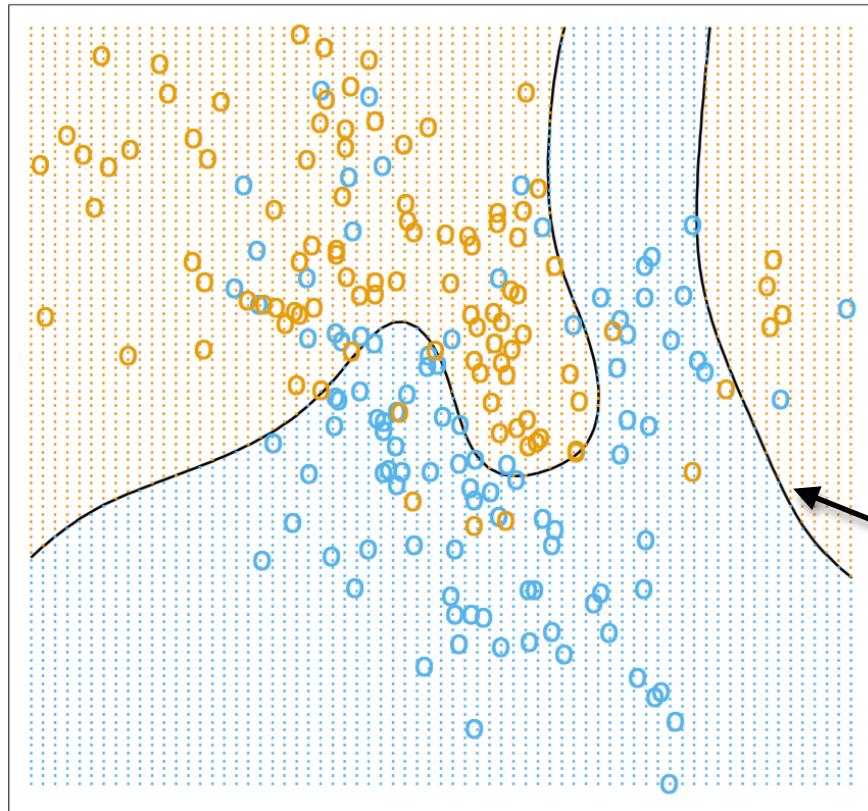


# Nearest Neighbor

Machine Learning – CSE446  
Kevin Jamieson  
University of Washington

May 6, 2019

# Some data, Bayes Classifier



Training data:

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

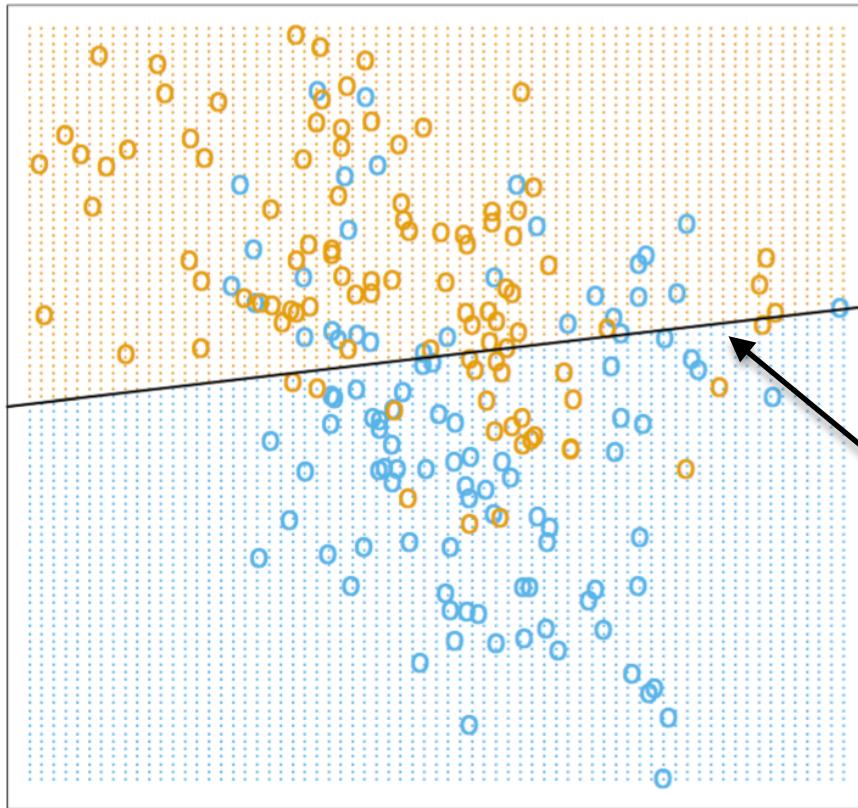
Optimal “Bayes” classifier:

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

- [Orange dotted pattern] Predicted label: +1
- [Blue dotted pattern] Predicted label: -1

Figures stolen from Hastie et al

# Linear Decision Boundary



Training data:

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

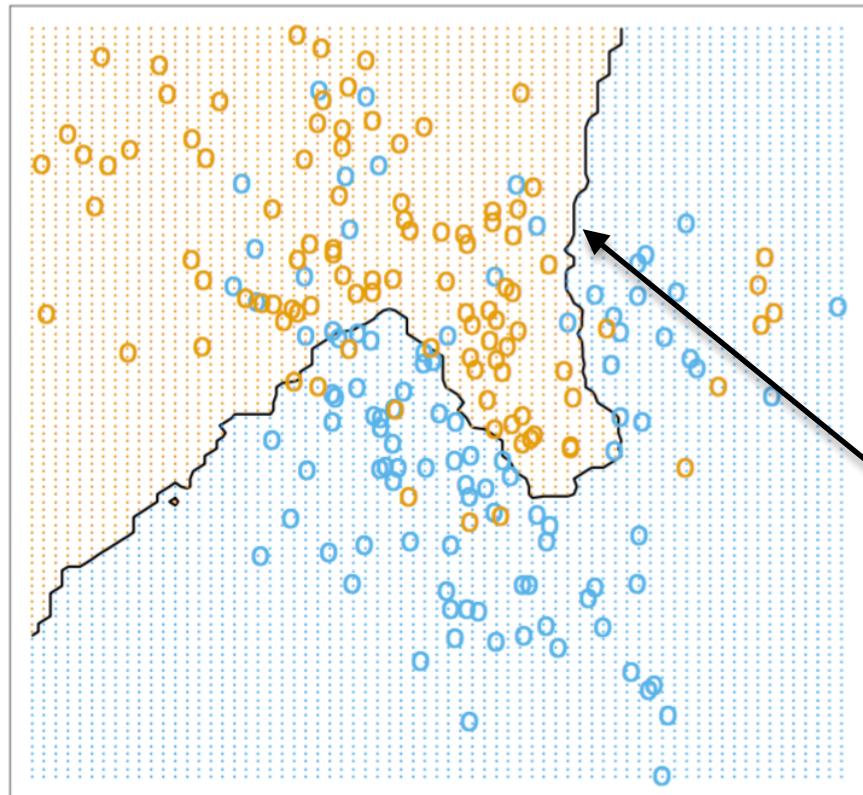
Learned:

Linear Decision boundary

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

- [Orange square] Predicted label: +1
- [Blue square] Predicted label: -1

# 15 Nearest Neighbor Boundary



Training data:

○ True label: +1

○ True label: -1

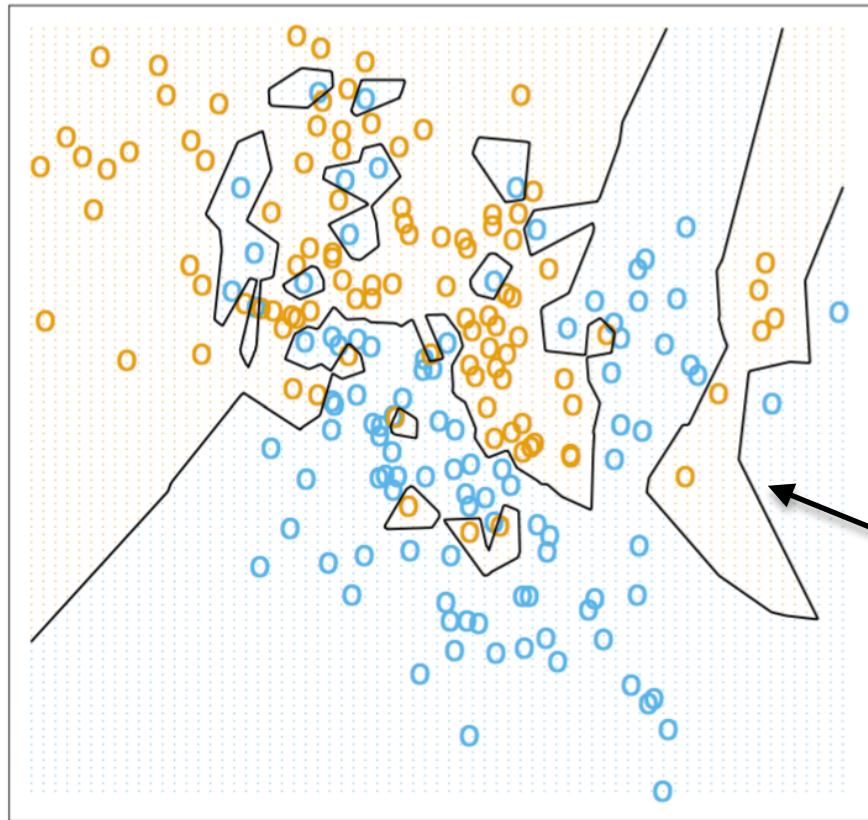
Learned:

**15** nearest neighbor decision  
boundary (majority vote)

□ Predicted label: +1

□ Predicted label: -1

# 1 Nearest Neighbor Boundary



Training data:

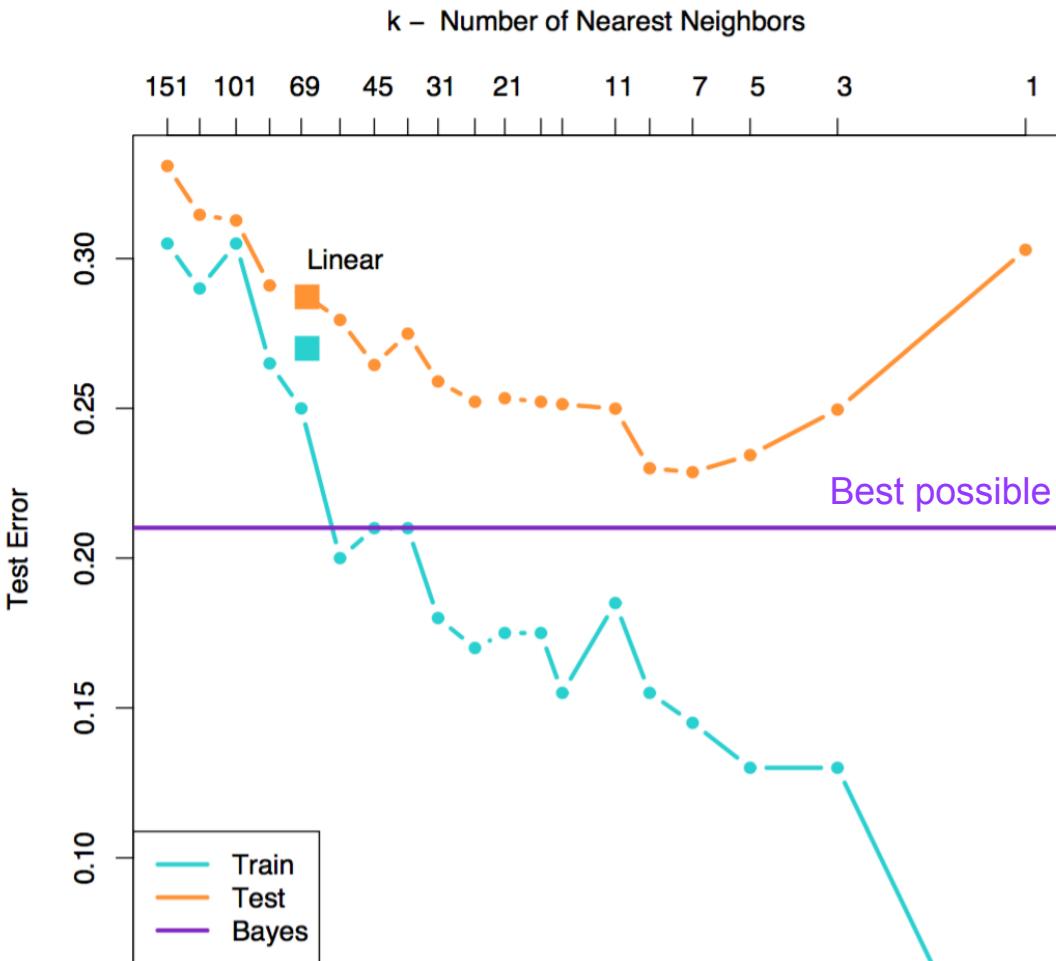
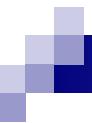
- True label: +1
- True label: -1

Learned:

1 nearest neighbor decision  
boundary (majority vote)

- [Orange square] Predicted label: +1
- [Blue square] Predicted label: -1

# k-Nearest Neighbor Error



Bias-Variance tradeoff

As  $k \rightarrow \infty$ ?

Bias:

Variance:

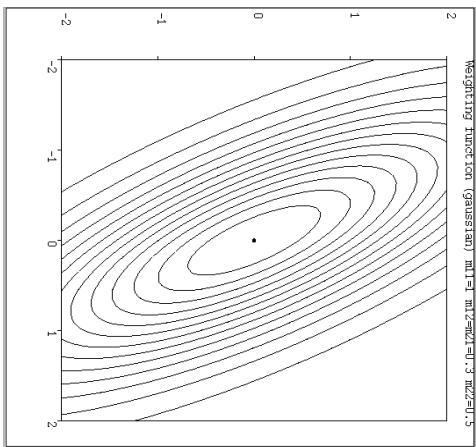
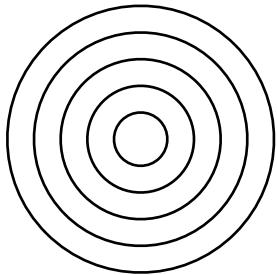
As  $k \rightarrow 1$ ?

Bias:

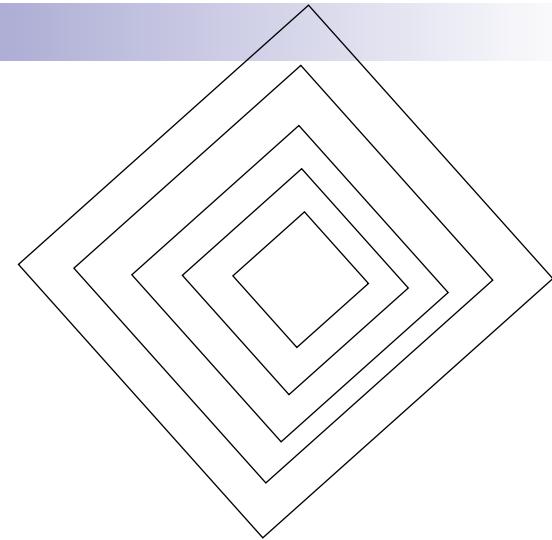
Variance:

# Notable distance metrics (and their level sets)

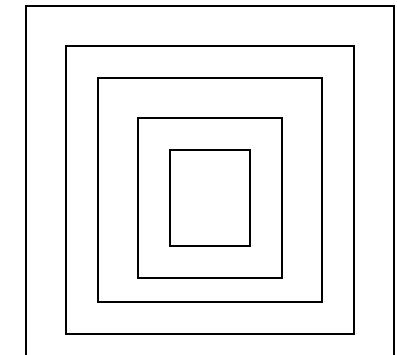
$L_2$  norm



Mahalanobis



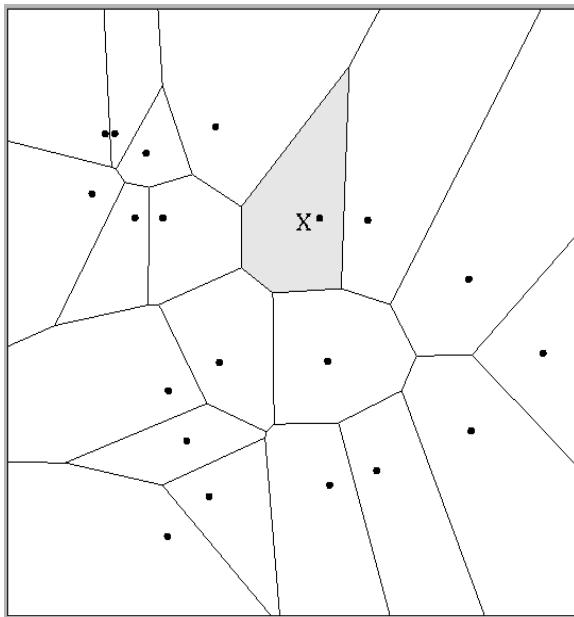
$L_1$  norm (taxi-cab)



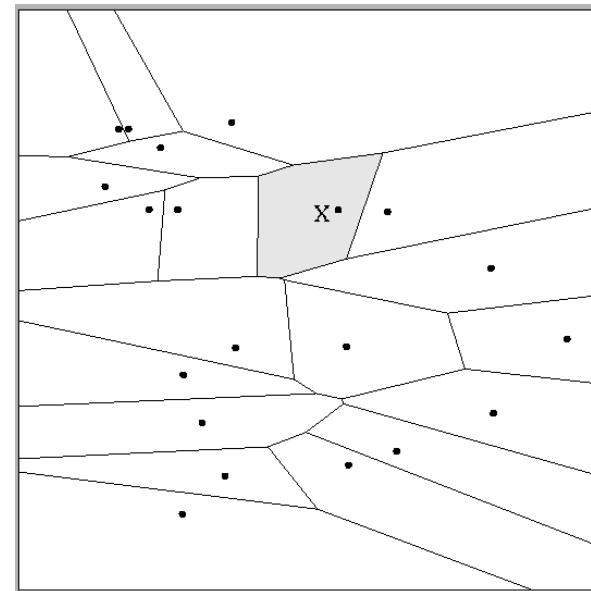
$L$ -infinity (max) norm

# 1 nearest neighbor

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



$$Dist(\mathbf{x}^i, \mathbf{x}^j) = (x_1^i - x_1^j)^2 + (x_2^i - x_2^j)^2$$



$$Dist(\mathbf{x}^i, \mathbf{x}^j) = (x_1^i - x_1^j)^2 + (3x_2^i - 3x_2^j)^2$$

The relative scalings in the distance metric affect region shapes

# 1 nearest neighbor guarantee

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}$$

As  $n \rightarrow \infty$  assume the  $x_i$ 's become *dense* in  $\mathbb{R}^d$  and  $\mathbb{P}(Y = 1|X = x)$  is *smooth*

As  $x_a \rightarrow x_b$  we have  $\mathbb{P}(Y_a = 1|X_a = x_a) \rightarrow \mathbb{P}(Y_b = 1|X_b = x_b)$

# 1 nearest neighbor guarantee

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}$$

As  $n \rightarrow \infty$  assume the  $x_i$ 's become *dense* in  $\mathbb{R}^d$  and  $\mathbb{P}(Y = 1|X = x)$  is *smooth*

As  $x_a \rightarrow x_b$  we have  $\mathbb{P}(Y_a = 1|X_a = x_a) \rightarrow \mathbb{P}(Y_b = 1|X_b = x_b)$

If  $p_* = \max_{y=0,1} \mathbb{P}(Y_b = y|X_b = x_b)$  then the Bayes Error =  $1 - p_*$

1-nearest neighbor error =

$$\lim_{n \rightarrow \infty} \mathbb{P}(\widehat{f}_{1NN}(x_a) \neq Y_a | X_a = x_a) = \mathbb{P}(Y_b \neq Y_a | X_a = x_b, X_b = x_b)$$

# 1 nearest neighbor guarantee

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}$$

As  $n \rightarrow \infty$  assume the  $x_i$ 's become *dense* in  $\mathbb{R}^d$  and  $\mathbb{P}(Y = 1|X = x)$  is *smooth*

As  $x_a \rightarrow x_b$  we have  $\mathbb{P}(Y_a = 1|X_a = x_a) \rightarrow \mathbb{P}(Y_b = 1|X_b = x_b)$

If  $p_* = \max_{y=0,1} \mathbb{P}(Y_b = y|X_b = x_b)$  then the Bayes Error =  $1 - p_*$

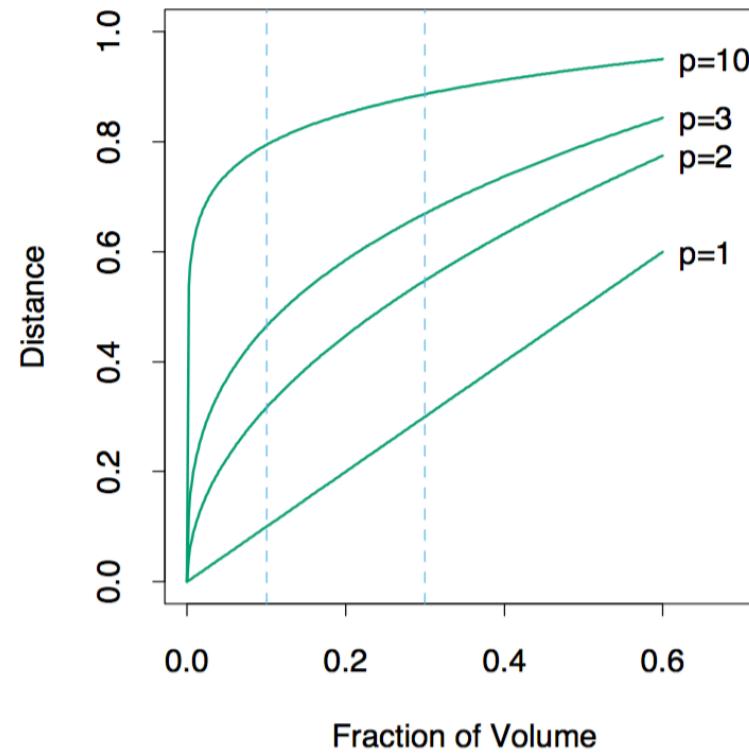
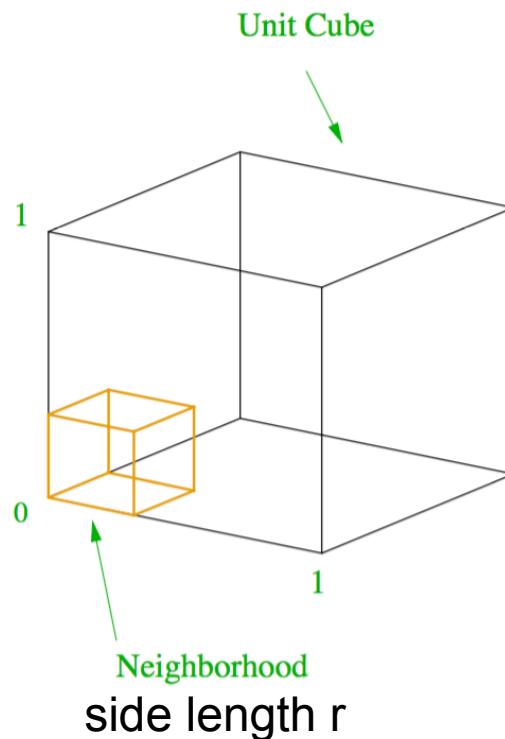
1-nearest neighbor error =

$$\begin{aligned} & \lim_{n \rightarrow \infty} \mathbb{P}(\widehat{f}_{1NN}(x_a) \neq Y_a | X_a = x_a) = \mathbb{P}(Y_b \neq Y_a | X_a = x_b, X_b = x_b) \\ &= \mathbb{P}(Y_b = 1 | X_b = x_b) \mathbb{P}(Y_a = 0 | X_a = x_b) + \mathbb{P}(Y_b = 0 | X_b = x_b) \mathbb{P}(Y_a = 1 | X_a = x_b) \\ &= 2p_*(1 - p_*) \leq 2(1 - p_*) \end{aligned}$$

As  $n \rightarrow \infty$ , then 1-NN rule error is at most twice the Bayes error!

[Cover, Hart, 1967]

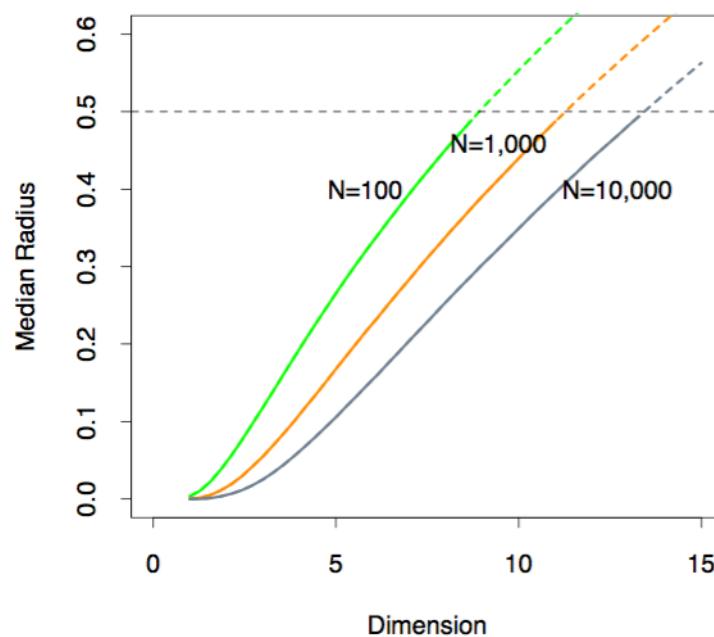
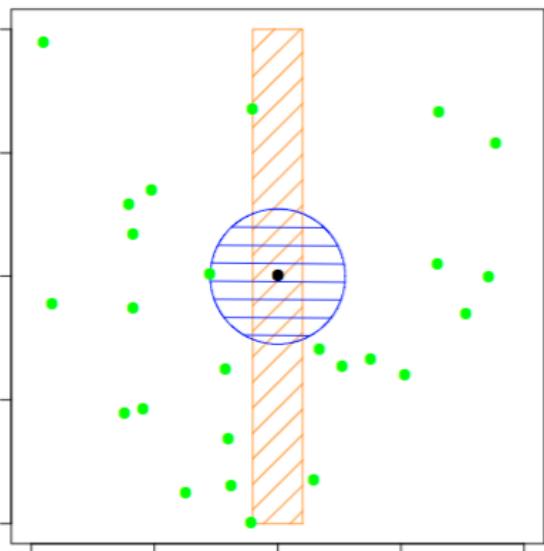
# Curse of dimensionality Ex. 1



$X$  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  $[-.5, .5]^p$ .

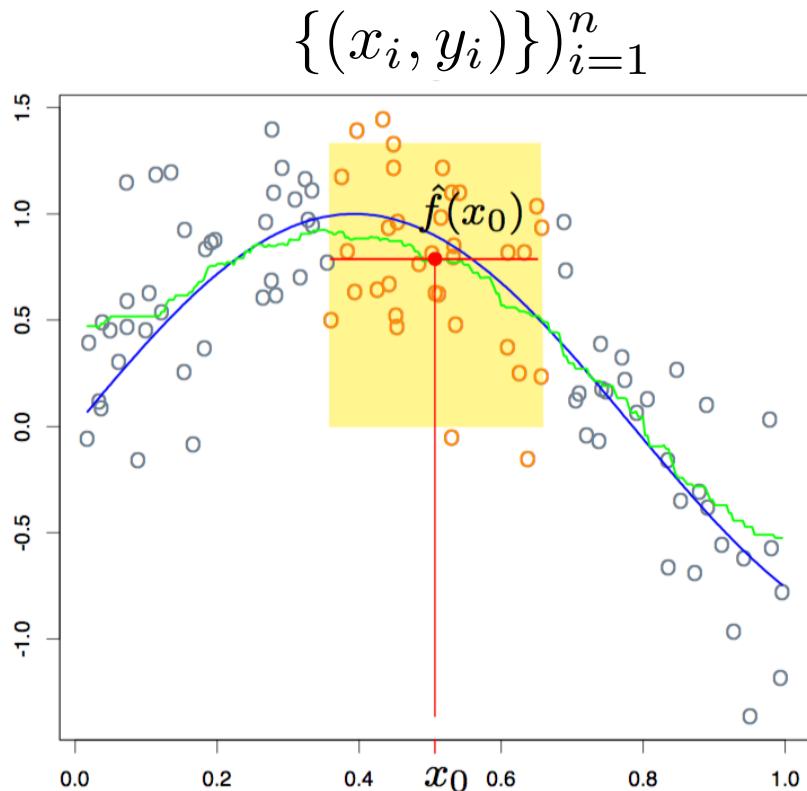


What is the median distance from a point at origin to its 1NN?

# Takeaways

- Very simple to explain and implement. No training!
- You can use other forms of distance (not just Euclidean)
- With a lot of data, “local methods” have strong, simple theoretical guarantees.
- Without a lot of data, neighborhoods aren’t “local” and methods suffer.

# Nearest neighbor regression

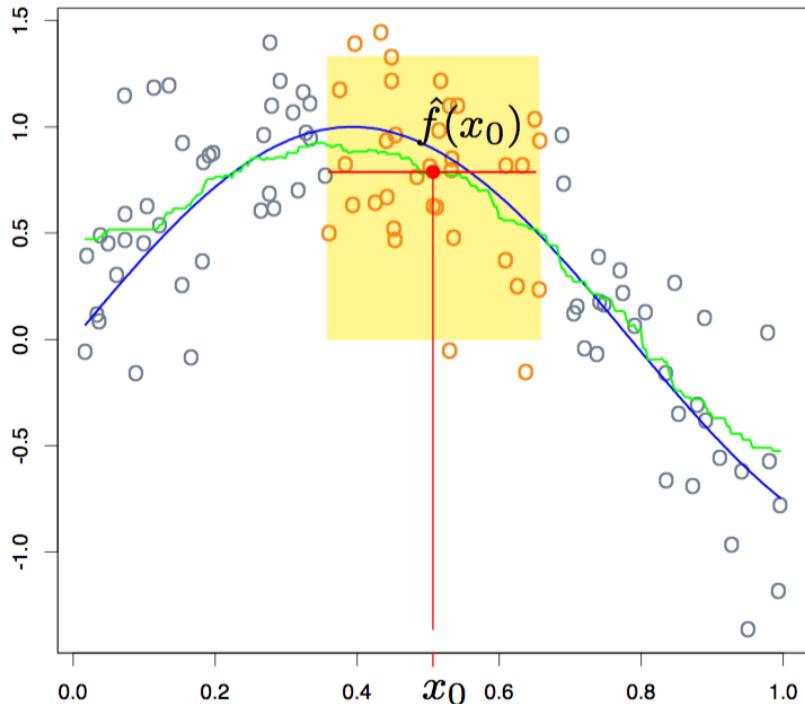


$\mathcal{N}_k(x_0) = k$ -nearest neighbors of  $x_0$

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

# Nearest neighbor regression

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

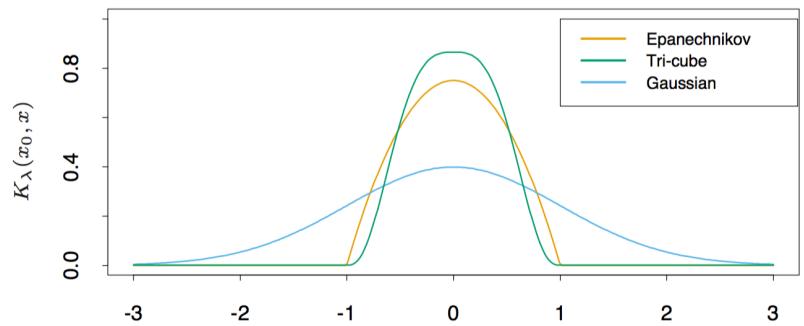


$\mathcal{N}_k(x_0) = k$ -nearest neighbors of  $x_0$

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

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

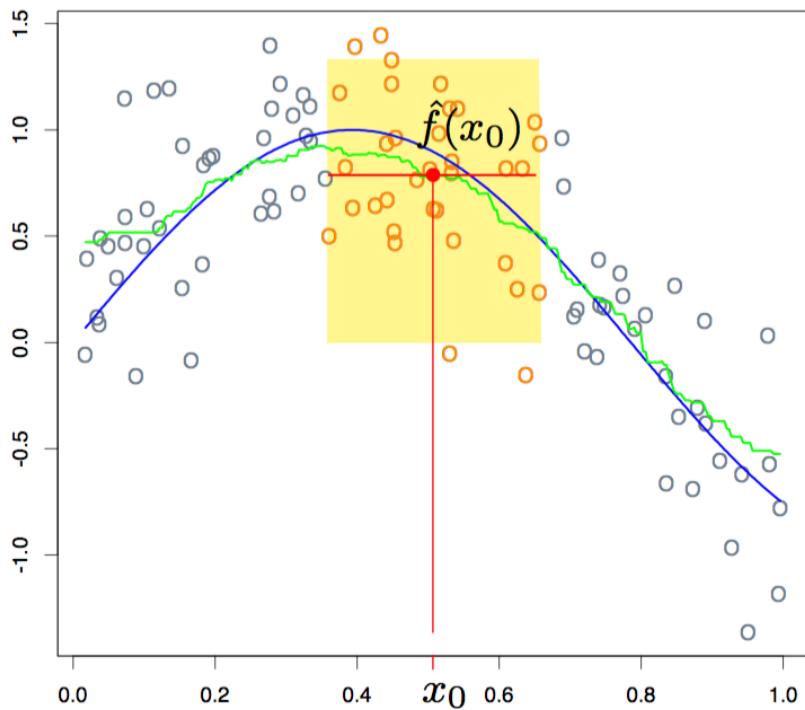
Kernel smoothing:  $K(x, y)$



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

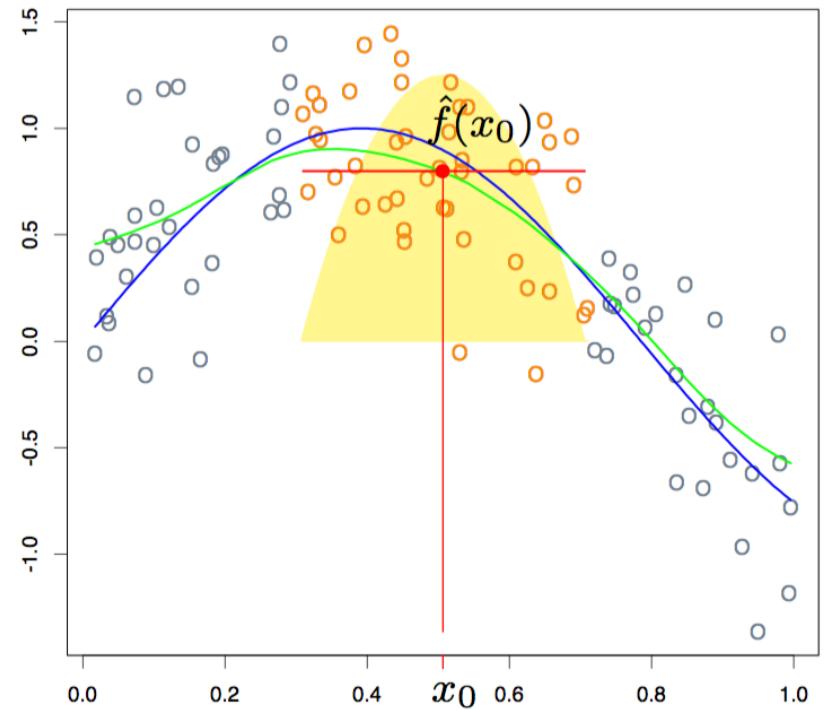
# Nearest neighbor regression

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



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

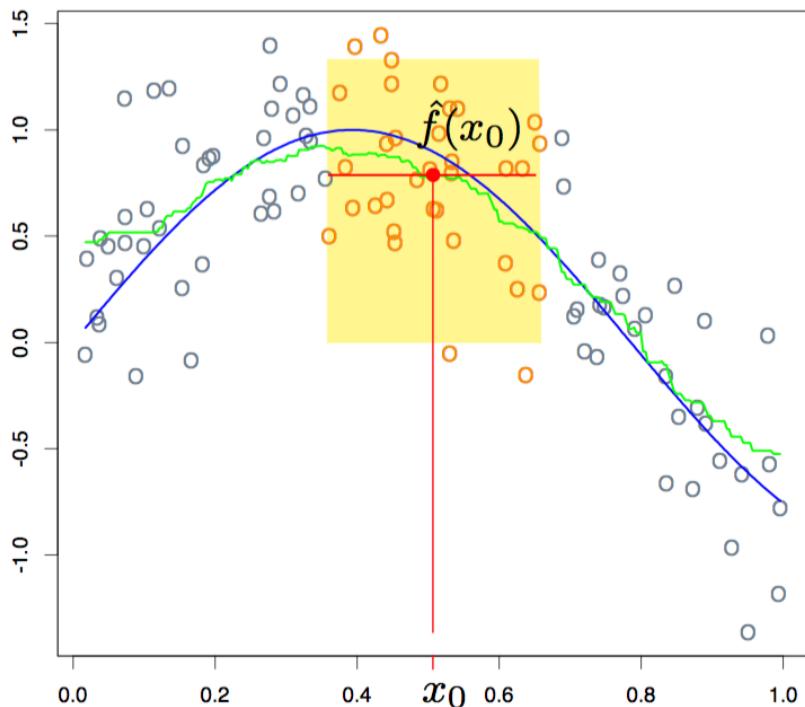
$$\hat{f}(x_0) = \sum_{x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} y_i$$



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

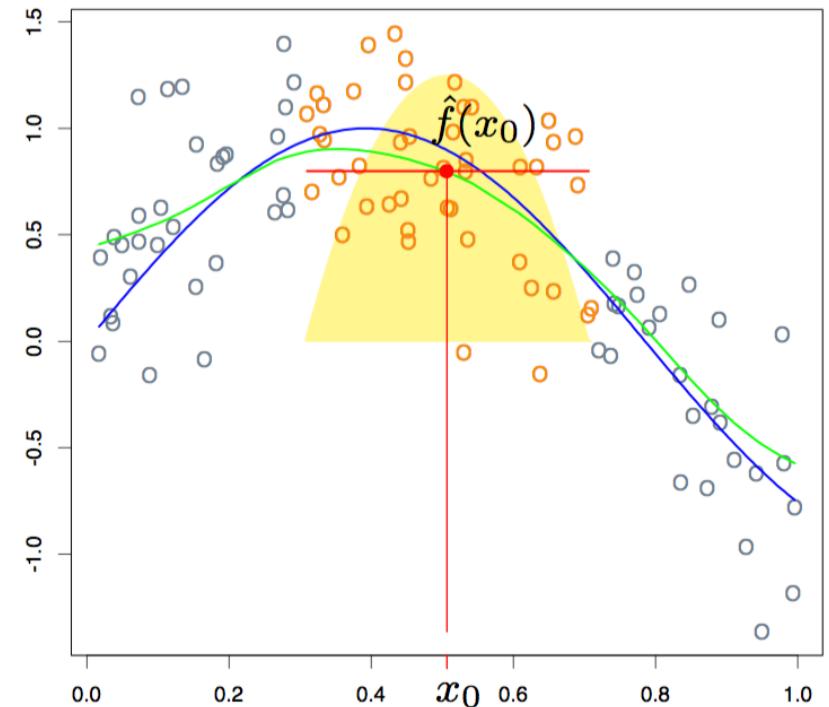
# Nearest neighbor regression

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



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

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

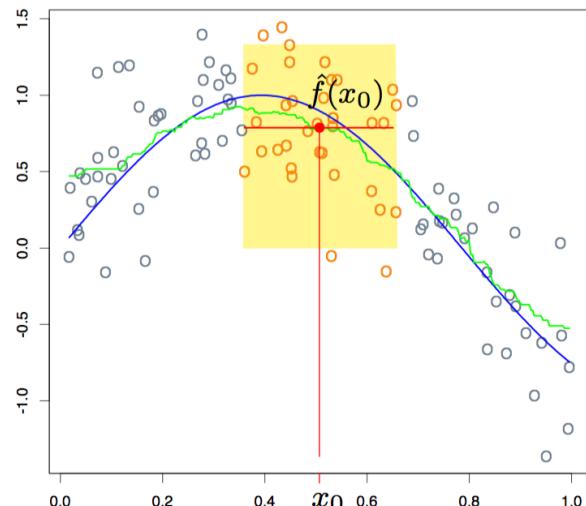


Why just average them?

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

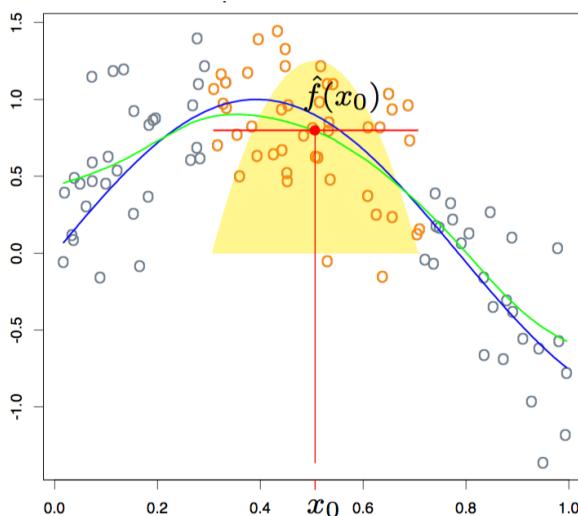
# Nearest neighbor regression

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

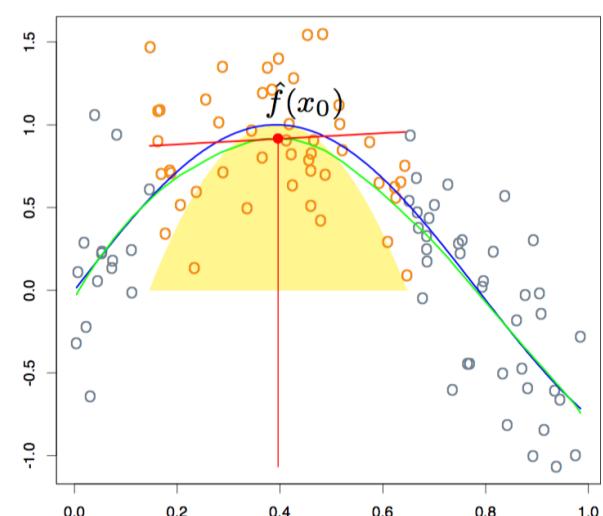


$\mathcal{N}_k(x_0)$  =  $k$ -nearest neighbors of  $x_0$

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



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



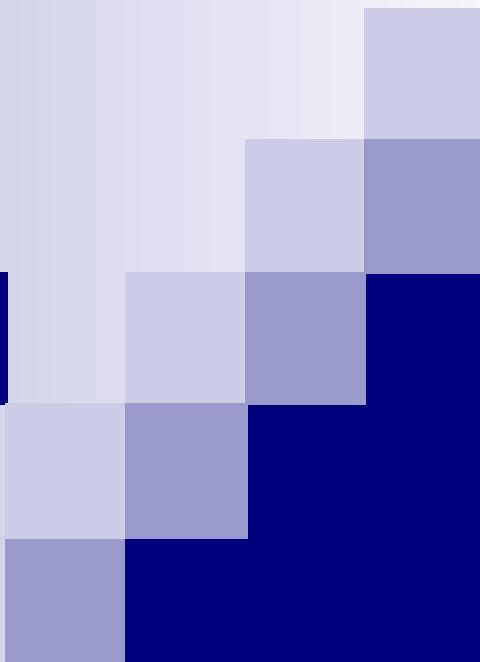
$$\hat{f}(x_0) = b(x_0) + w(x_0)^T x_0$$

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

**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)
- 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.
- Without a lot of data, neighborhoods aren’t “local” and methods suffer.



# Kernels

Machine Learning – CSE446  
Kevin Jamieson  
University of Washington

May 6, 2019

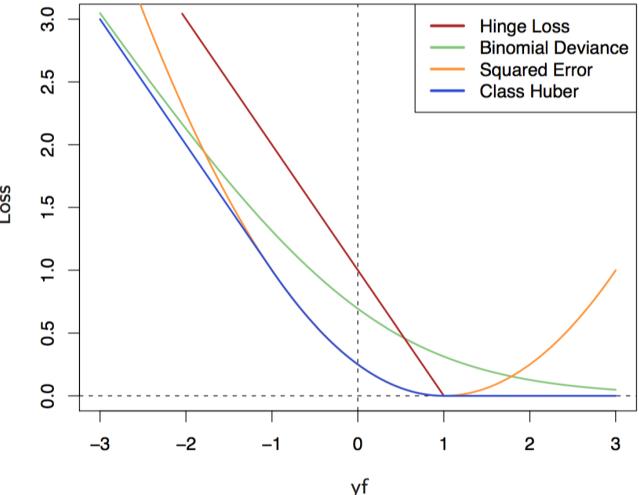
# Machine Learning Problems



- Have a bunch of iid data of the form:

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

- Learning a model's parameters:  
Each  $\ell_i(w)$  is convex.



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

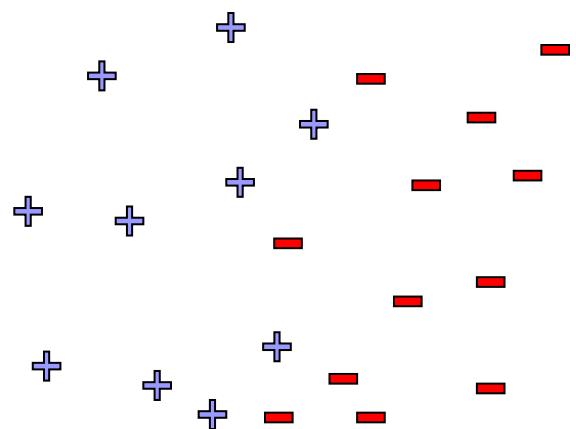
Hinge Loss:  $\ell_i(w) = \max\{0, 1 - y_i x_i^T w\}$

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

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

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

# Dot-product of polynomials

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

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

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

# Dot-product of polynomials

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

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

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

General  $d$  :

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

# Kernel Trick

$$\hat{w} = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda \|w\|_2^2$$

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

# Kernel Trick

$$\hat{w} = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda \|w\|_2^2$$

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

$$\hat{\alpha} = \arg \min_{\alpha} \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$

# Kernel Trick

$$\hat{w} = \arg \min_w \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda \|w\|_2^2$$

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

$$\begin{aligned}\hat{\alpha} &= \arg \min_{\alpha} \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle \\ &= \arg \min_{\alpha} \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j) \\ &= \arg \min_{\alpha} \|\mathbf{y} - \mathbf{K}\alpha\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha\end{aligned}$$

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

# Why regularization?

Typically,  $\mathbf{K} \succ 0$ . What if  $\lambda = 0$ ?

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

# Why regularization?

Typically,  $\mathbf{K} \succ 0$ . What if  $\lambda = 0$ ?

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

Unregularized kernel least squares can (over) fit **any data!**

$$\hat{\alpha} = \mathbf{K}^{-1} \mathbf{y}$$

# 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  $K(x, x')$ ?
- Sufficient:

$K(x, x')$  is a valid kernel if there exists  $\phi(x)$  such that  $K(x, x') = \phi(x)^T \phi(x')$

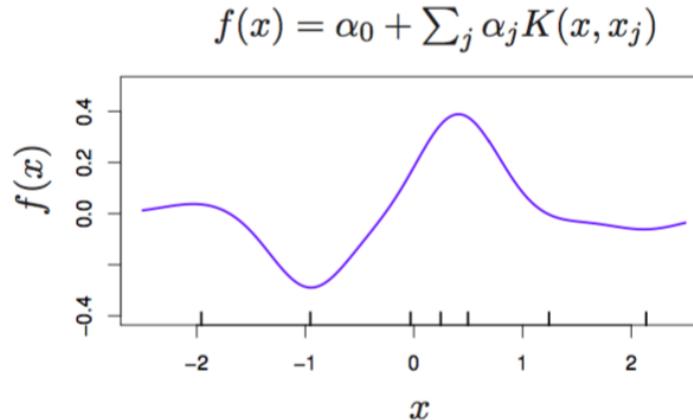
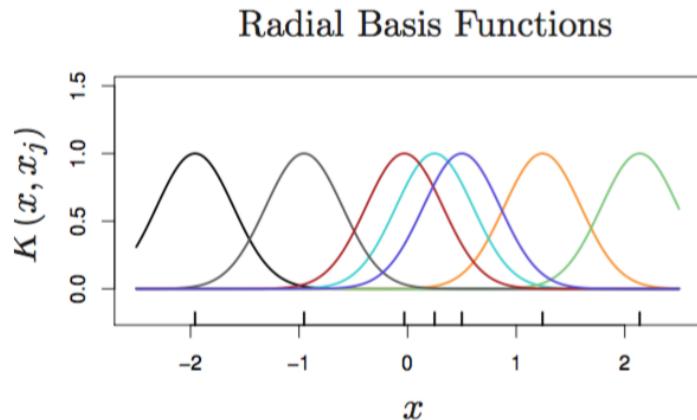
- Mercer's Theorem:

$K(x, x')$  is a valid kernel if and only if  $\mathbf{K}$  is symmetric and positive semi-definite for any pointset  $(x_1, \dots, x_n)$  where  $\mathbf{K}_{i,j} = K(x_i, x_j)$ .

# RBF Kernel

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

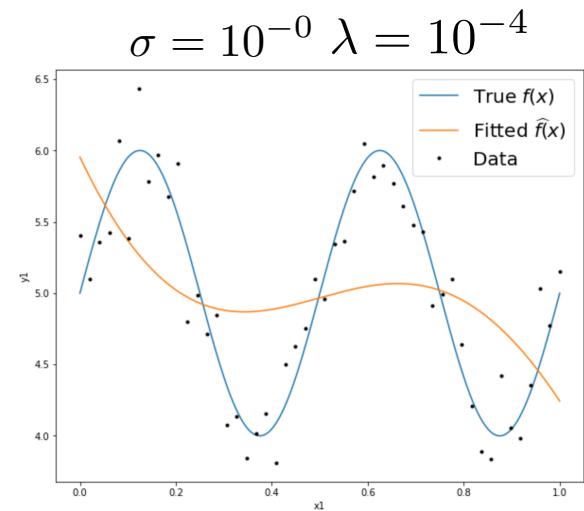
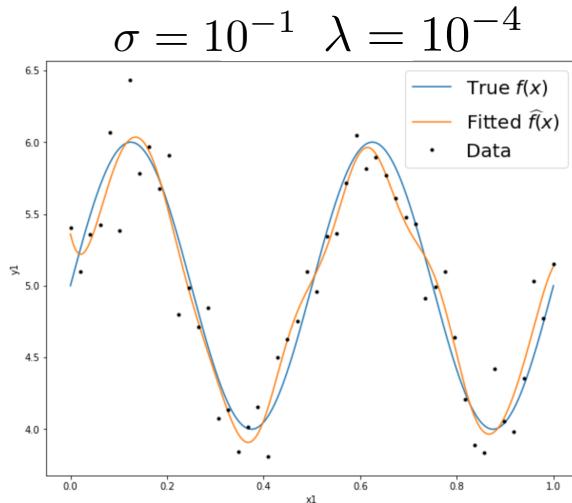
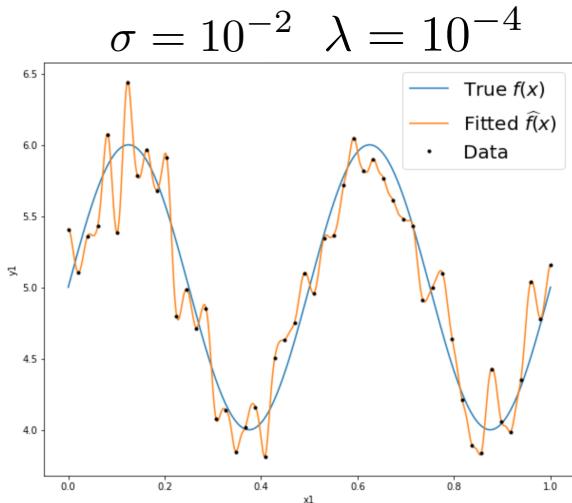
- Note that this is like weighting “bumps” on each point like kernel smoothing but now we **learn** the weights



# RBF Kernel

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

The bandwidth sigma has an enormous effect on fit:

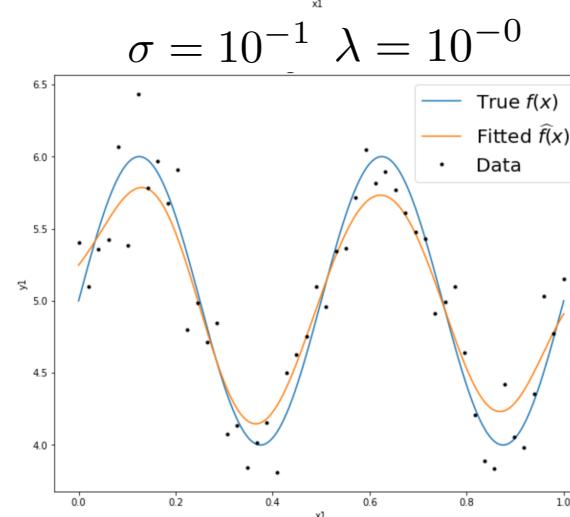
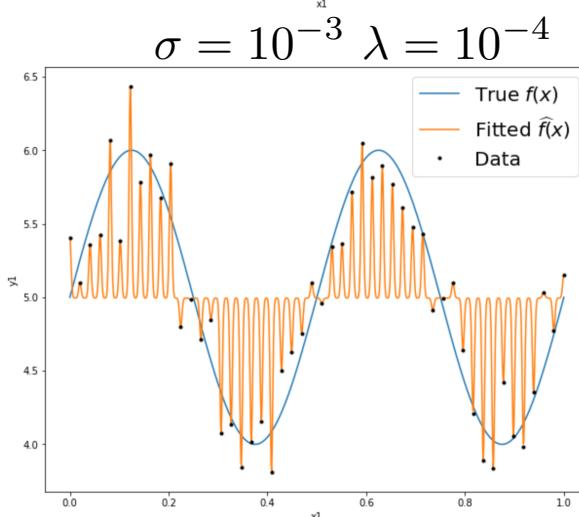
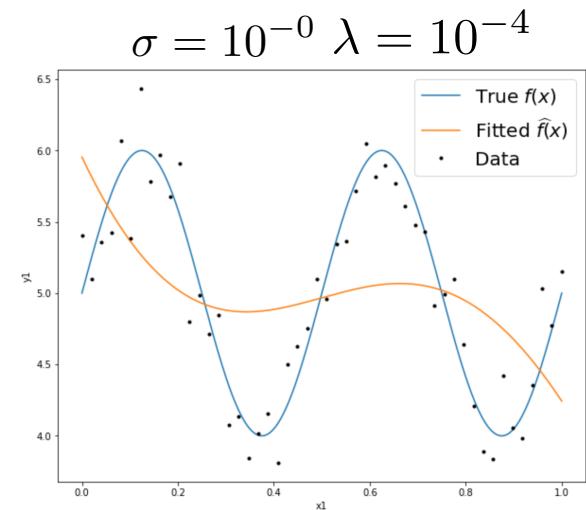
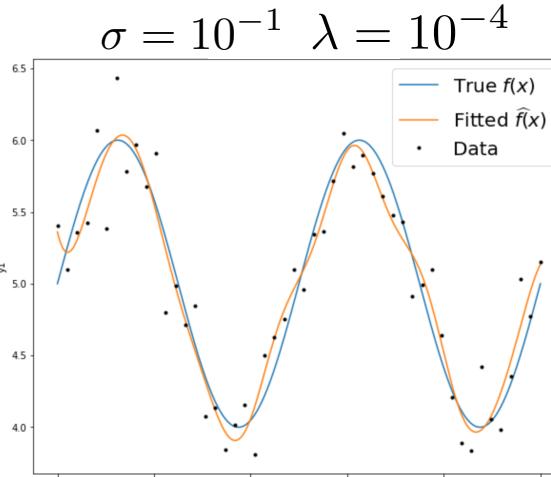
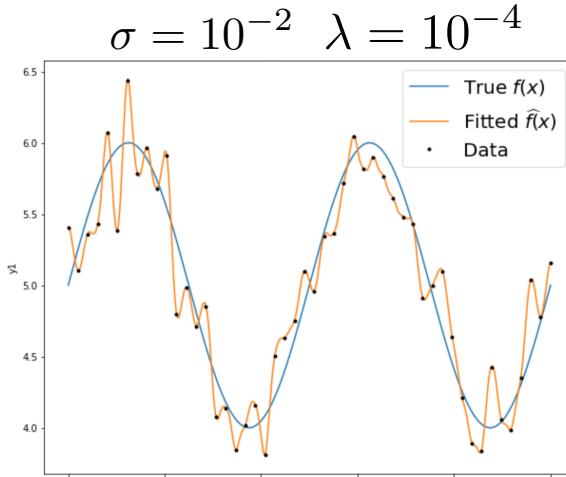


$$\hat{f}(x) = \sum_{i=1}^n \hat{\alpha}_i K(x_i, x)$$

# RBF Kernel

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

The bandwidth sigma has an enormous effect on fit:



$$\hat{f}(x) = \sum_{i=1}^n \hat{\alpha}_i K(x_i, x)$$

# RBF Kernel

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

Basis representation in 1d?

$$[\phi(x)]_i = \frac{1}{\sqrt{i!}} e^{-\frac{x^2}{2}} x^i \quad \text{for } i = 0, 1, \dots$$

$$\begin{aligned}\phi(x)^T \phi(x') &= \sum_{i=0}^{\infty} \left( \frac{1}{\sqrt{i!}} e^{-\frac{x^2}{2}} x^i \right) \left( \frac{1}{\sqrt{i!}} e^{-\frac{(x')^2}{2}} (x')^i \right) \\ &= e^{-\frac{x^2 + (x')^2}{2}} \sum_{i=0}^{\infty} \frac{1}{i!} (xx')^i \\ &= e^{-|x-x'|^2/2}\end{aligned}$$

If n is very large, allocating an n-by-n matrix is tough. Can we truncate the above sum to approximate the kernel?

# RBF kernel and random features

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

$$e^{jz} = \cos(z) + j \sin(z)$$

Recall HW1 where we used the feature map:

$$\phi(x) = \begin{bmatrix} \sqrt{2} \cos(w_1^T x + b_1) \\ \vdots \\ \sqrt{2} \cos(w_p^T x + b_p) \end{bmatrix} \quad \begin{aligned} w_k &\sim \mathcal{N}(0, 2\gamma I) \\ b_k &\sim \text{uniform}(0, \pi) \end{aligned}$$

$$\begin{aligned} \mathbb{E}\left[\frac{1}{p} \phi(x)^T \phi(y)\right] &= \frac{1}{p} \sum_{k=1}^p \mathbb{E}[2 \cos(w_k^T x + b_k) \cos(w_k^T y + b_k)] \\ &= \mathbb{E}_{w,b}[2 \cos(w^T x + b) \cos(w^T y + b)] \end{aligned}$$

# RBF kernel and random features

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

$$e^{jz} = \cos(z) + j \sin(z)$$

Recall HW1 where we used the feature map:

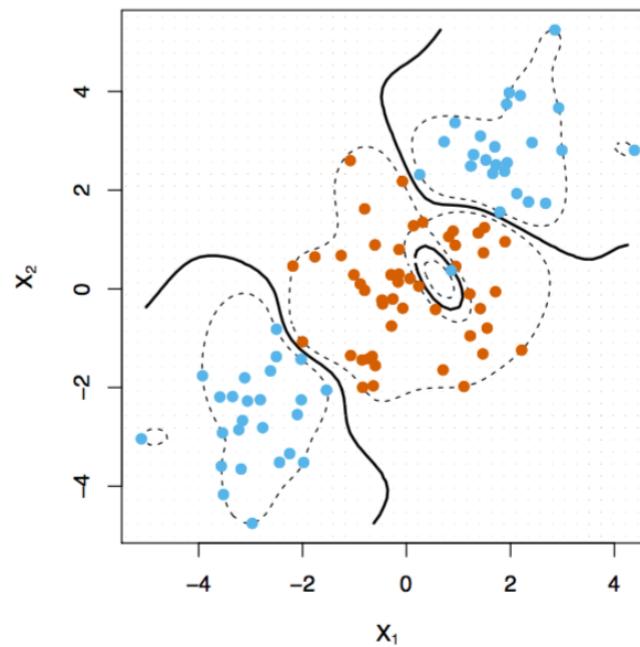
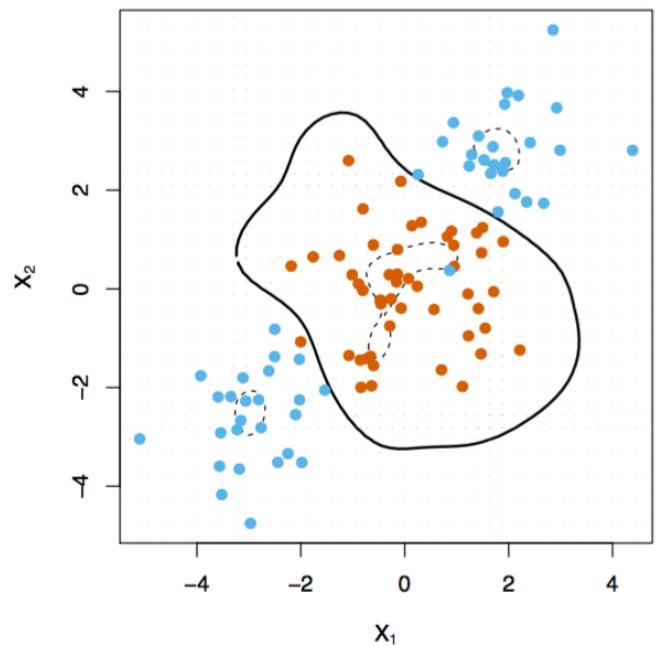
$$\phi(x) = \begin{bmatrix} \sqrt{2} \cos(w_1^T x + b_1) \\ \vdots \\ \sqrt{2} \cos(w_p^T x + b_p) \end{bmatrix} \quad \begin{aligned} w_k &\sim \mathcal{N}(0, 2\gamma I) \\ b_k &\sim \text{uniform}(0, \pi) \end{aligned}$$

$$\begin{aligned} \mathbb{E}\left[\frac{1}{p} \phi(x)^T \phi(y)\right] &= \frac{1}{p} \sum_{k=1}^p \mathbb{E}[2 \cos(w_k^T x + b_k) \cos(w_k^T y + b_k)] \\ &= \mathbb{E}_{w,b}[2 \cos(w^T x + b) \cos(w^T y + b)] \\ &= e^{-\gamma \|x-y\|_2^2} \end{aligned}$$

[Rahimi, Recht NIPS 2007]  
“NIPS Test of Time Award, 2018”

# RBF Classification

$$\widehat{w} = \sum_{i=1}^n \max\{0, 1 - y_i(b + x_i^T w)\} + \lambda \|w\|_2^2$$
$$\min_{\alpha, b} \sum_{i=1}^n \max\{0, 1 - y_i(b + \sum_{j=1}^n \alpha_j \langle x_i, x_j \rangle)\} + \lambda \sum_{i,j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$



# Wait, infinite dimensions?

- Isn't everything separable there? How are we not overfitting?
- Regularization! Fat shattering  $(R/\text{margin})^2$

# String Kernels

Example from Efron and Hastie, 2016

Amino acid sequences of different lengths:

$x_1$

IPT SALV KET LALLS THRT LLI ANET LRI PVP VHK NHQL CTEE IFQ GIG TLE SQT VQGG TV  
ERLF KNLS LIKKY IDG QKK CGEERR RVN QFL DY **LQE** FLG VMN TEWI

$x_2$

PHRR DLCS RSRI WLARK I RSDL TALTES YVKH QGLW SELTEA ER **LQE** NLQAY RTFH VLLA  
RLLED QQV HFT PTEG DFHQ AIHT LLQVA AFAY QIEEL MILLEY KIPR NEAD GMLF EKK  
LWGL KV **LQE** LSQ WTVRSI HDL RFIS SHQT GIP

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

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



# Bootstrap

Machine Learning – CSE446  
Kevin Jamieson  
University of Washington

May 6, 2019

# Limitations of CV

- An 80/20 split throws out a relatively large amount of data if only have, say, 20 examples.
- Test error is informative, but how accurate is this number? (e.g., 3/5 heads vs. 30/50)
- How do I get confidence intervals on statistics like the median or variance of a distribution?
- Instead of the error for the entire dataset, what if I want to study the error for a *particular example x*?

# Limitations of CV

- An 80/20 split throws out a relatively large amount of data if only have, say, 20 examples.
- Test error is informative, but how accurate is this number? (e.g., 3/5 heads vs. 30/50)
- How do I get confidence intervals on statistics like the median or variance of a distribution?
- Instead of the error for the entire dataset, what if I want to study the error for a *particular example x*?

The Bootstrap: Developed by Efron in 1979.

# Bootstrap: basic idea

Given dataset drawn iid samples with CDF  $F_Z$ :

$$\mathcal{D} = \{z_1, \dots, z_n\} \stackrel{i.i.d.}{\sim} F_Z$$

We compute a *statistic* of the data to get:  $\hat{\theta} = t(\mathcal{D})$

# Bootstrap: basic idea

Given dataset drawn iid samples with CDF  $F_Z$ :

$$\mathcal{D} = \{z_1, \dots, z_n\} \stackrel{i.i.d.}{\sim} F_Z$$

We compute a *statistic* of the data to get:  $\hat{\theta} = t(\mathcal{D})$

For  $b=1, \dots, B$  define the *bth bootstrapped* dataset as drawing  $n$  samples **with replacement** from  $D$

$$\mathcal{D}^{*b} = \{z_1^{*b}, \dots, z_n^{*b}\} \stackrel{i.i.d.}{\sim} \hat{F}_{Z,n}$$

and the *bth bootstrapped statistic* as:  $\theta^{*b} = t(\mathcal{D}^{*b})$

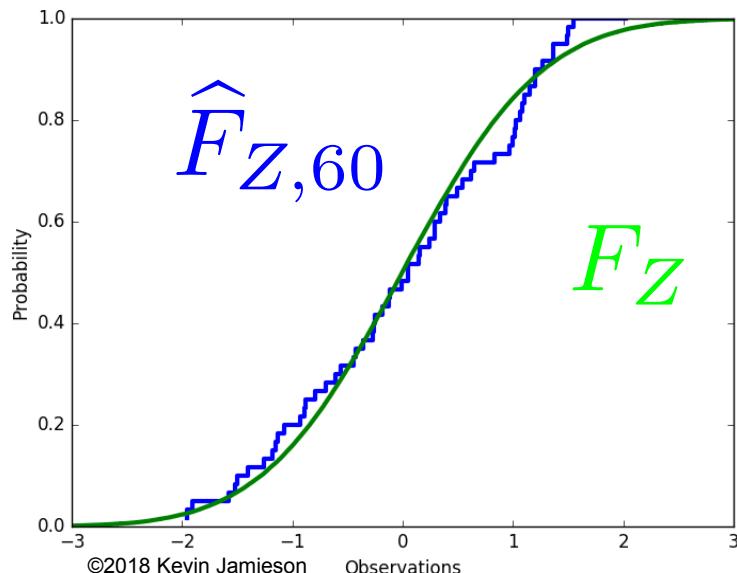
# Bootstrap: basic idea

Given dataset drawn iid samples with CDF  $F_Z$ :

$$\mathcal{D} = \{z_1, \dots, z_n\} \stackrel{i.i.d.}{\sim} F_Z \quad \hat{\theta} = t(\mathcal{D})$$

For  $b=1, \dots, B$ , samples sampled **with replacement** from  $D$

$$\mathcal{D}^{*b} = \{z_1^{*b}, \dots, z_n^{*b}\} \stackrel{i.i.d.}{\sim} \hat{F}_{Z,n} \quad \theta^{*b} = t(\mathcal{D}^{*b})$$



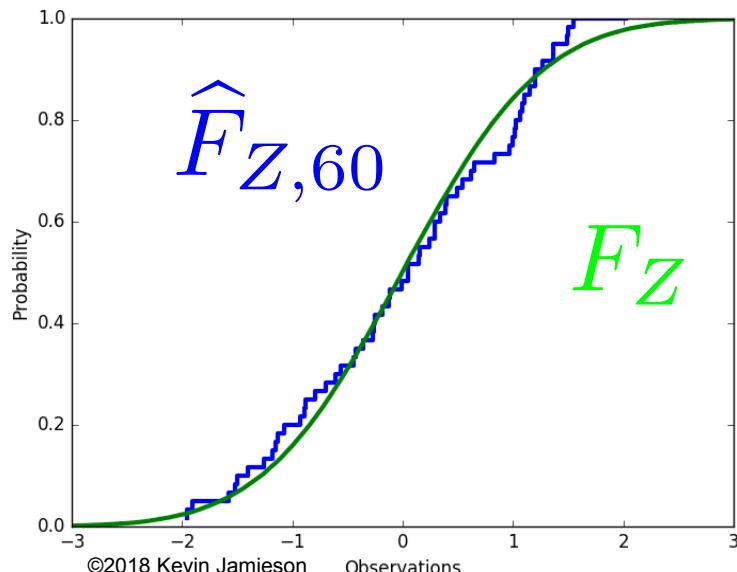
# Bootstrap: basic idea

Given dataset drawn iid samples with CDF  $F_Z$ :

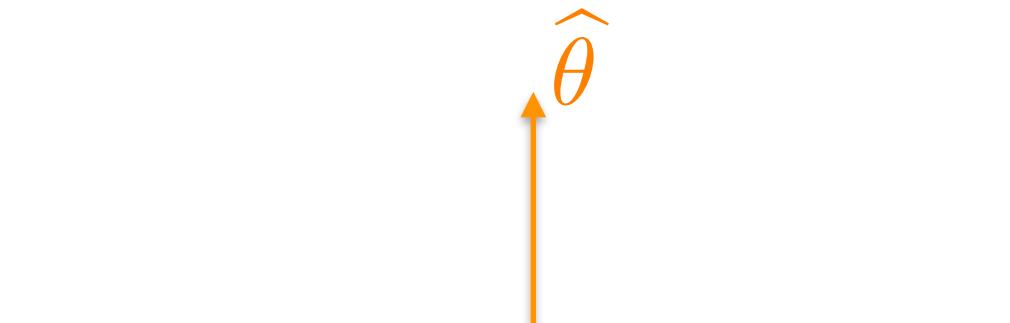
$$\mathcal{D} = \{z_1, \dots, z_n\} \stackrel{i.i.d.}{\sim} F_Z \quad \hat{\theta} = t(\mathcal{D})$$

For  $b=1, \dots, B$ , samples sampled **with replacement** from  $D$

$$\mathcal{D}^{*b} = \{z_1^{*b}, \dots, z_n^{*b}\} \stackrel{i.i.d.}{\sim} \hat{F}_{Z,n} \quad \theta^{*b} = t(\mathcal{D}^{*b})$$



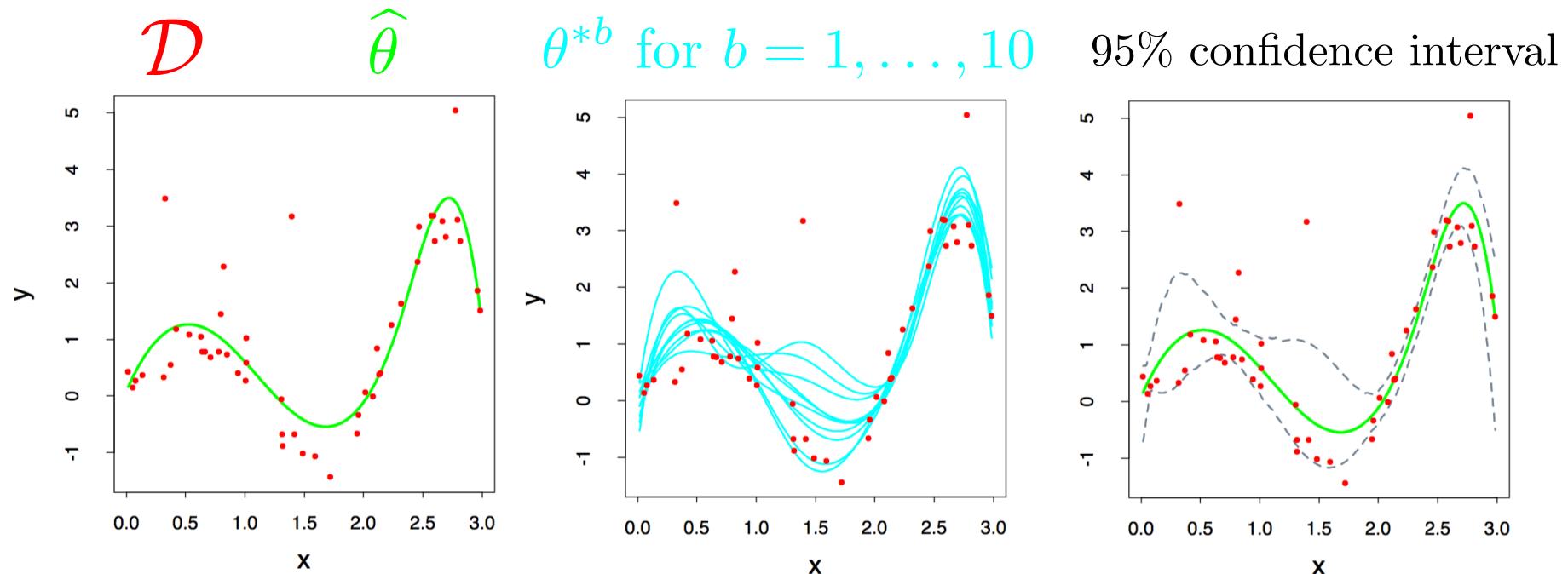
$$\sup_x |\hat{F}_n(x) - F(x)| \rightarrow 0 \quad \text{as } n \rightarrow \infty$$



# Applications

Common applications of the bootstrap:

- Estimate parameters that escape simple analysis like the variance or median of an estimate
- Confidence intervals
- Estimates of error for a particular example:



Figures from Hastie et al

# Takeaways

Advantages:

- Bootstrap is **very** generally applicable. Build a confidence interval around **anything**
- **Very** simple to use
- Appears to give meaningful results even when the amount of data is very small
- Very strong **asymptotic theory** (as num. examples goes to infinity)

# Takeaways

## Advantages:

- Bootstrap is **very** generally applicable. Build a confidence interval around **anything**
- **Very** simple to use
- Appears to give meaningful results even when the amount of data is very small
- Very strong **asymptotic theory** (as num. examples goes to infinity)

## Disadvantages

- Very few meaningful finite-sample guarantees
- Potentially **computationally intensive**
- Reliability relies on test statistic and rate of convergence of empirical CDF to true CDF, which is unknown
- Poor performance on “extreme statistics” (e.g., the max)

Not perfect, but better than nothing.

# Warm up: risk prediction with logistic regression

- Boss gives you a bunch of data on loans defaulting or not:

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

- You model the data as:  $P(Y = y|x, w) = \frac{1}{1 + \exp(-y w^T x)}$
- And compute the maximum likelihood estimator:

$$\hat{w}_{MLE} = \arg \max_w \prod_{i=1}^n P(y_i|x_i, w)$$

For a new loan application  $x$ , boss recommends to give loan if your model says they will repay it with probability at least .95 (i.e. low risk):

Give loan to  $x$  if  $\frac{1}{1 + \exp(-\hat{w}_{MLE}^T x)} \geq .95$

- One year later only half of loans are paid back and the bank folds. What might have happened?

How would you use the bootstrap to do this differently?