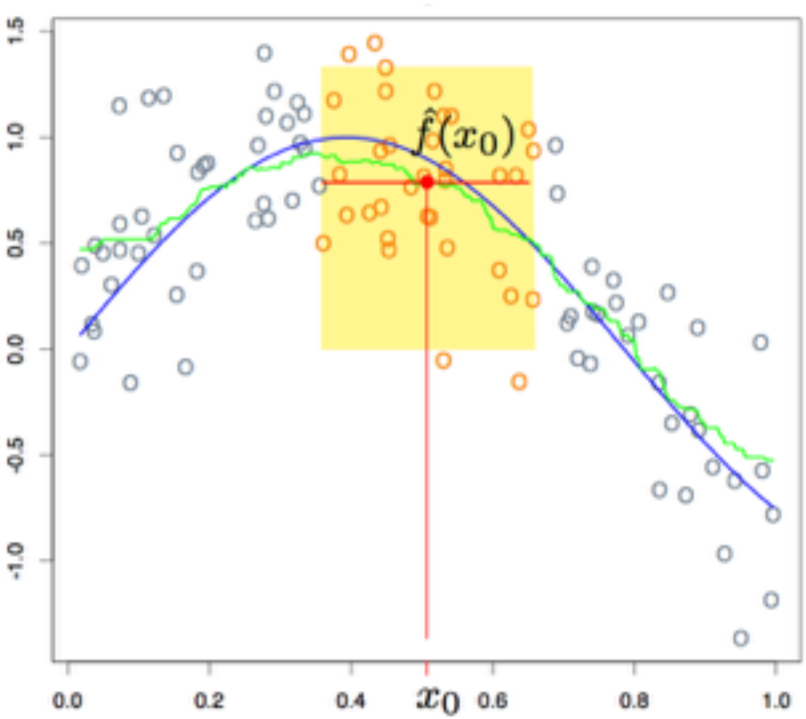
$\mathcal{N}_k(x_0) = k$-nearest neighbors of x_0

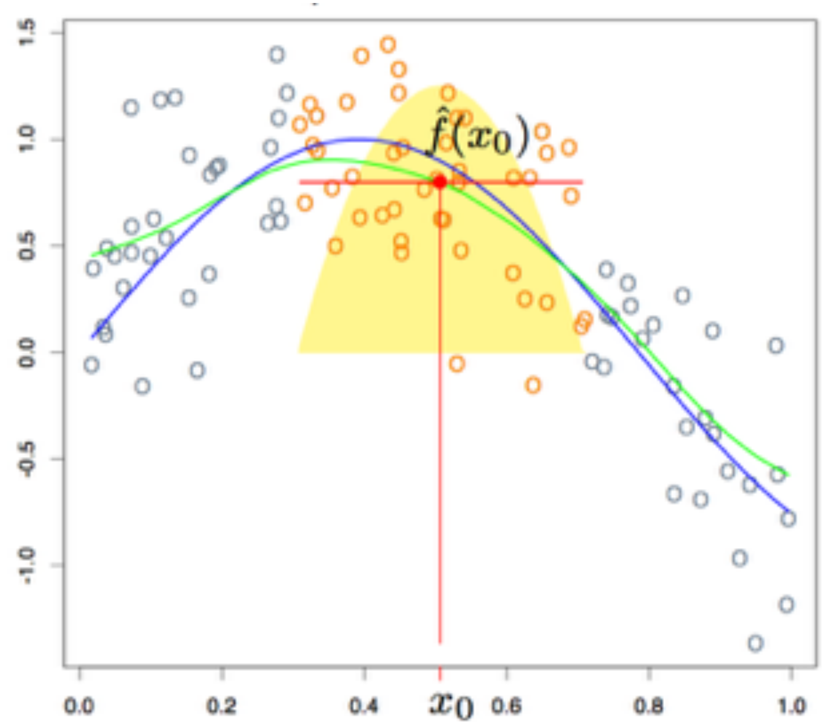
$$\widehat{f}(x_0) = \sum_{x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} \, y_i$$

$$\widehat{f}(x_0) = \frac{\sum_{i=1}^n K(x_0, x_i) y_i}{\sum_{i=1}^n K(x_0, x_i)}$$

Nearest neighbor regression

$\{(x_i, y_i)\}_{i=1}^n$





$\mathcal{N}_k(x_0) = k\text{-nearest neighbors of } x_0$

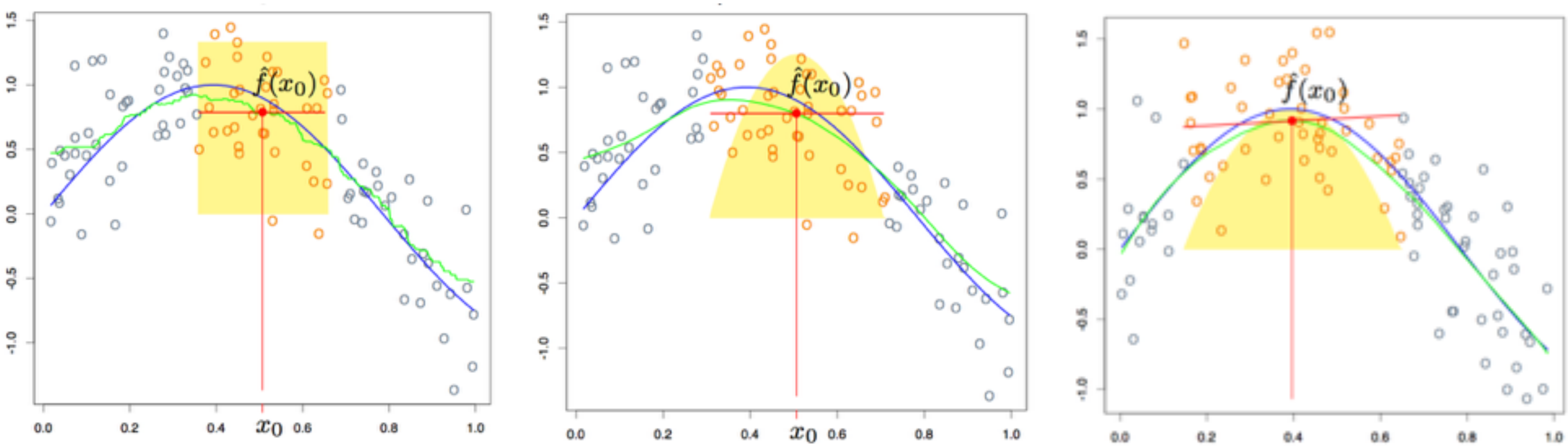
$$\widehat{f}(x_0) = \sum_{x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} y_i$$

Why just average them?

$$\widehat{f}(x_0) = \frac{\sum_{i=1}^n K(x_0, x_i) y_i}{\sum_{i=1}^n K(x_0, x_i)}$$

Nearest neighbor regression

$$\{(x_i, y_i)\}_{i=1}^n$$



$\mathcal{N}_k(x_0) = k\text{-nearest neighbors of } x_0$

$$\hat{f}(x_0) = \sum_{x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} y_i$$

$$\hat{f}(x_0) = \frac{\sum_{i=1}^n K(x_0, x_i) y_i}{\sum_{i=1}^n K(x_0, x_i)}$$

$$\hat{f}(x_0) = b(x_0) + w(x_0)^T x_0$$

$$w(x_0), b(x_0) = \arg\min_{w,b} \sum_{i=1}^n K(x_0, x_i)(y_i - (b + w^T x_i))^2$$

Local Linear Regression

Nearest Neighbor Overview

- Very simple to explain and implement
- No training! But finding nearest neighbors in large dataset at test can be computationally demanding (kD-trees help)

Nearest Neighbor Overview

- Very simple to explain and implement
- No training! But finding nearest neighbors in large dataset at test can be computationally demanding (kD-trees help)
- You can use other forms of distance (not just Euclidean)
- Smoothing with Kernels and local linear regression can improve performance (at the cost of higher variance)

Nearest Neighbor Overview

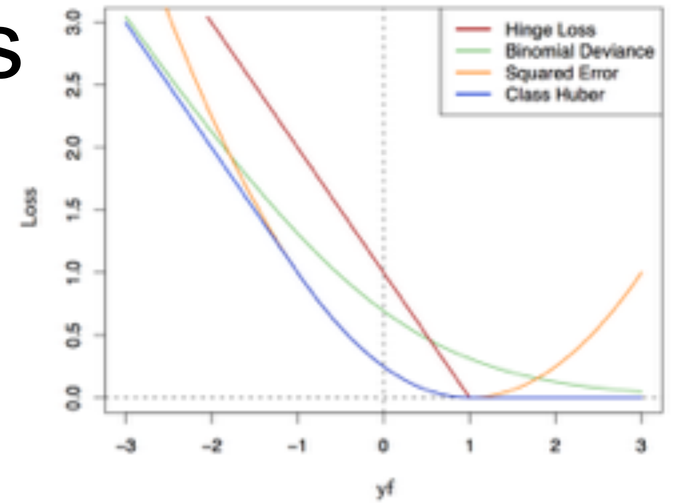
- Very simple to explain and implement
- No training! But finding nearest neighbors in large dataset at test can be computationally demanding (kD-trees help)
- You can use other forms of distance (not just Euclidean)
- Smoothing with Kernels and local linear regression can improve performance (at the cost of higher variance)
- With a lot of data, “local methods” have strong, simple theoretical guarantees. With not a lot of data, neighborhoods aren’t “local” and methods suffer.

Kernels

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 26, 2017

Machine Learning Problems



- Have a bunch of iid data of the form:

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Learning a model's parameters:

$$\sum_{i=1}^n \ell_i(w)$$

Each $\ell_i(w)$ is convex.

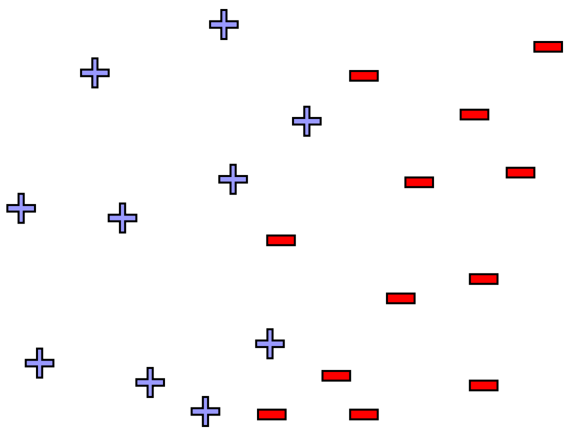
Hinge Loss: $\ell_i(w) = \max\{0, 1 - y_i x_i^T w\}$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

All in terms of inner products! Even nearest neighbor can use inner products!

What if the data is not linearly separable?



Use features of features of features of features….

$$\phi(x) : \mathbb{R}^d \rightarrow \mathbb{R}^p$$

Feature space can get really large really quickly!

Dot-product of polynomials

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \text{polynomials of degree}$ exactly d

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

Dot-product of polynomials

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \text{polynomials of degree}$ exactly d

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

$$d = 2 : \phi(u) = \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_1 u_2 \\ u_2 u_1 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1^2 v_1^2 + u_2^2 v_2^2 + 2 u_1 u_2 v_1 v_2$$

Dot-product of polynomials

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) =$ polynomials of degree exactly d

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

$$d = 2 : \phi(u) = \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_1 u_2 \\ u_2 u_1 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1^2 v_1^2 + u_2^2 v_2^2 + 2 u_1 u_2 v_1 v_2$$

General d :

Dimension of $\phi(u)$ is roughly p^d if $u \in \mathbb{R}^p$

Observation

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda ||w||_w^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$

Why?

Observation

$$\arg\min_{\alpha} ||\mathbf{K}\alpha - \mathbf{y}||_2^2 + \lambda\alpha^T\mathbf{K}\alpha$$

Common kernels

- Polynomials of degree exactly d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian (squared exponential) kernel

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

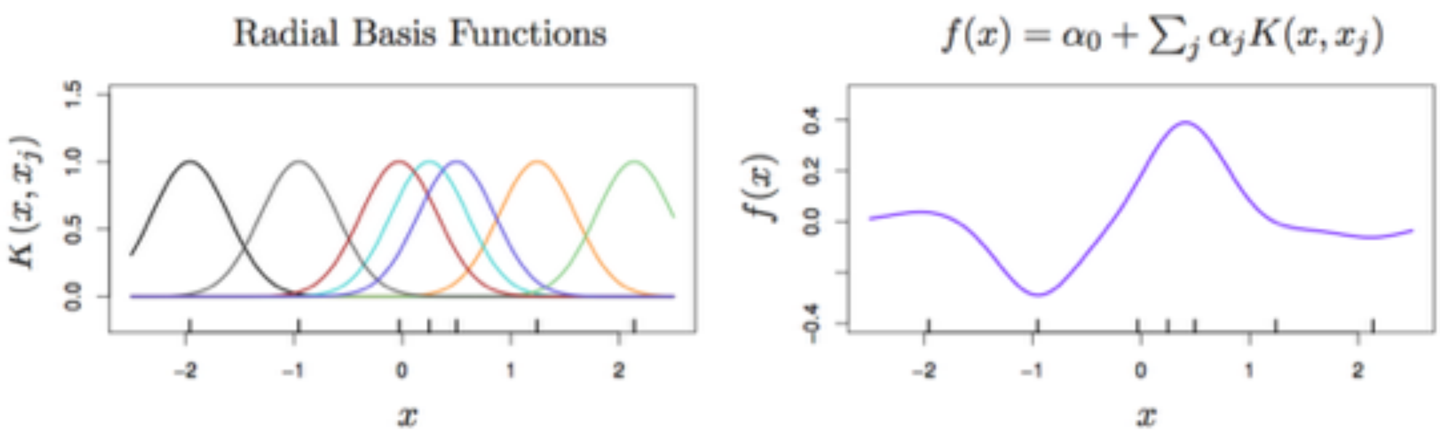
Mercer's Theorem

- When do we have a valid Kernel K(x,x')?
- Definition 1: when it is an inner product

- Mercer's Theorem:
  - K(x,x') is a valid kernel if and only if K is a positive semi-definite.
  - PSD in the following sense:

RBF Kernel $K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$

- Note that this is like weighting "bumps" on each point like kernel smoothing but now we **learn** the weights

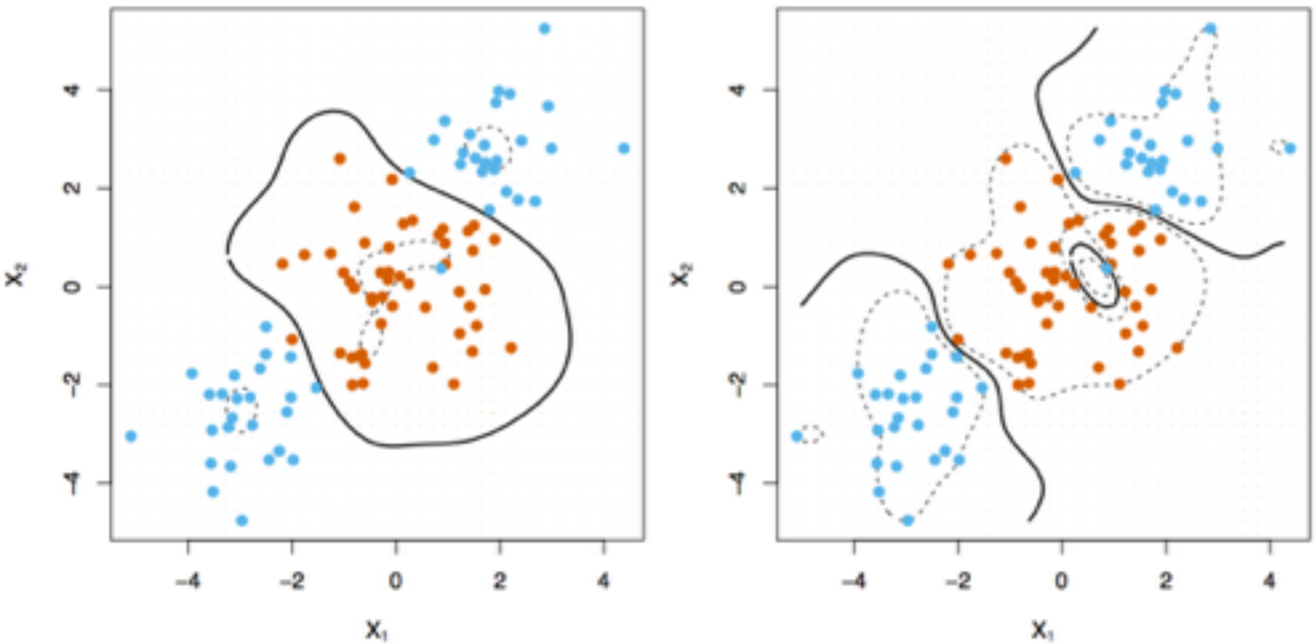


- Is there an inner product representation of K(x,y)?

Classification

$$\widehat{w} = \sum_{i=1}^n \max\{0, 1 - y_i(b + x_i^T w)\} + \lambda ||w||_2^2$$

$$\min_{\alpha,b} \sum_{i=1}^n \max\{0, 1 - y_i(b + \sum_{j=1}^n \alpha_j \langle x_i, x_j \rangle)\} + \lambda \sum_{i,j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$



RBF kernel Secretly random features

$$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta) + \cos(\alpha-\beta)$$

$$b \sim \text{uniform}(0, \pi) \qquad w \sim \mathcal{N}(0, 2\gamma)$$

$$\phi(x) = \sqrt{2}\cos(w^T x + b)$$

$$\mathbb{E}_{w,b}[\phi(x)^T \phi(y)] =$$

RBF kernel Secretly random features

$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta) + \cos(\alpha-\beta)$

$b \sim \text{uniform}(0, \pi)$ $\qquad$ $w \sim \mathcal{N}(0, 2\gamma)$

$$\phi(x) = \sqrt{2}\cos(w^T x + b)$$

$$\mathbb{E}_{w,b}[\phi(x)^T \phi(y)] = \quad e^{-\gamma||x-y||_2^2}$$

[Rahimi, Recht 2007]

Hint: use Euler's formula $e^{jz} = \cos(z) + j\sin(z)$

Wait, infinite dimensions?

- Isn't everything separable there? How are we not overfitting?
- Regularization! Fat shattering (R/margin)^2
- What about sparsity?

String Kernels

Example from Efron and Hastie, 2016

Amino acid sequences of different lengths:

x1

IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDYLQEFLGVMNTEWI

x2

PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAERLQENLQAYRTFHVLLA
RLLEDQQVHFTPTEGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK
LWGLKVLQELSQWTVRSIHDLRFISSHQTGIP

All subsequences of length 3 (of possible 20 amino acids) $20^3 = 8{,}000$

$$h^3_{\text{LQE}}(x_1) = 1 \text{ and } h^3_{\text{LQE}}(x_2) = 2.$$

Least squares, tradeoffs