# PCA: Efficient computation and some cool applications

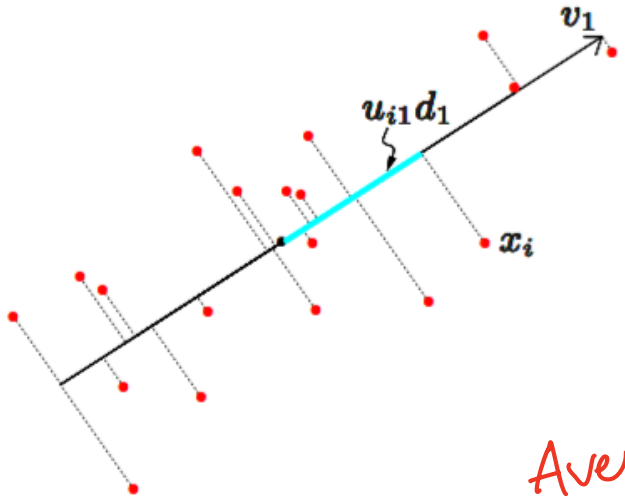
# PCA: a high-fidelity linear projection

Given $x_i \in \mathbb{R}^d$ and some $q < d$ consider

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} ||(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})||^2.$$

where $\mathbf{V}_q = [v_1, v_2, \ldots, v_q]$ is orthonormal:

$$\mathbf{V}_q^T \mathbf{V}_q = I_q$$




$\mathbf{V}_q$ are the first $q$ eigenvectors of $\Sigma$

$\mathbf{V}_q$ are the first q *principal components*

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T$$

Principal Component Analysis (PCA) projects $(\mathbf{X} - \mathbf{1}\bar{x}^T)$ down onto $\mathbf{V}_q$

$$(\mathbf{X} - \mathbf{1}\bar{x}^T)\mathbf{V}_q = \mathbf{U}_q \mathrm{diag}(d_1, \ldots, d_q) \qquad \mathbf{U}_q^T \mathbf{U}_q = I_q$$

# PCA on MNIST

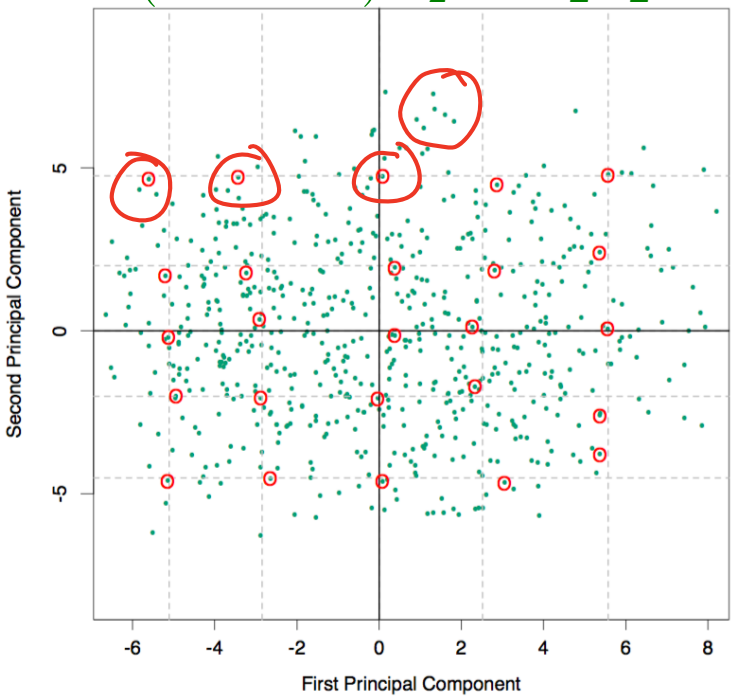
$\mathbf{V}_q$ are the first $q$ eigenvectors of $\Sigma$ and *SVD* $\mathbf{X} - \mathbf{1}\bar{x}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T$

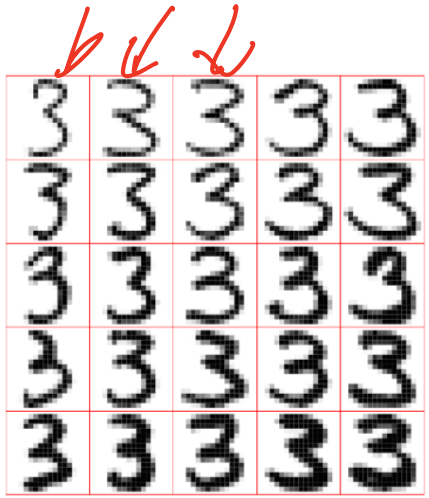
Handwritten 3's, 16x16 pixel image so that $x_i \in \mathbb{R}^{256}$

$$\hat{f}(\lambda) = \bar{x} + \lambda_1 v_1 + \lambda_2 v_2$$

$$= \text{<image>} + \lambda_1 \cdot \text{<image>} + \lambda_2 \cdot \text{<image>}.$$

$(\mathbf{X} - \mathbf{1}\bar{x}^T)\mathbf{V}_2 = \mathbf{U}_2\mathbf{S}_2 \in \mathbb{R}^{n \times 2}$



$\text{diag}(\mathbf{S})$

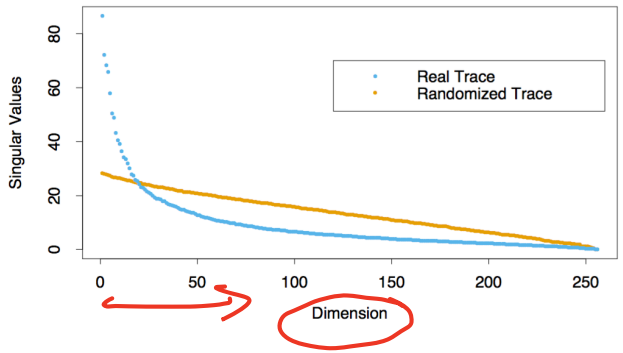
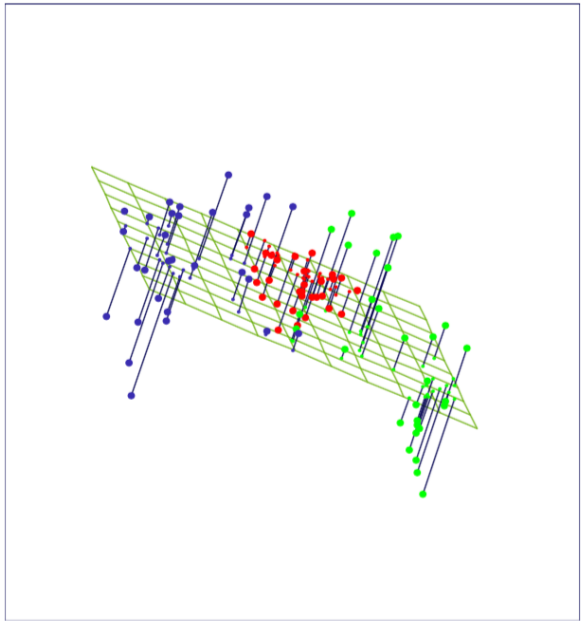


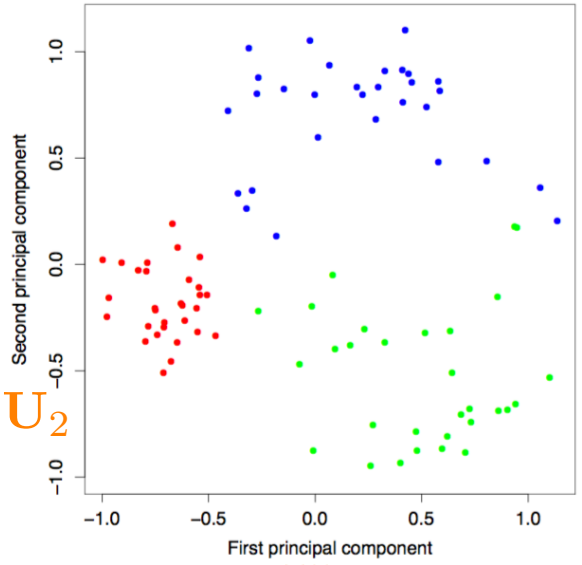

**FIGURE 14.24.** *The 256 singular values for the digitized threes, compared to those for a randomized version of the data (each column of* $\mathbf{X}$ *was scrambled).*

# SVD and PCA

$\mathbf{V}_q$ are the first $q$ eigenvectors of $\Sigma$ and *SVD* $\mathbf{X} - \mathbf{1}\bar{x}^T = \mathbf{USV}^T$



$\mathbf{X} - \mathbf{1}\bar{x}^T$




$\mathbf{U}_2$

$\mathbf{U}_1$

# How do we compute the principal components?

1. Power iteration
2. **Solving for a singular value decomposition (SVD)**

# Singular Value Decomposition (SVD)

MM : $O(n^\omega)$

**Theorem (SVD)**: Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with rank $r \leq \min\{m, n\}$. Then $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ where $\mathbf{S} \in \mathbb{R}^{r \times r}$ is diagonal with positive entries, $\mathbf{U}^T\mathbf{U} = I$, $\mathbf{V}^T\mathbf{V} = I$.

$\mathbf{U}^T\mathbf{U} = I$ : U has orthonormal rows

$\mathbf{V}^T\mathbf{V} = I$ : V has orthonormal columns

$\begin{bmatrix} + & & & 0 \\ & + & & \\ & & + & \\ 0 & & & + \end{bmatrix}$

$\mathbf{A}^T\mathbf{A}v_i = \sigma_i^2 v_i$

$r \times r \times r \times 1 \rightarrow r \times 1$

$\mathbf{A}\mathbf{A}^T u_i = \sigma_i^2 u_i$

$\mathbf{A}\mathbf{A}^T = S^2$

$\mathbf{A}^T\mathbf{A} = (USV^T)^T(USV^T)$

$= VS^TU^TUSV^T$

$= VS^2V^T$

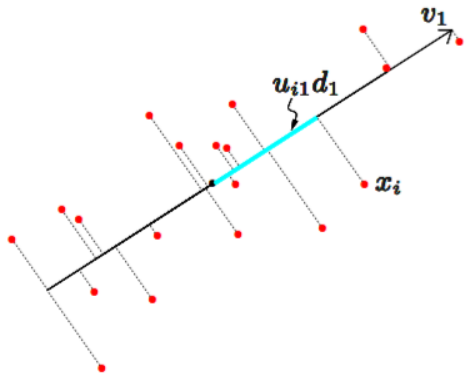
$= S^2$

# How do we compute the principal components?

1. **Power iteration**
2. Solving for a singular value decomposition (SVD)

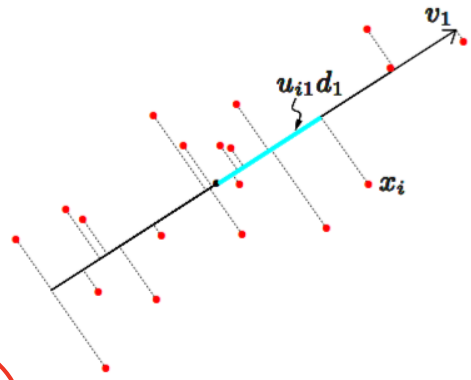
# Power method - one vector at a time

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \qquad v_* = \arg\max_{v} \; v^T \Sigma v$$



# Power method - one vector found iteratively

$$\Sigma := \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \qquad v_* = \arg\max_{v} \; v^T \Sigma v$$



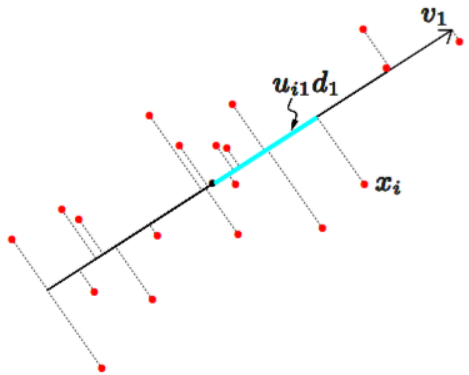

$z_0 \sim \mathcal{N}(0, I)$

Iterate: $z_{t+1} = \dfrac{\Sigma z_t}{\|\Sigma z_t\|_2}$

To analyze write: $\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T \qquad z_t =: \mathbf{V}\alpha_t$

# Power method - analysis

$$\Sigma := \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T \qquad v_* = \arg\max_{v} v^T \Sigma v$$




$$z_0 \sim \mathcal{N}(0, I)$$

Iterate: $$z_{t+1} = \frac{\Sigma z_t}{\|\Sigma z_t\|_2}$$

To analyze write: $$\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T \qquad z_t =: \mathbf{V}\alpha_t$$

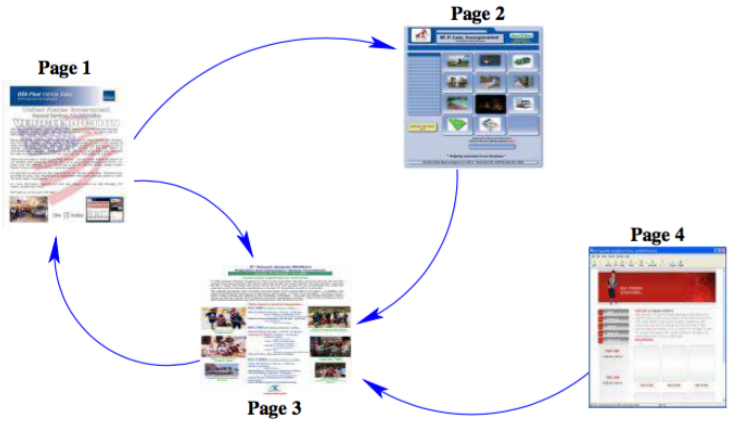
$$\alpha_{t+1} = \mathbf{V}^T z_{t+1} = \frac{\mathbf{V}^T \Sigma z_t}{\|\Sigma z_t\|} = \frac{\mathbf{D}\alpha_t}{\|\mathbf{D}\alpha_t\|} = \frac{\mathbf{D}^2\alpha_{t-1}}{\|\mathbf{D}^2\alpha_{t-1}\|} = \frac{\mathbf{D}^t\alpha_0}{\|\mathbf{D}^t\alpha_0\|}$$

$$\mathbf{D}^t = (\mathbf{D}_{1,1})^t(\mathbf{D}/\mathbf{D}_{1,1})^t \to (\mathbf{D}_{1,1})^t \mathbf{e}_1\mathbf{e}_1^T \text{ since } \mathbf{D}_{i,i}/\mathbf{D}_{1,1} < 1$$

# Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$




Google PageRank of page i:

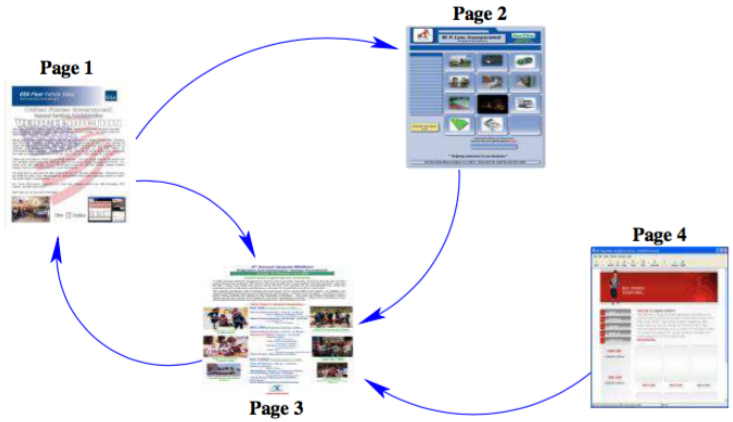
$$p_i = (1-\lambda) + \lambda \sum_{j=1}^{n} \frac{L_{i,j}}{c_j} p_j$$

$$c_j = \sum_{k=1}^{n} L_{j,k}$$

# Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



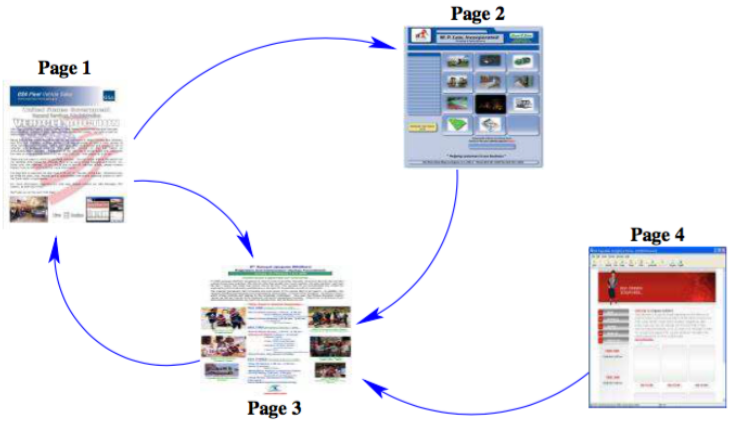

Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

# Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$




Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Set arbitrary normalization: $\mathbf{1}^T\mathbf{p} = n$ so that

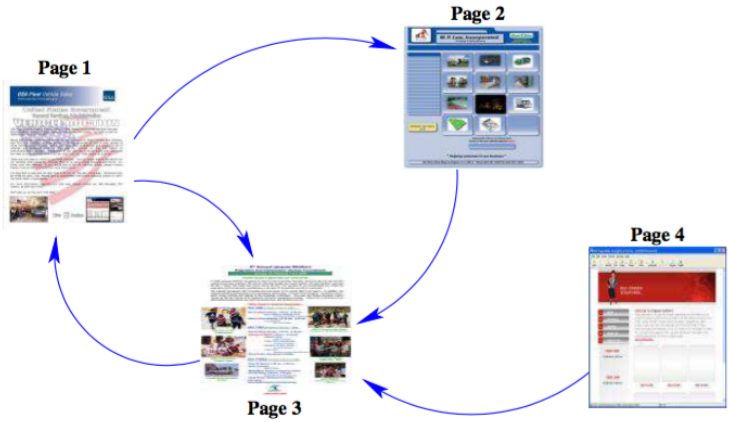
$$\mathbf{p} = \left((1-\lambda)\mathbf{1}\mathbf{1}^T/n + \lambda \mathbf{L}\mathbf{D}_c^{-1}\right)\mathbf{p}$$

$$=: \mathbf{A}\mathbf{p}$$

# Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$




Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Set arbitrary normalization: $\mathbf{1}^T\mathbf{p} = n$ so that

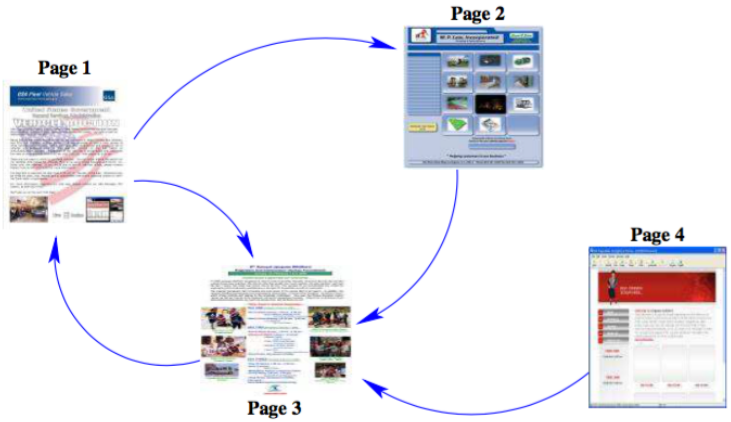
$$\mathbf{p} = \left((1-\lambda)\mathbf{1}\mathbf{1}^T/n + \lambda \mathbf{L}\mathbf{D}_c^{-1}\right)\mathbf{p}$$
$$=: \mathbf{A}\mathbf{p}$$

$\mathbf{p}$ is an eigenvector of $\mathbf{A}$ with eigenvalue 1! And by the properties stochastic matrices, it corresponds to the *largest* eigenvalue

# Markov chains - PageRank

$$L_{i,j} = \mathbf{1}\{\text{page } j \text{ points to page } i\}$$

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$




Google PageRank of pages given by:

$$\mathbf{p} = (1-\lambda)\mathbf{1} + \lambda \mathbf{L}\mathbf{D}_c^{-1}\mathbf{p}$$

Set arbitrary normalization: $\mathbf{1}^T\mathbf{p} = n$ so that

$$\begin{aligned} \mathbf{p} &= \left((1-\lambda)\mathbf{1}\mathbf{1}^T/n + \lambda \mathbf{L}\mathbf{D}_c^{-1}\right)\mathbf{p} \\ &=: \mathbf{A}\mathbf{p} \end{aligned}$$

$\mathbf{p}$ is an eigenvector of $\mathbf{A}$ with eigenvalue 1! And by the properties stochastic matrices, it corresponds to the *largest* eigenvalue

Solve using power method:

$$\mathbf{p}_{k+1} = \frac{\mathbf{A}\mathbf{p}_k}{\mathbf{1}^T\mathbf{A}\mathbf{p}_k/n} \qquad \mathbf{p}_0 \sim \text{uniform}([0,1]^n)$$

# PCA and SVD take-aways

PCA finds a $q$-dimensional representation with:

- Highest variance in any $q$-dimensional space
- Lowest reconstruction error
- spanned by the top $q$ eigenvectors of covariance matrix

How to find the top $q$ eigenvectors?

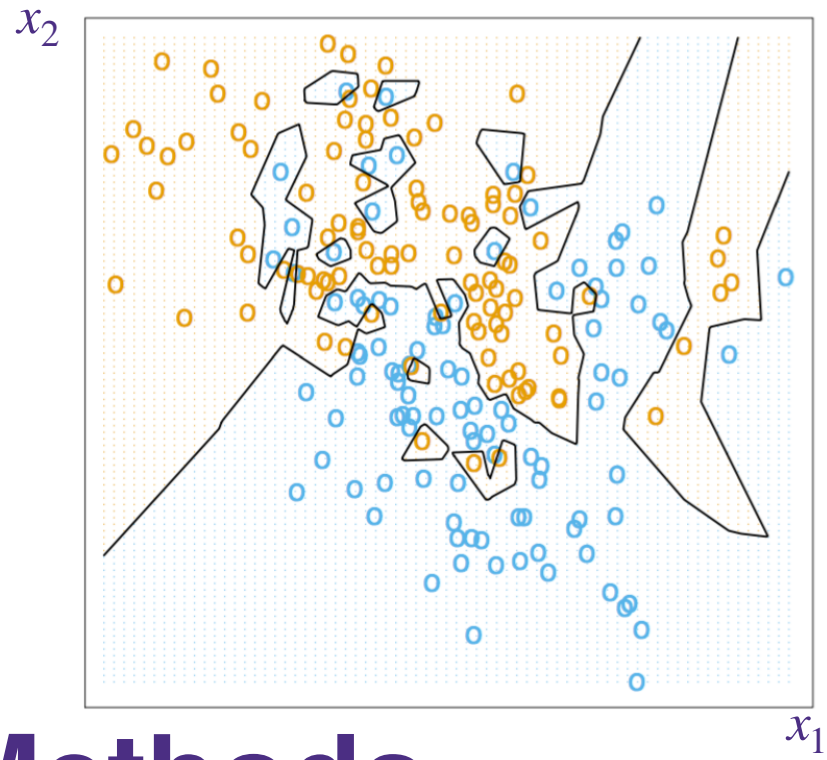
- SVD: $(X - I\,\mu) := A = U\,S\,V^T$
  - ***V are the eigenvectors of $A^TA$***
  - U are the eigenvectors of $AA^T$
- Power method

This is one way to represent data in lower dimensions: there are others with other properties

- E.g., that approximately maintain pairwise distances

# Miscellaneous fun stuff!

# Nonparametric models for classification



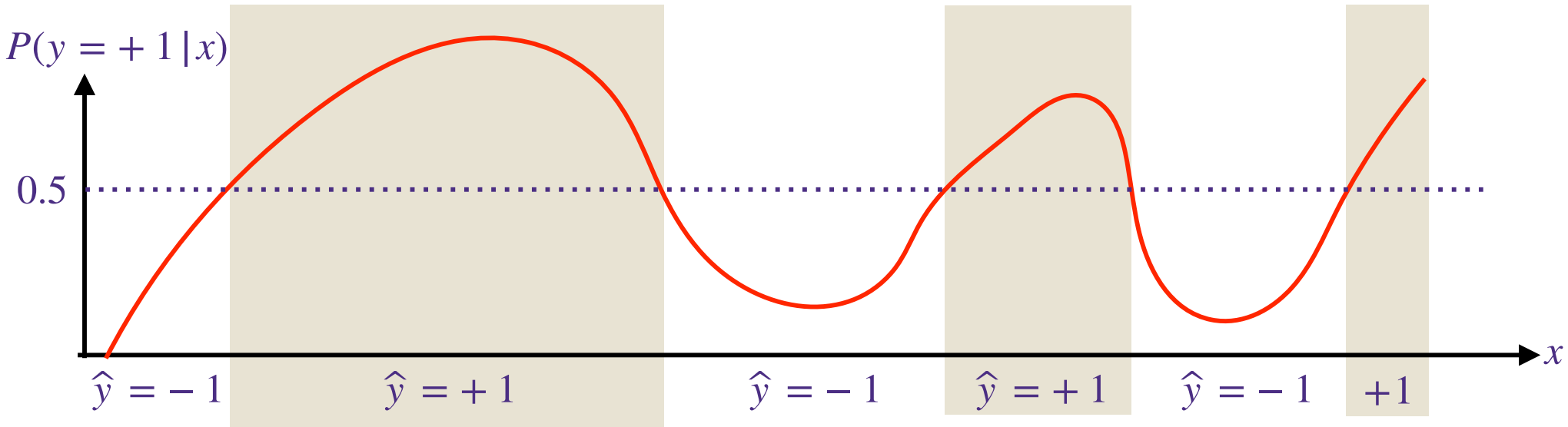

# Nearest Neighbor Methods

- Yet another non-linear model
  - Kernel method
  - Neural Network
  - Nearest Neighbor method
- A model is called “parametric” if the number of parameters do not depend on the number of samples
- A model is called “non-parametric” if the number of parameters increase with the number of samples

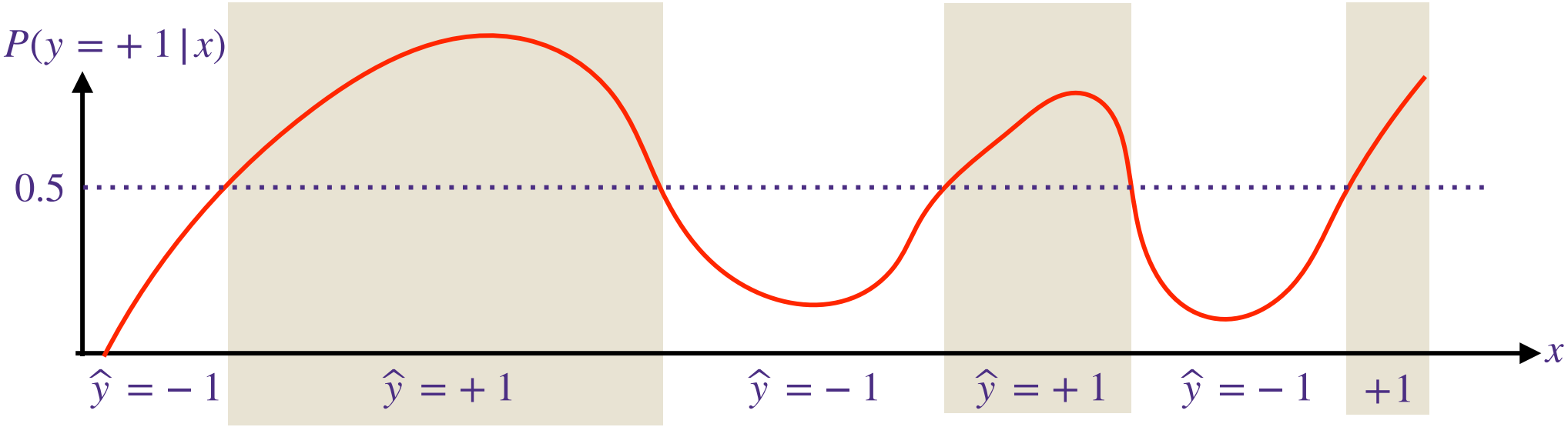
# Recall Bayes optimal classifier

- Consider an example of binary classification on 1-dimensional $x \in \mathbb{R}$
- The problem is fully specified by the ground truths $P_{X,Y}(x, y)$
- Suppose for simplicity that $P_Y(y = +1) = P_Y(y = -1) = 1/2$
- Bayes optimal classifier minimizes the conditional error $P(\hat{y} \neq y \,|\, x)$ for every $x$, which can be written explicitly as

$$\hat{y} = \begin{cases} +1 & \text{if } P(+1 \,|\, x) > P(-1 \,|\, x) \\ -1 & \text{if } P(+1 \,|\, x) < P(-1 \,|\, x) \end{cases}$$



# In practice we do not have $P(x, y)$

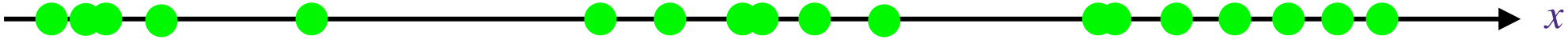



- Bayes optimal classifier $\hat{y} = +1$ if $P(+1 \mid x) > P(-1 \mid x)$
  $-1$ if $P(+1 \mid x) < P(-1 \mid x)$

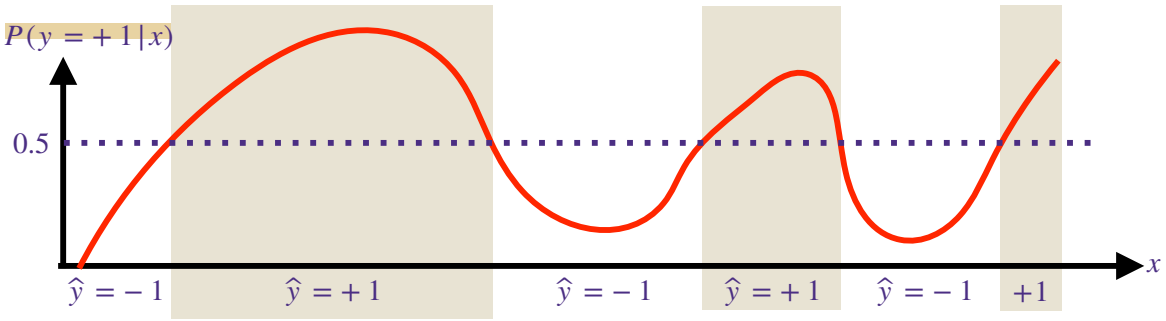
- How do we compare $P(y = +1 \mid x)$ and $P(y = -1 \mid x)$ from samples?

samples with $y = +1$


samples with $y = -1$



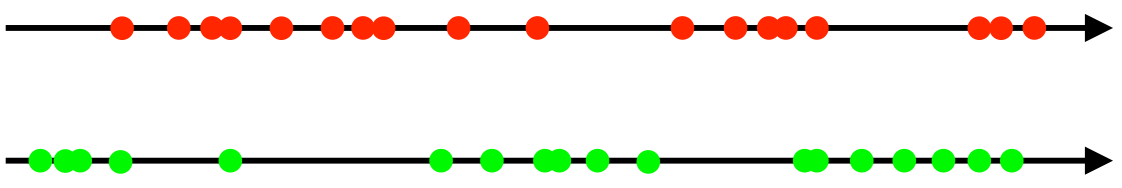
# One way to approximate Bayes Classifier = local statistics




- Bayes optimal classifier

$$\hat{y} = +1 \text{ if } P(+1\,|\,x) > P(-1\,|\,x)$$
$$-1 \text{ if } P(+1\,|\,x) < P(-1\,|\,x)$$

decision is based on $\dfrac{P(x, y=+1)}{P(x, y=-1)}$




- $k$-nearest neighbors classifier considers the $k$-nearest neighbors and takes a majority vote

$\hat{y} = +1$, if (# of +1 samples) > (# of -1 samples)

$-1$, if (# of +1 samples) < (# of -1 samples)

- Decision is based on $\dfrac{\text{\# of +1 samples}}{\text{\# of -1 samples}}$

- Denote the $n_r^+$ as the number of samples within distance $r$ from $x$ with label $+1$, then

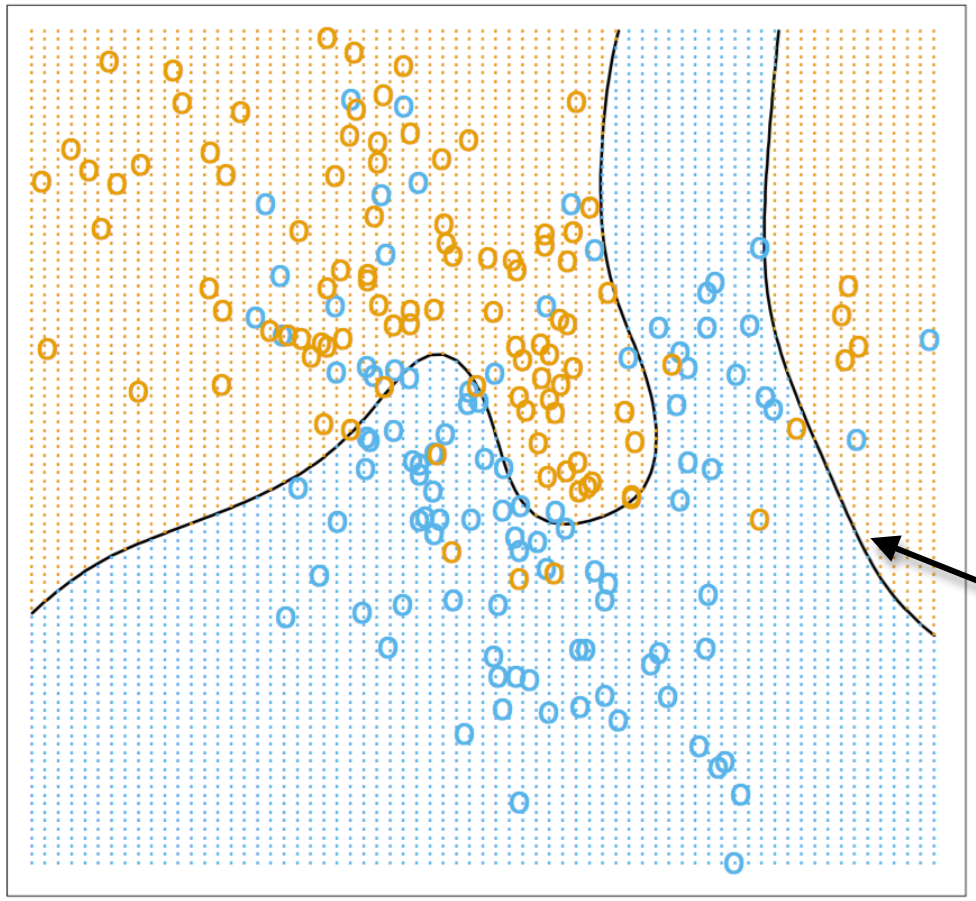
$$\frac{n_r^+}{n} \longrightarrow 2r \times P(x, y=+1)$$

as we increase $n$ and decrease $r$.

- If we take $r$ to be the distance to the $k$-th neighbor from $x$, then

$$\frac{\text{\# of +1 samples}}{\text{\# of -1 samples}} \longrightarrow \frac{P(x, y=+1)}{P(x, y=-1)}$$

# Some data, Bayes Classifier



Training data:

True label: +1

True label: -1

Optimal "Bayes" classifier:

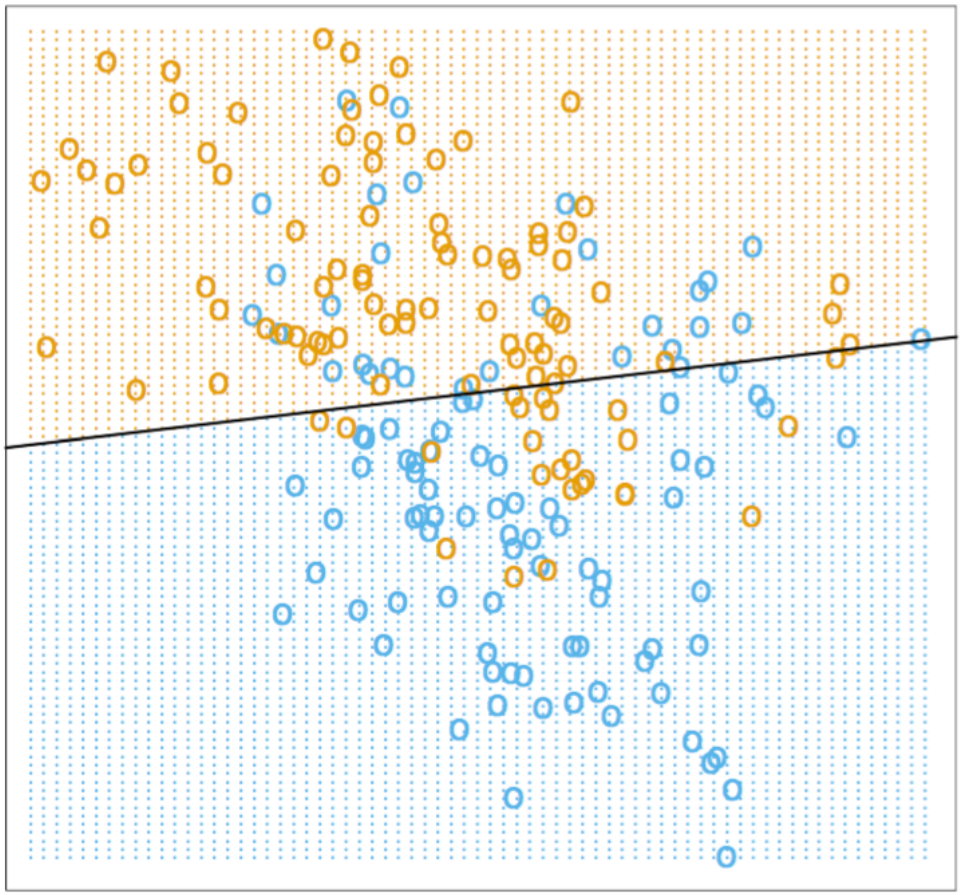
$$\mathbb{P}(Y = 1|X = x) = \frac{1}{2}$$

Predicted label: +1

Predicted label: -1

Figures from Hastie et al.

# Linear Decision Boundary



Training data:

True label: +1

True label: -1

Learned:

Linear Decision boundary

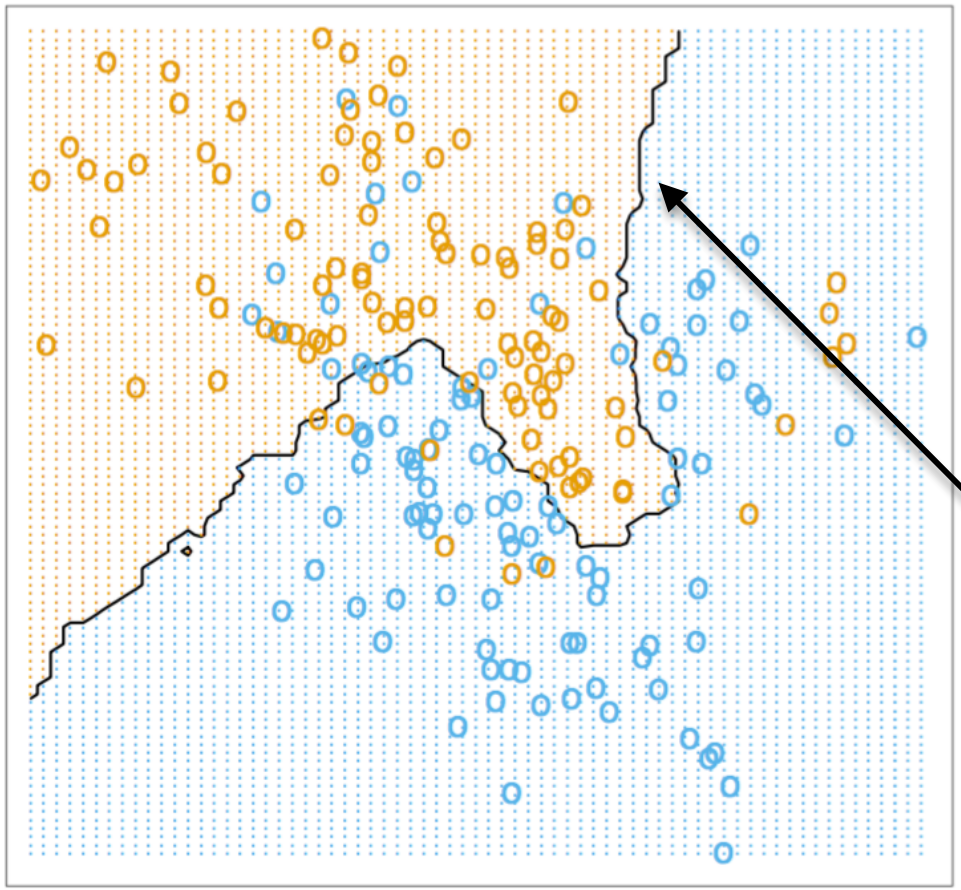
$$x^T w + b = 0$$

Predicted label: +1

Predicted label: -1

Figures from Hastie et al

# $k$=15 Nearest Neighbor Boundary



Training data:

True label: +1

True label: -1

Learned:

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

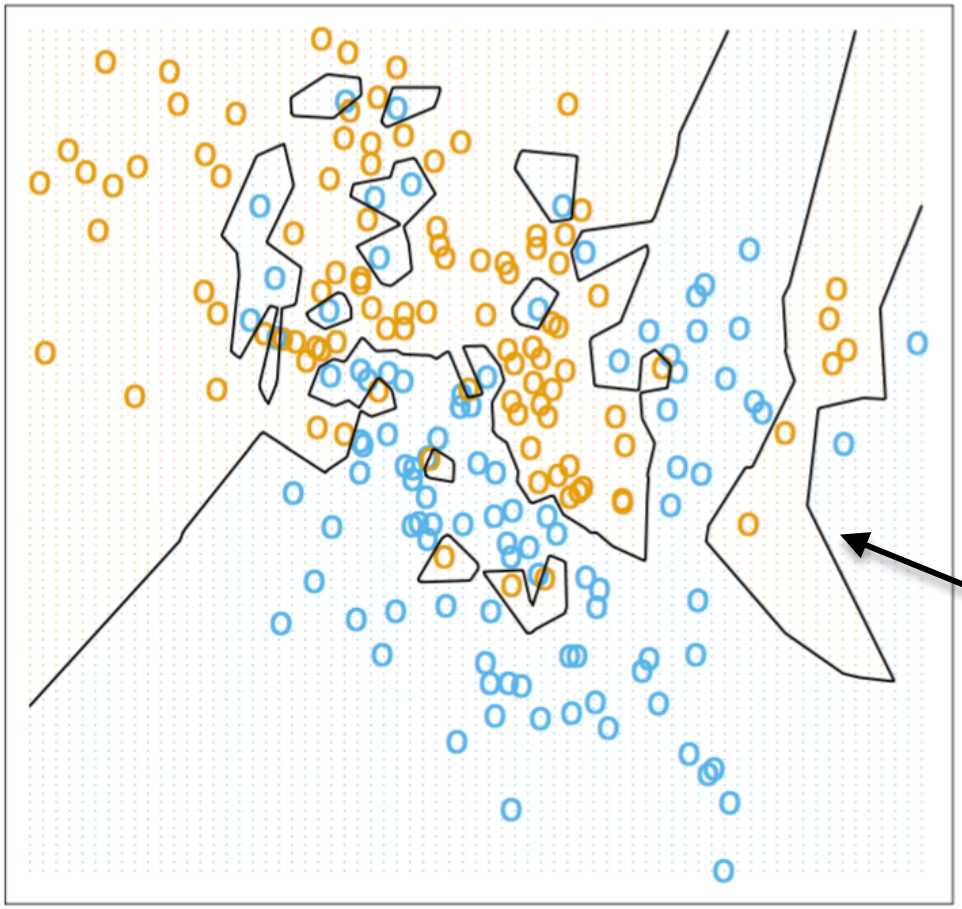
Predicted label: +1

Predicted label: -1

- Nearest neighbor gives non-linear decision boundaries
- What happens if we use a small $k$ or a large $k$?

Figures from Hastie et al

# k=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

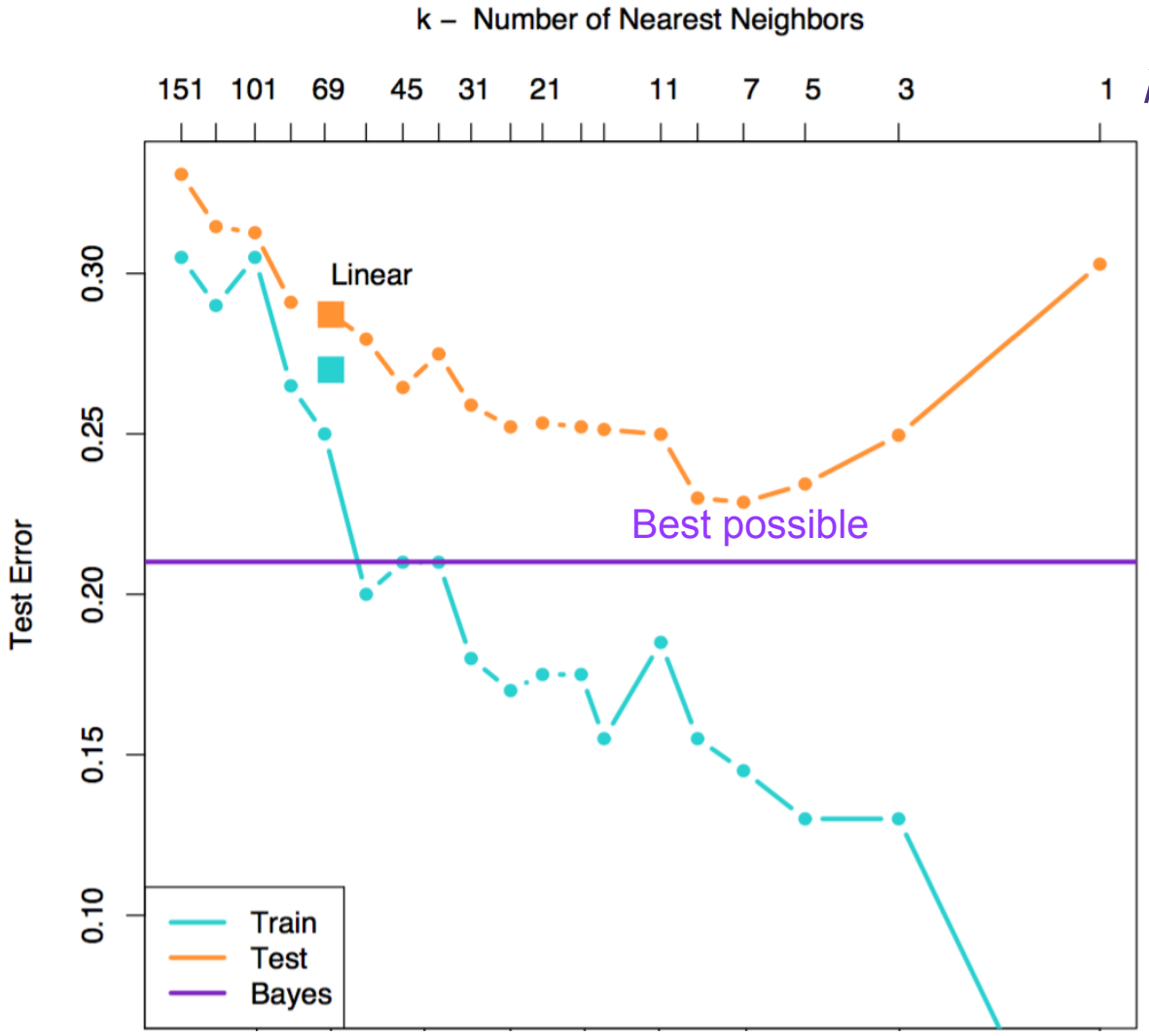
- With a small $k$, we tend to overfit.

Figures from Hastie et al

# k-Nearest Neighbor Error

Model complexity low

Model complexity high




Bias-Variance tradeoff

As k->infinity?

Bias:

Variance:

As k->1?

Bias:

Variance:

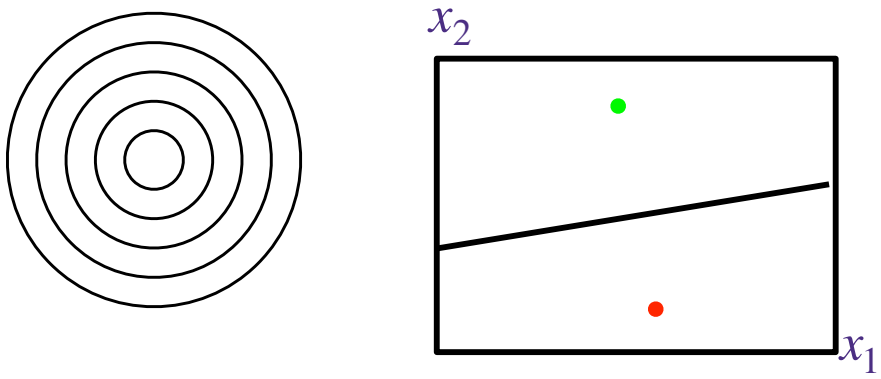
Figures from Hastie et al

- The error achieved by Bayes optimal classifier provides a lower bound on what any estimator can achieve

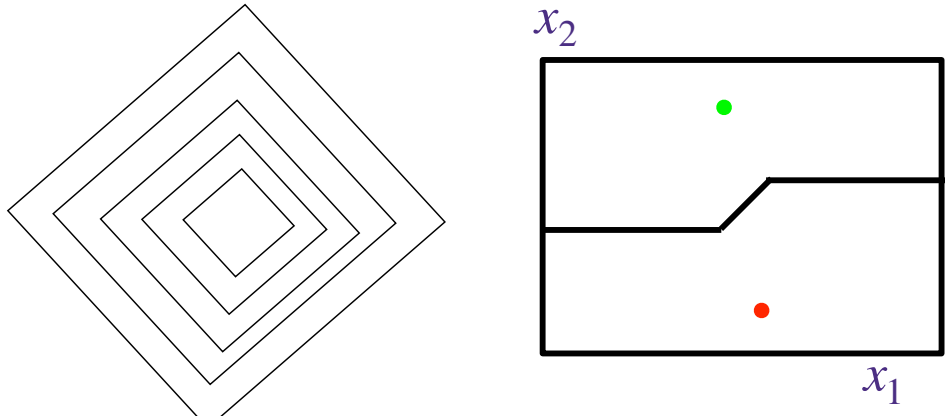
# Notable distance metrics (and their level sets)

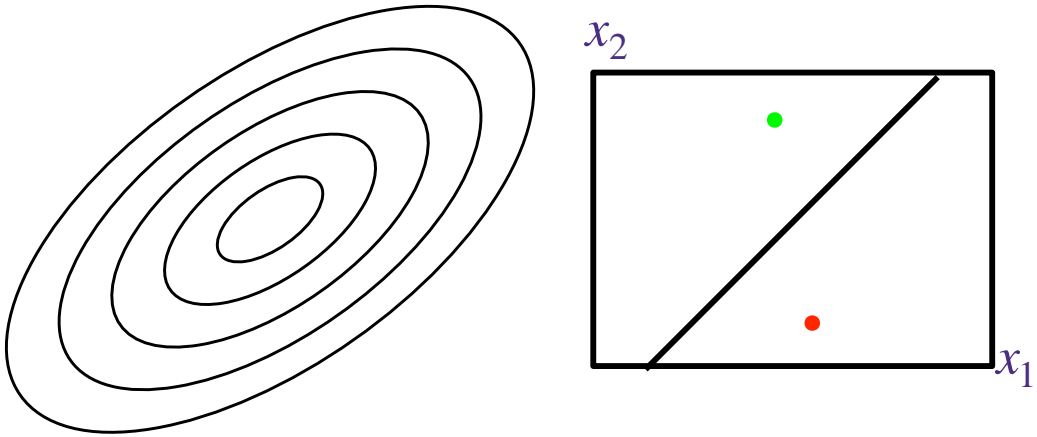
Consider 2 dimensional example with 2 data points with labels green, red,
and we show $k = 1$nearest neighbor decision boundaries for various choices of distances

**$L_2$ norm :** $d(x, y) = \|x - y\|_2$

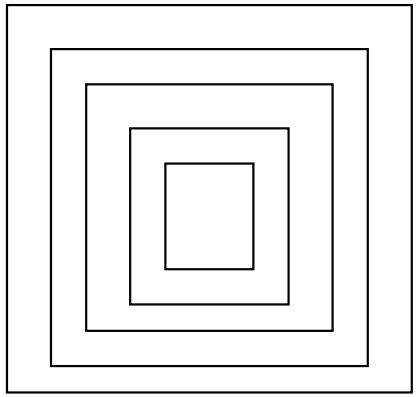



**$L_1$ norm (taxi-cab)**

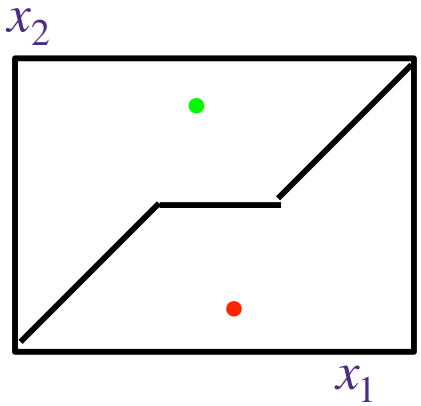



**Mahalanobis norm:** $d(x, y) = (x - y)^T M (x - y)$

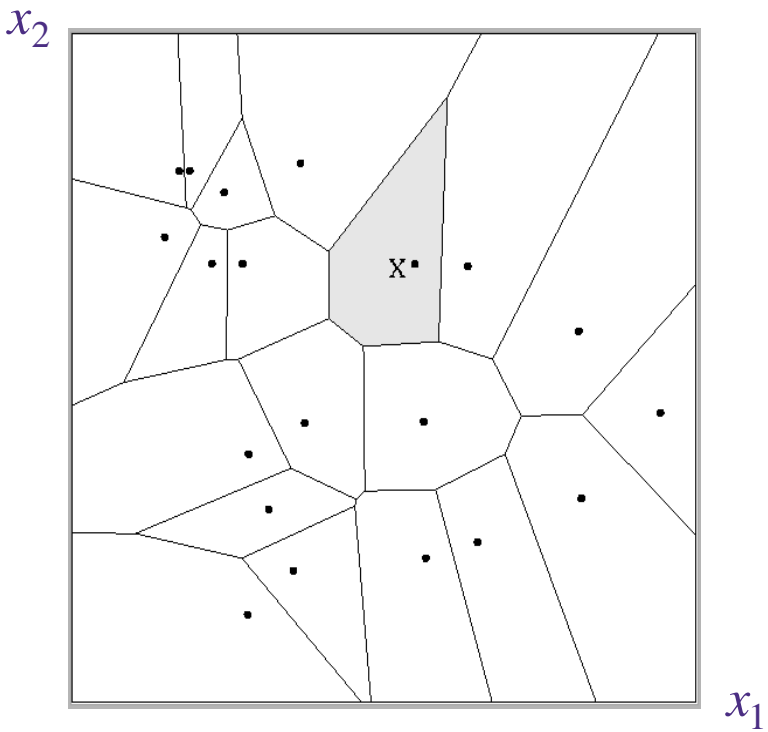


**L-infinity *(max) norm***

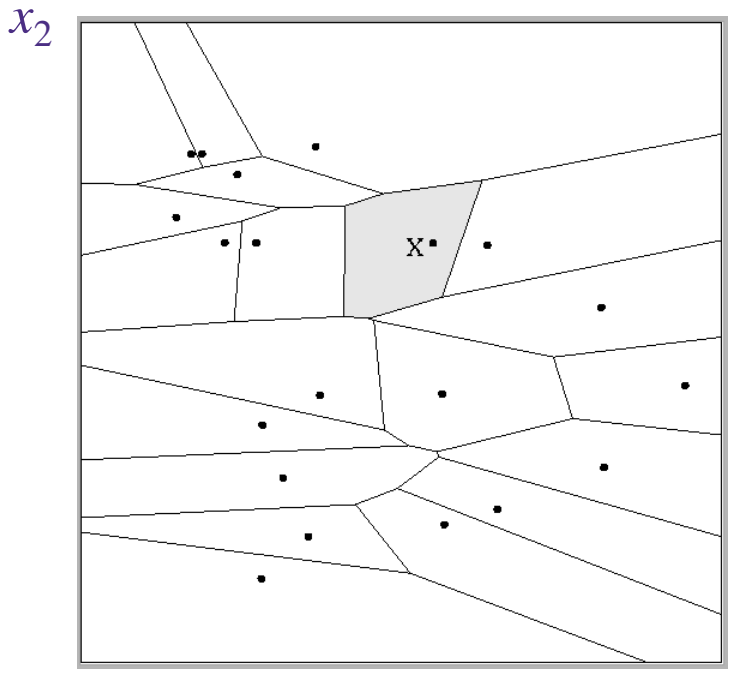


# $k = 1$ nearest neighbor

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 - classification

$$\{(x_i, y_i)\})_{i=1}^n \qquad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\} \qquad (x_i, y_i) \overset{iid}{\sim} P_{XY}$$

**Theorem**[Cover, Hart, 1967] If $P_X$ is supported everywhere in $\mathbb{R}^d$ and $P(Y = 1|X = x)$ is smooth everywhere, then as $n \to \infty$ the 1-NN classification rule has error at most twice the Bayes error rate.

# 1 nearest neighbor guarantee - classification

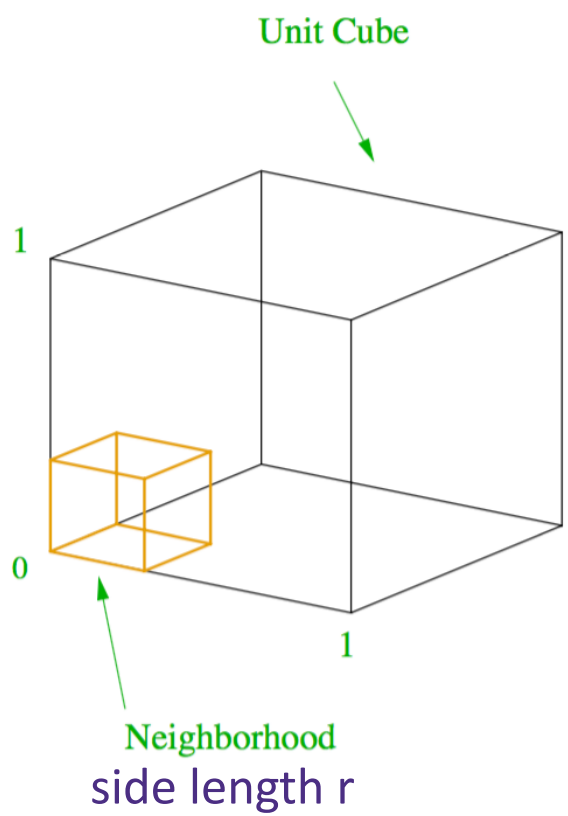
$$\{(x_i, y_i)\})_{i=1}^n \qquad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\} \qquad (x_i, y_i) \overset{iid}{\sim} P_{XY}$$

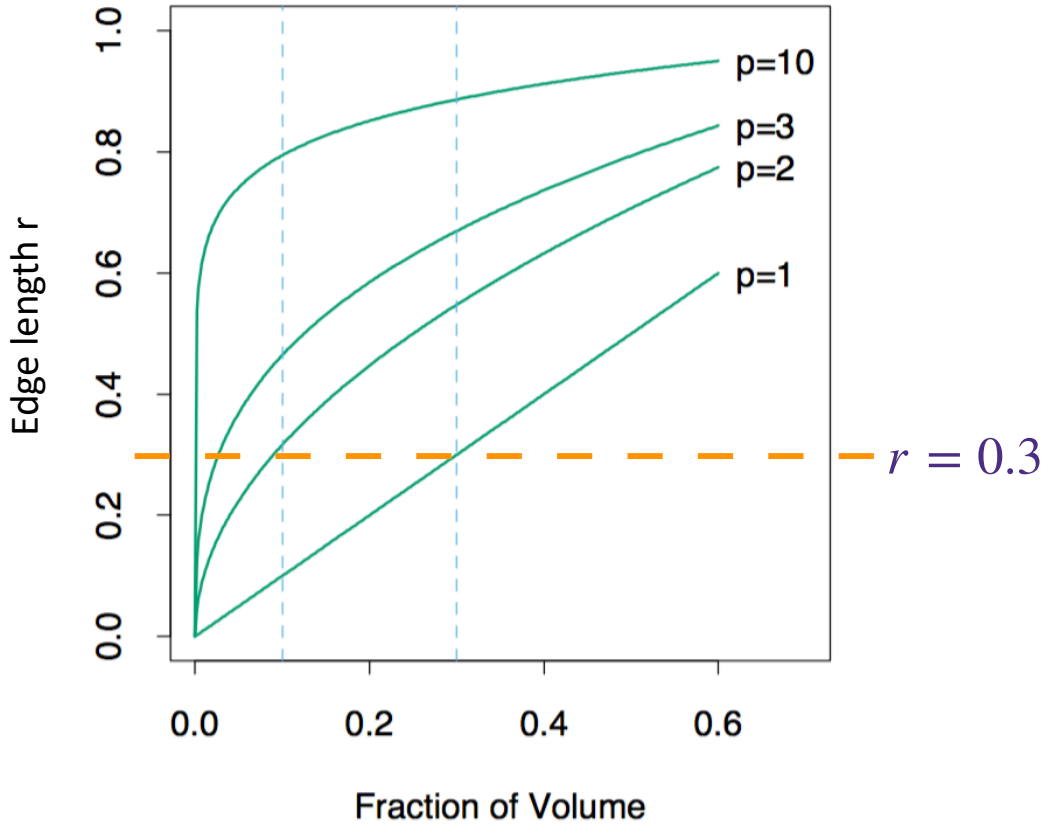
**Theorem**[Cover, Hart, 1967] If $P_X$ is supported everywhere in $\mathbb{R}^d$ and $P(Y = 1|X = x)$ is smooth everywhere, then as $n \to \infty$ the 1-NN classification rule has error at most twice the Bayes error rate.

- Let $x_{NN}$ denote the nearest neighbor at a point $x$
- First note that as $n \to \infty, P(y = +1 \,|\, x_{NN}) \to P(y = +1 \,|\, x)$
- Let $p^* = \min\{P(y = +1 \,|\, x), P(y = -1 \,|\, x)\}$ denote the Bayes error rate
- At a point $x$,
  - Case 1: nearest neighbor is $+1$, which happens with $P(y = +1 \,|\, x)$ and the error rate is $P(y = -1 \,|\, x)$
  - Case 2: nearest neighbor is $+1$, which happens with $P(y = -1 \,|\, x)$ and the error rate is $P(y = +1 \,|\, x)$
- The average error of a 1-NN is

$$P(y = +1 \,|\, x)\, P(y = -1 \,|\, x) + P(y = -1 \,|\, x)\, P(y = +1 \,|\, x) \;= 2p^*(1 - p^*)$$

# Curse of dimensionality Ex. 1



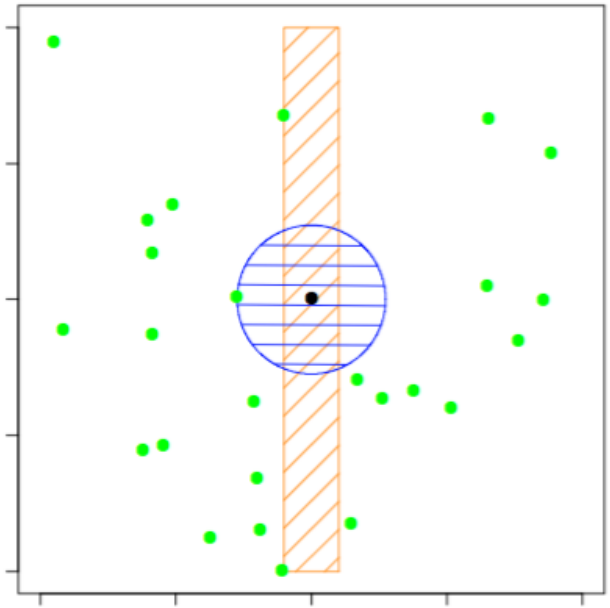



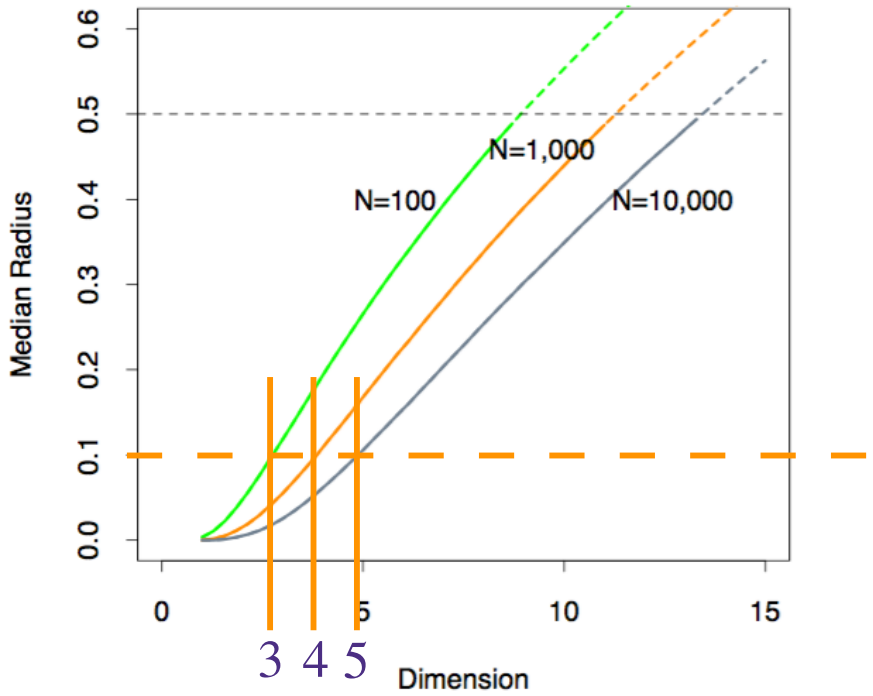

$X$ is uniformly distributed over $[0,1]^p$. What is $\mathbb{P}(X \in [0,r]^p)$?

How many samples do we need so that a nearest neighbor is within a cube of side length $r$?

# Curse of dimensionality Ex. 2

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

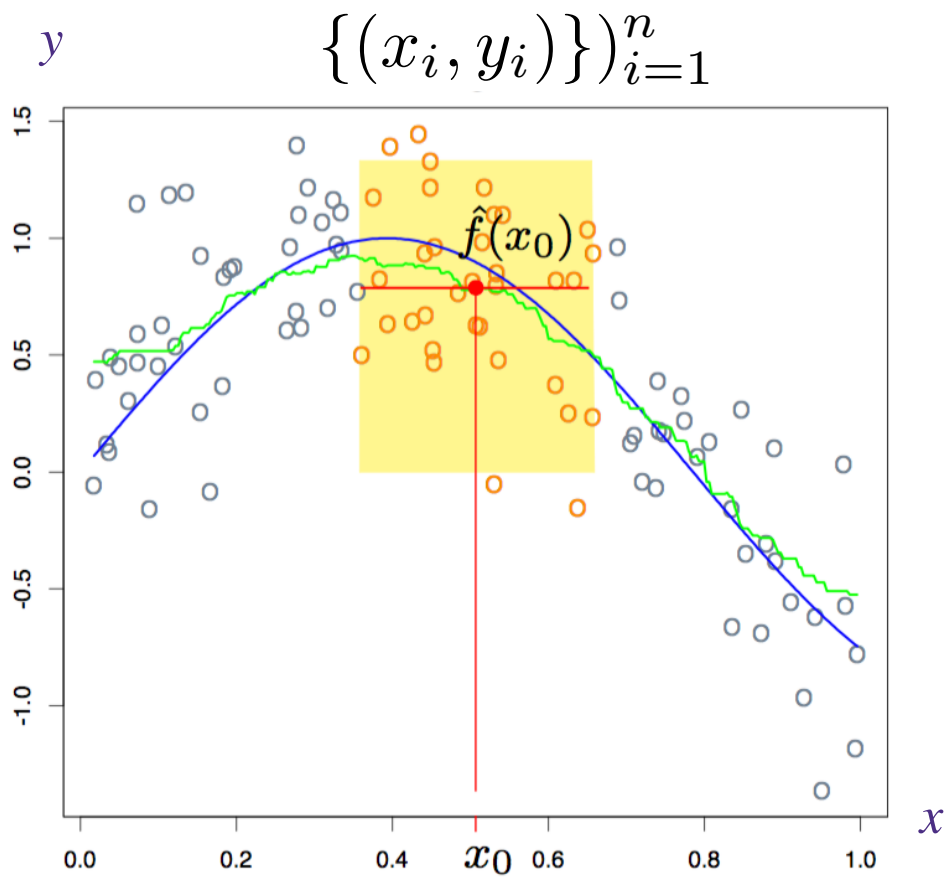





What is the median distance from a point at origin to its 1NN?

How many samples do we need so that a median Euclidean distance is within $r$?

# Nearest neighbor regression




- What is the optimal classifier that minimizes MSE $\mathbb{E}[(\hat{y} - y)^2]$?

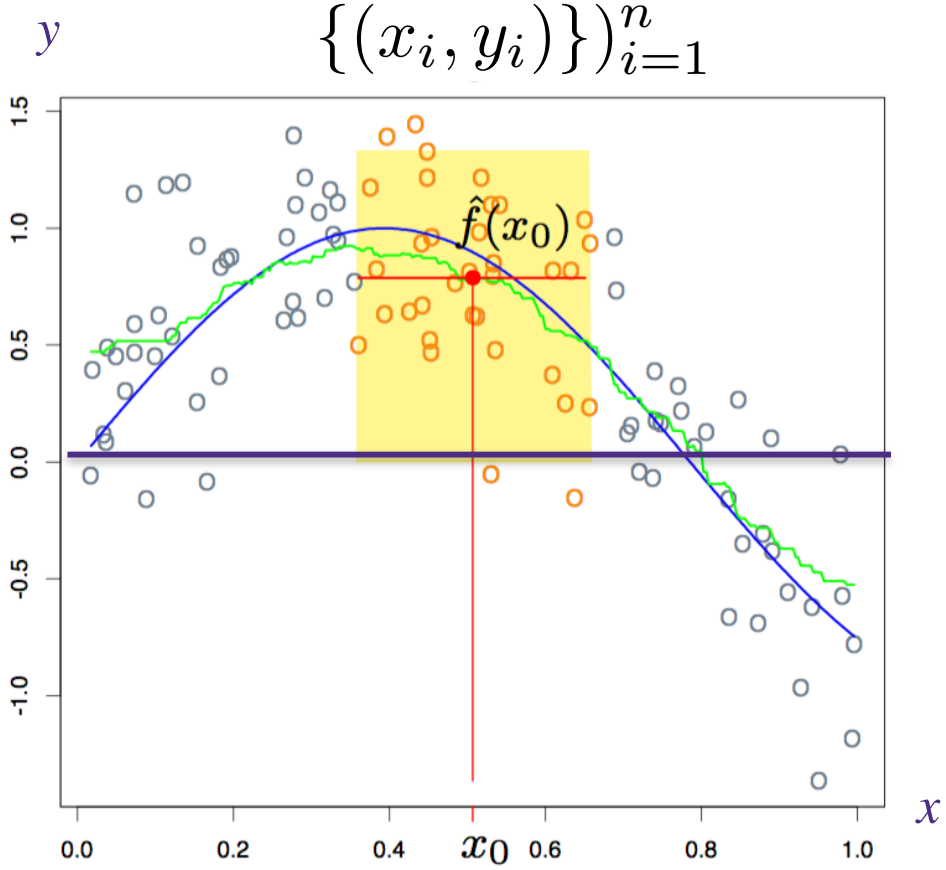
$$\hat{y} = \mathbb{E}[y \mid x]$$

- $k$-nearest neighbor regressor is

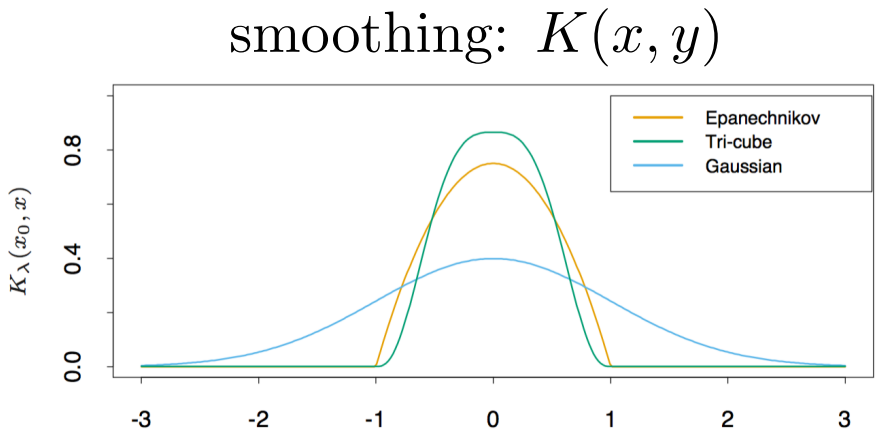
$$\hat{f}(x) = \frac{1}{k} \sum_{j \in \text{nearest neighbor}} y_j$$

$$= \frac{\sum_{i=1}^n y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^n \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

# Nearest neighbor regression




In nearest neighbor methods, the "weight" changes abruptly

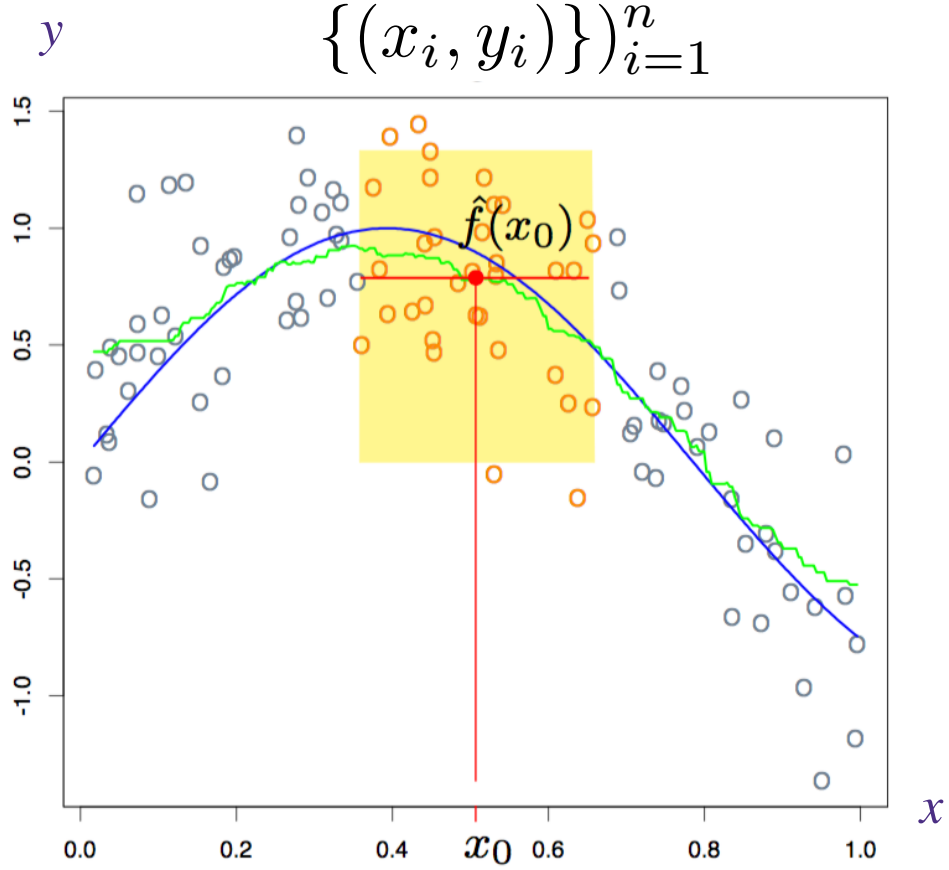


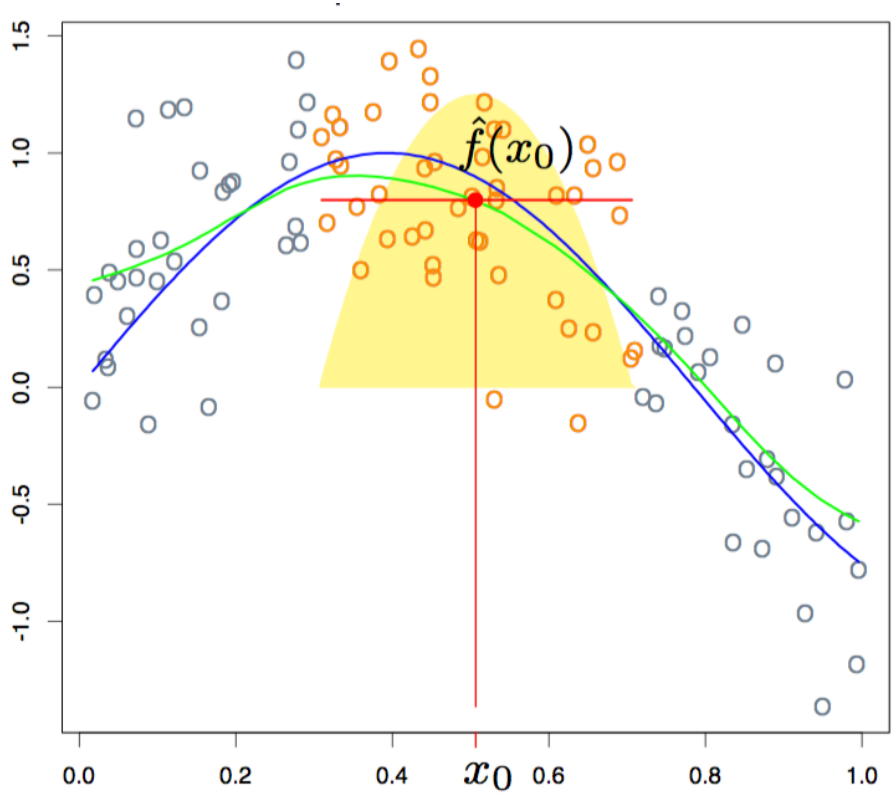

- $k$-nearest neighbor regressor is

$$\hat{f}(x_0) = \frac{\sum_{i=1}^n y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^n \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

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

# Nearest neighbor regression



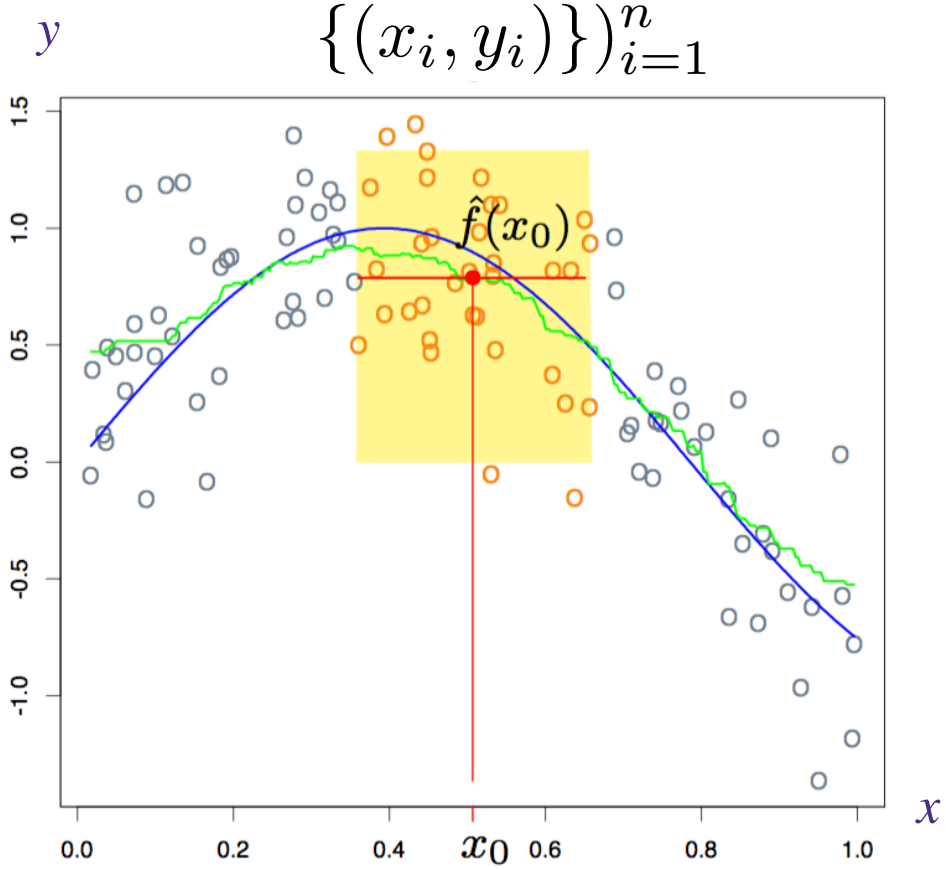



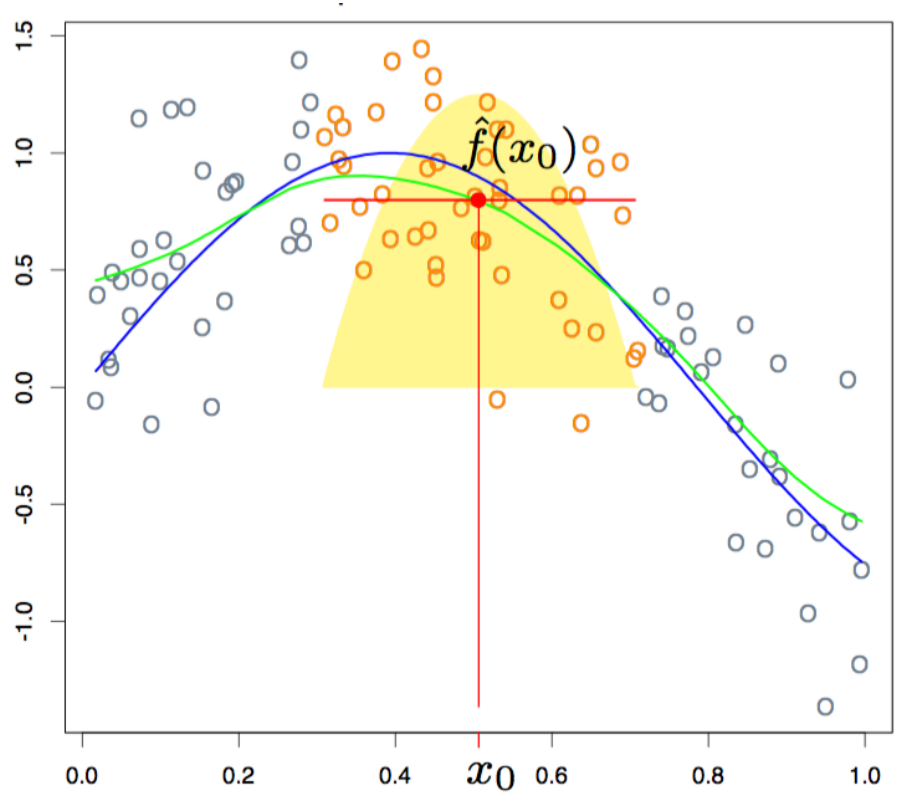

- $k$-nearest neighbor regressor is

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{n} y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^{n} \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

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

# Nearest neighbor regression




Why just average them?

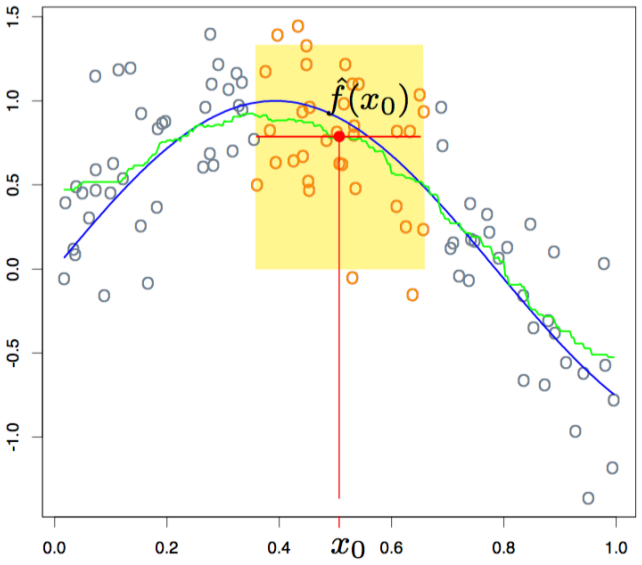
- $k$-nearest neighbor regressor is

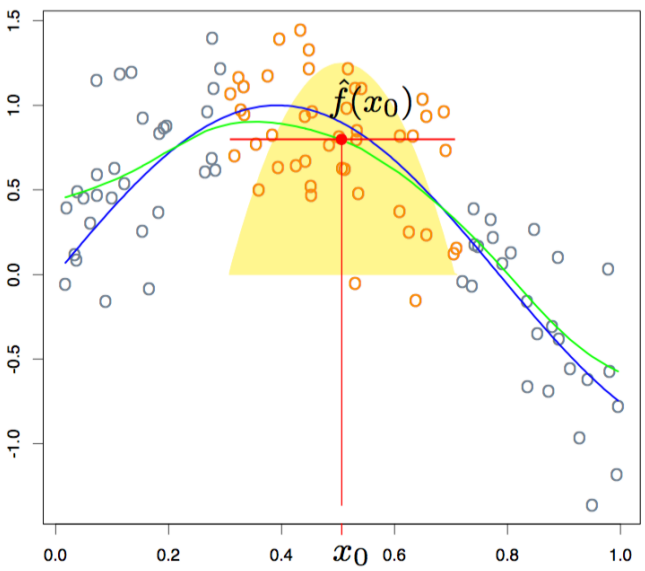
$$\hat{f}(x_0) = \frac{\sum_{i=1}^{n} y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^{n} \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

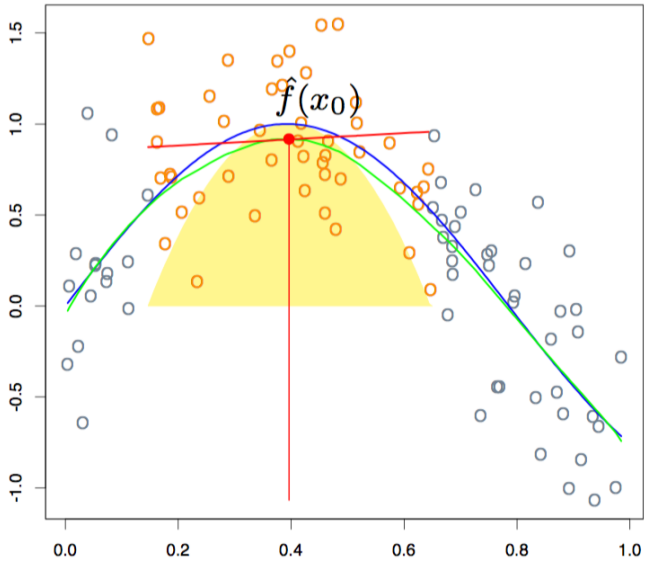
$$\widehat{f}(x_0) = \frac{\sum_{i=1}^{n} K(x_0, x_i) y_i}{\sum_{i=1}^{n} K(x_0, x_i)}$$

# Nearest neighbor regression

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




- $k$-nearest neighbor regressor is

$$\hat{f}(x_0) = \frac{\sum_{i=1}^n y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^n \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

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

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

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

***Local Linear Regression***

# Nearest Neighbor Overview

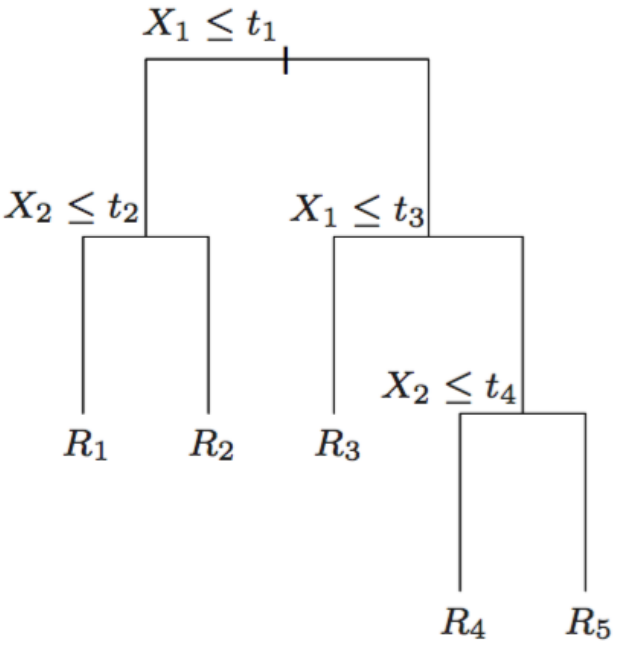
- Very simple to explain and implement
- No training! But finding nearest neighbors in large dataset at test can be computationally demanding (KD-trees help)
- You can use other forms of distance (not just Euclidean)
- Smoothing and local linear regression can improve performance (at the cost of higher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

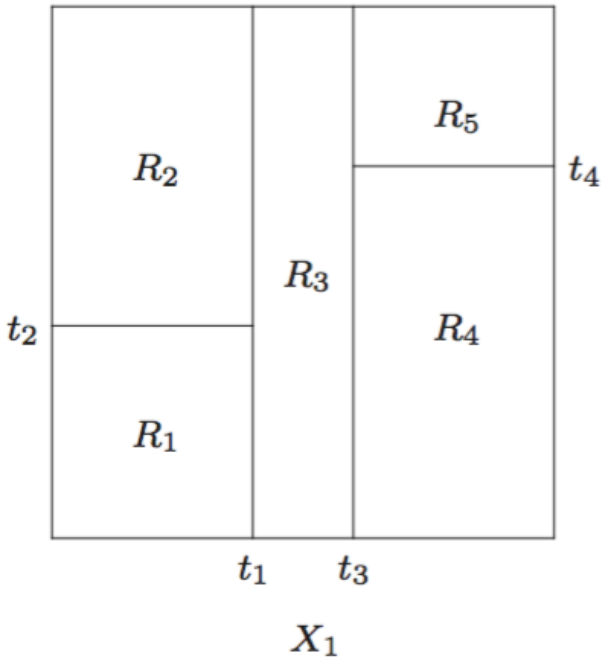
# Questions?

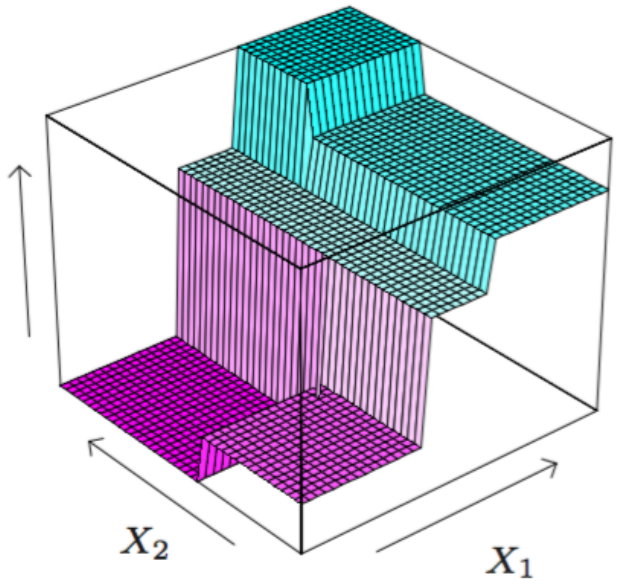
# Trees

# Trees

Example: binary tree with splits along axes




$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

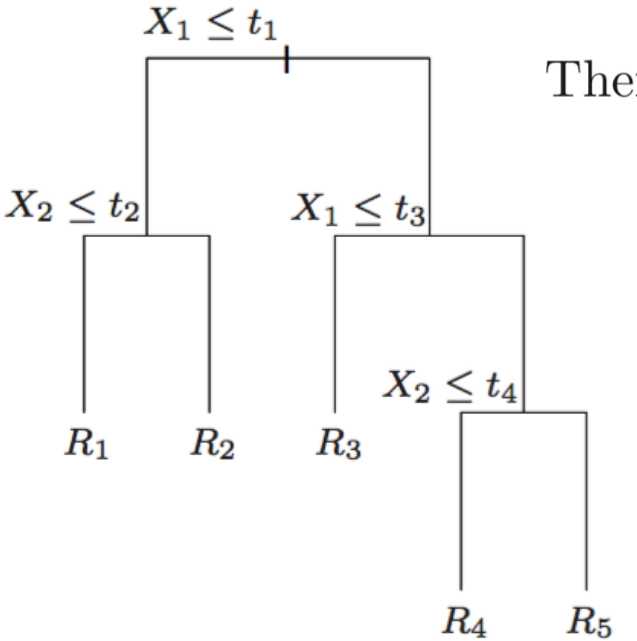
# Regression Trees

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m).$$

Binary tree with splits along axes.

How do you build the tree / find the splits?



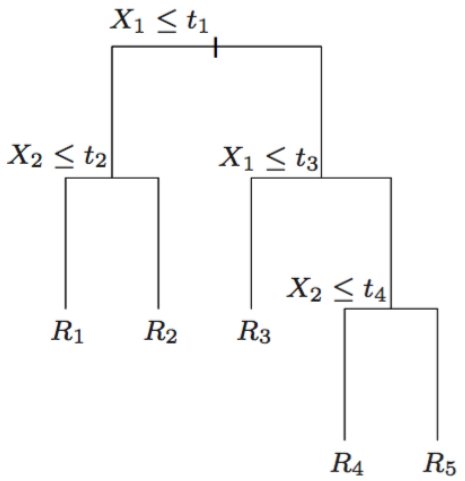

$$R_1(j,s) = \{X | X_j \le s\} \text{ and } R_2(j,s) = \{X | X_j > s\}.$$

Then we seek the splitting variable $j$ and split point $s$ that solve

$$\min_{j,\ s}\left[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right].$$

# Learning decision trees

- Start from empty decision tree
- Split on next best attribute (feature)
  - Use, for example, information gain to select attribute
  - Split on $\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$
- Recurse
- Prune

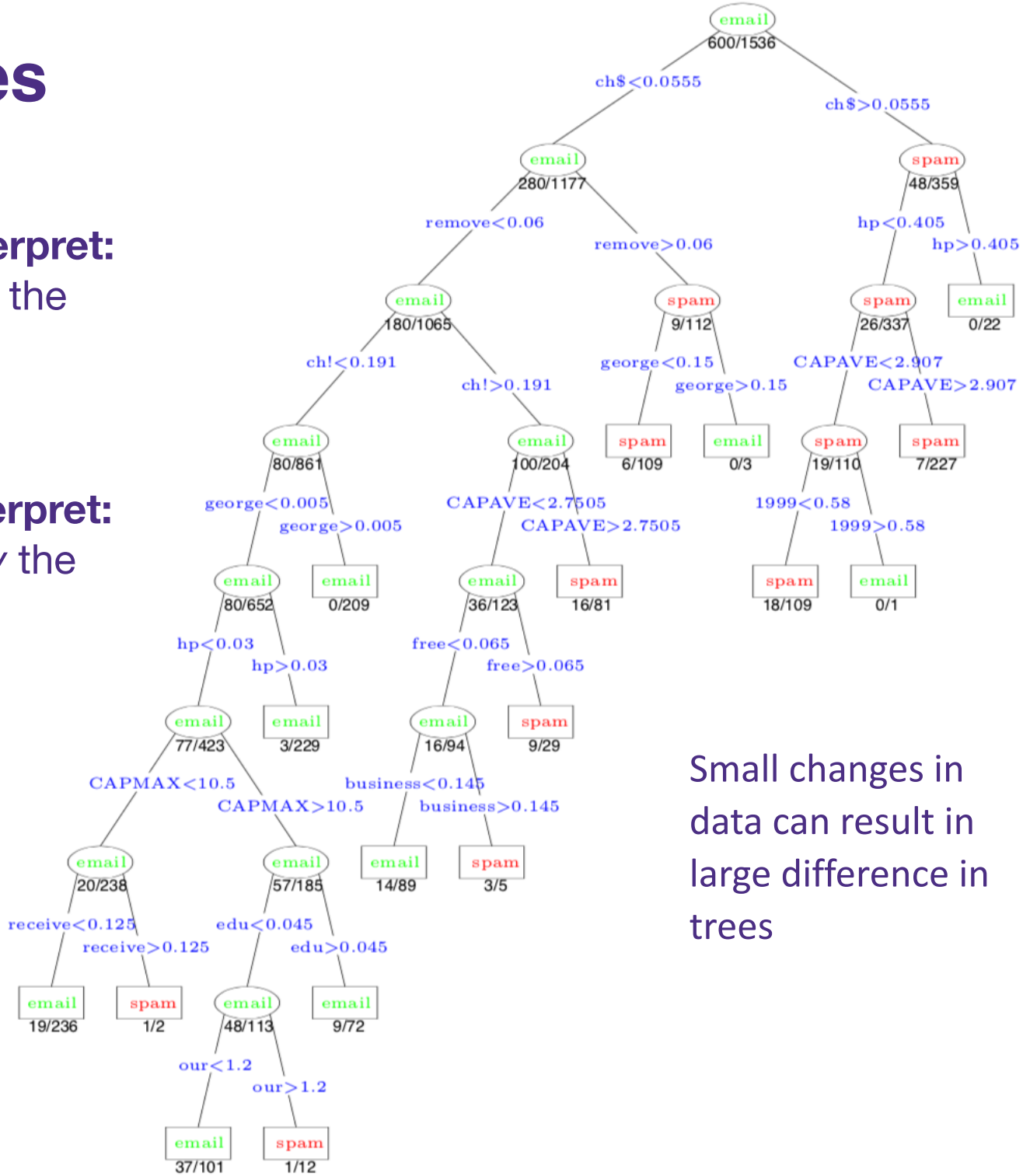



$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

# Decision Trees

**Trees are easy to interpret:**
- You can explain *how* the classifier came to the conclusion it did

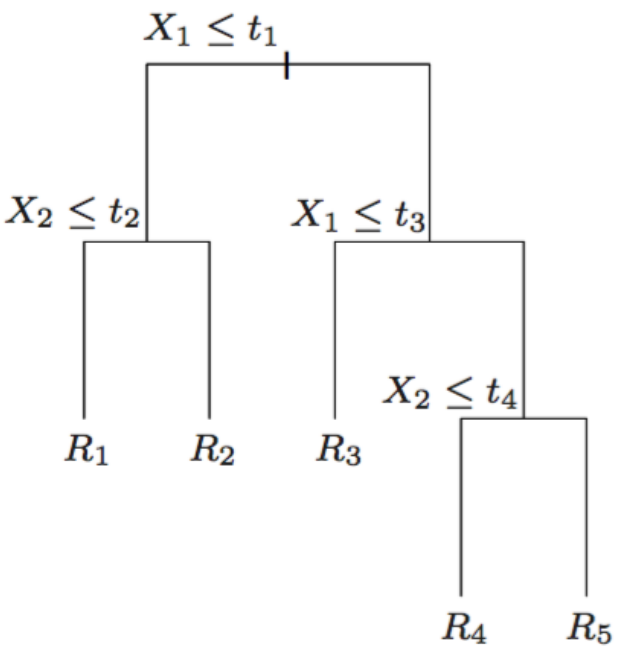
**Trees are hard to interpret:**
- Tough to explain *why* the classifier came to the conclusion it did




Small changes in data can result in large difference in trees

# Trees

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$



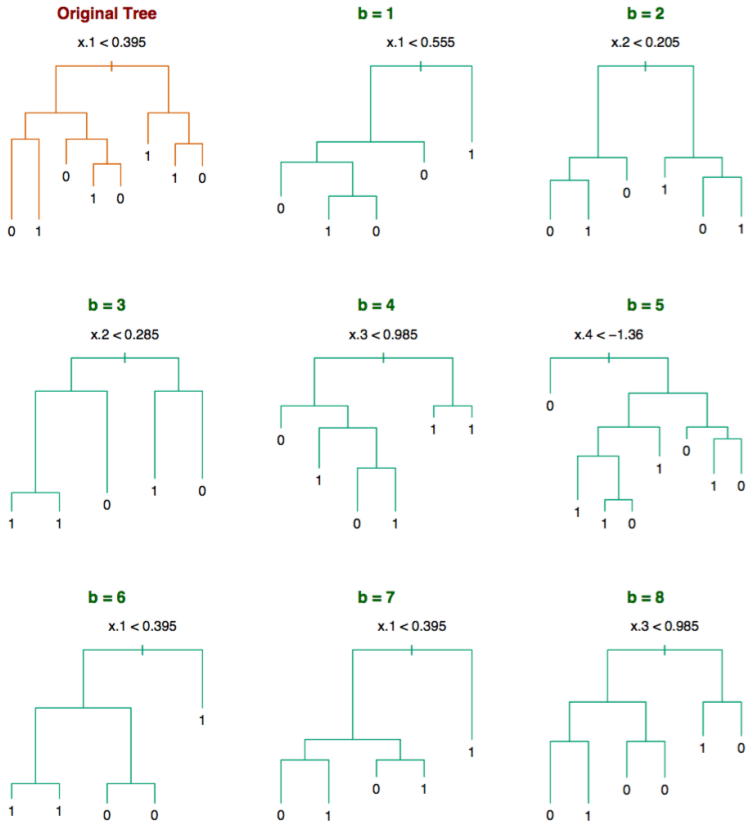

- Trees
  - **have low bias, high variance**
  - deal with categorial variables well
  - intuitive, interpretable
  - good software exists
  - Some theoretical guarantees

# Random Forests

# Random Forests

Tree methods have **low bias** but **high variance**.

One way to reduce variance is to construct a lot of “lightly correlated” trees and average them:




“Bagging:” Bootstrap aggregating

# Random Forests

**Algorithm 15.1** *Random Forest for Regression or Classification.*

1. For $b = 1$ to $B$:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.

   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.

      i. Select $m$ variables at random from the $p$ variables.

      ii. Pick the best variable/split-point among the $m$.

      iii. Split the node into two daughter nodes.

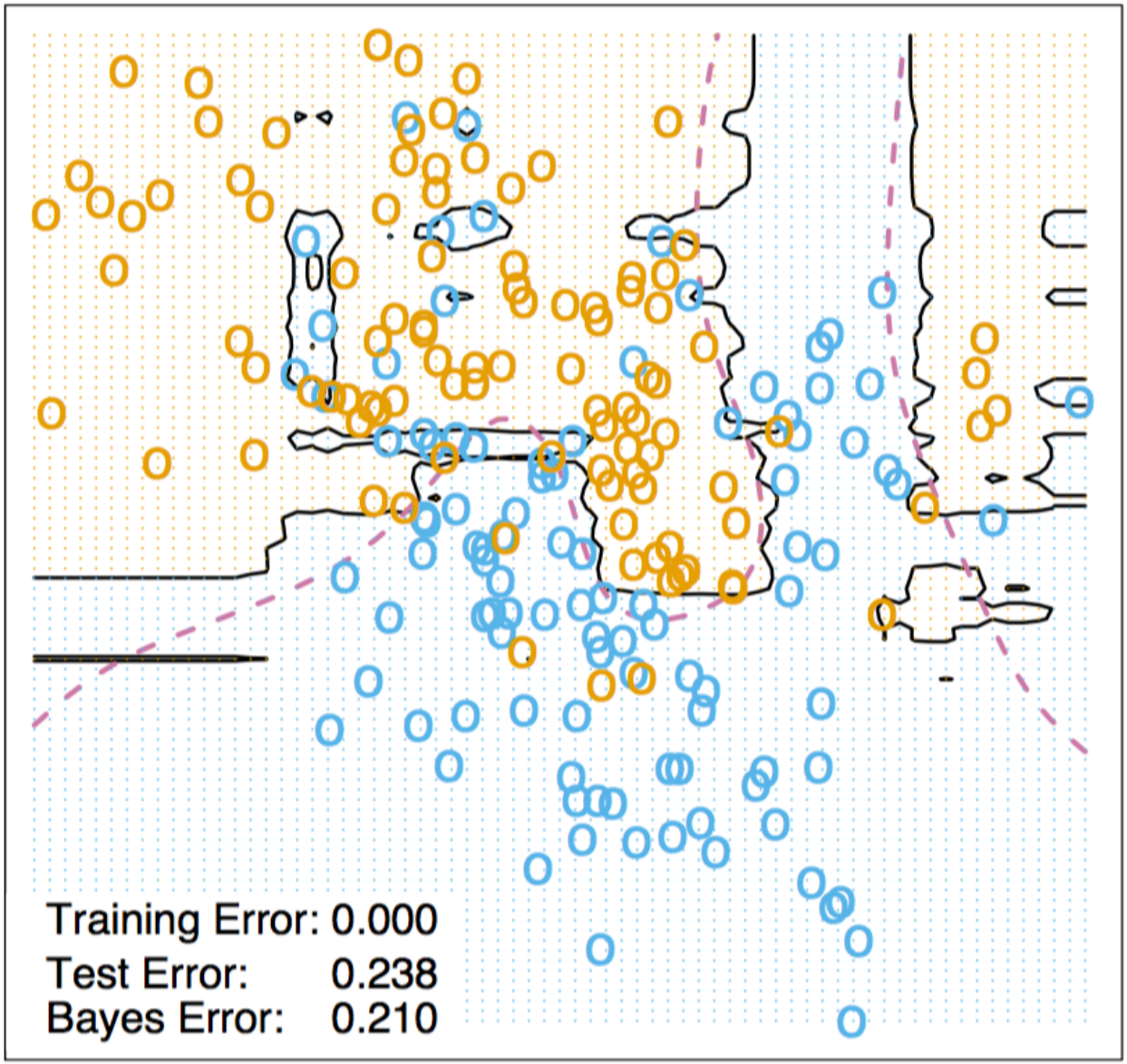
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point $x$:

*Regression:* $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

*Classification:* Let $\hat{C}_b(x)$ be the class prediction of the $b$th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

# Random Forest - Decision Boundary Example



# Random Forest

Given random variables $Y_1, Y_2, \ldots, Y_B$ with
$\mathbb{E}[Y_i] = y$, $\mathbb{E}[(Y_i - y)^2] = \sigma^2$, $\mathbb{E}[(Y_i - y)(Y_j - y)] = \rho\sigma^2$

$\sigma^2$ **Variance of individual predictor**

**Assume bias = 0**

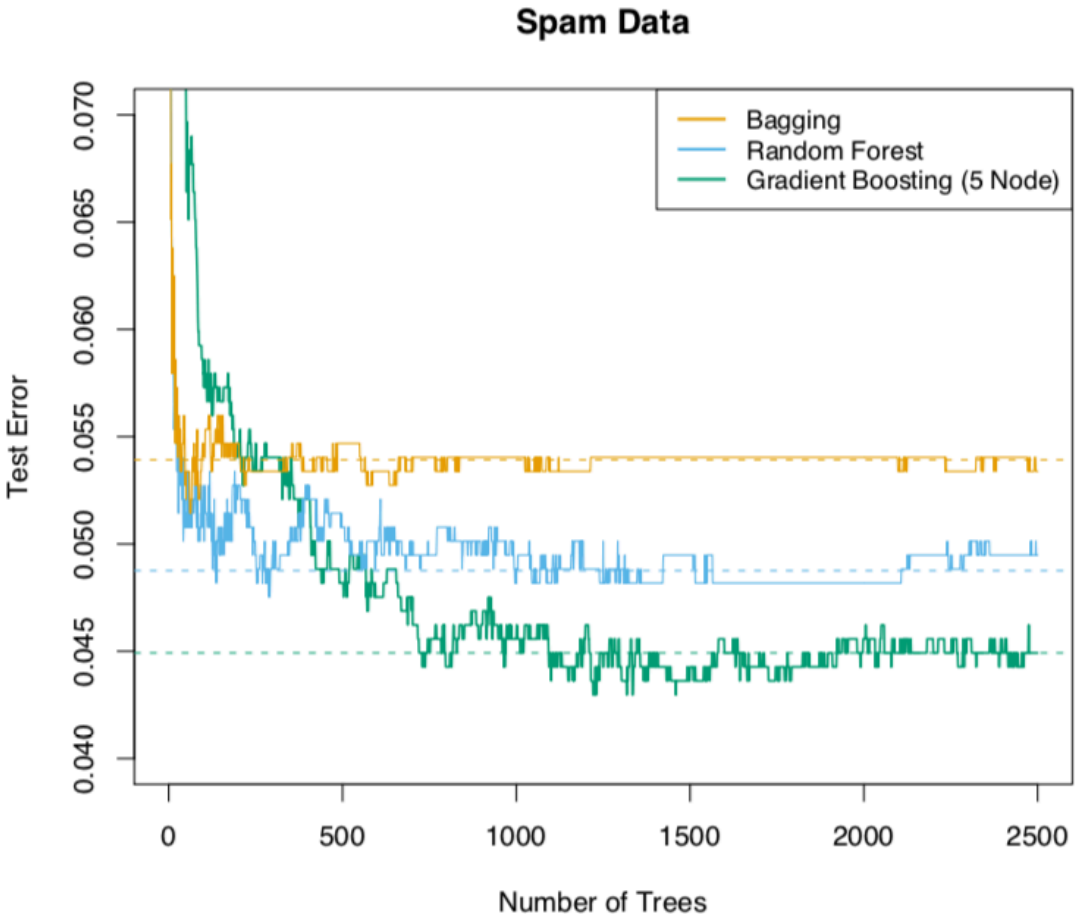
$\rho\sigma^2$ **Correlation between predictors**

The Yi's are identically distributed but **not** independent

$$\mathbb{E}[(\frac{1}{B}\sum_{i=1}^{B} Y_i - y)^2] =$$

# Random Forest

The power of weakly correlated predictors:




Bagging: Averaged trees trained on bootstrapped datasets that used **all d variables**

Random forest: Averaged trees trained on bootstrapped datasets that used **m < d random variables**

Gradient boosting: ignore for now

**Takeaway: reducing correlation improves performance!**

# Random Forests

- Random Forests
  - **have low bias, low variance**
  - deal with categorial variables well
  - not that intuitive or interpretable
  - Notion of confidence estimates
  - good software exists
  - Some theoretical guarantees
  - **works well with default hyperparameters**

# Boosting and Additive Models

# Boosting

- 1988 Kearns and Valiant: “Can **weak learners** be combined to create a **strong learner?**”

Weak learner definition (informal):

An algorithm $\mathcal{A}$ is a *weak learner* for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $\mathcal{A}$ correctly classifies $h$ with error at most $1/2 - \gamma$

- 1990 Robert Schapire: “Yup!”
- 1995 Schapire and Freund: “Practical for 0/1 loss” AdaBoost
- 2001 Friedman: “Practical for arbitrary losses”
- 2014 Tianqi Chen: “Scale it up!” XGBoost

# Additive models

- Given: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$
- Generate **random** functions: $\phi_t : \mathbb{R}^d \to \mathbb{R} \quad t = 1, \ldots, p$
- Learn some weights: $$\widehat{w} = \arg\min_w \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$$
- Classify new data: $$f(x) = \text{sign}\left(\sum_{t=1}^p \widehat{w}_t \phi_t(x)\right)$$

# Additive models

- Given: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$
- Generate **random** functions: $\phi_t : \mathbb{R}^d \to \mathbb{R} \quad t = 1, \ldots, p$
- Learn some weights: $$\widehat{w} = \arg\min_w \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$$
- Classify new data: $$f(x) = \text{sign}\left(\sum_{t=1}^p \widehat{w}_t \phi_t(x)\right)$$

An interpretation:
Each $\phi_t(x)$ is a classification rule that we are assigning some weight $\widehat{w}_t$

$$\widehat{w}, \widehat{\phi}_1, \ldots, \widehat{\phi}_t = \arg \min_{w, \phi_1, \ldots, \phi_p} \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$$

is in general computationally hard

# Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters $\gamma$

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \le \gamma_2\}$

**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

# Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters $\gamma$

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \le \gamma_2\}$

**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

**AdaBoost:** $b(x, \gamma)$: classifiers to $\{-1, 1\}$

$$L(y, f(x)) = \exp(-y f(x))$$

# Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters $\gamma$

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$

**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

**Boosted Regression Trees:** $L(y, f(x)) = (y - f(x))^2$

$b(x, \gamma)$: regression trees

# Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters $\gamma$ Examples: $b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$

**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

**Boosted Regression Trees:** $L(y, f(x)) = (y - f(x))^2$

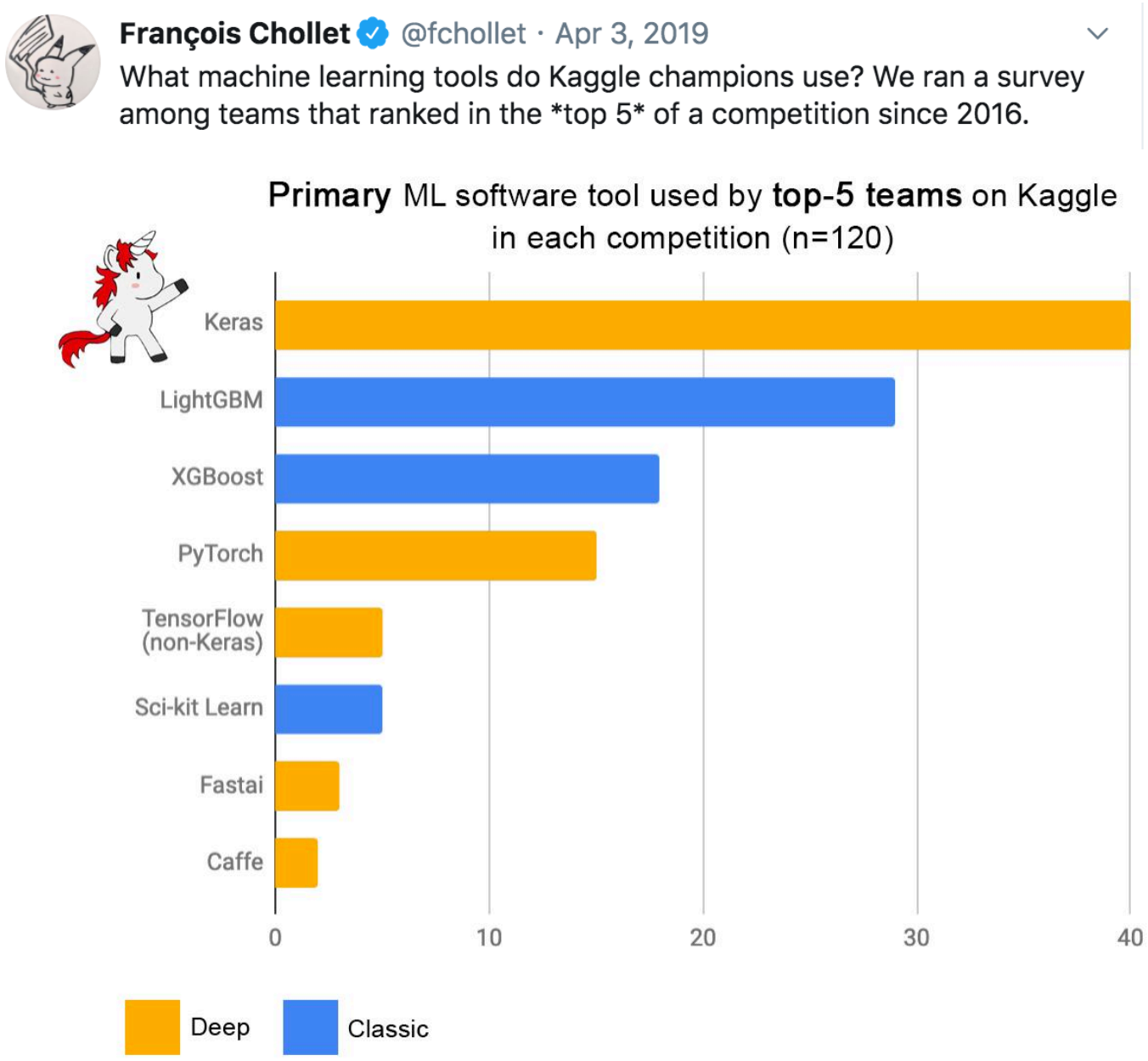
$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \qquad r_{im} = y_i - f_{m-1}(x_i) \end{aligned}$$

Efficient: No harder than learning regression trees!

# Additive models

- Boosting is popular at parties: Invented by theorists, heavily adopted by practitioners.
- Computationally efficient with “weak” learners. But can also use trees! Boosting can scale.
- Gradient boosting generalization with good software packages (e.g., *XGBoost)*. Effective on Kaggle

# Additive models



# Bagging versus Boosting

- Bagging *averages* many **low-bias**, **lightly dependent** classifiers to reduce the variance
- Boosting *learns* linear combination of **high-bias**, **highly dependent** classifiers to reduce error

# Last slide of the quarter!