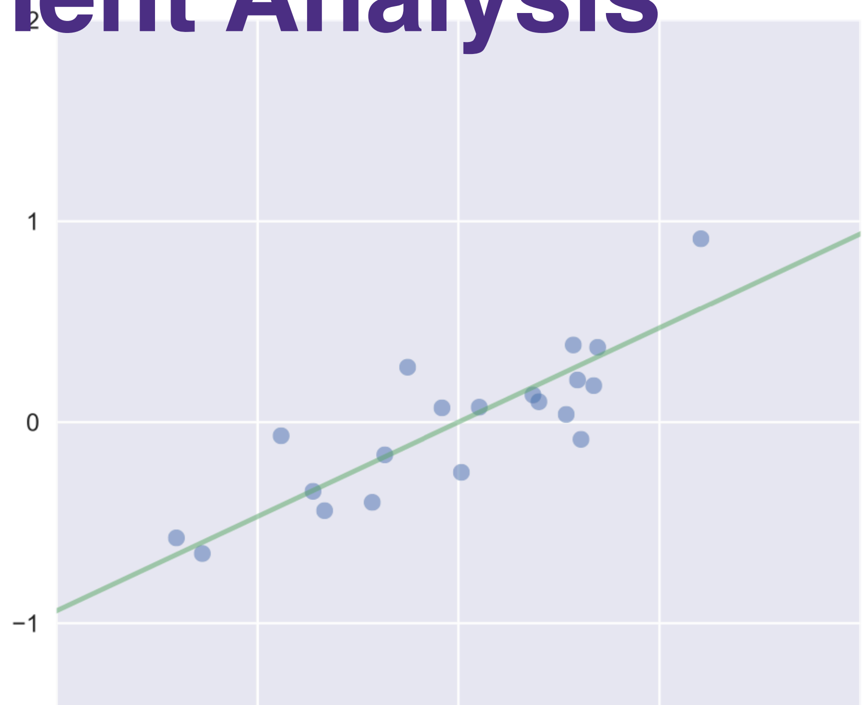
Principal Component Analysis



Principal components is the subspace that minimizes the reconstruction error

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

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

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

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

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

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

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

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

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

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

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

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

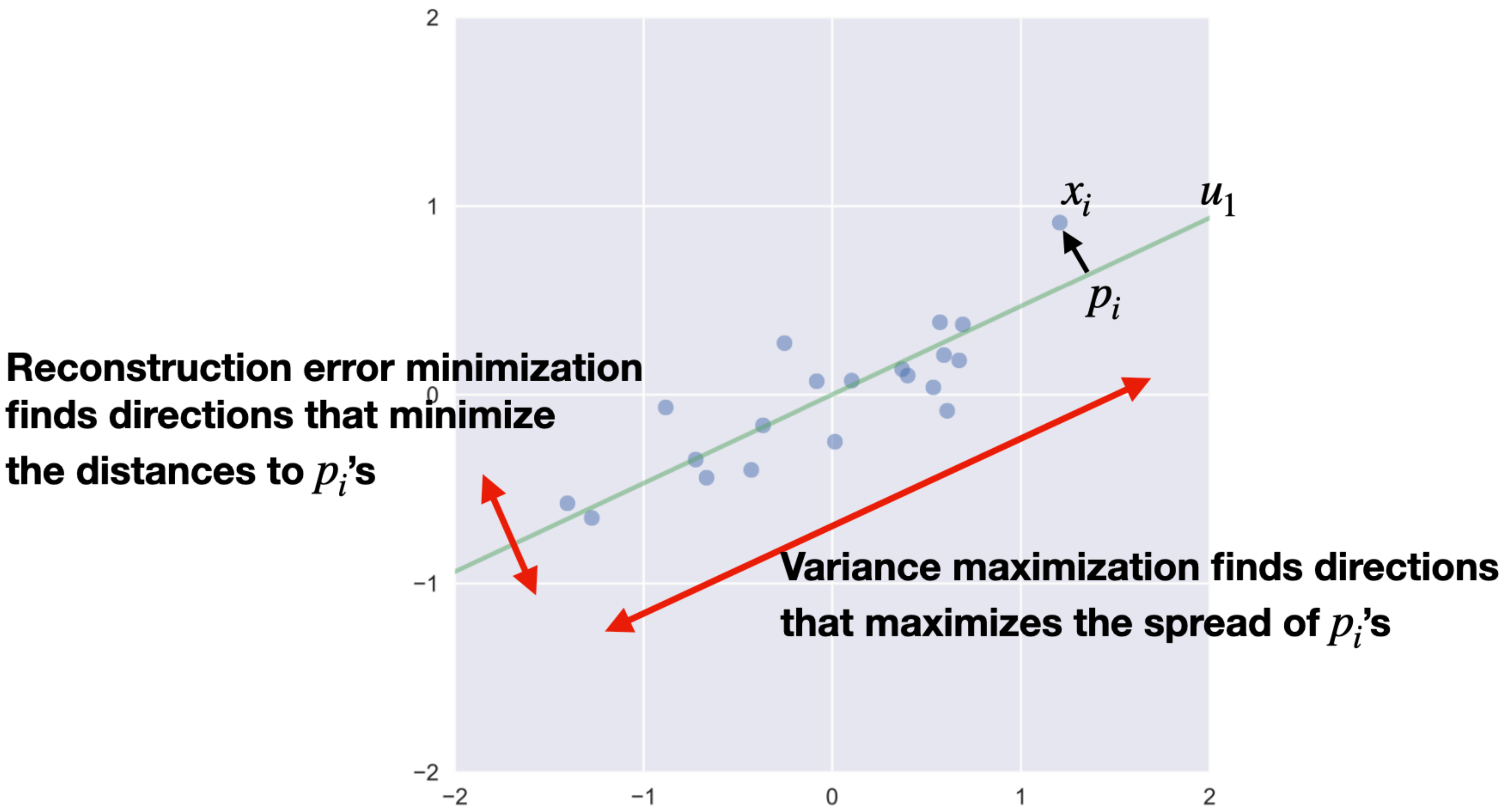
$\Updownarrow$

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

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

Variance maximization vs. reconstruction error minimization

- both give the same principal components as optimal solution



Maximizing variance to find principal components

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

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

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

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

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

Maximizing variance to find principal components

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

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

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

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

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

- 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

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

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

- 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

$$\text{maximize}_u \quad \underbrace{u^T \mathbf{C} u - \lambda \|u\|_2^2}_{F_\lambda(u)} \qquad (c)$$

- 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, the optimal u has $\|u\|_2 = 1$

Solving the unconstrained optimization

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

- to find such λ and the corresponding u, we solve the unconstrained optimization, by setting the gradient to zero

$$\nabla_u F_\lambda(u) \;=\; 2\mathbf{C}u - 2\lambda u \qquad = \; 0$$

- the candidate solution satisfies: $\mathbf{C}u = \lambda u$, i.e. an eigenvector of $\mathbf{C}$

$$\text{maximize}_u \; u^T \mathbf{C} u$$

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

- let $(\lambda^{(1)}, u^{(1)})$ denote the largest eigenvalue and corresponding eigenvector of $\mathbf{C}$, with norm one, i.e. $\|u^{(1)}\|_2^2 = 1$
- The maximum is achieved when $u = u^{(1)}$

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 use 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
$$a_i = \mathbf{U}^T(x_i - \bar{x}) \in \mathbb{R}^r$$

Singular Value Decomposition (SVD)

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, $\mathbf{U}^T\mathbf{U} = I$, $\mathbf{V}^T\mathbf{V} = I$.

What is $A^T A v_i =$ $\qquad$ $AA^T =$

What is $AA^T u_i =$ $\qquad$ $A^T A =$

- v_i's are the r eigen vectors of $A^T A$ with corresponding eigen values S_{jj}^2's
- u_i's are the r eigen vectors of AA^T with corresponding eigen values S_{jj}^2's
- Computing SVD takes $O(mnr)$ operations

Singular Value Decomposition (SVD)

- Consider a full rank matrix $A \in \mathbb{R}^{m \times n}$ whose SVD is $A = USV^T$, and we want to find the best rank-r approximation of A that minimizes the error

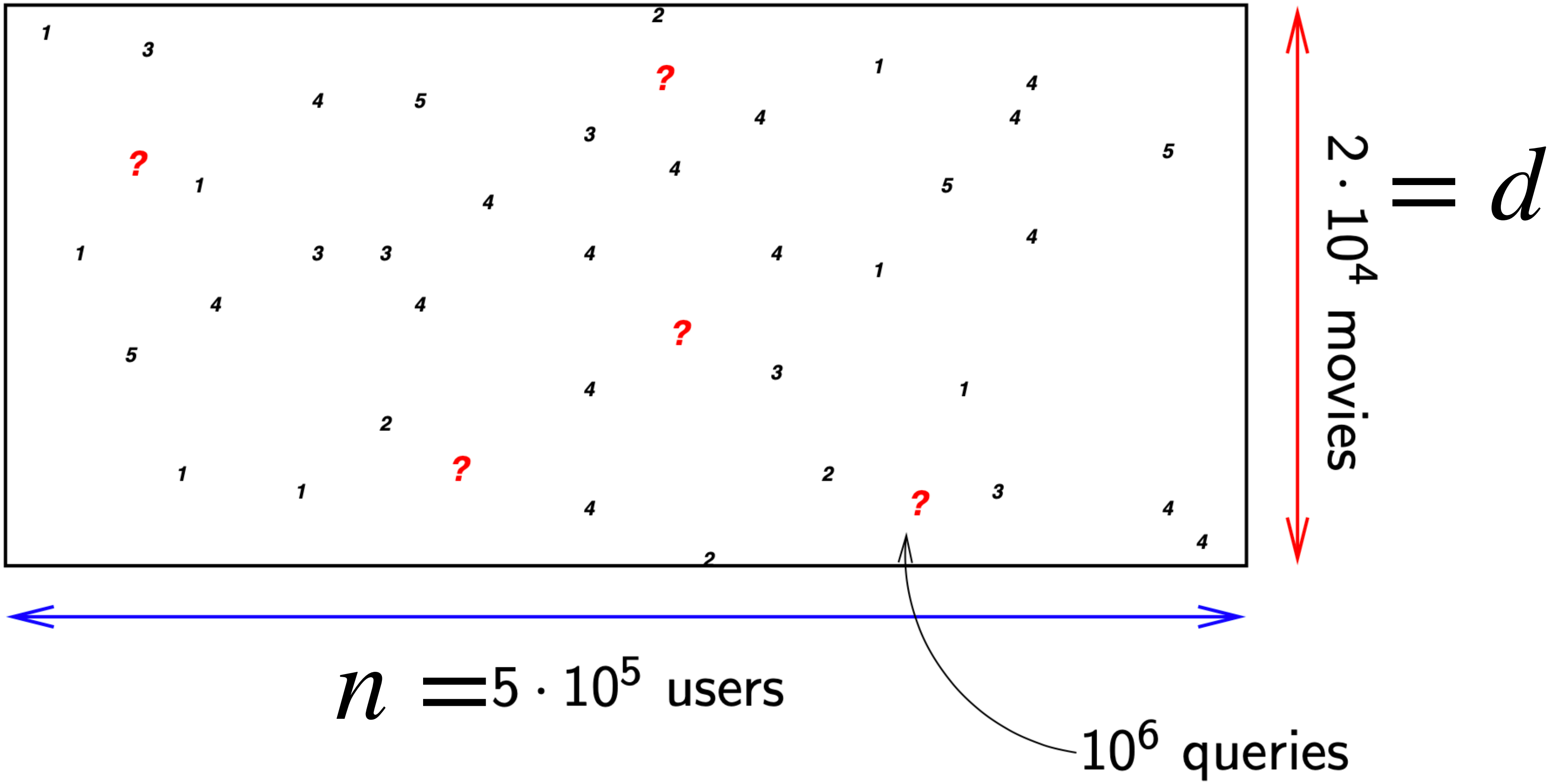
$$\text{minimize}_{L \in \mathbb{R}^{m \times n}} \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{i,j} - L_{i,j})^2$$

$$\text{subject to } \operatorname{rank}(L) = r$$

- The optimal rank-r approximation is $U_{1:r} S_{1:r,1:r} V_{1:r}^T$

Matrix completion for recommendation systems

Netflix challenge dataset



- 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 problem

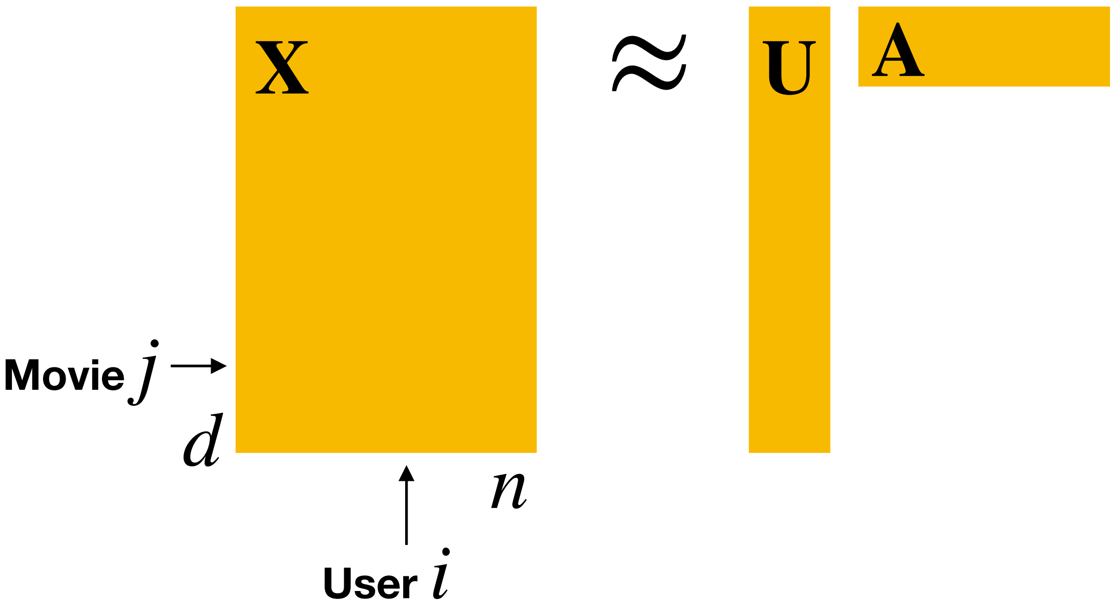
- however, the ratings are not arbitrary, but people with similar tastes rate similarly
- such structure can be modeled using low dimensional representation of the data as follows
- we will find a set of principal component vectors $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r] \ \in \mathbb{R}^{d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

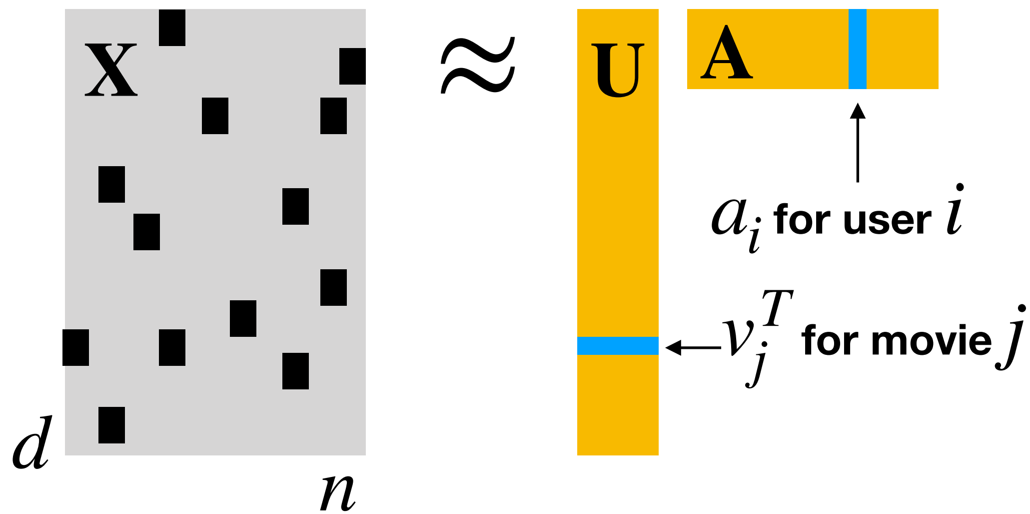
- let $\mathbf{X} = [x_1 \quad x_2 \quad \cdots \quad x_n] \in \mathbb{R}^{d \times n}$ be the ratings matrix, and assume it is fully observed, i.e. we know all the entries
- then we want to find $\mathbf{U} \in \mathbb{R}^{d \times r}$ and $\mathbf{A} = [a_1 \quad a_2 \quad \cdots \quad a_n] \in \mathbb{R}^{r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



- if we **observe all entries** of $\mathbf{X}$, then we can find the best rank-r approximation with 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 $\langle v_j, a_i \rangle = v_j^T a_i$

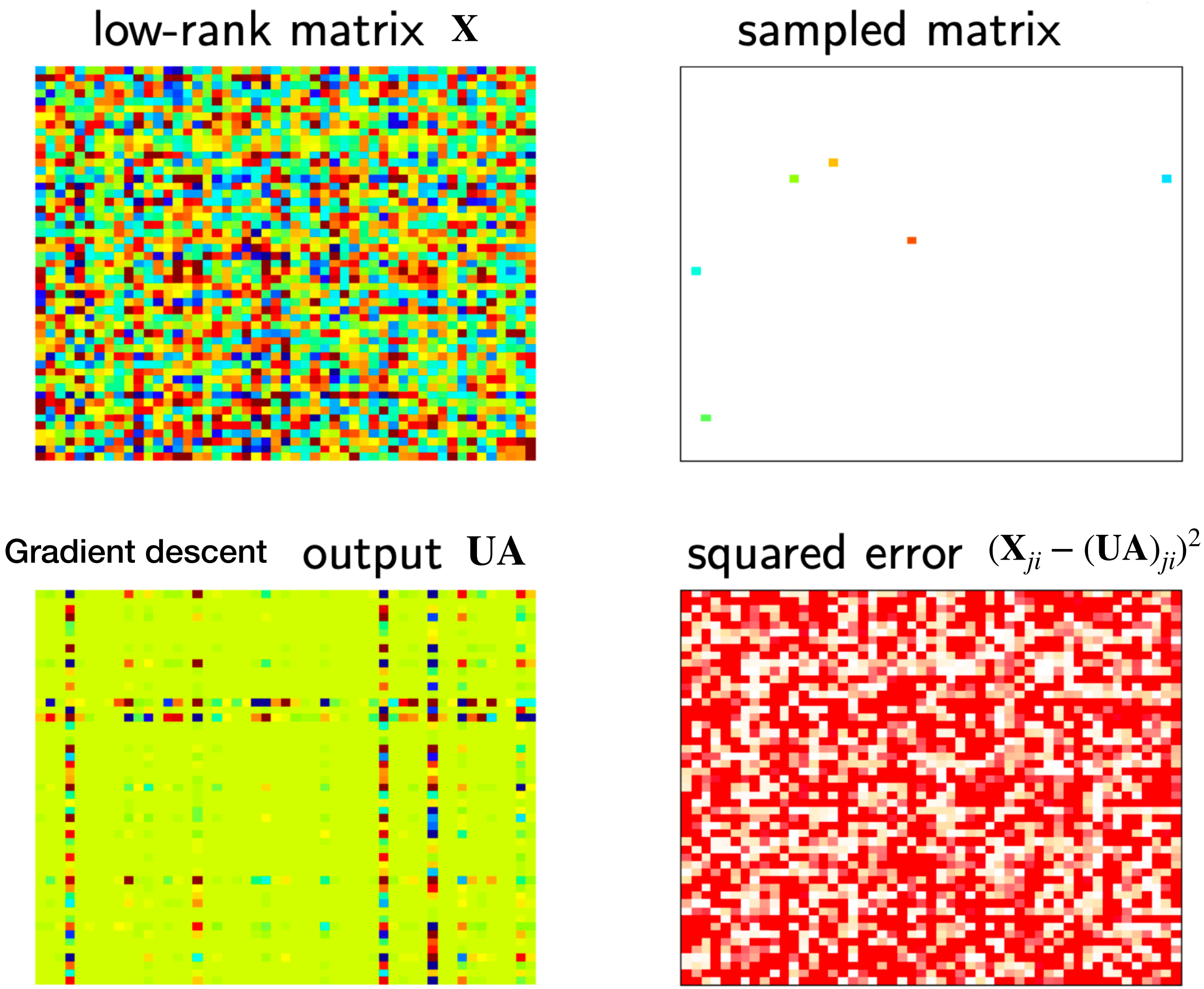
Matrix completion

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

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

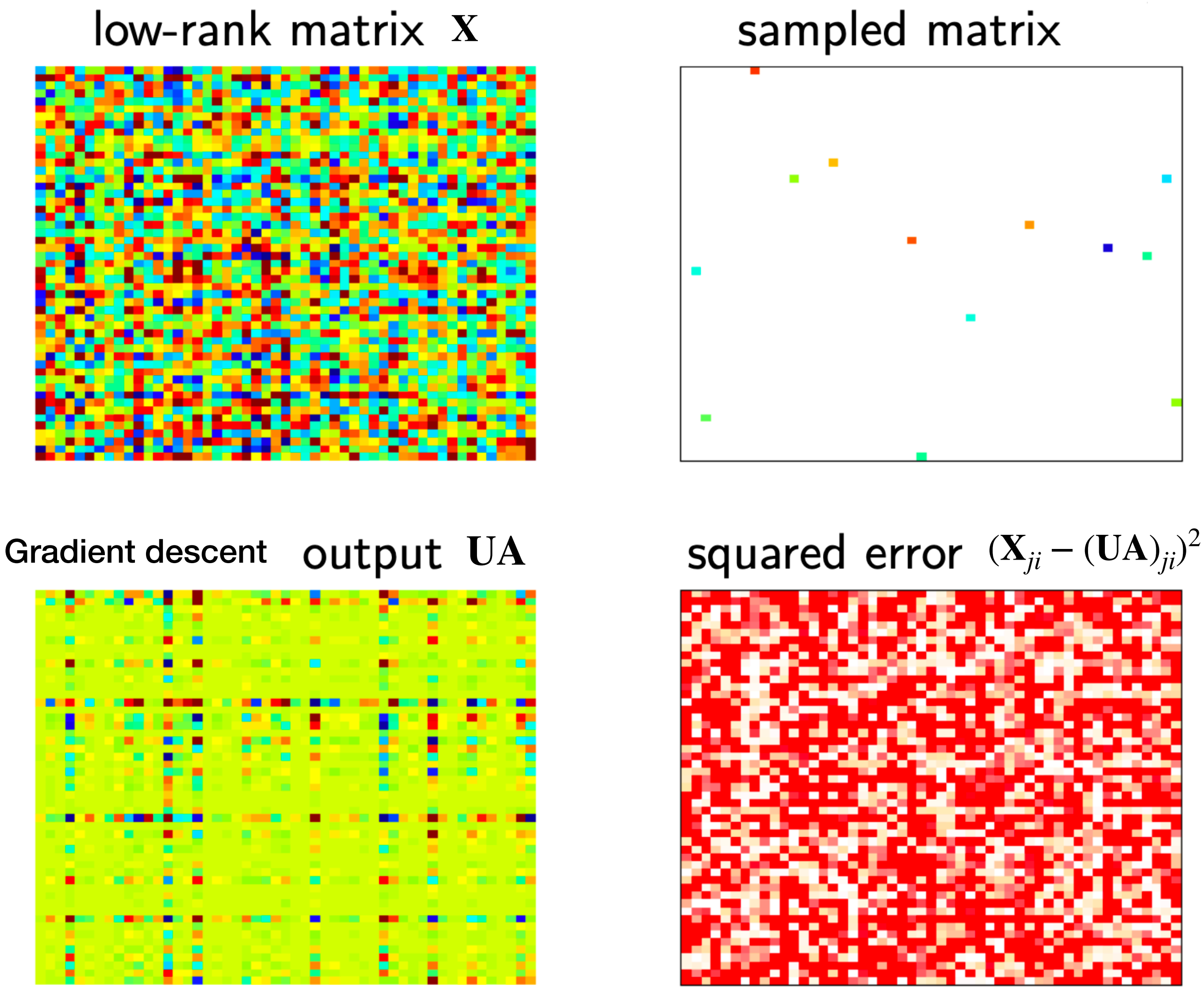
- this can be solved, for example via gradient descent or alternating minimization
- this can be quite accurate, with small number of samples

Example: 2000×2000 rank-8 random matrix



0.25% sampled

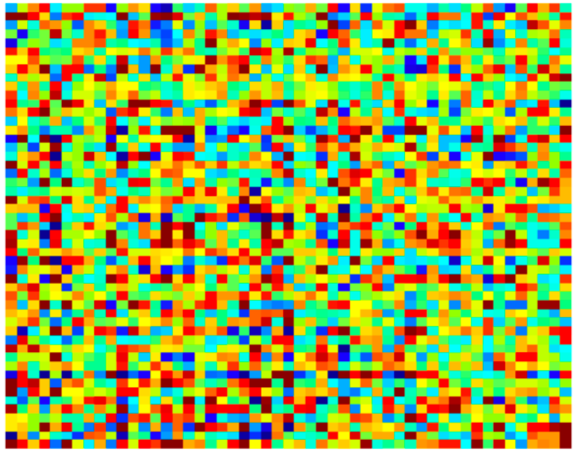
Example: 2000 $\times$ 2000 rank-8 random matrix



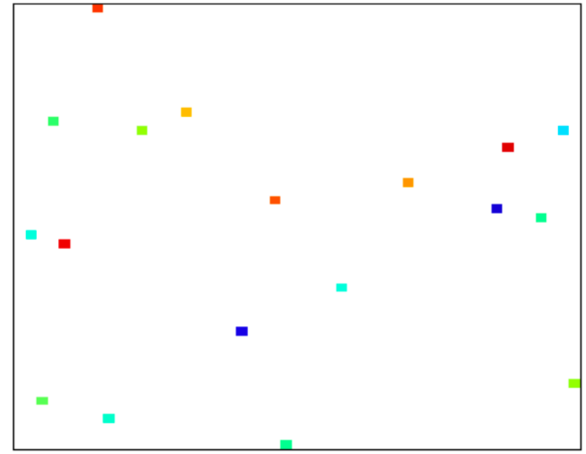
0.50% sampled

Example: 2000×2000 rank-8 random matrix

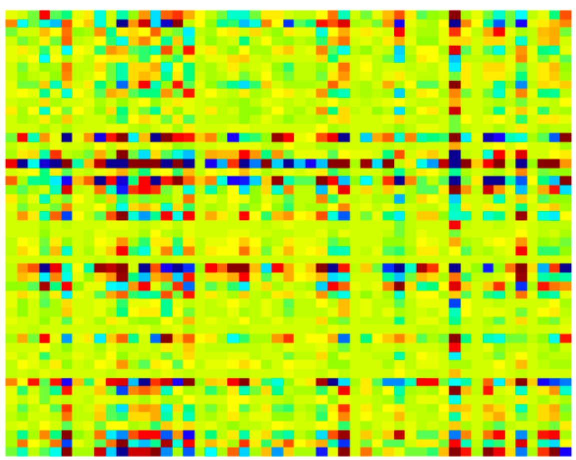
low-rank matrix $\mathbf{X}$



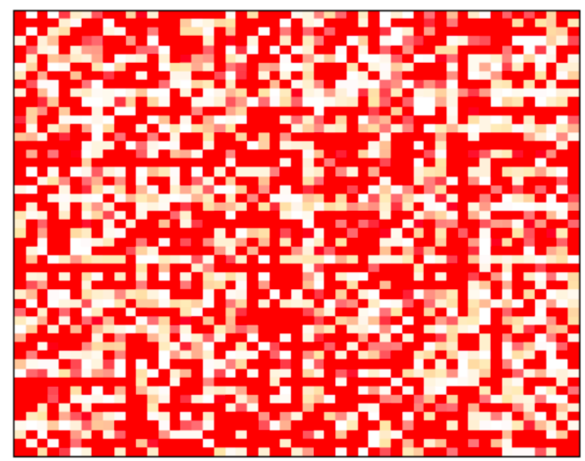
sampled matrix



Gradient descent output $\mathbf{UA}$



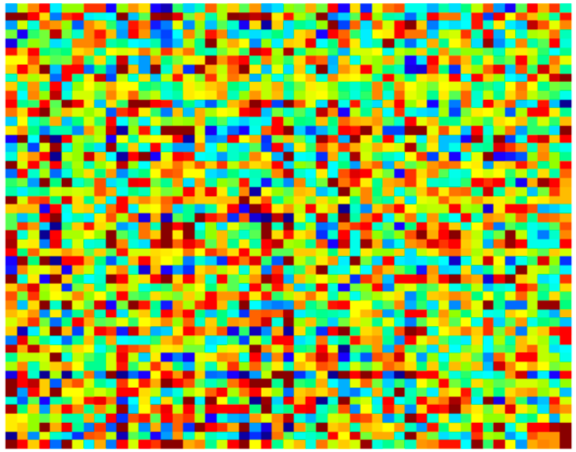
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



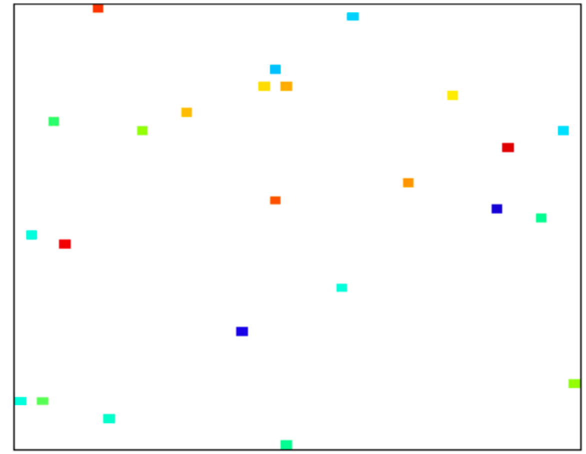
0.75% sampled

Example: 2000 × 2000 rank-8 random matrix

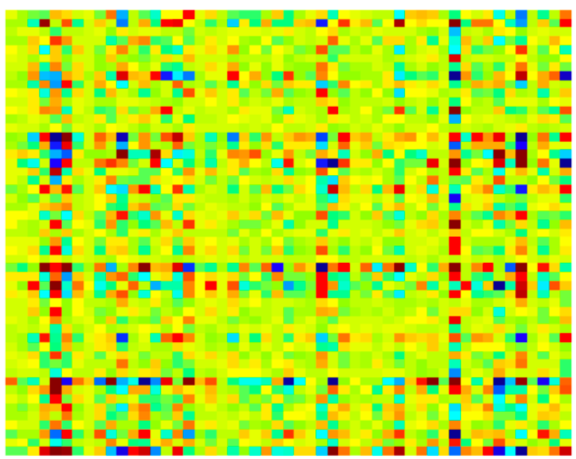
low-rank matrix $\mathbf{X}$



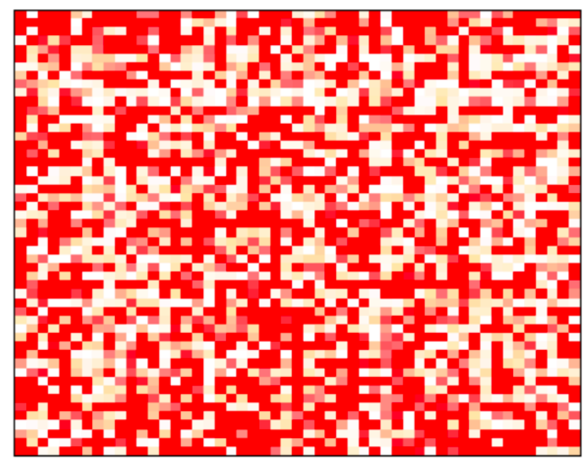
sampled matrix



Gradient descent output $\mathbf{UA}$



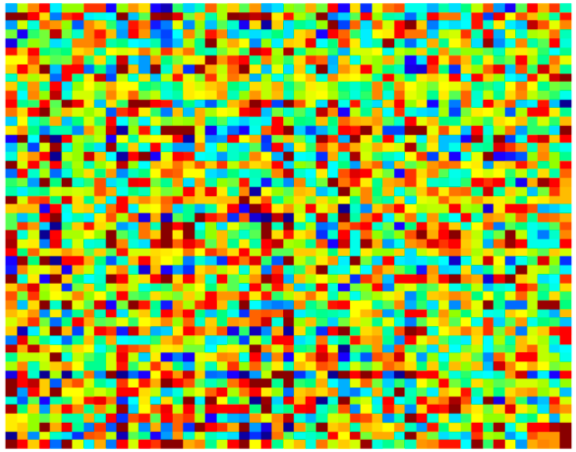
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



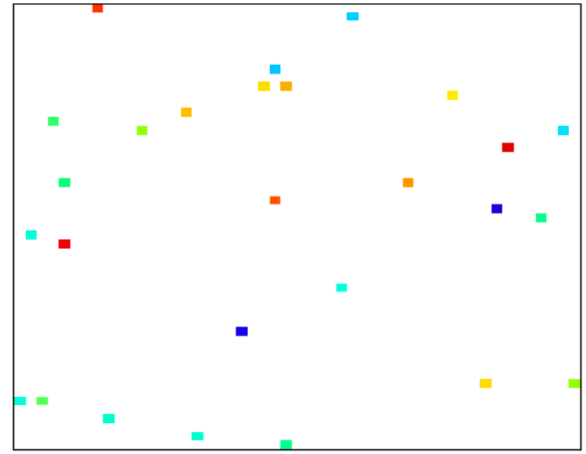
1.00% sampled

Example: 2000×2000 rank-8 random matrix

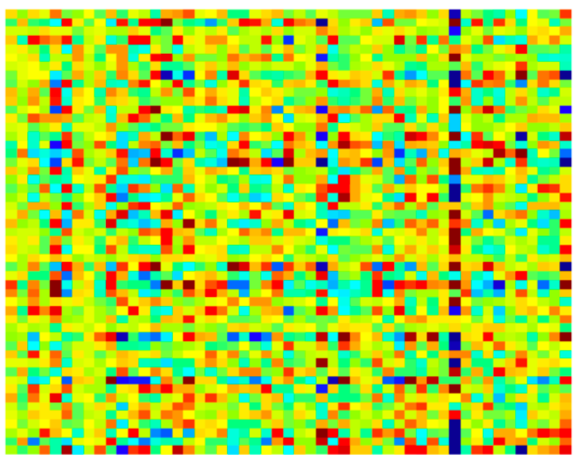
low-rank matrix $\mathbf{X}$



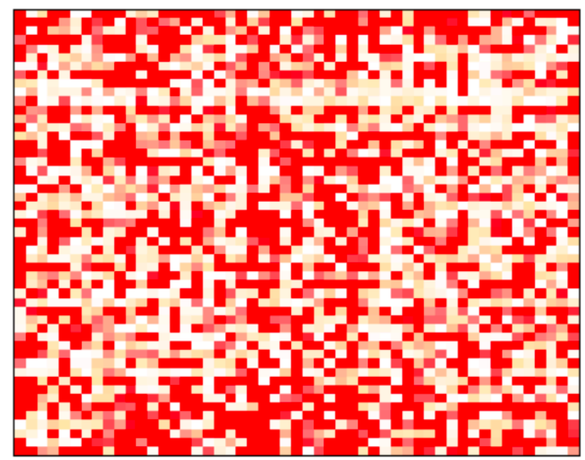
sampled matrix



Gradient descent output $\mathbf{UA}$



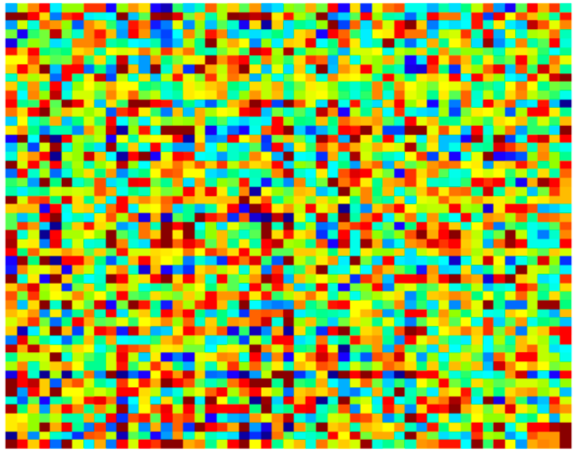
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



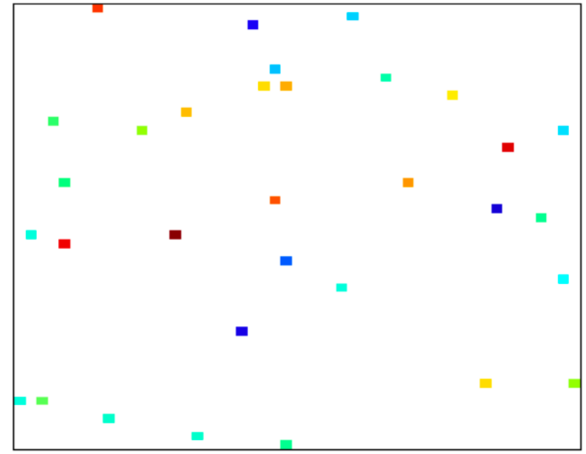
1.25% sampled

Example: 2000×2000 rank-8 random matrix

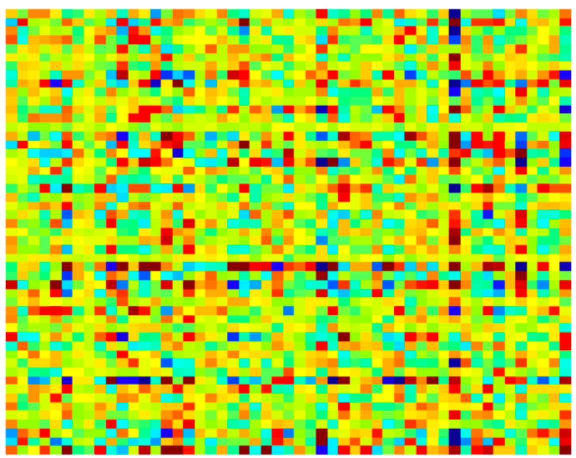
low-rank matrix $\mathbf{X}$



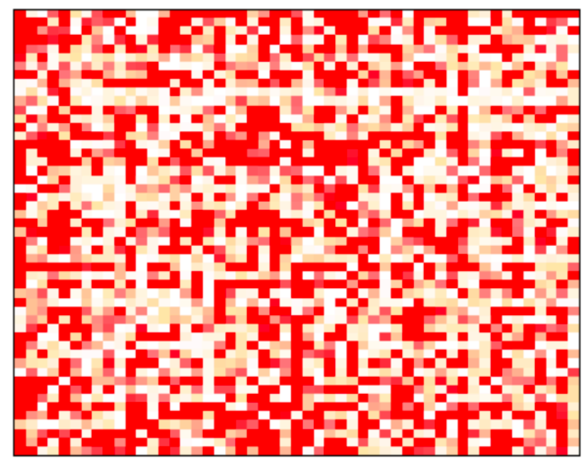
sampled matrix



Gradient descent output $\mathbf{UA}$



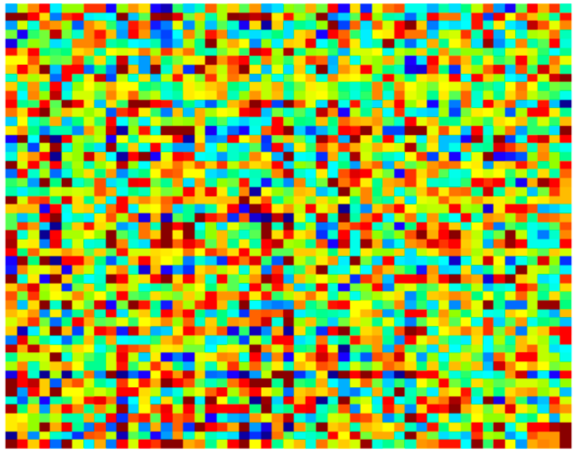
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



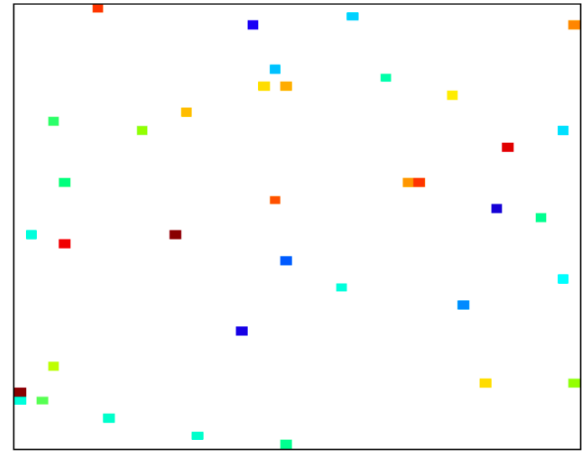
1.50% sampled

Example: 2000×2000 rank-8 random matrix

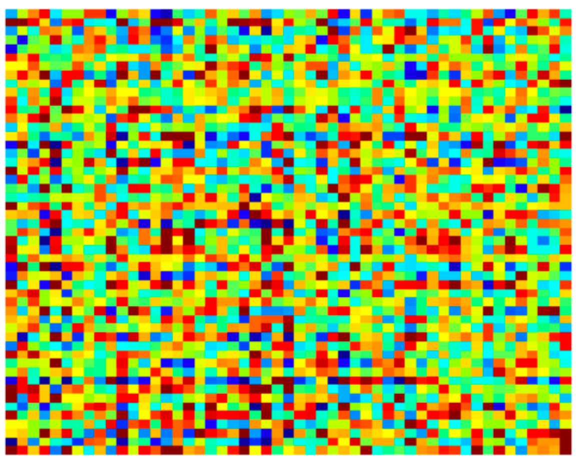
low-rank matrix $\mathbf{X}$



sampled matrix



Gradient descent output $\mathbf{UA}$



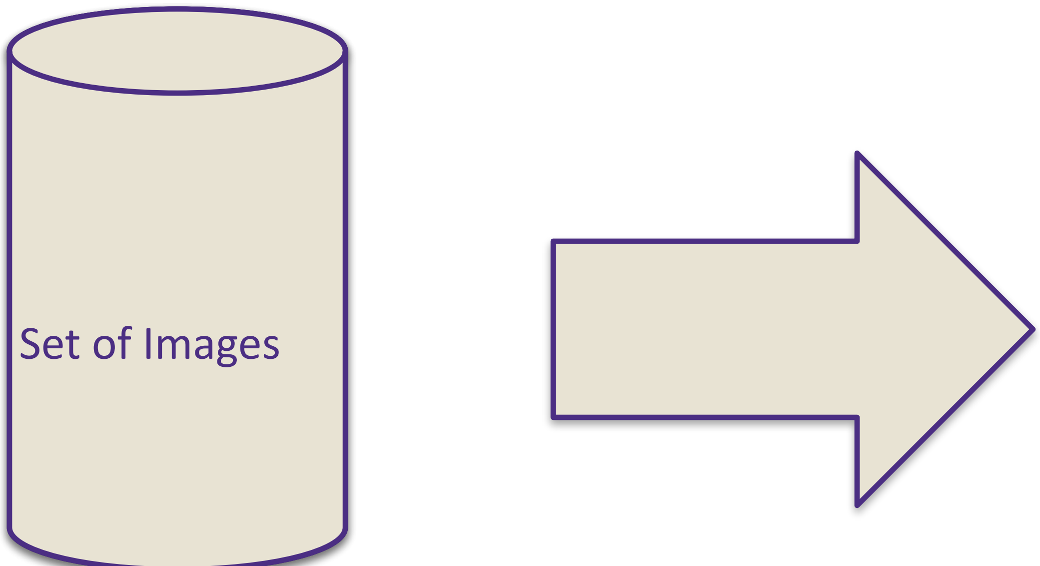
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$

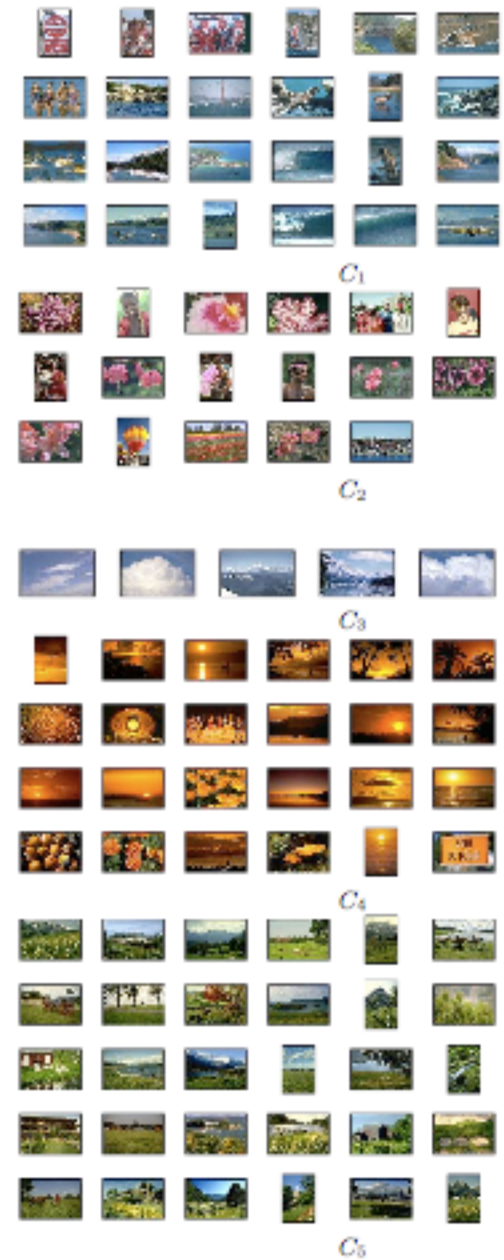
1.75% sampled

Clustering with k-means

UNIVERSITY *of* WASHINGTON

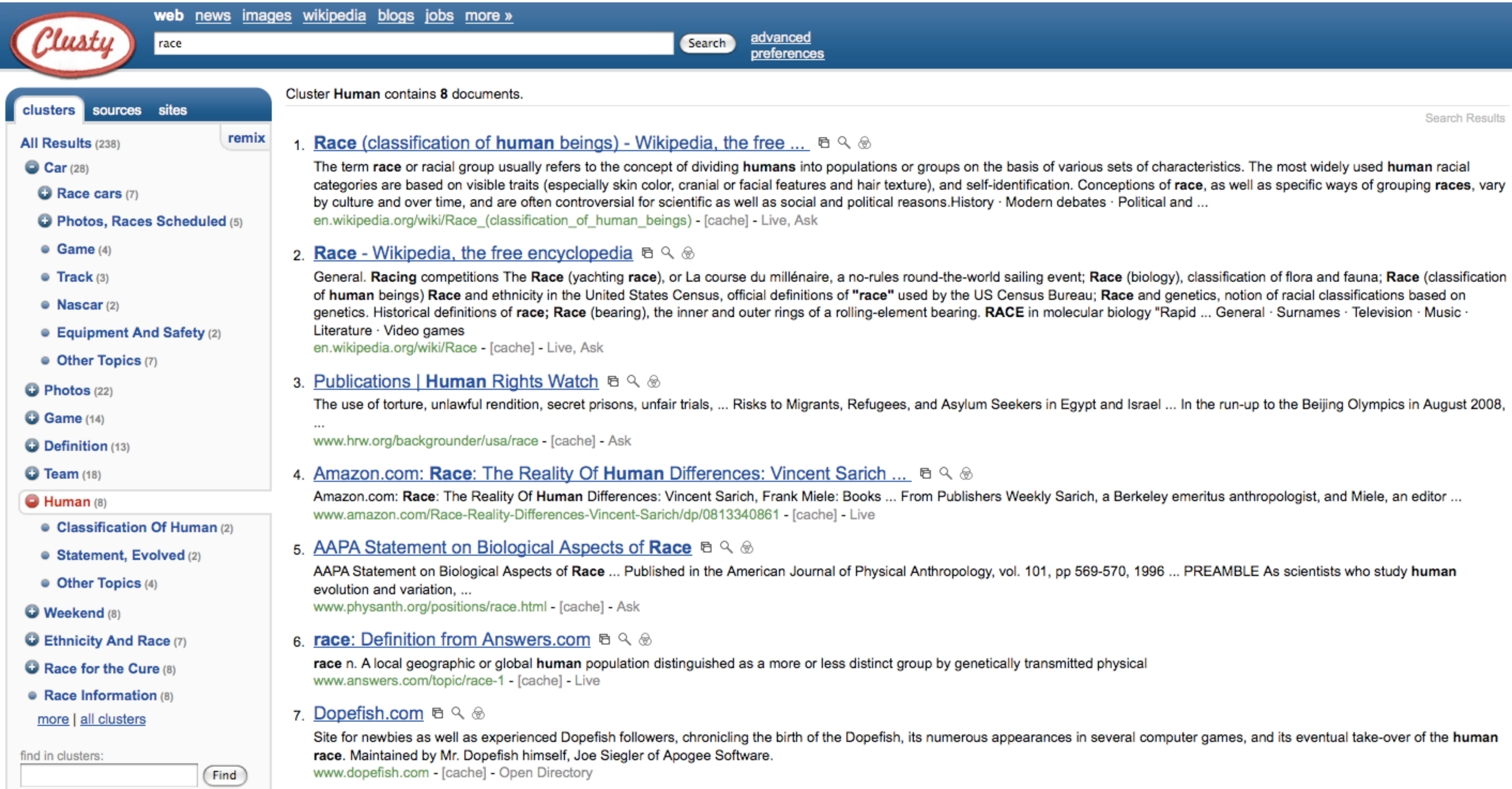
Clustering images



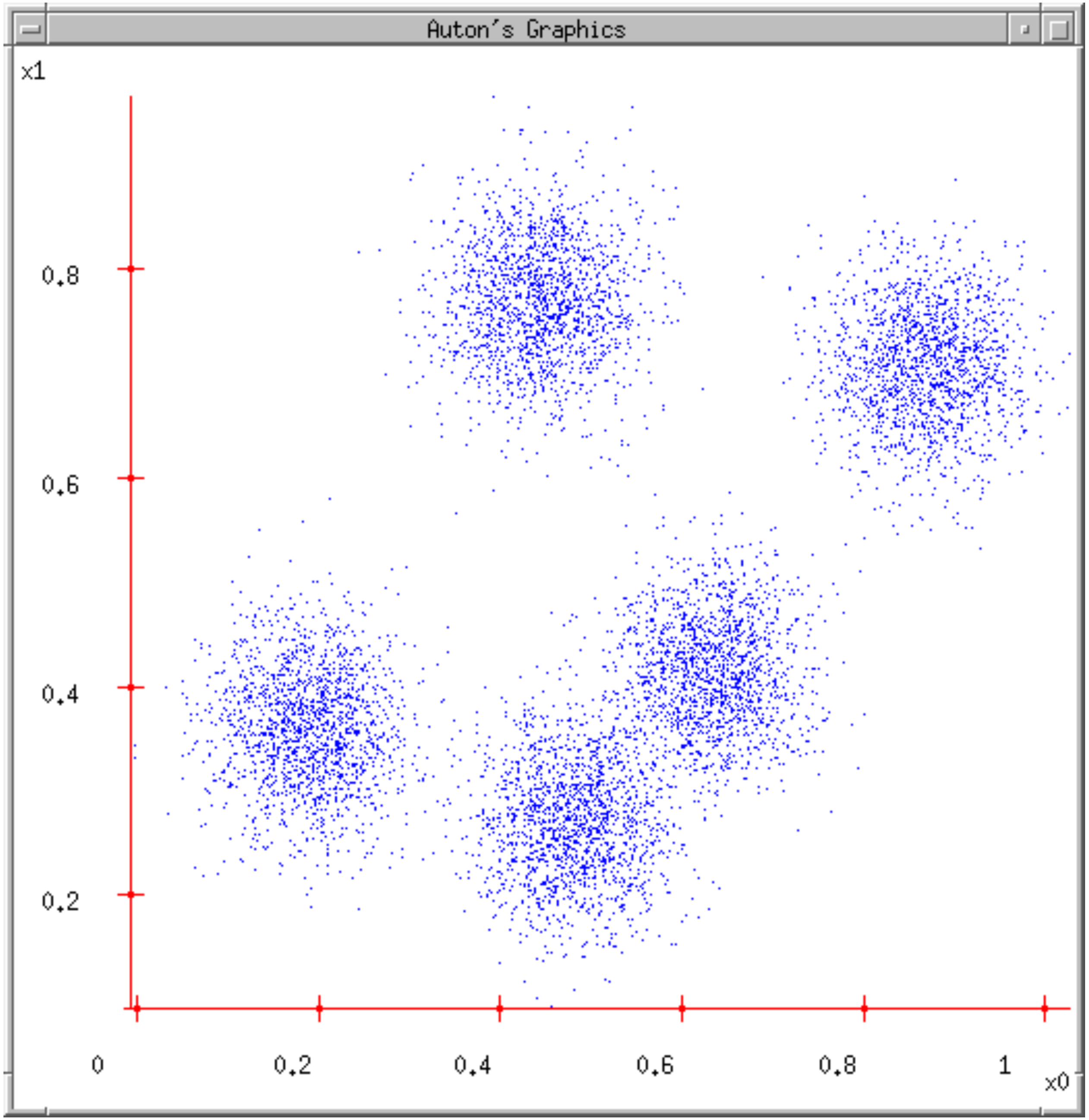


[Goldberger et al.]

Clustering web search results

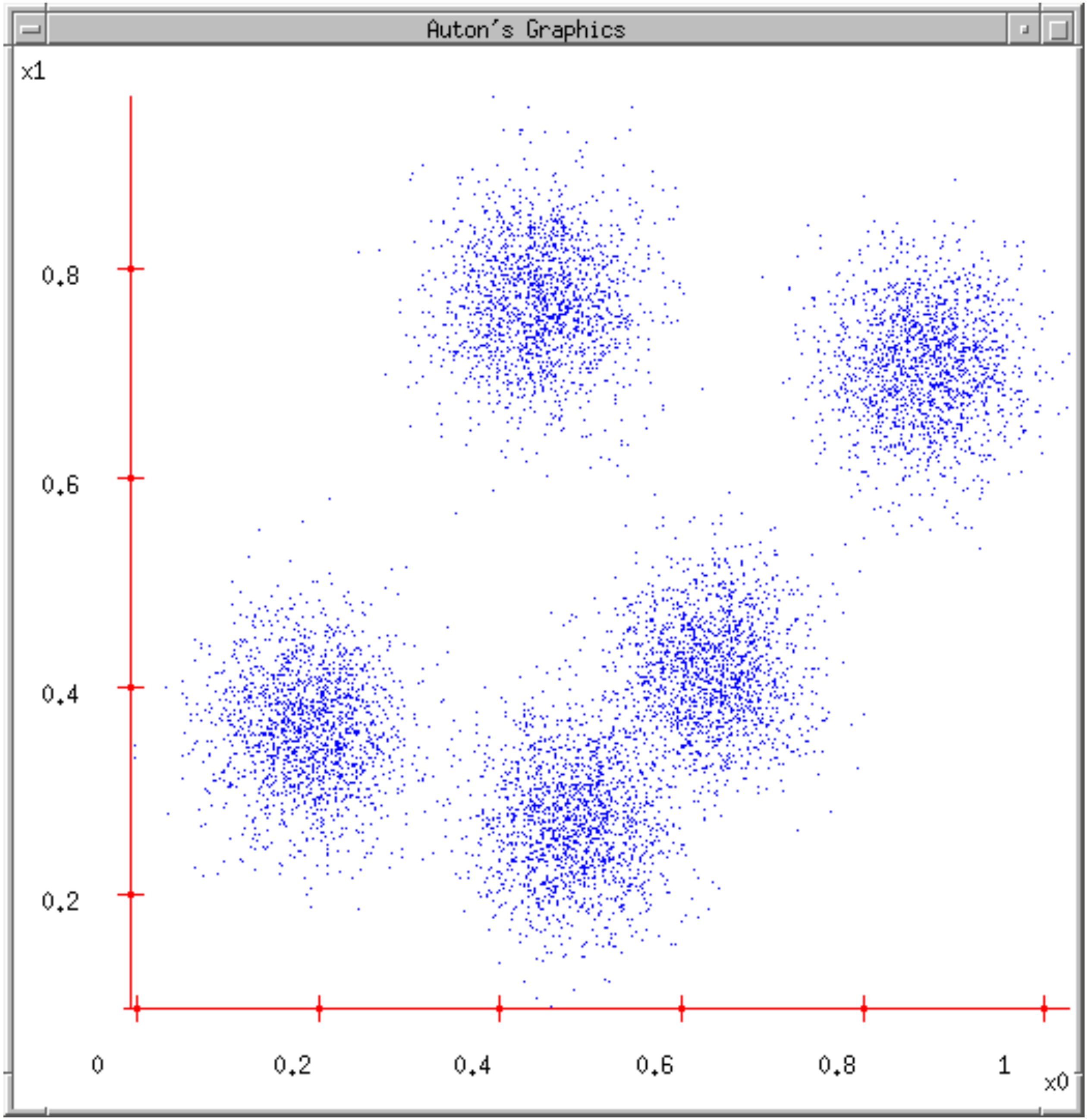


Some Data



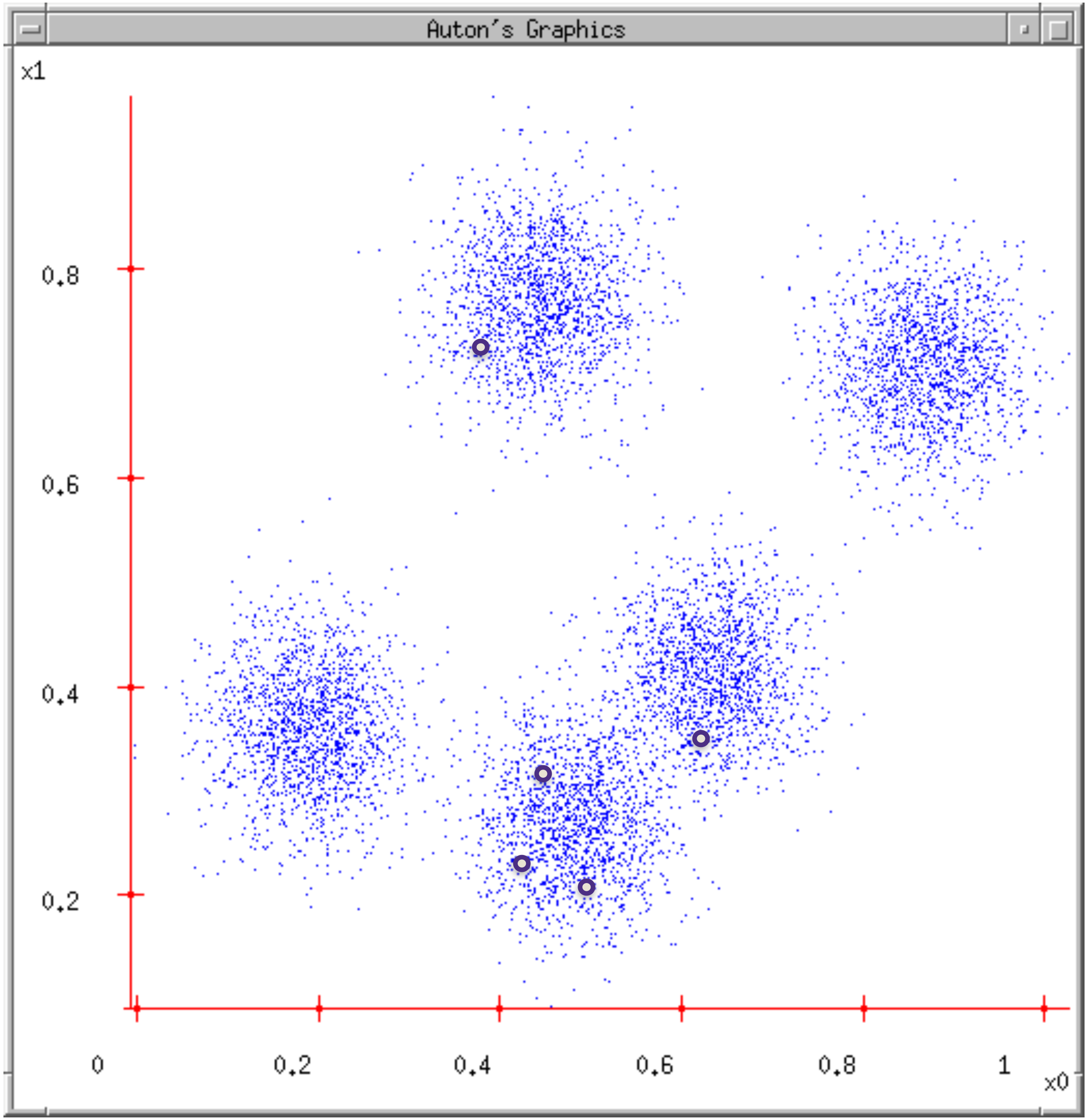
K-means

1. Ask user how many clusters they'd like. (e.g. k=5)



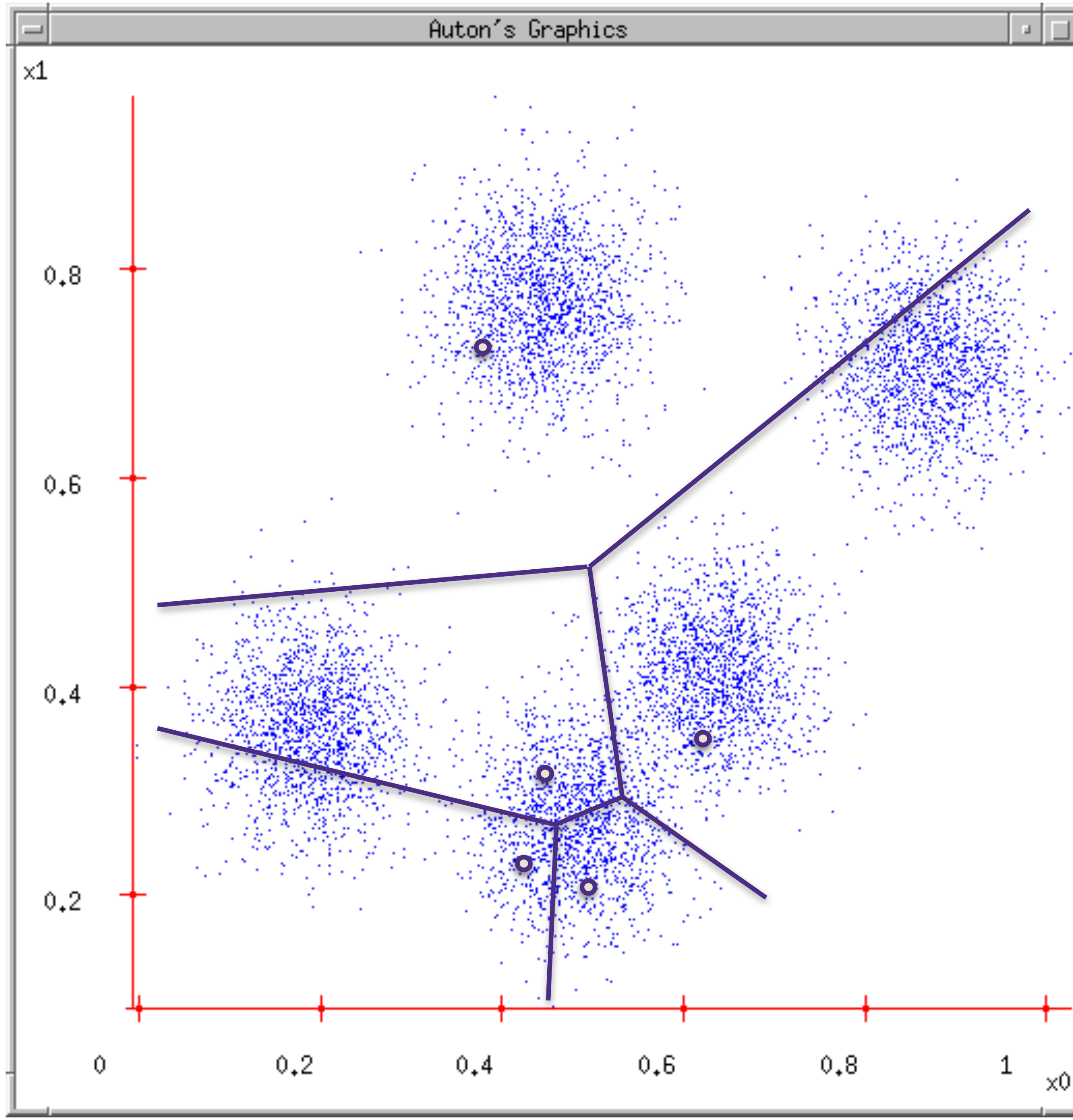
K-means

1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations



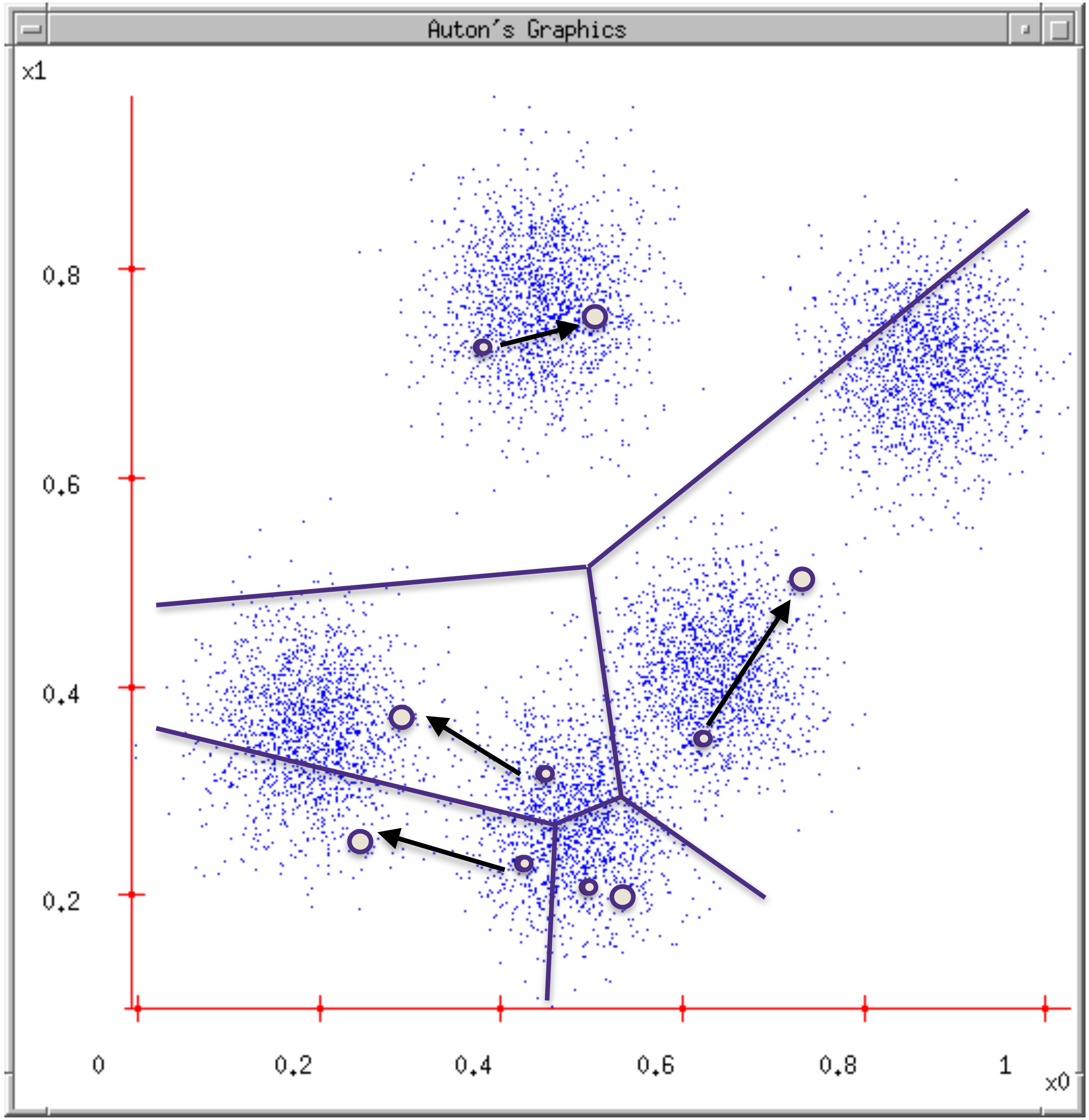
K-means

1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



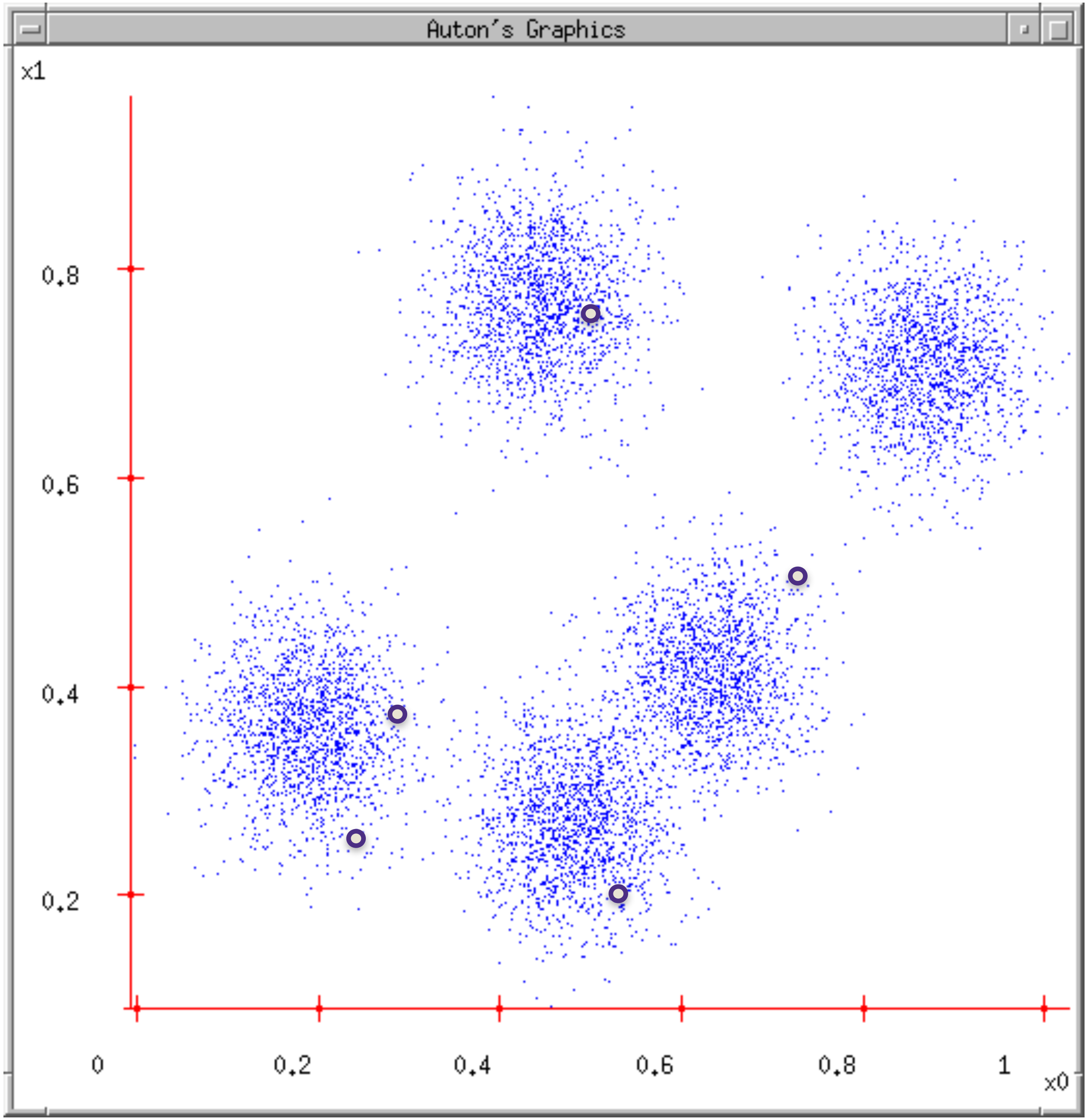
K-means

1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

1. Ask user how many clusters they'd like. (e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



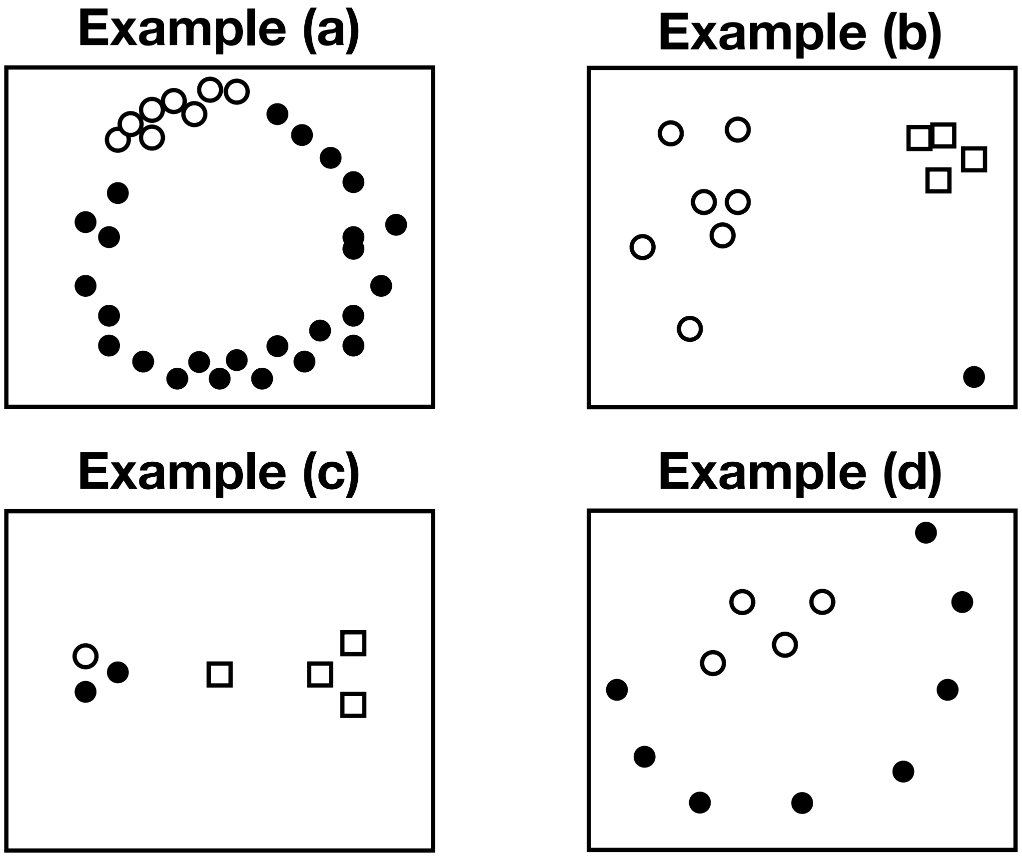
K-means

> **Randomly initialize k centers**
- $\mu(0) = \mu_1(0), \ldots, \mu_k(0)$

> **Classify: Assign each point j∈{1,…N} to nearest center:**
- $$C^{(t)}(j) \leftarrow \arg\min_i ||\mu_i - x_j||^2$$

> **Recenter: μ_i becomes centroid of its point:**
- $$\mu_i^{(t+1)} \leftarrow \arg\min_\mu \sum_{j:C(j)=i} ||\mu - x_j||^2$$
- **Equivalent to $\mu_i \leftarrow$ average of its points!**

Which one is a snapshot of a converged k-means



Does k-means converge??

- k-means is trying to minimize the following objective

- Optimize potential function:

$$\min_{\mu} \min_{C} F(\mu, C) = \min_{\mu} \min_{C} \sum_{i=1}^{k} \sum_{j:C(j)=i} ||\mu_i - x_j||^2$$

- Via alternating minimization
  - Fix μ, optimize C

Does k-means converge??

> k-means is trying to minimize the following objective

> Optimize potential function:

$$\min_{\mu} \min_{C} F(\mu, C) = \min_{\mu} \min_{C} \sum_{i=1}^{k} \sum_{j:C(j)=i} ||\mu_i - x_j||^2$$

> Via alternating minimization
>
> > Fix C, optimize μ

Does k-means converge??

- there is only a finite set of values that $\{C(j)\}_{j=1}^{n}$ can take (k^n is large but finite)
- so there is only finite, k^n at most, values for cluster-centers also
- each time we update them, we will never increase the objective function $\sum_{i=1}^{k} \sum_{j:C(j)=i} \|x_j - \mu_i\|_2^2$
- the objective is lower bounded by zero
- after at most k^n steps, the algorithm must converge (as the assignments $\{C(j)\}_{j=1}^{n}$ cannot return to previous assignments in the course of k-means iterations)

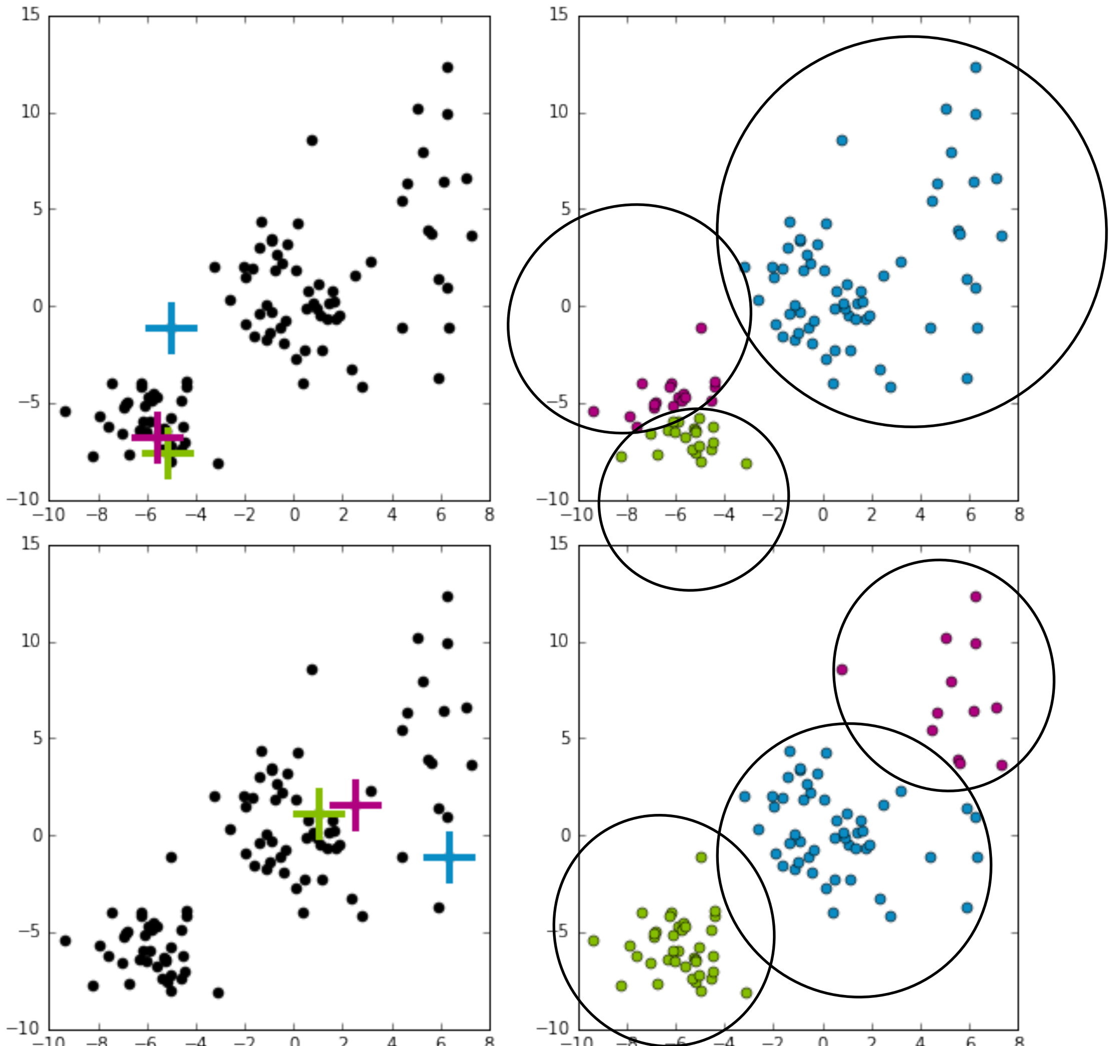
downsides of k-means

- it requires the number of clusters K to be specified by us
- the final solution depends on the initialization
  (does not find global minimum of the objective)

Initial position of centers   final converged assignment

Trial 1

Trial 2



k-means++: a smart initialization

Smart initialization:

1. Choose first cluster center uniformly at random from data points

2. Repeat ***K-1*** times

   3. For each data point x_i, compute distance d_i to nearest cluster center

   4. Choose new cluster center from amongst data points, with probability of x_i being chosen proportional to $(d_i)^2$

- apply standard K-means after the initialization