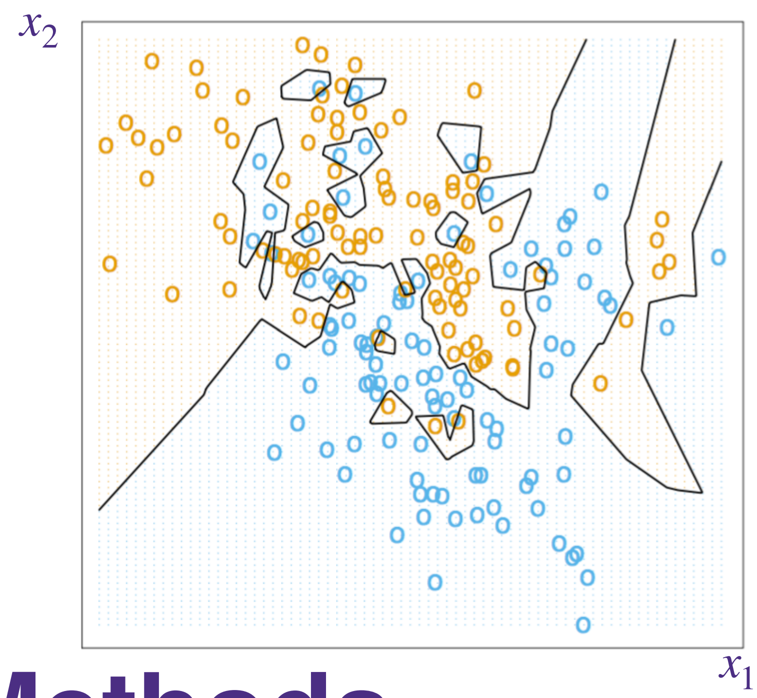
- Homework 3, due Saturday, February 26 midnight



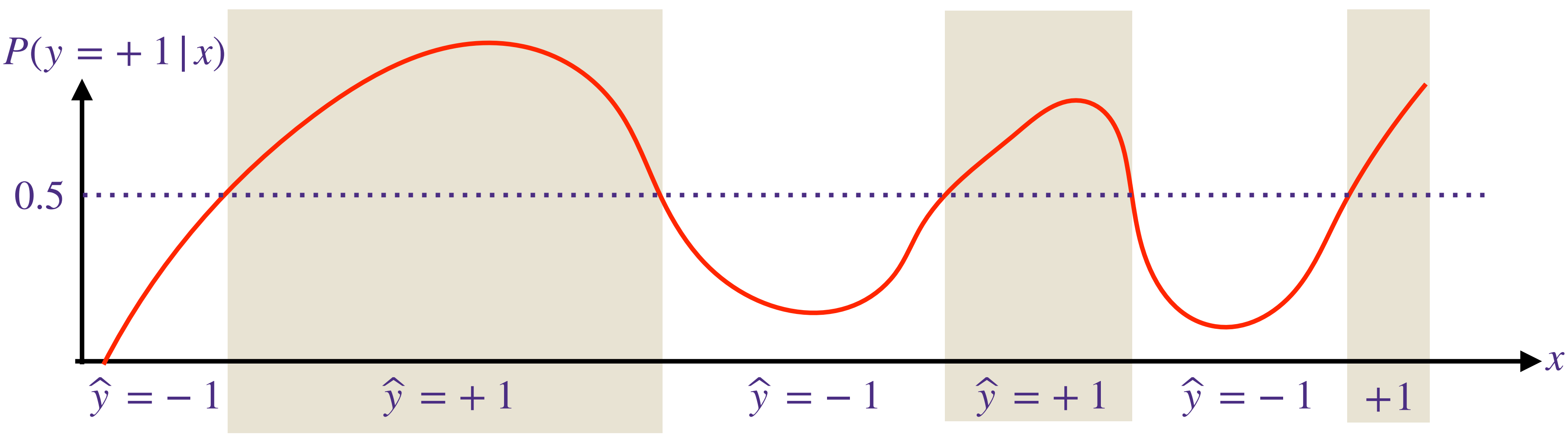
Lecture 21:
Nearest Neighbor Methods

- 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

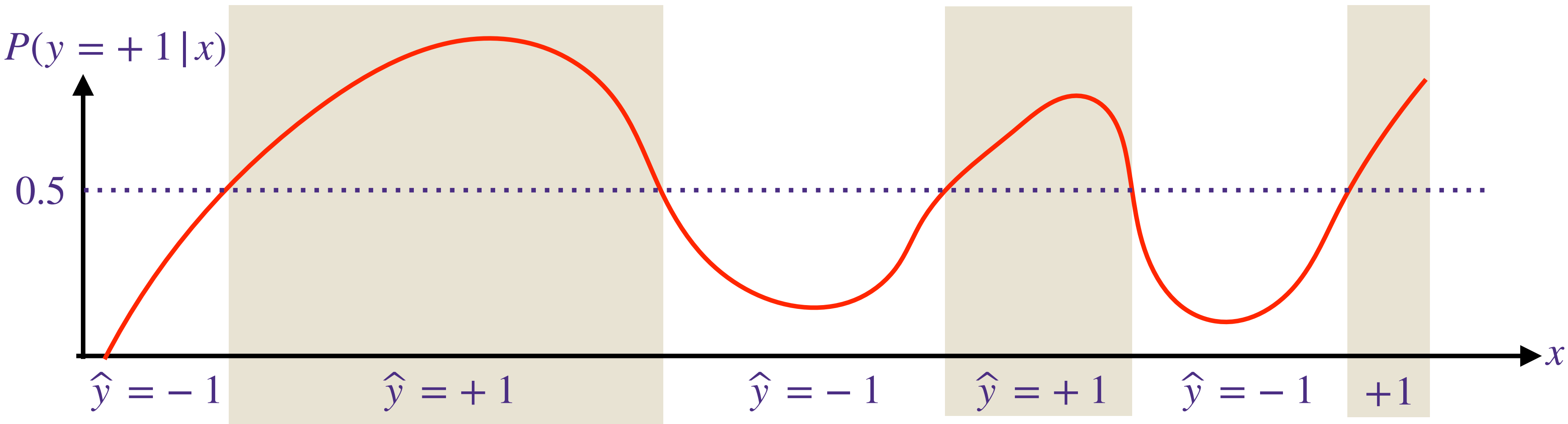
Recall Bayes optimal classifier

- 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 $P(\hat{y} \neq y \mid x)$ for every x, which can be written explicitly as

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



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

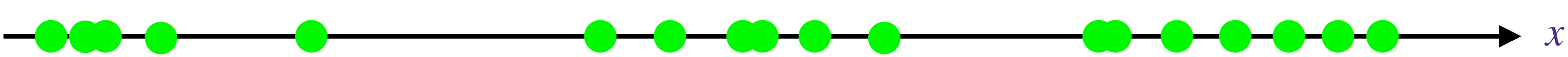


- Bayes optimal classifier $\hat{y} = \begin{cases} +1 & \text{if } P(+1 \mid x) > P(-1 \mid x) \\ -1 & \text{if } P(+1 \mid x) < P(-1 \mid x) \end{cases}$

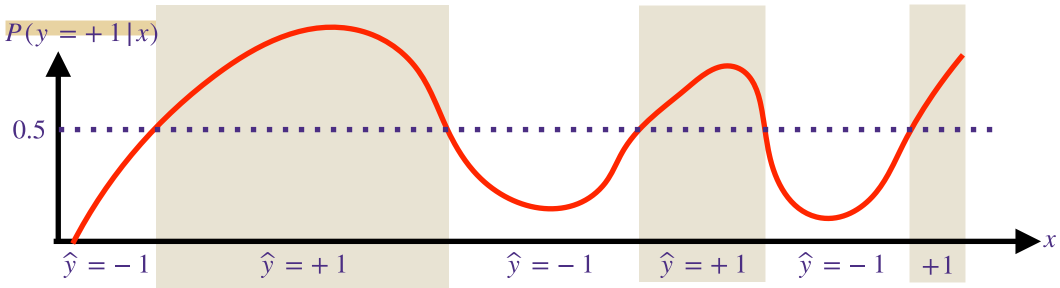
- How do we compare $P(y = +1 \mid x)$ and $P(y = -1 \mid x)$ from samples?

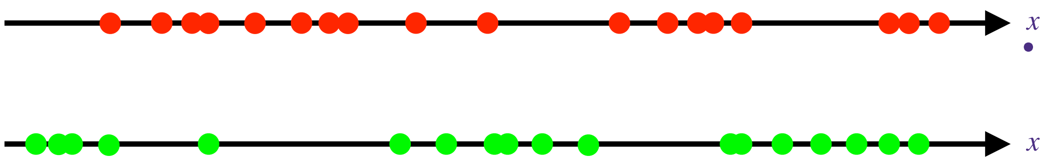
samples with $y = +1$

samples with $y = -1$



One way to approximate Bayes Classifier = local statistics





- Bayes optimal classifier

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

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

$$\hat{y} = \begin{cases} +1, & \text{if (\# of +1 samples)} > \text{(\# of -1 samples)} \\ -1, & \text{if (\# of +1 samples)} < \text{(\# of -1 samples)} \end{cases}$$

- 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

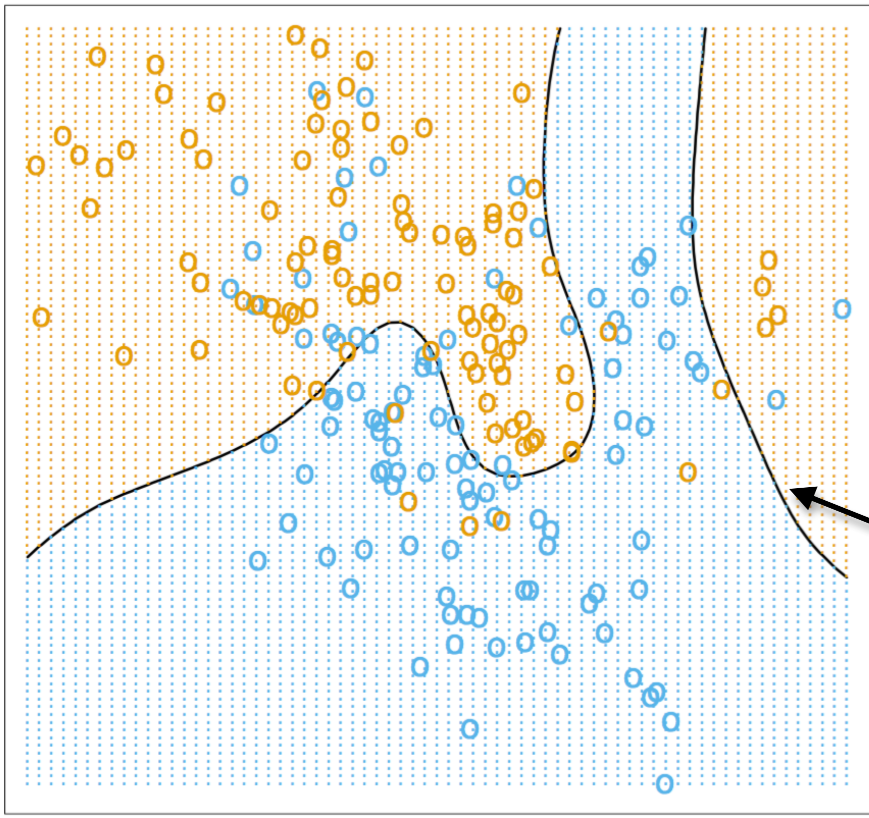
$$\frac{n_r^+}{n} \longrightarrow 2r \times P(x, y=+1)$$

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

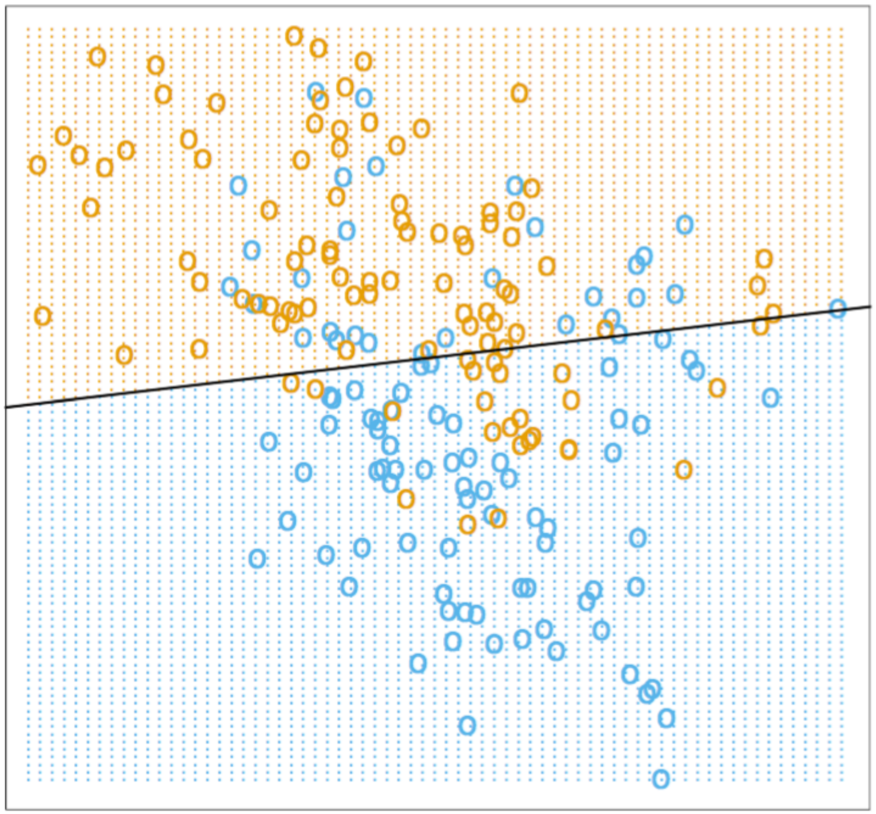
Optimal "Bayes" classifier:

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

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

Figures from Hastie et al.

Linear Decision Boundary



Training data:

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

Learned:

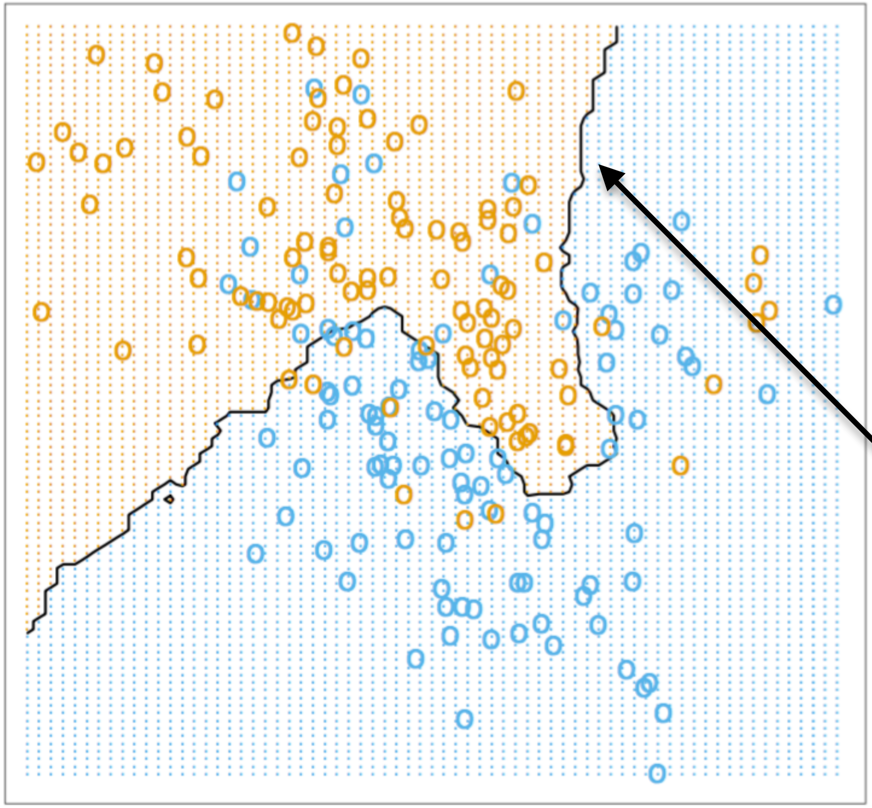
Linear Decision boundary

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

- 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)

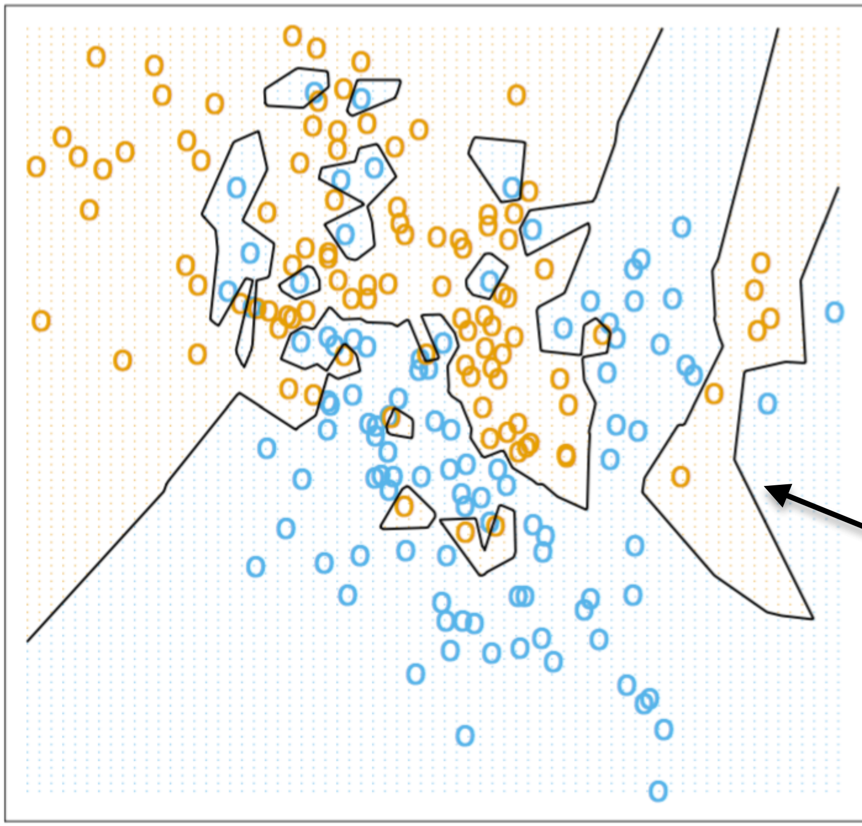

Predicted label: +1


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

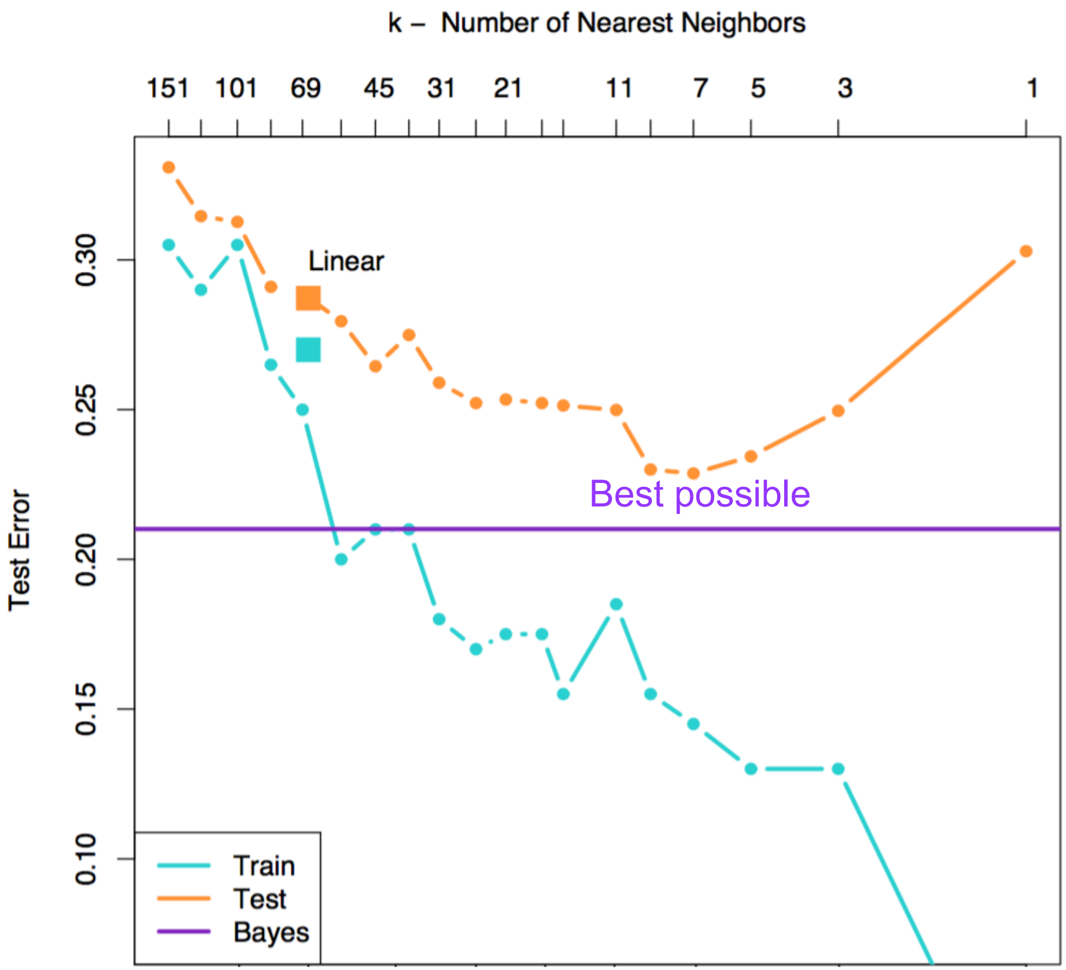
- With a small k, we tend to overfit.

Figures from Hastie et al

k-Nearest Neighbor Error

Model complexity low

Model complexity high



k Bias-Variance tradeoff

As k->infinity?

Bias:

Variance:

As k->1?

Bias:

Variance:

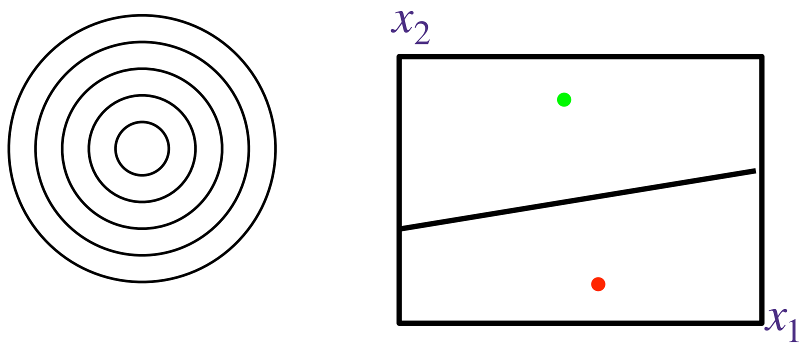
Figures from Hastie et al

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

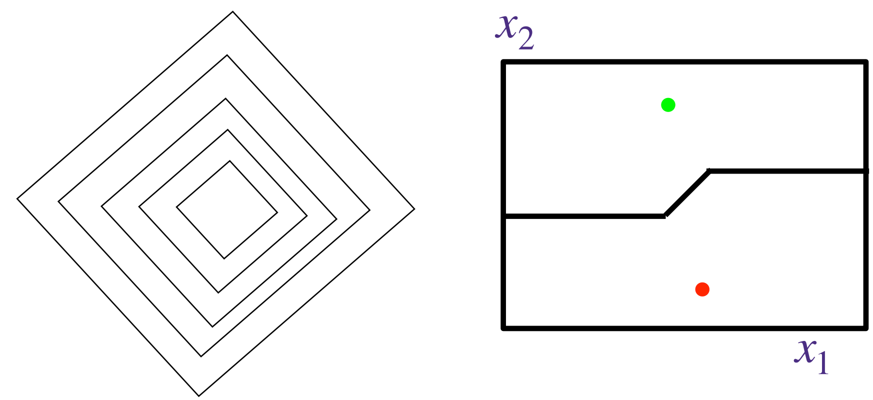
Notable distance metrics (and their level sets)

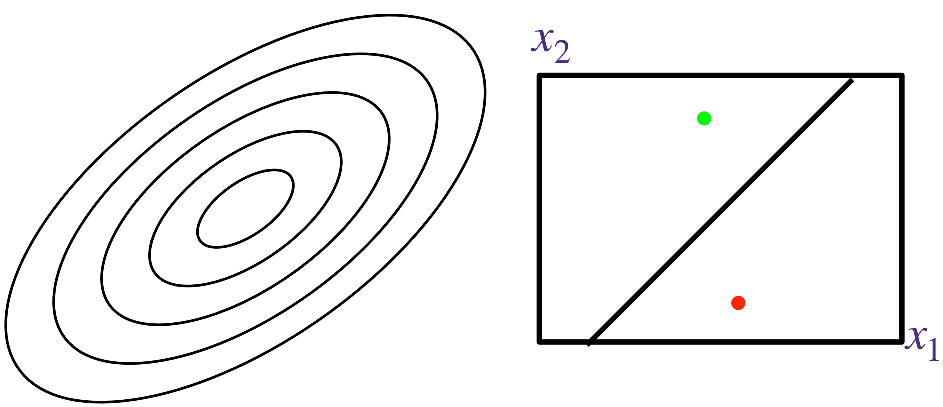
Consider 2 dimensional example with 2 data points with labels green, red,
and we show $k = 1$ nearest neighbor decision boundaries for various choices of distances

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

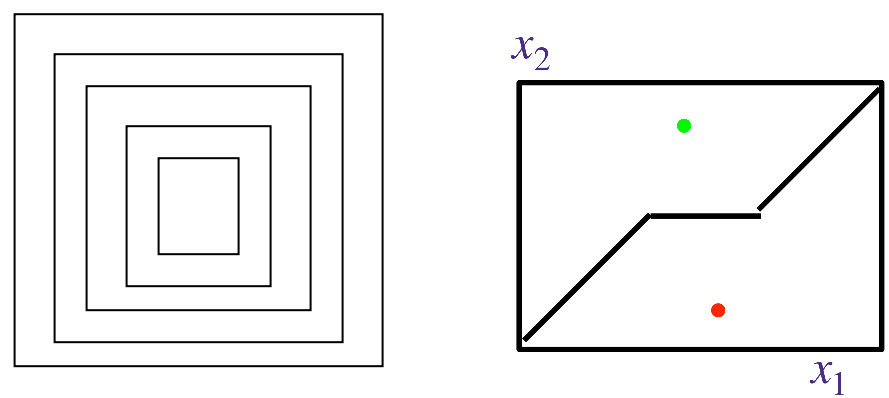


L_1 norm (taxi-cab)





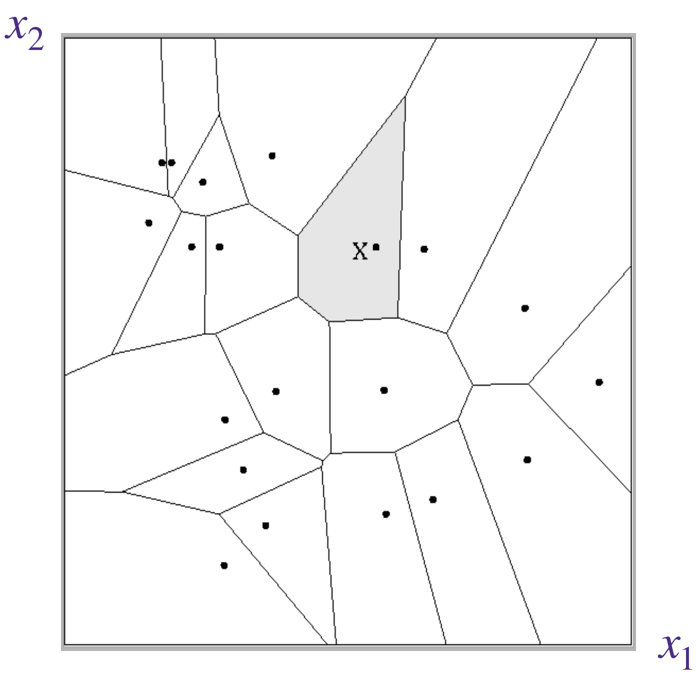
Mahalanobis norm: $d(x, y) = (x - y)^T M (x - y)$



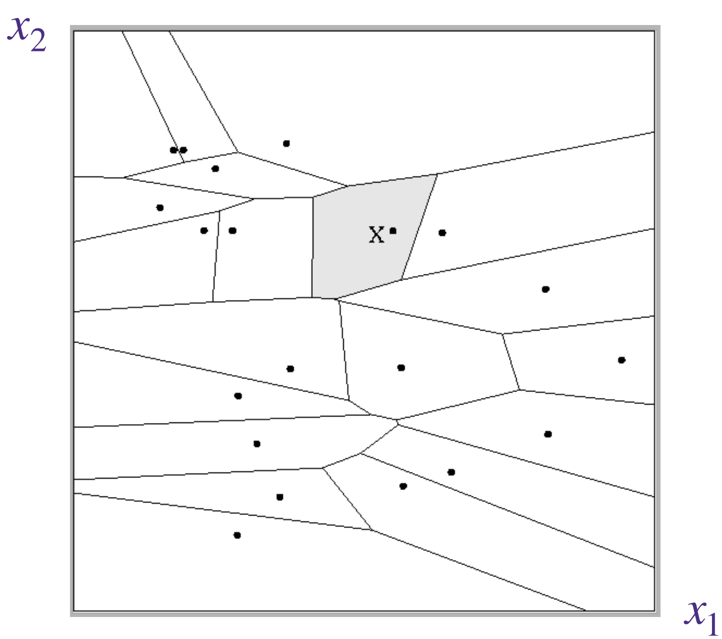
L-infinity *(max) norm*

$k = 1$ 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

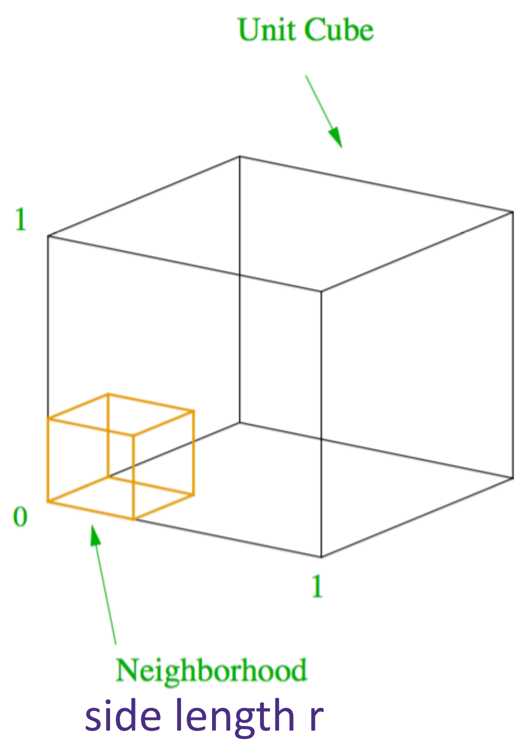
$$\{(x_i, y_i)\}_{i=1}^{n} \qquad x_i \in \mathbb{R}^d, \quad y_i \in \{0,1\} \qquad (x_i, y_i) \overset{iid}{\sim} P_{XY}$$

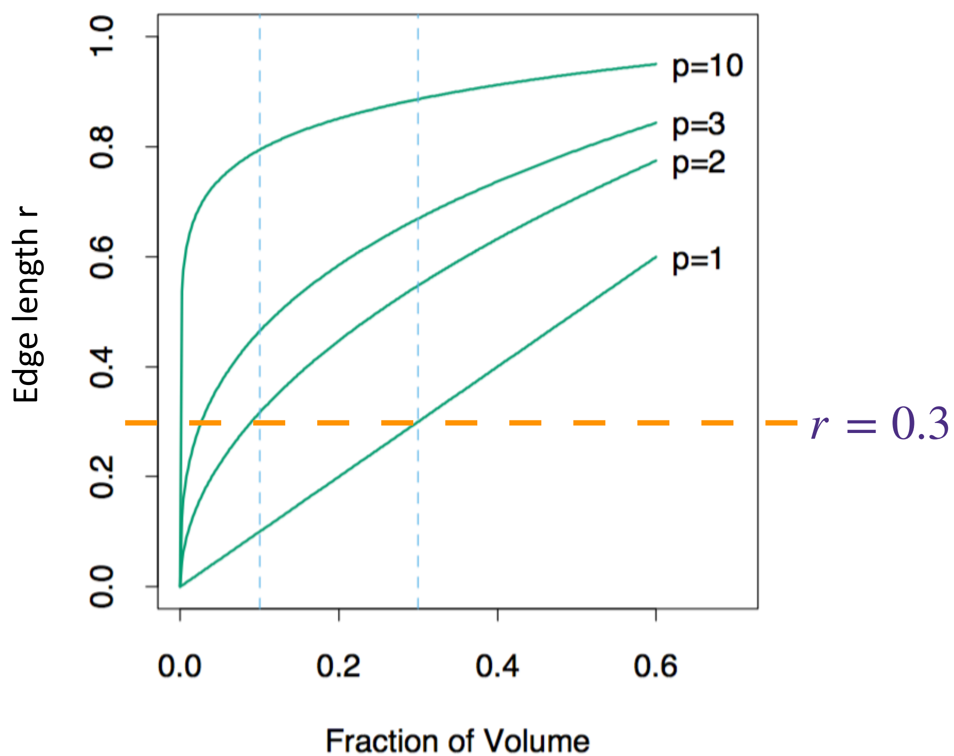
Theorem[Cover, Hart, 1967] If P_X is supported everywhere in $\mathbb{R}^d$ and $P(Y = 1|X = x)$ is smooth everywhere, then as $n \to \infty$ the 1-NN classification rule has error at most twice the Bayes error rate.

- 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



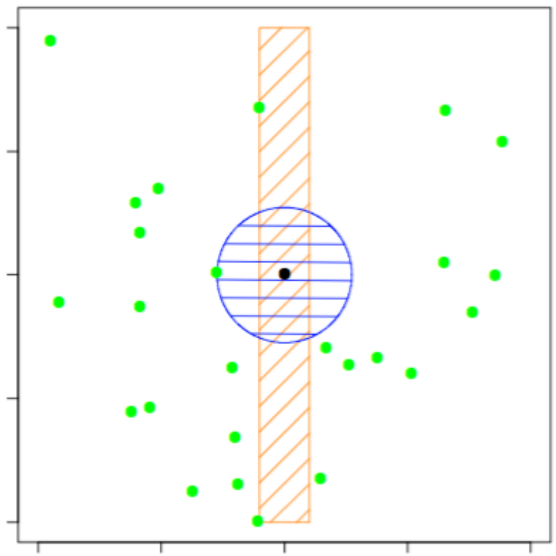


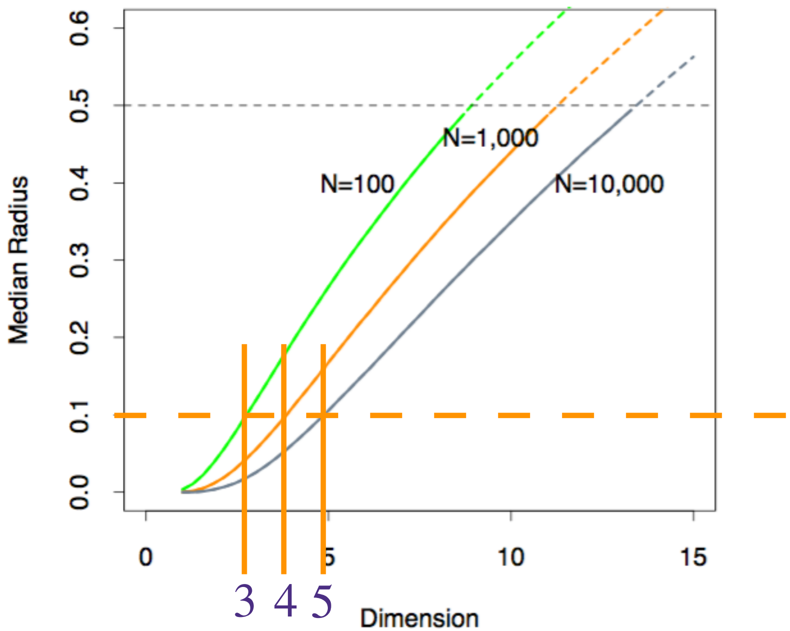
X is uniformly distributed over $[0,1]^p$. What is $\mathbb{P}(X \in [0,r]^p)$?

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

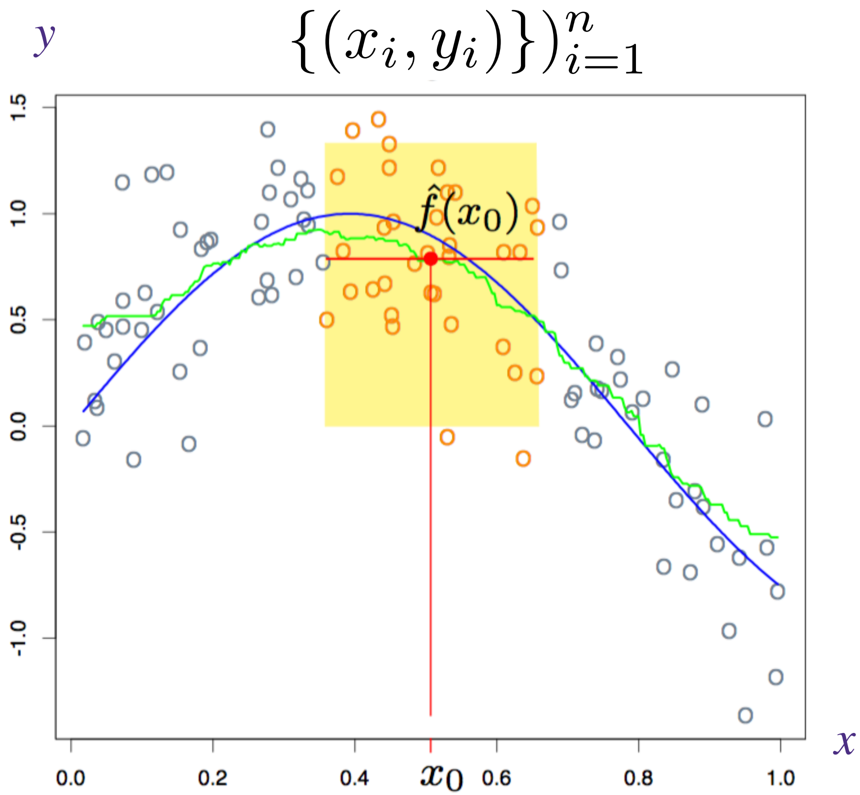




What is the median distance from a point at origin to its 1NN?

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 $\mathbb{E}[(\hat{y} - y)^2]$?

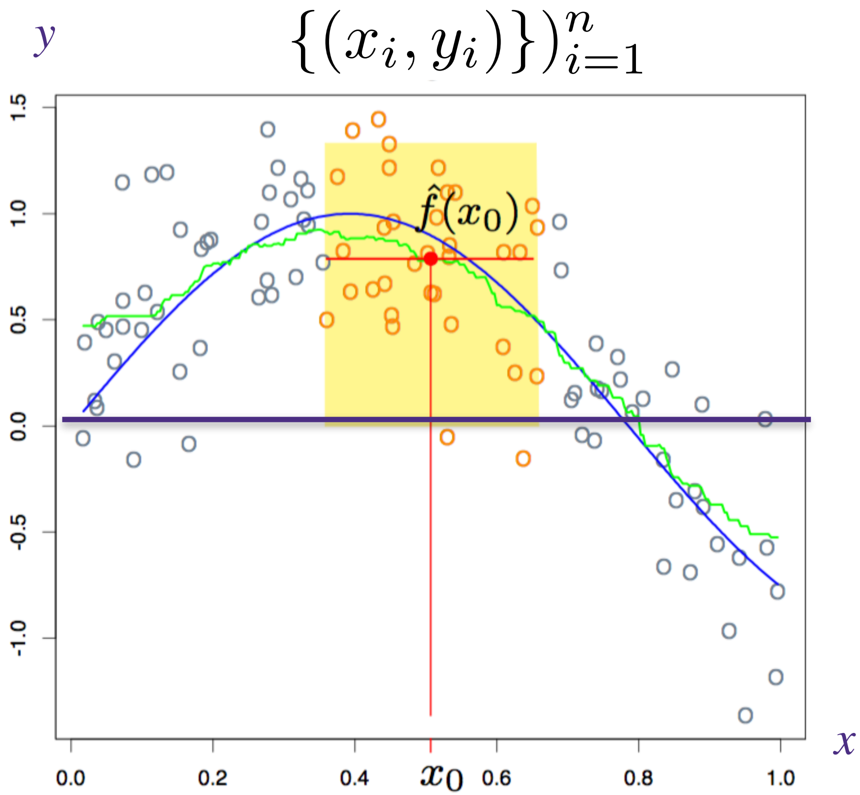
$$\hat{y} = \mathbb{E}[y \mid x]$$

- k-nearest neighbor regressor is

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

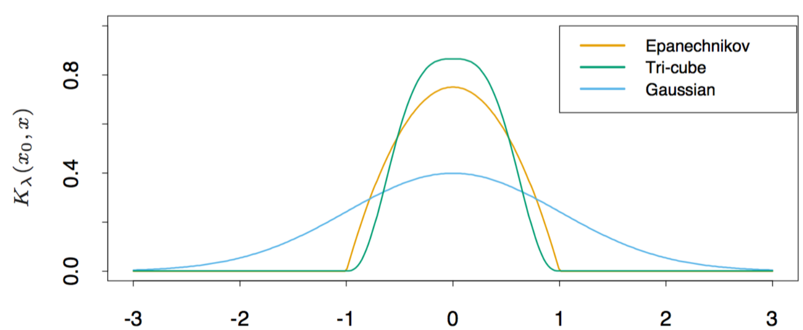
$$= \frac{\sum_{i=1}^n y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^n \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

Nearest neighbor regression



In nearest neighbor methods, the "weight" changes abruptly

smoothing: $K(x,y)$

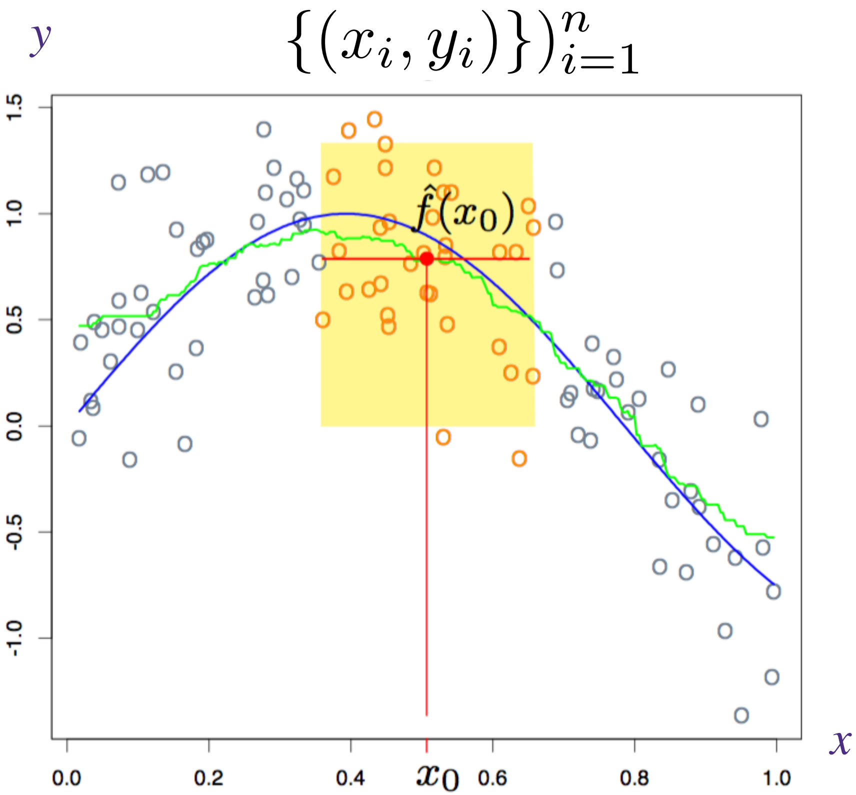


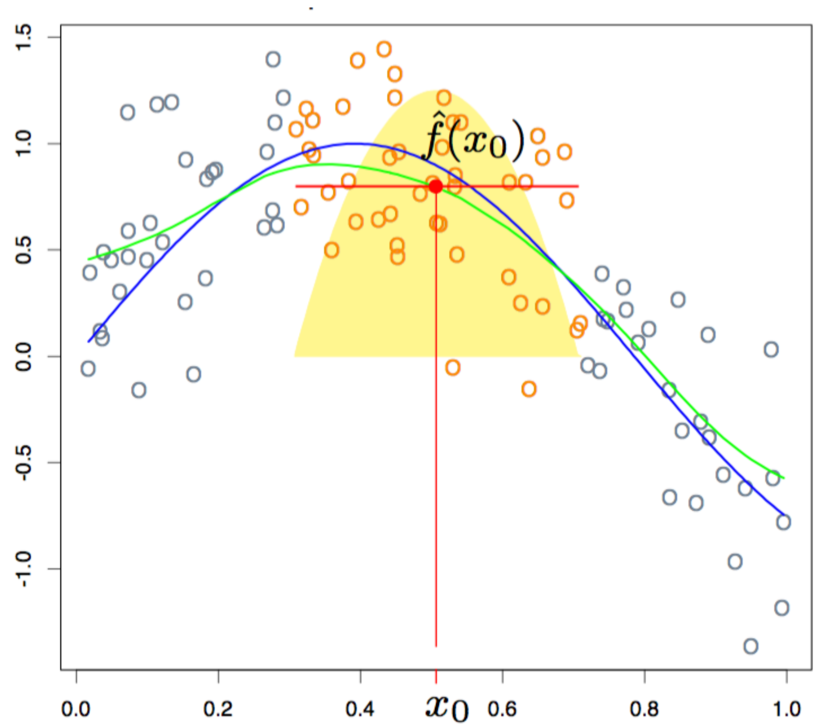
- k-nearest neighbor regressor is

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

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

Nearest neighbor regression



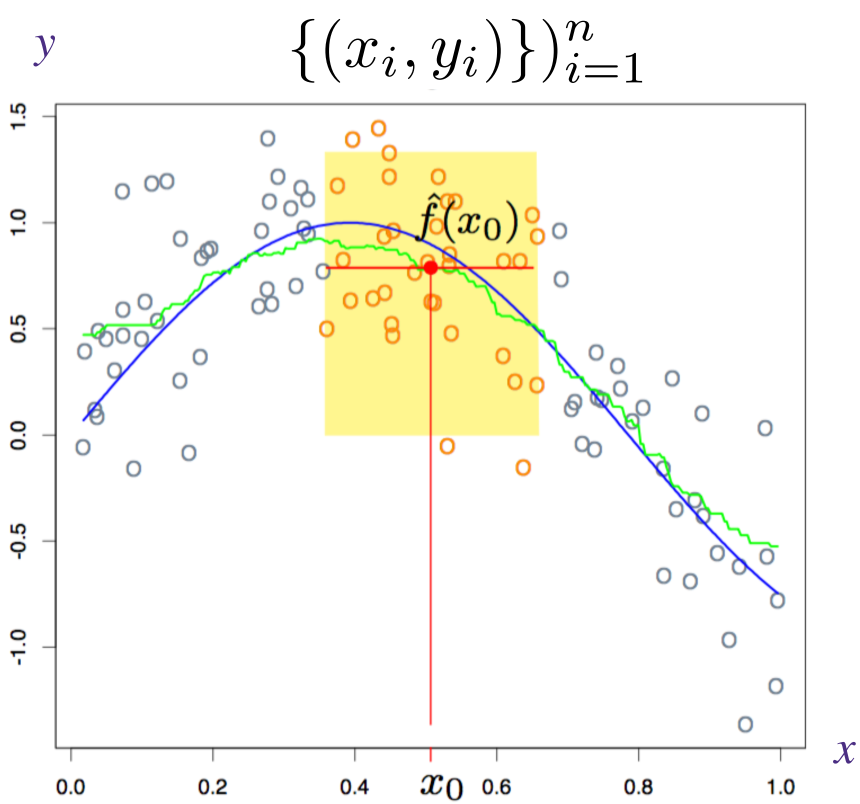


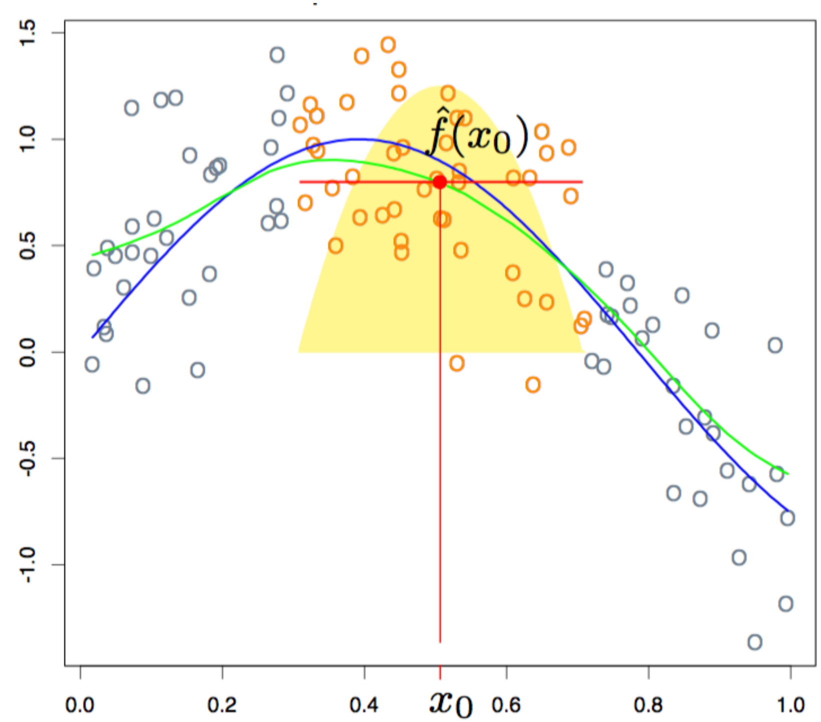
- k-nearest neighbor regressor is

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

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

Nearest neighbor regression





Why just average them?

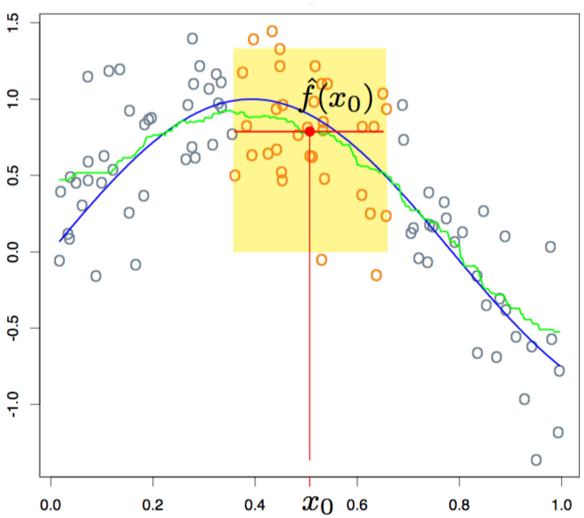
- k-nearest neighbor regressor is

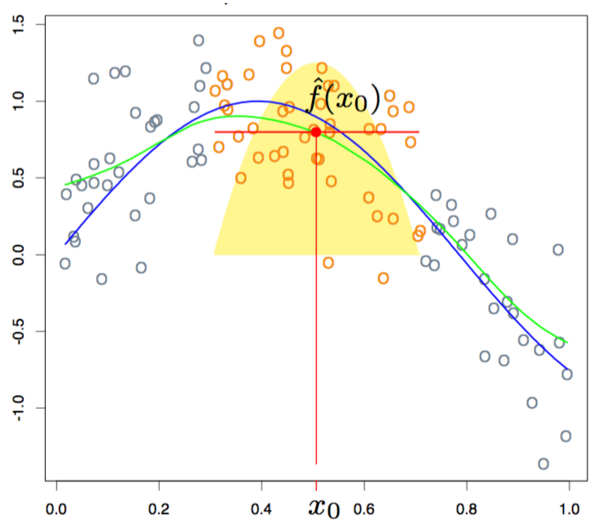
$$\hat{f}(x_0) = \frac{\sum_{i=1}^n y_i \times \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}{\sum_{i=1}^n \text{Ind}(x_i \text{ is a } k \text{ nearest neighbor})}$$

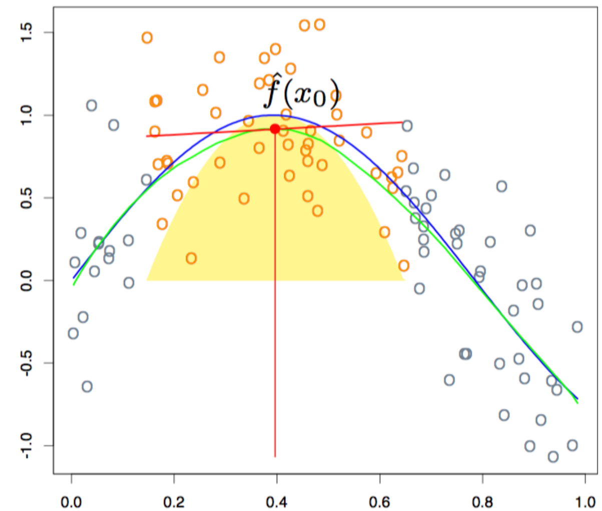
$$\widehat{f}(x_0) = \frac{\sum_{i=1}^n K(x_0, x_i) y_i}{\sum_{i=1}^n K(x_0, x_i)}$$

Nearest neighbor regression

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







- k-nearest neighbor regressor is

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

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

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

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

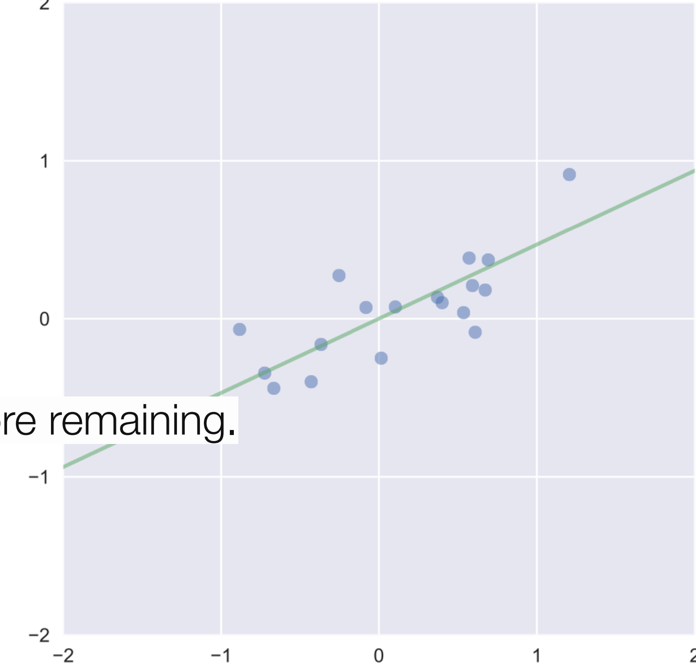
Local Linear Regression

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?

- Homework 3, due Sunday, February 27 midnight
- We will add more office hours on Saturday and Sunday
- Schedule on Canvas (and more coming)
    - Thai Hoang Saturday 9-10 AM
    - Hugh Sun Saturday 1:30-2:30 PM
    - Sewoong Oh Sunday 10-11 AM
- Homework 4, due Sunday, March 13th Midnight
- You are allowed only 3 late days for HW4 even if you have more remaining.



Lecture 22: Principal Component Analysis

- Supervised Learning with labelled data $\{(x_i, y_i)\}_{i=1}^n$
    - Goal: fit a function to predict y
    - Regression/Classification
    - Linear models / Kernels / Nearest Neighbor / Neural Networks
- **Unsupervised Learning** with unlabelled data $\{x_i\}_{i=1}^n$
    - Goal: find pattern in clouds of data $\{x_i\}_{i=1}^n$
    - Principal Component Analysis
    - Clustering

Motivation: dimensionality reduction

- it takes $n \times d$ memory to store data $\{x_i\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$
- but many real data have patterns that repeat over samples
- Can we exploit this redundancy? Can we find some patterns and use them?
- Can we represent each image compactly,
  but still preserve most of information, by exploiting similarities?

d=32x32pixels per image

n images

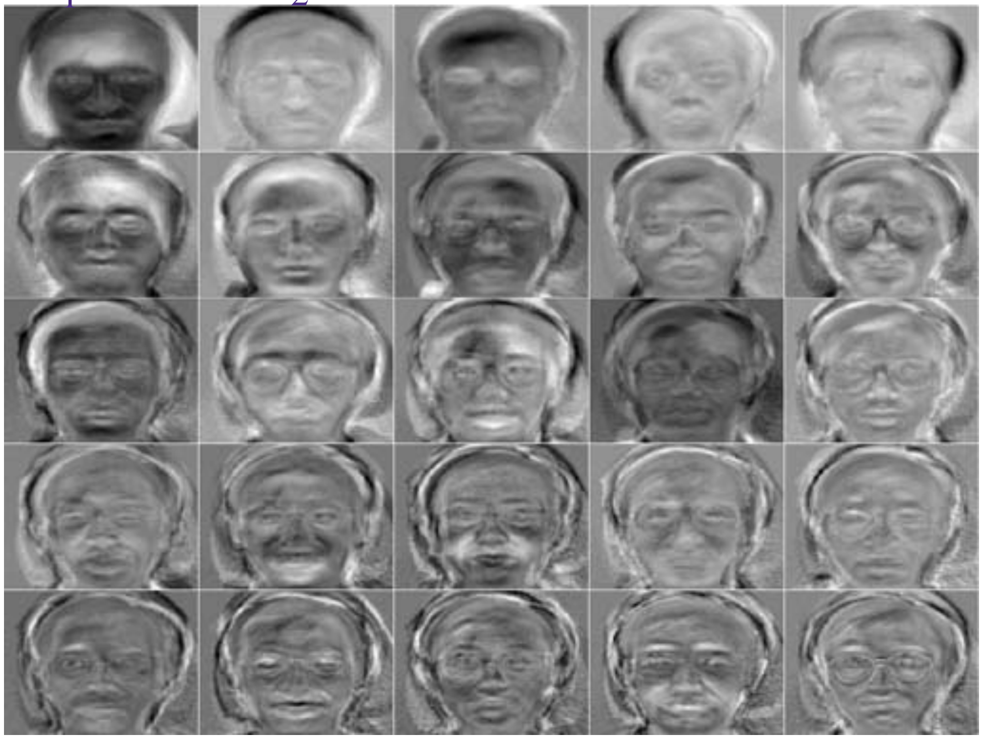
$d \times n$ real values to store the data

Principal component analysis finds a compact linear representation

- patterns that capture the distinct features of the samples is called **principal component** (to be formally defined later)
- we use $r = 25$ principal components

Principal components:

$u_1 \in \mathbb{R}^d$ u_2

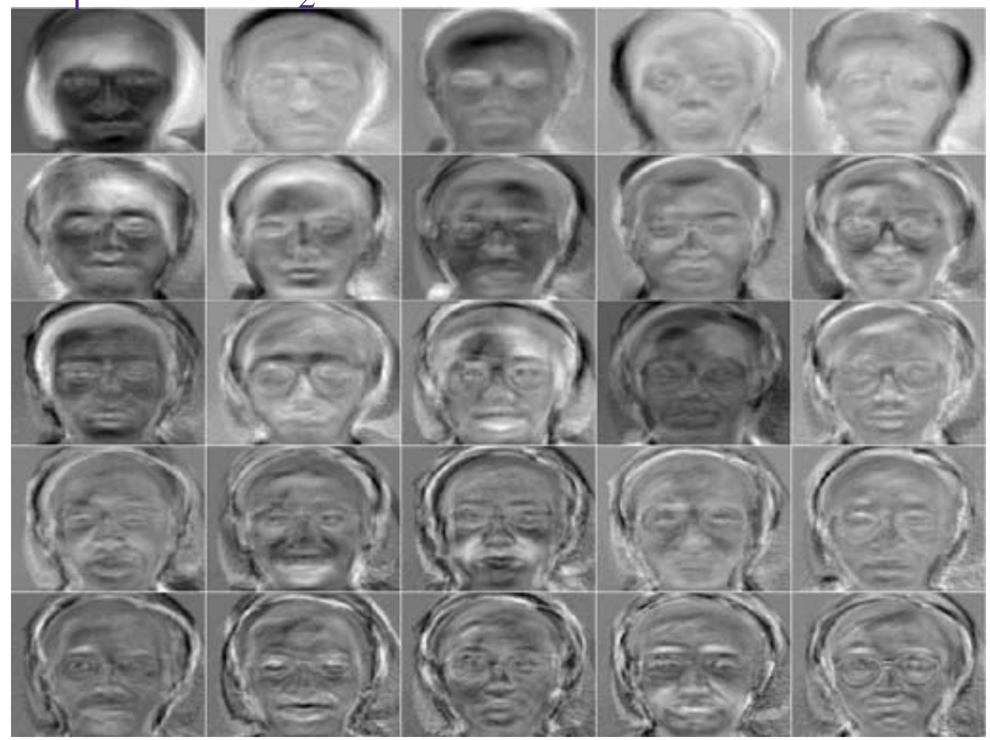


Principal component analysis finds a compact linear representation

- patterns that capture the distinct features of the samples is called **principal component** (to be formally defined later)
- we use $r = 25$ principal components
- we can represent each sample as a **weighted linear combination** of the principal components, and just store the weights (as opposed to all pixel values)

Principal components:

$u_1 \in \mathbb{R}^d$ $\quad u_2$



$$\approx \; a[1]u_1 + a[2]u_2 + \cdots + a[25]u_{25}$$

- Each image is now represented by $r = 25$ numbers $a = (a[1], \ldots, a[25])$
- To store n images, it requires memory of only $d \times r + r \times n \;\ll\; d \times n$

$$1{,}000 \times 25 + 25 \times n \qquad\qquad 1{,}000 \times n$$

10 principal components give a pretty good reconstruction of a face

average face $\bar{x} + a[1]u_1$ $\bar{x} + a[1]u_1 + a[2]u_2$

$\bar{x}$ r = 1 r = 2 r = 3 r = 4

r = 7 r = 8 r = 9

r = 10

Ground truths real face

Assumption

- Notice how we started with the average face $\bar{x} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$
- PCA is applied to $\{x_i - \bar{x}\}_{i=1}^{n}$
- For simplicity, we will assume that x_i's are centered such that $\dfrac{1}{n}\sum_{i=1}^{n} x_i = 0$
- otherwise, without loss of generality,
  everything we do can be applied to the re-centered version of the data,
  i.e. $\{x_i - \bar{x}\}_{i=1}^{n}$, with $\bar{x} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$

How do we define the principal components?

- Dimensionality reduction (for some $r \ll d$):
  we would like to have a set of orthogonal directions $u_1, \ldots, u_r \in \mathbb{R}^d$, with $\|u_j\|_2 = 1$for all j to uniquely define principal components when we can,
  such that each data can be represented as linear combination of those direction vectors, i.e.

$$x_i \quad \approx \quad p_i = a_i[1]u_1 + \cdots + a_i[r]u_r$$

d=32x32

$$x_i = \begin{bmatrix} x_i[1] \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ x_i[d] \end{bmatrix} \longrightarrow a_i = \begin{bmatrix} a_i[1] \\ \vdots \\ a_i[r] \end{bmatrix}$$

Dimensionality Reduction

- Which choice of the principal components, $\{u_1, \ldots, u_r\}$, are better?
- But first, how do we find a_i given x_i and $\{u_1, \ldots, u_r\}$?

How do we find the principal components?

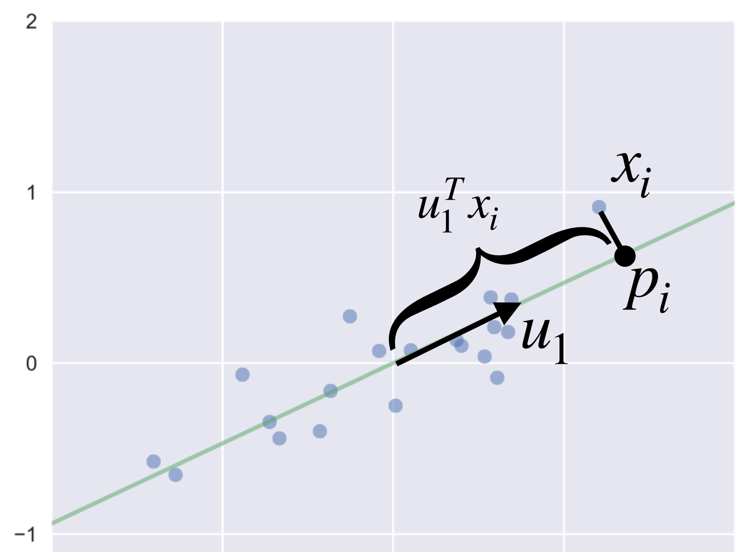
- Dimensionality reduction (for some $r \ll d$):
  we would like to have a set of orthogonal directions $u_1, \ldots, u_r \in \mathbb{R}^d$, with $\|u_j\|_2 = 1$ for all j, such that each data can be represented as linear combination of those direction vectors, i.e.
  $$x_i \;\approx\; p_i \;=\; a_i[1]u_1 + \cdots + a_i[r]u_r$$

$$x_i = \begin{bmatrix} x_i[1] \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ x_i[d] \end{bmatrix} \longrightarrow a_i = \begin{bmatrix} a_i[1] \\ \vdots \\ a_i[r] \end{bmatrix}$$

- those directions that minimize the average reconstruction error for a dataset is called the **principal components**

- given a choice of $u_1, \ldots, u_r$,
  the best representation p_i of x_i
  is the projection of the point onto
  the subspace spanned by u_j's, i.e.

$$a_i[j] = u_j^T x_i$$
$$p_i = \sum_{j=1}^{r} \underbrace{(u_j^T x_i)}_{a_i[j]} u_j$$



- we will use these without proving it

Principal components is the subspace that minimizes the reconstruction error

$$\underset{u_1,\ldots,u_r}{\text{minimize}} \ \frac{1}{n}\sum_{i=1}^{n} \|x_i - p_i\|_2^2$$

$$\text{subject to } \ \|u_j\|_2 = 1 \text{ for all } j \text{ and } u_j^T u_\ell = 0 \text{ for all } j \neq \ell$$

$$p_i = \sum_{j=1}^{r} (u_j^T x_i) u_j = \sum_{j=1}^{r} u_j u_j^T x_i = \Big(\sum_{j=1}^{r} u_j u_j^T\Big) x_i = \mathbf{U}\mathbf{U}^T x_i$$

$$\text{where } \mathbf{U} = [u_1 \ \ u_2 \ \ \cdots \ \ u_r] \in \mathbb{R}^{d\times r}$$

$$\underset{U}{\text{minimize}} \ \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mathbf{U}\mathbf{U}^T x_i\|_2^2$$

$$\text{subject to } \ \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

- Small rank r gives efficiency and large r gives less reconstruction error
- Q. How do we solve this optimization?

Minimizing reconstruction error to find principal components

$$\underset{U}{\text{minimize}} \; \frac{1}{n} \sum_{i=1}^{n} \| x_i - \mathbf{U}\mathbf{U}^T x_i \|_2^2$$

$$\text{subject to} \; \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r \times r}$$

Minimizing reconstruction error to find principal components

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i - UU^T x_i\|_2^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left\{ \|x_i\|_2^2 - 2x_i^T UU^T x_i + x_i^T U \underbrace{U^T U}_{=\mathbf{I}} U^T x_i \right\}$$

$$= \underbrace{\frac{1}{n}\sum_{i=1}^{n} \|x_i\|_2^2}_{\text{does not depend on } U} - \frac{1}{n}\sum_{i=1}^{n} x_i^T UU^T x_i$$

$$= C - \sum_{j=1}^{r} \underbrace{\frac{1}{n}\sum_{i=1}^{n} (u_j^T x_i)^2}_{\text{Variance in direction } u_j}$$

Recall we assumed x_i's are centered, i.e., zero-mean

Minimize Reconstruction Error

$$\underset{U}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mathbf{U}\mathbf{U}^T x_i\|_2^2$$

$$\text{subject to} \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

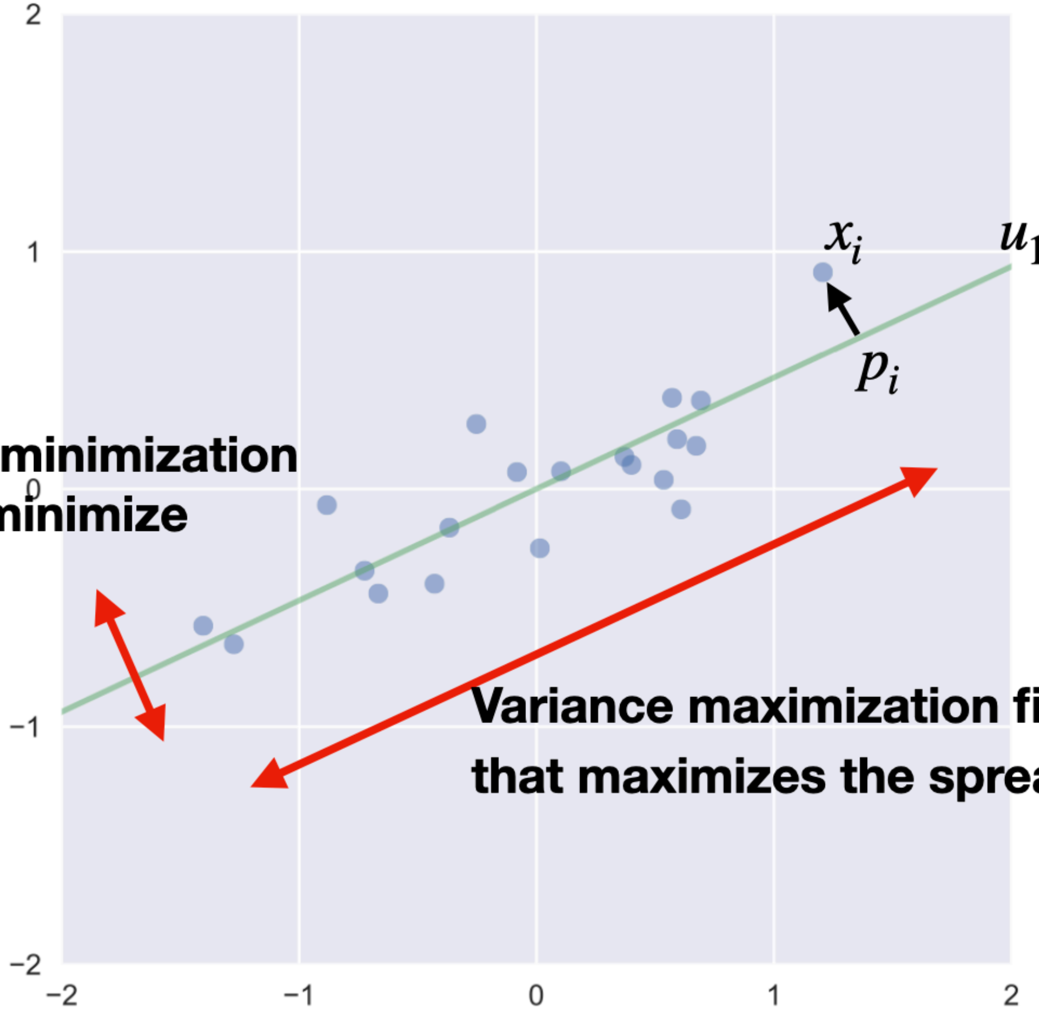
Maximizing Variance captured in principal directions

$$\underset{U}{\text{maximize}} \quad \sum_{j=1}^{r} \frac{1}{n}\sum_{i=1}^{n} (u_j^T x_i)^2$$

$$\text{subject to} \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

Variance maximization vs. reconstruction error minimization

- both give the same principal components as optimal solution, because $\text{Error}^2 + \text{Variance} = \|x_i\|_2^2$



Reconstruction error minimization finds directions that minimize the distances to p_i's

Variance maximization finds directions that maximizes the spread of p_i's

Maximizing variance to find principal components

$$\underset{U}{\text{maximize}} \quad \sum_{j=1}^{r} \frac{1}{n} \sum_{i=1}^{n} (u_j^T x_i)^2$$

$$\text{subject to} \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r \times r}$$

We will solve it for $r = 1$ case,
and the general case follows similarly

$$\underset{u:\|u\|_2=1}{\text{maximize}} \quad \frac{1}{n} \sum_{i=1}^{n} (u^T x_i)^2$$

$$\underset{u:\|u\|_2=1}{\text{maximize}} \quad u^T C u$$

How do you find u?

Maximizing variance to find principal components

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (a)$$

$$\textbf{subject to} \quad \|u\|_2^2 = 1$$

- we first claim that this optimization problem has the same optimal solution as the following **inequality constrained** problem

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (b)$$

$$\textbf{subject to} \quad \|u\|_2^2 \leq 1$$

- Why?

Maximizing variance to find principal components

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (a)$$

$$\textbf{subject to} \quad \|u\|_2^2 = 1$$

- we first claim that this optimization problem has the same optimal solution as the following **inequality constrained** problem

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (b)$$

$$\textbf{subject to} \quad \|u\|_2^2 \leq 1$$

- the reason is that, because $u^T \mathbf{C} u \geq 0$ for all $u \in \mathbb{R}^d$, the optimal solution of (b) has to have $\|u\|_2^2 = 1$
- if it did not have $\|u\|_2^2 = 1$, say $\|u\|_2^2 = 0.9$, then we can just multiply this u by a constant factor of $\sqrt{10/9}$ and increase the objective by a factor of $10/9$ while still satisfying the constraints

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (b)$$

subject to $\|u\|_2^2 \leq 1$

- we are maximizing the variance, while **keeping u small**
- this can be reformulated as an unconstrained problem, with Lagrangian encoding, to move the constraint into the objective

$$\text{maximize}_{u \in \mathbb{R}^d} \; \underbrace{u^T \mathbf{C} u - \lambda \|u\|_2^2}_{F_\lambda(u)} \qquad (c)$$

- this encourages small u as we want, and we can make this connection precise: there exists a (unknown) choice of λ such that the optimal solution of (c) is the same as the optimal solution of (b)
- further, for this choice of λ, exists an optimal u^* with $\|u^*\|_2 = 1$

Solving the unconstrained optimization

$$\text{maximize}_{u \in \mathbb{R}^d} \quad \underbrace{u^T \mathbf{C} u - \lambda \|u\|_2^2}_{F_\lambda(u)}$$

- to find such λ and the corresponding u, we solve the unconstrained optimization, by setting the gradient to zero
$$\nabla F_\lambda(u) = 2\mathbf{C}u - 2\lambda u = 0$$
- the candidate solution satisfies: $\mathbf{C}u = \lambda u$, i.e. an eigenvector of $\mathbf{C}$
- let $(\lambda^{(1)}, u^{(1)})$ denote the largest eigenvalue and corresponding eigenvector of $\mathbf{C}$,
- We will normalize the eigenvector such that $\|u^{(1)}\|_2^2 = 1$
- Selecting $\lambda = \lambda^{(1)}$, the maximum value of zero is achieved when $u = u^{(1)}$, why?
- No other choice of λ gives a solution with $\|u\|_2 = 1$

The principal component analysis

- so far we considered finding ONE principal component $u \in \mathbb{R}^d$
- it is the eigenvector corresponding to the maximum eigenvalue of the covariance matrix

$$\mathbf{C} = \frac{1}{n}\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d \times d}$$

- We can also use the Singular Value Decomposition (SVD) to find such eigen vector
- note that is the data is not centered at the origin, we should re-center the data before applying SVD

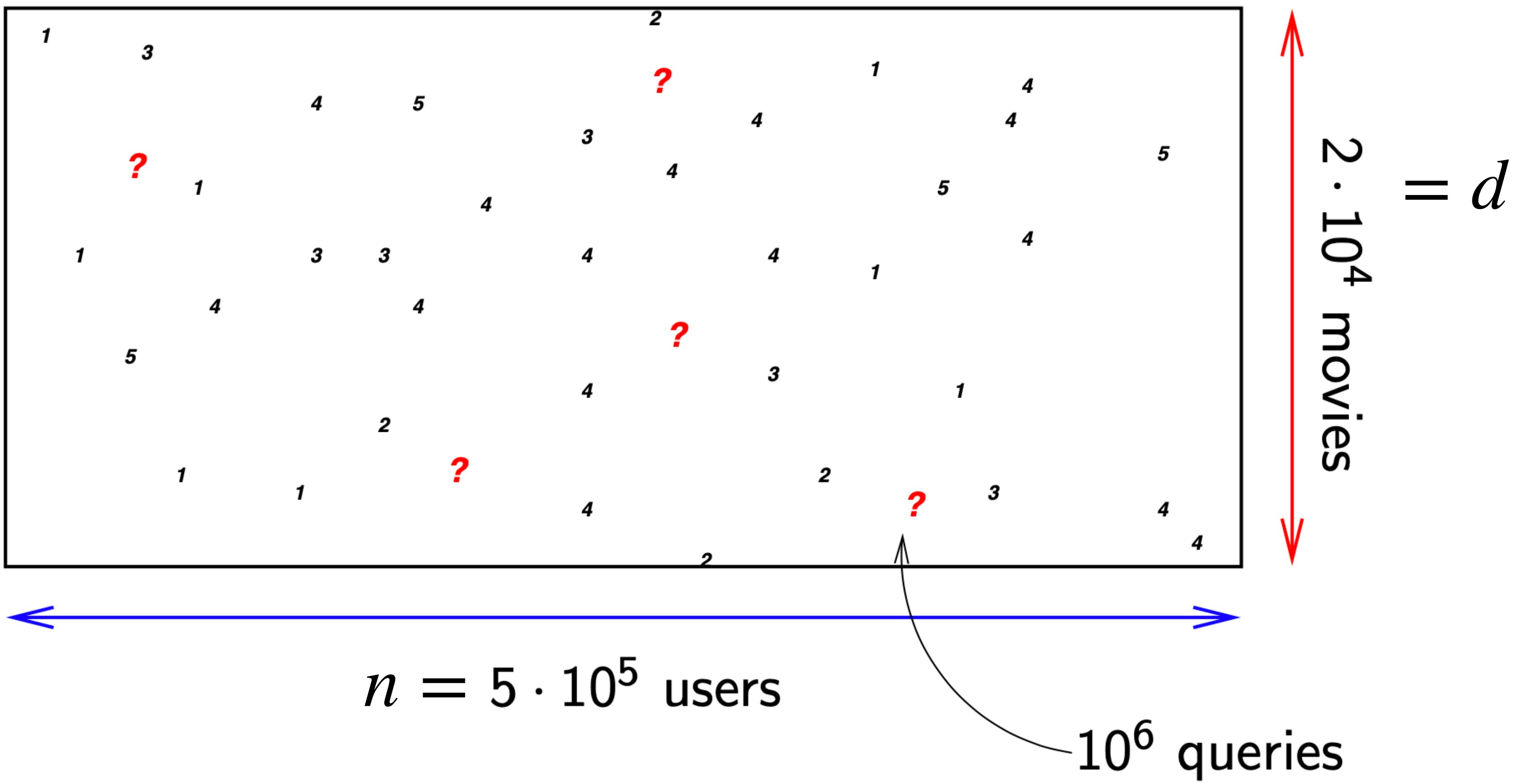
- in general we define and use multiple principal components
- if we need r principal components, we take r eigenvectors corresponding to the largest r eigenvalues of $\mathbf{C}$

Algorithm: Principal Component Analysis

- **input**: data points $\{x_i\}_{i=1}^n$, target dimension $r \ll d$
- **output**: r-dimensional subspace U
- **algorithm**:
  - compute mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
  - compute covariance matrix
  $$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T$$
  - let $(u_1, \ldots, u_r)$ be the set of (normalized) eigenvectors with corresponding to the largest r eigenvalues of $\mathbf{C}$
  - return $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r]$

- further the data points can be represented compactly via
$$a_i = \mathbf{U}^T(x_i - \bar{x}) \in \mathbb{R}^r$$

Matrix completion for recommendation systems



- users provide ratings on a few movies, and we want to predict the missing entries in this ratings matrix, so that we can make recommendations
- without any assumptions, the missing entries can be anything, and no prediction is possible

Matrix completion

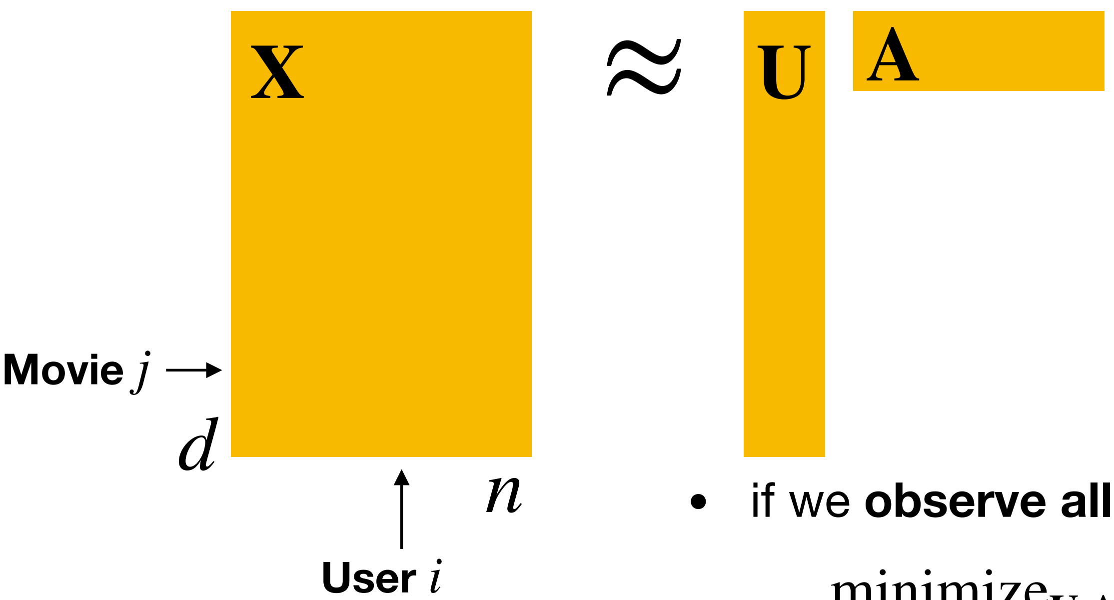
- however, the ratings are not arbitrary, but people with similar tastes rate similarly
- such structure can be modeled using low dimensional representation of the data as follows
- we will find a set of principal component vectors $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r] \ \in \mathbb{R}^{d \times r}$
- such that that ratings $x_i \in \mathbb{R}^d$ of user i, can be represented as

$$\begin{aligned} x_i &= a_i[1]u_1 + \cdots a_i[r]u_r \\ &= \mathbf{U}a_i \end{aligned}$$

  for some lower-dimensional $a_i \in \mathbb{R}^r$ for i-th user and some $r \ll d$
- for example, $u_1 \in \mathbb{R}^d$ means how horror movie fans like each of the d movies,
- and $a_i[1]$ means how much user i is fan of horror movies

Matrix completion

- let $\mathbf{X} = [x_1 \quad x_2 \quad \cdots \quad x_n] \in \mathbb{R}^{d \times n}$ be the ratings matrix, and assume it is fully observed, i.e. we know all the entries
- then we want to find $\mathbf{U} \in \mathbb{R}^{d \times r}$ and $\mathbf{A} = [a_1 \quad a_2 \quad \cdots \quad a_n] \in \mathbb{R}^{r \times n}$ that approximates $\mathbf{X}$



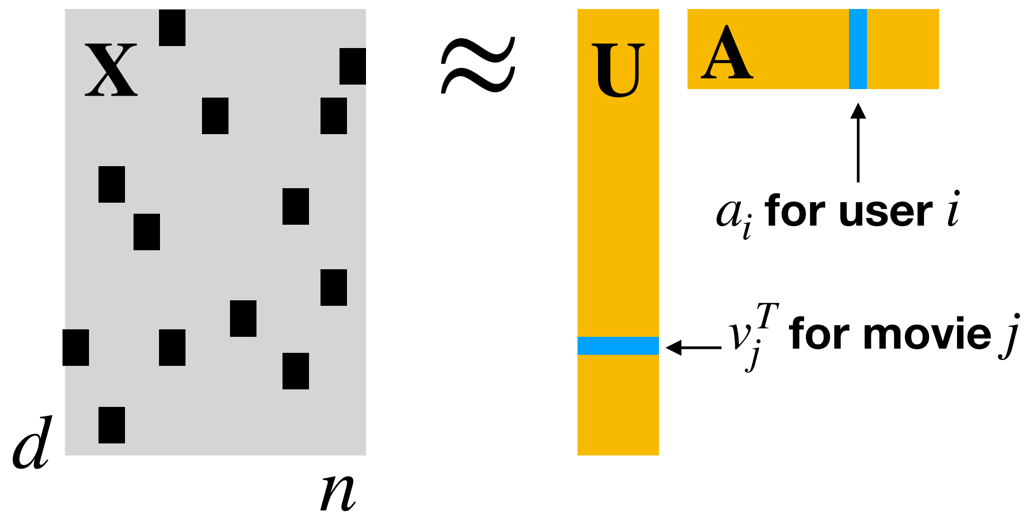
- if we **observe all entries** of $\mathbf{X}$, then we can solve

$$\text{minimize}_{\mathbf{U},\mathbf{A}} \sum_{i=1}^{n} \|x_i - \mathbf{U}a_i\|_2^2$$

which can be solved using PCA (i.e. SVD)

Matrix completion

- in practice, we only observe $\mathbf{X}$ partially
- let $S_{\text{train}} = \{(i_\ell, j_\ell)\}_{\ell=1}^N$ denote N observed ratings for user i_ℓ on movie j_ℓ



- let v_j^T denote the j-th row of $\mathbf{U}$ and a_i denote i-th column of $\mathbf{A}$
- then user i's rating on movie j, i.e. $\mathbf{X}_{ji}$ is approximated by $v_j^T a_i$, which is the inner product of v_j (a column vector) and a column vector a_i
- we can also write it as $\langle v_j, a_i \rangle = v_j^T a_i$

Matrix completion

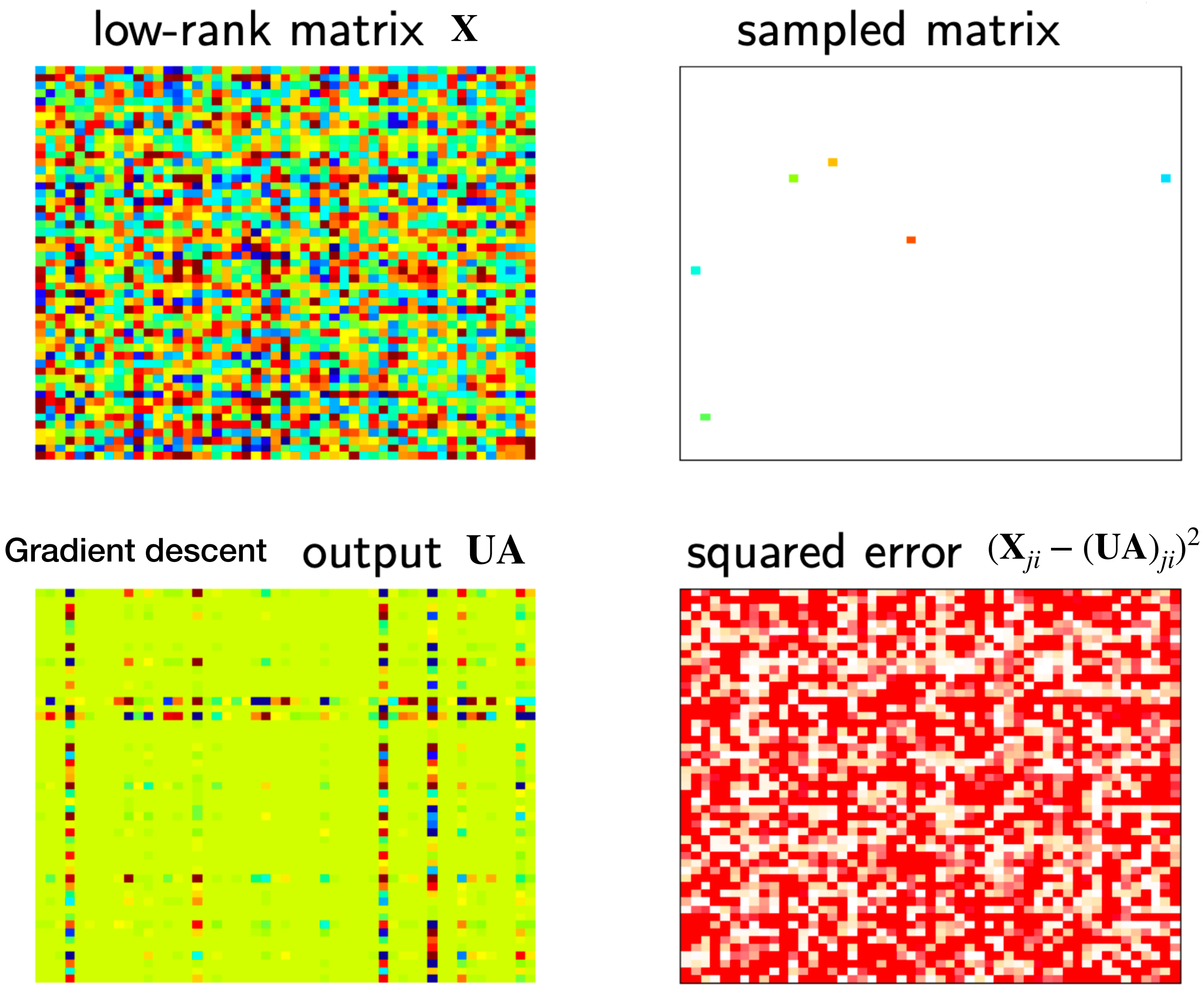
- a natural approach to fit v_j's and $a_i's$ to given training data is to solve

$$\text{minimize}_{\mathbf{U},\mathbf{A}} \sum_{(i,j)\in S_{\text{train}}} (\mathbf{X}_{ji} - v_j^T a_i)^2$$

- this can be solved, for example via gradient descent or alternating minimization
- this can be quite accurate, with small number of samples

Example: 2000 × 2000 rank-8 random matrix

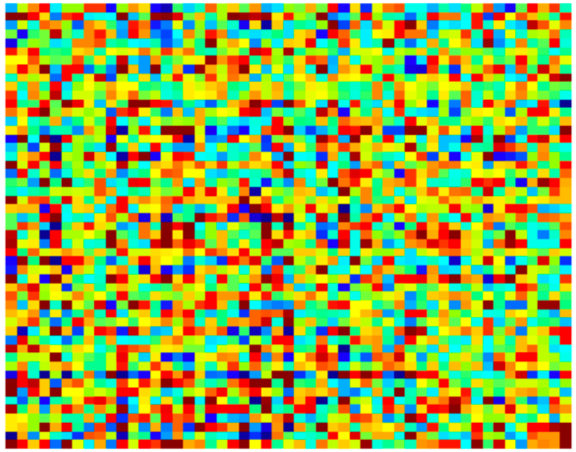
For illustration, we zoom in to a 50x50 submatrix



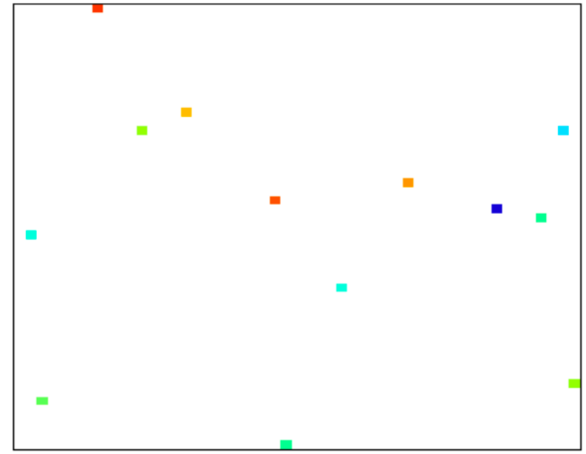
0.25% sampled

Example: 2000×2000 rank-8 random matrix

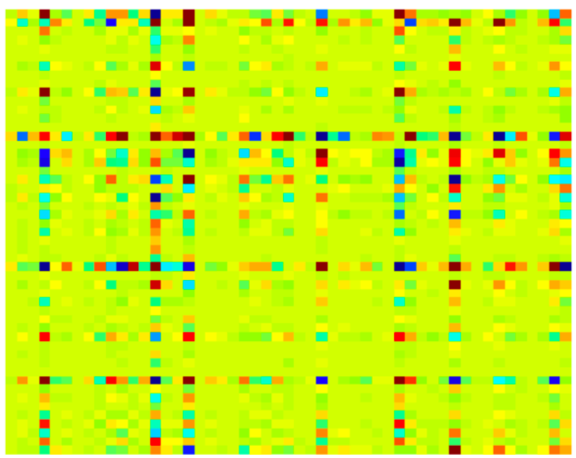
low-rank matrix $\mathbf{X}$



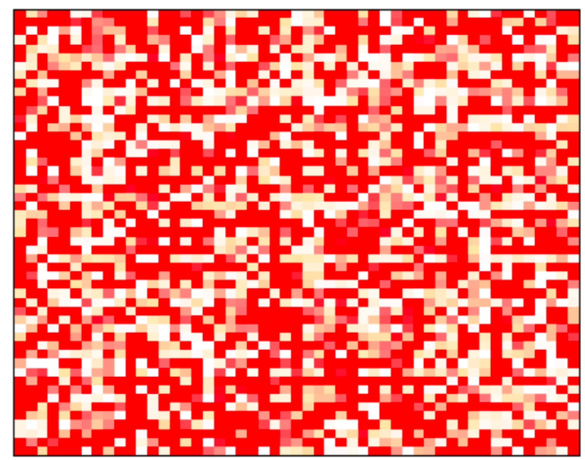
sampled matrix



Gradient descent output $\mathbf{UA}$

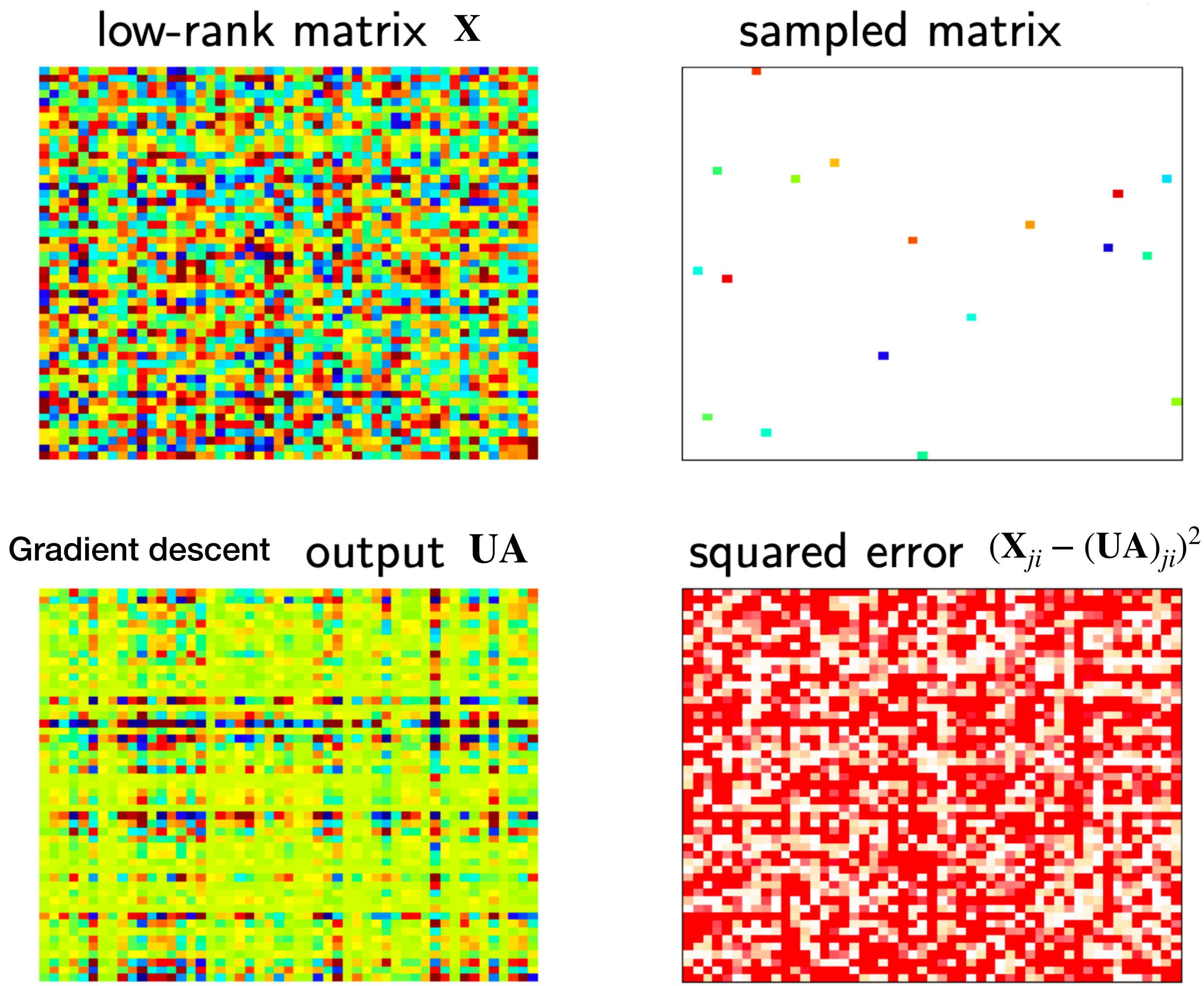


squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



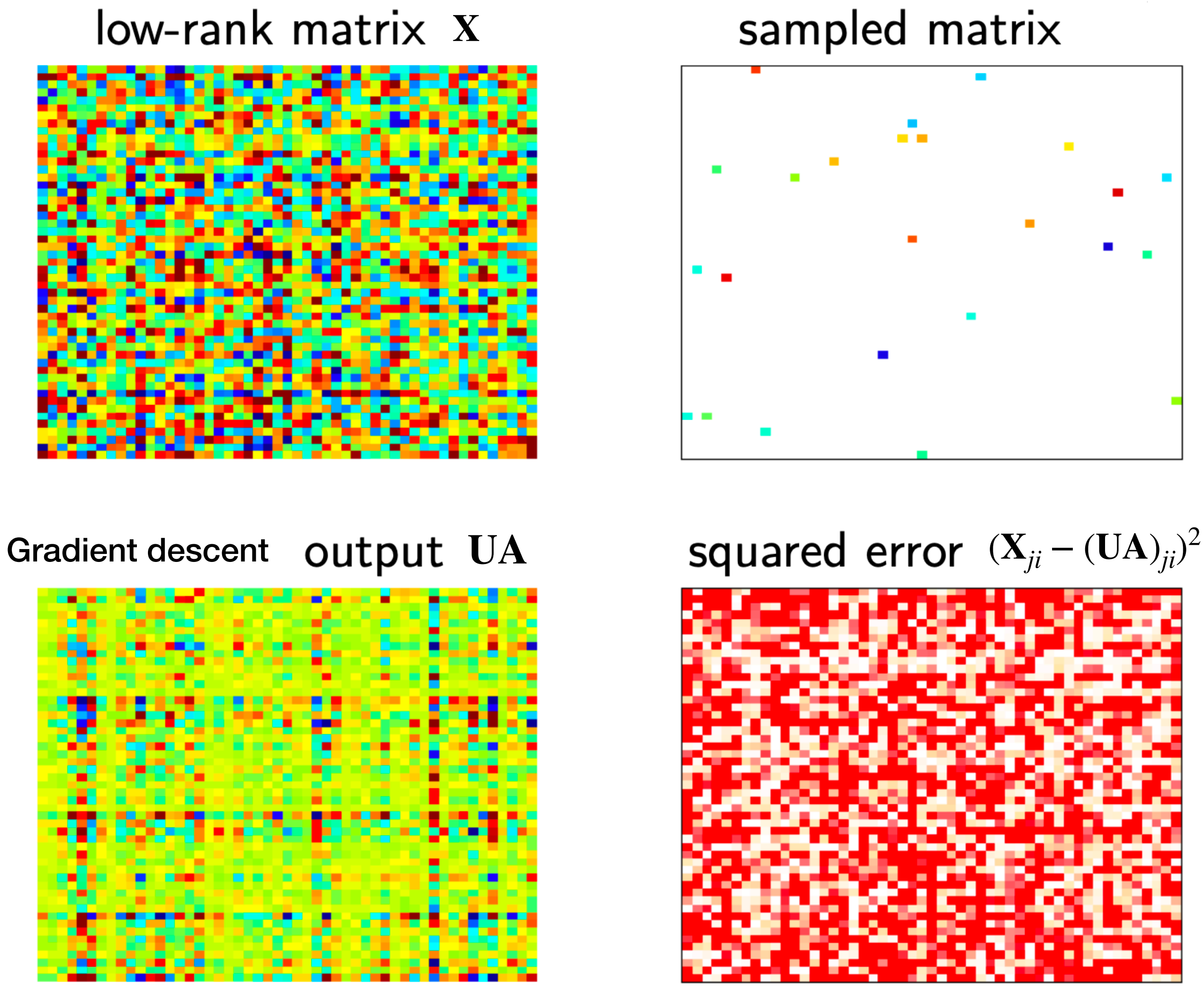
0.50% sampled

Example: 2000 × 2000 rank-8 random matrix



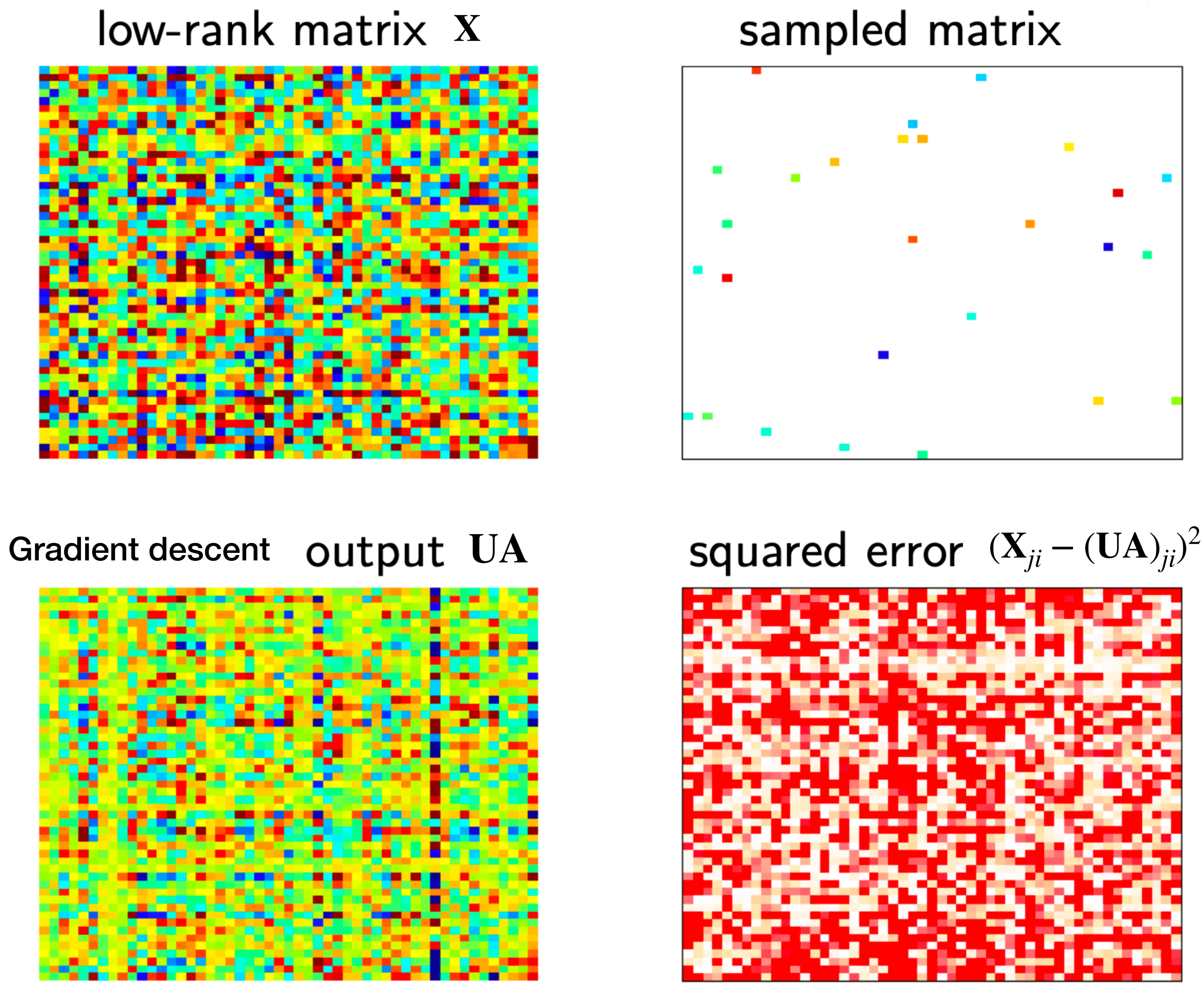
0.75% sampled

Example: 2000 × 2000 rank-8 random matrix



1.00% sampled

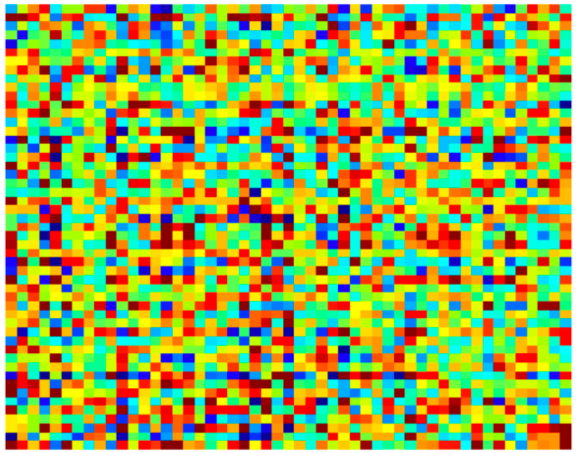
Example: 2000×2000 rank-8 random matrix



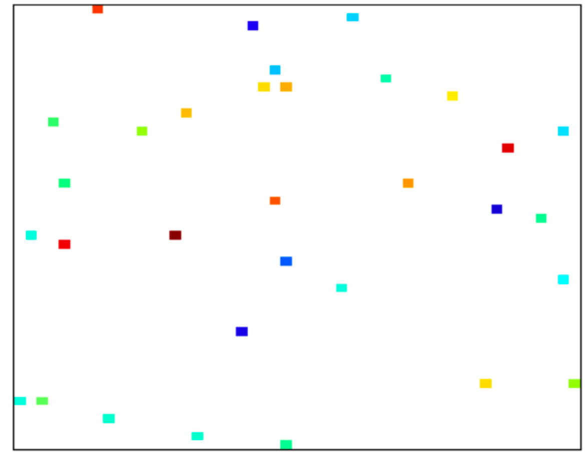
1.25% sampled

Example: 2000 $\times$ 2000 rank-8 random matrix

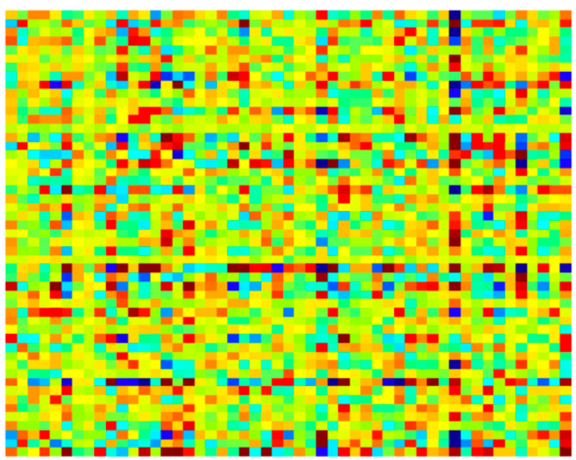
low-rank matrix $\mathbf{X}$



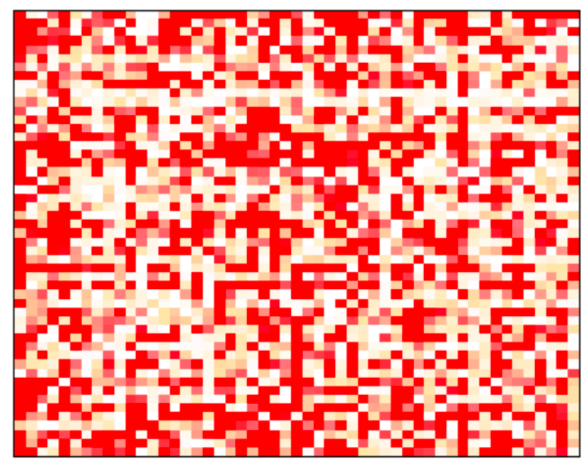
sampled matrix



Gradient descent output $\mathbf{UA}$



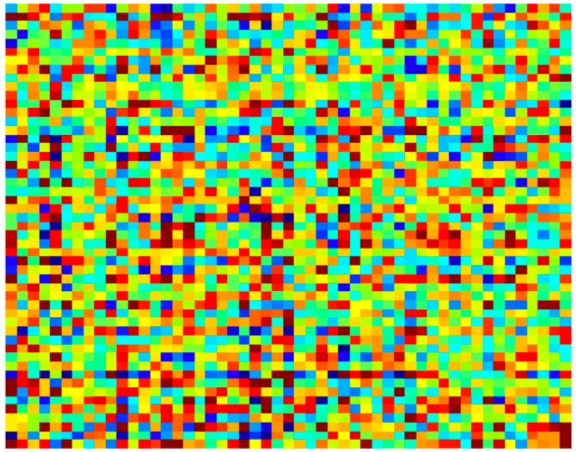
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



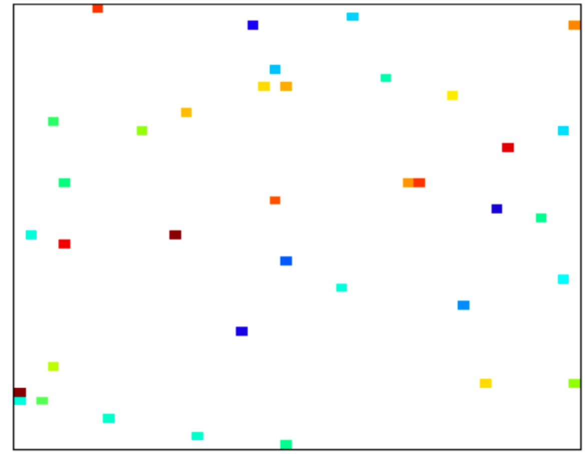
1.50% sampled

Example: 2000×2000 rank-8 random matrix

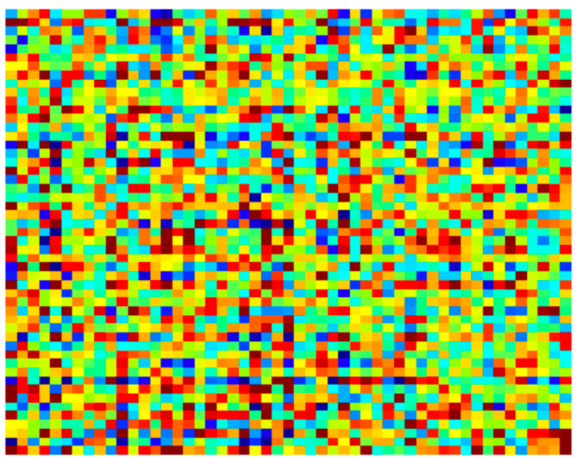
low-rank matrix $\mathbf{X}$



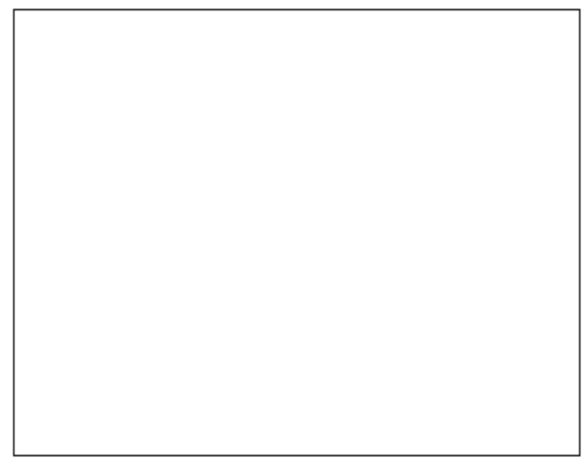
sampled matrix



Gradient descent output $\mathbf{UA}$



squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



1.75% sampled

Questions?