CSEP 573

Machine Learning II: Unsupervised Learning

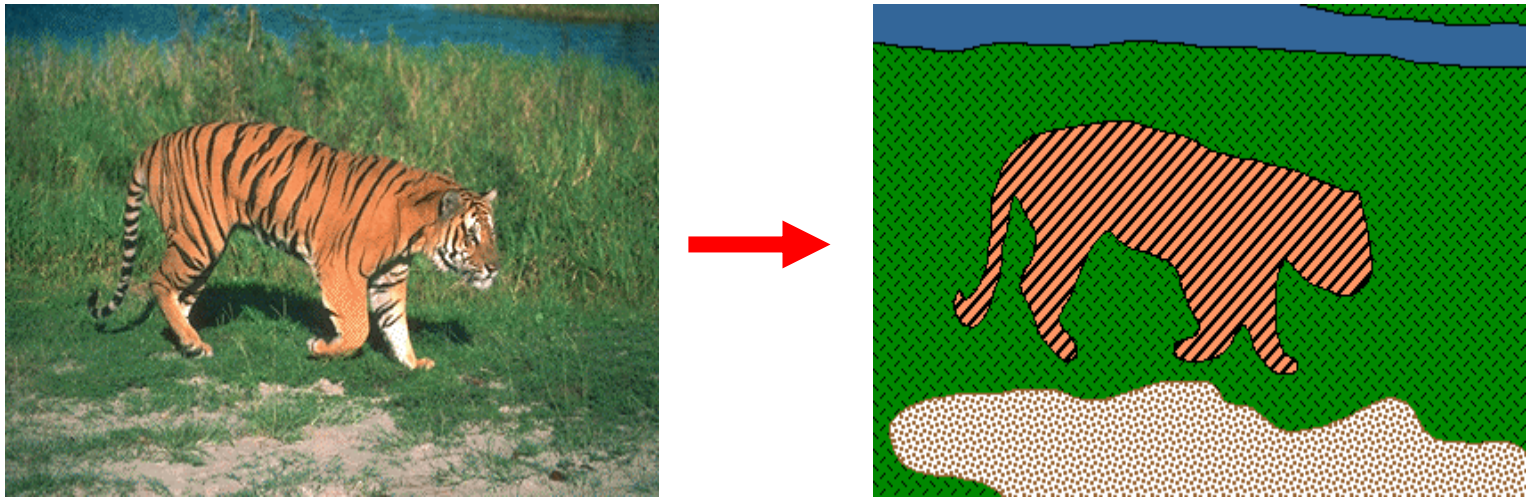
Our agenda today

Unsupervised Learning and Applications

- Clustering
  - Application: Image Segmentation
- Density Estimation and EM algorithm
- Dimensionality Reduction
  - Principal Component Analysis (PCA)
  - Applications: Image Compression, Face Recognition

Guest Lecture by Rawichote Chalodhorn: Applications of Learning in Robotics

Motivation: Image Segmentation in Computer Vision



Goal: Partition an image into its constituent "objects"

Idea: Image histograms

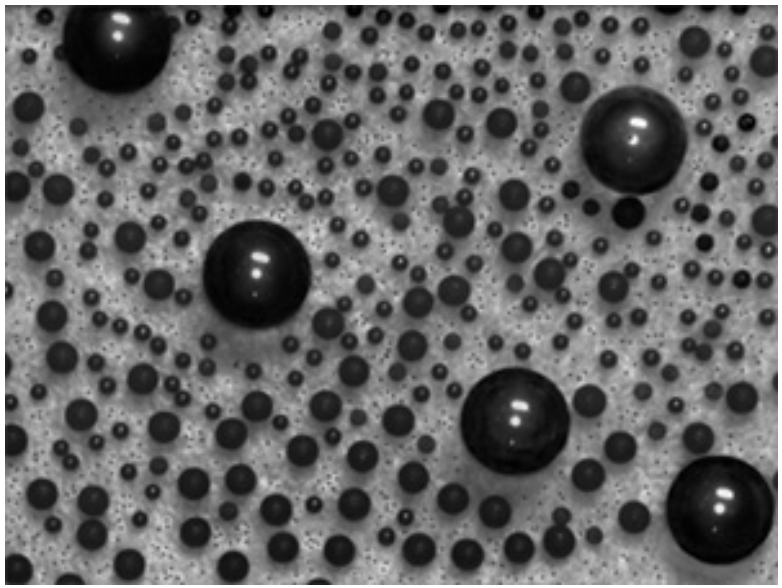
How many "orange" pixels are in this image?

- Look at the *histogram*
- A histogram counts the number of occurrences of each color
  - Given an image $F[x, y] \rightarrow RGB$
  - The histogram is $H_F[c] = |\{(x, y) \mid F[x, y] = c\}|$

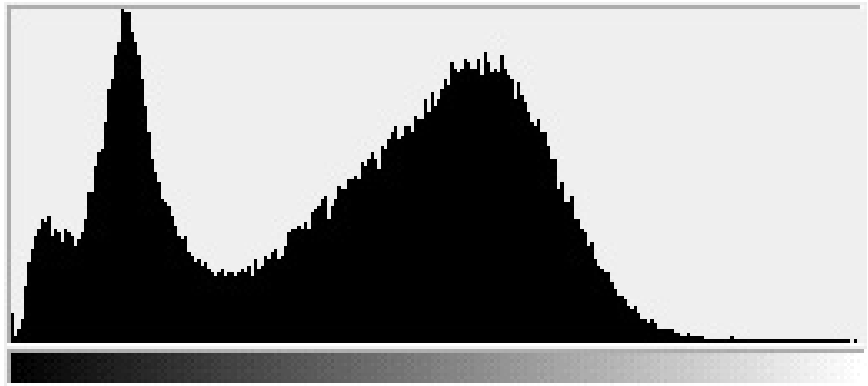
    i.e., for each color value c on the x-axis, plot # of pixels with that color on y-axis

Example Histogram of a Grayscale Image

Image



Histogram



Intensity bins

How Many Modes Are There?

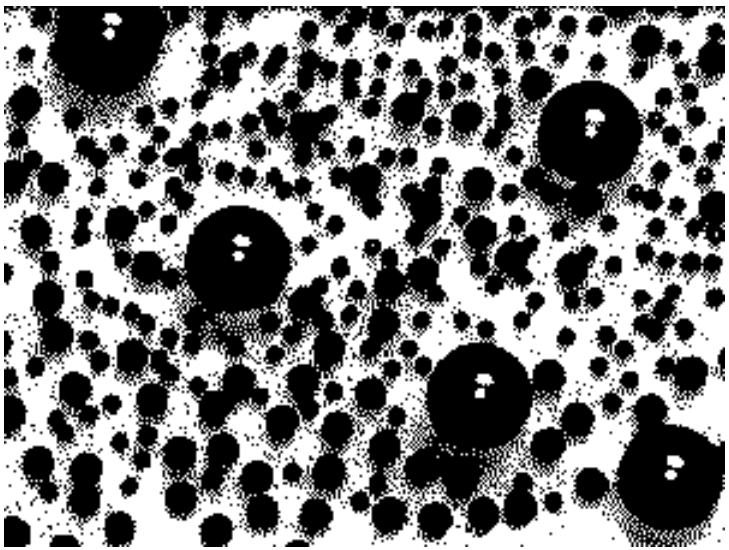
- Easy to see, hard to compute

Histogram-based segmentation

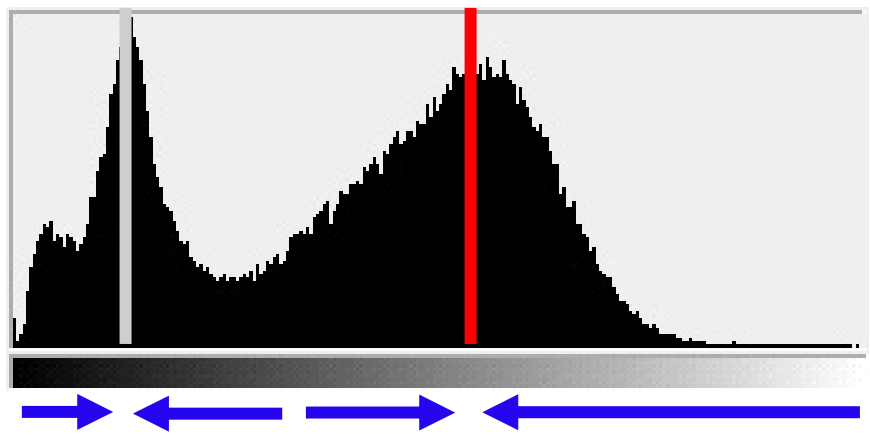
Idea: Break the image into K regions (segments) by

- **reducing the number of colors to K**
- **assigning each pixel to the closest color**

Here's what our image looks like if we use two colors (intensities)

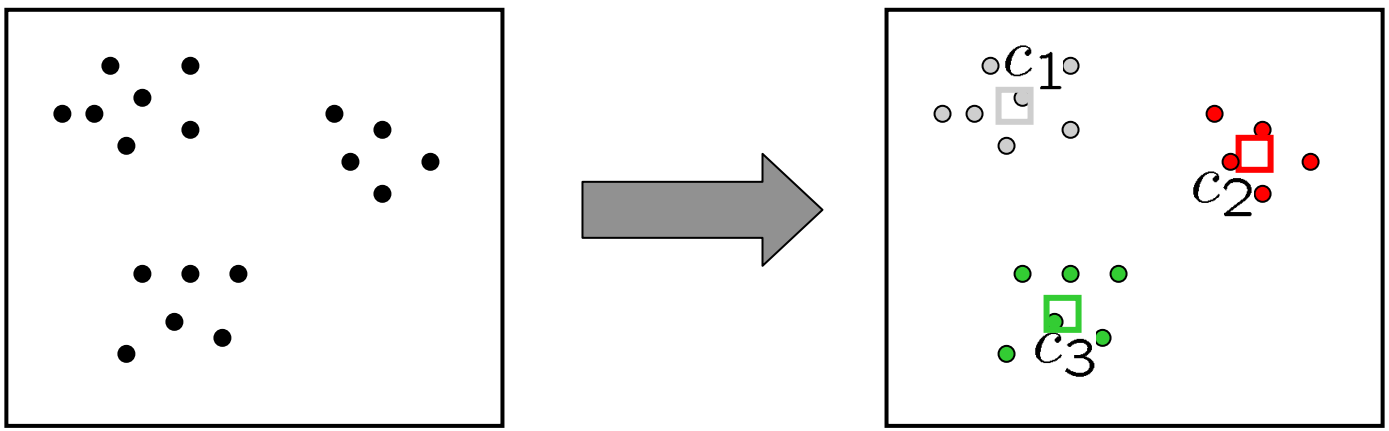


K = 2



Clustering

- Idea in previous slide can be formalized as *clustering*
- Problem: Given unlabeled data points $\{p_1, p_2, \ldots, p_N\}$, assign each data point p_j to one of K clusters
  - points within a cluster are "similar" (according to some metric)
- Example of *unsupervised learning* (no label given)

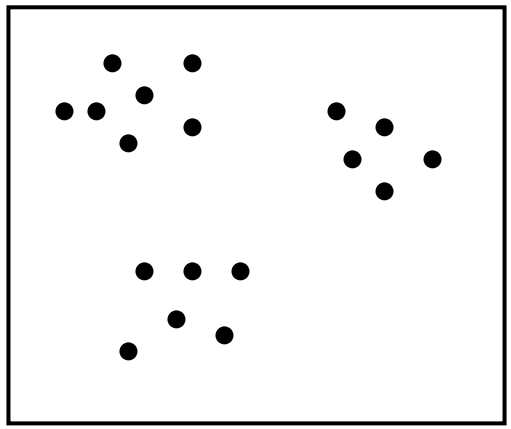


2D data points $\{p_1, p_2, \ldots\}$ K = 3 clusters

Why Clustering?

- **Lots of Applications!**
  - Biology: Discovering gene clusters with similar expression patterns, grouping homologous DNA sequences, etc.
  - Marketing: Grouping customers with similar traits for segmenting the market, product positioning etc.
  - Vision: Image segmentation, feature learning for recognition,…
  - Search result grouping (e.g, clusty.com)
  - Social network analysis (discovering user communities with similar interests)
  - Crime analysis (identification of "hot spots")
  - Many more!

Clustering: The Problem



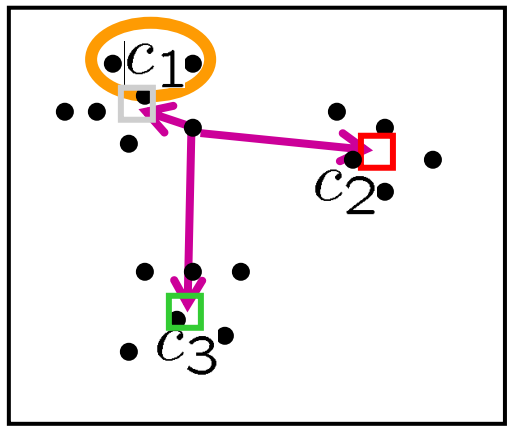
Given: Unlabeled data

Goal: Assign each point to the cluster it is most similar to

Suppose we are given the number of clusters *K* (= 3 here)

How do we assign each point to a cluster?

Suppose you are given the cluster centers c_i

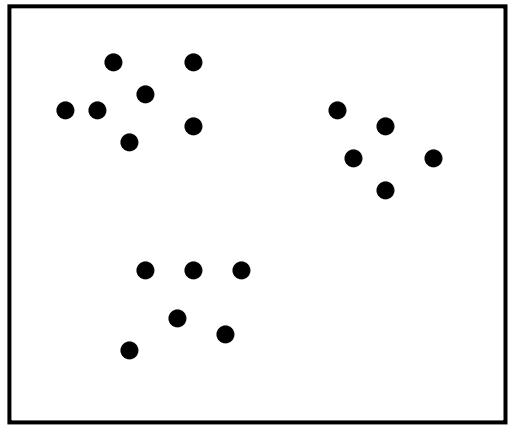


Q: how do you assign points to a cluster?

A: for each point p,

- Compute distance to cluster centers
- Choose the closest c_i

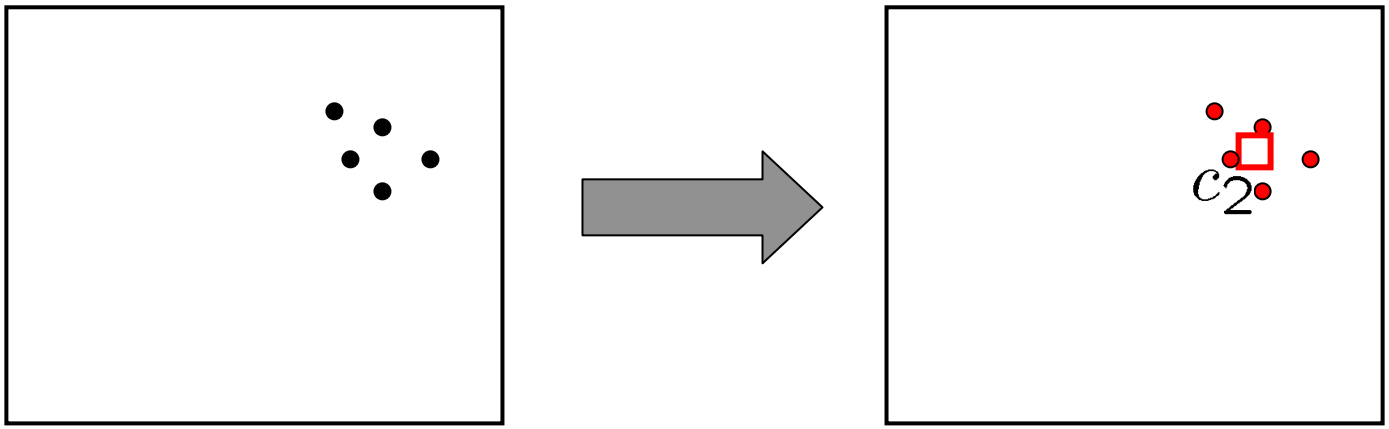
Suppose you are given the cluster centers c_i



But wait…you are not given the cluster centers!

How do you find them?

Finding the cluster center



Given a cluster of points, we can easily compute its center

(How?)

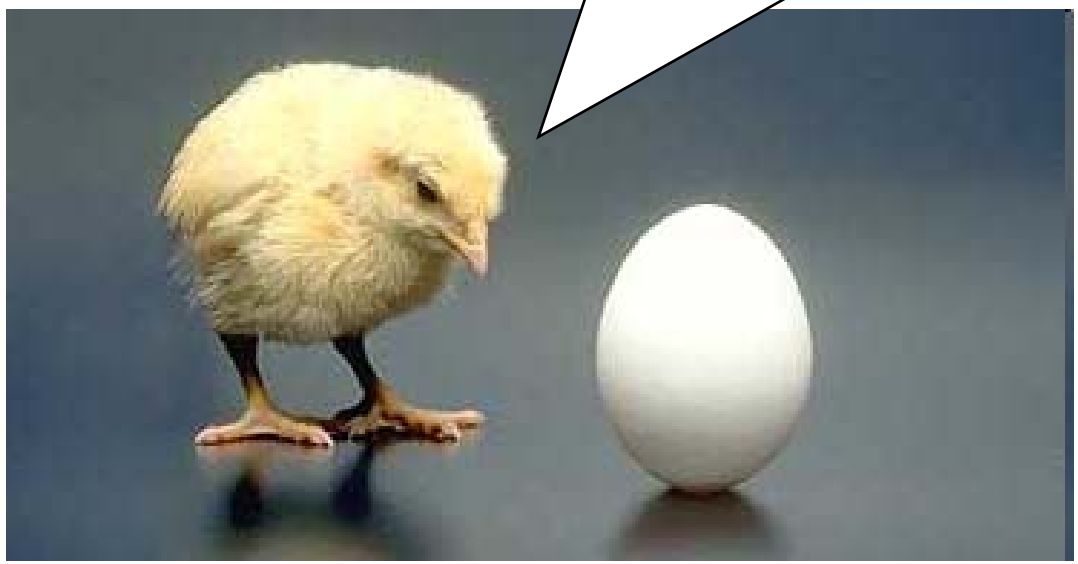
A chicken-or-egg problem?

Given cluster centers, we can assign points

To find centers, we need points assigned to a cluster

A way out of the impasse

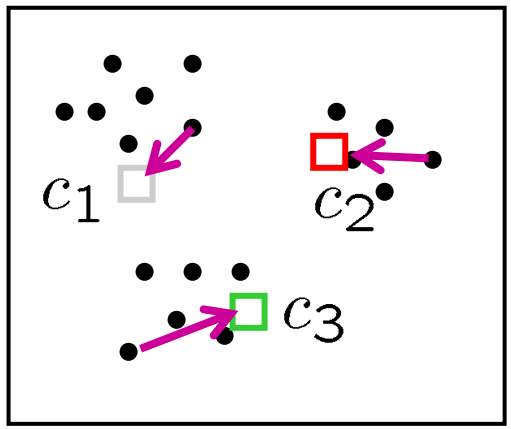
Why not alternate?
(between finding centers and assigning points)



Alternate between 2 steps

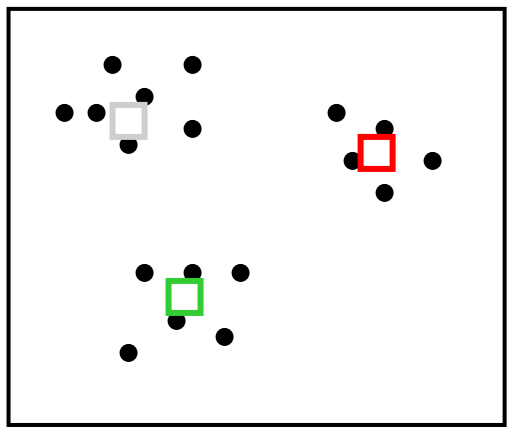
I. Given current estimate of cluster centers c_i:

Assign each point p to closest c_i



II. Given current assignment of points to clusters:

Choose c_i to be the mean of all the points in the cluster



K-means clustering

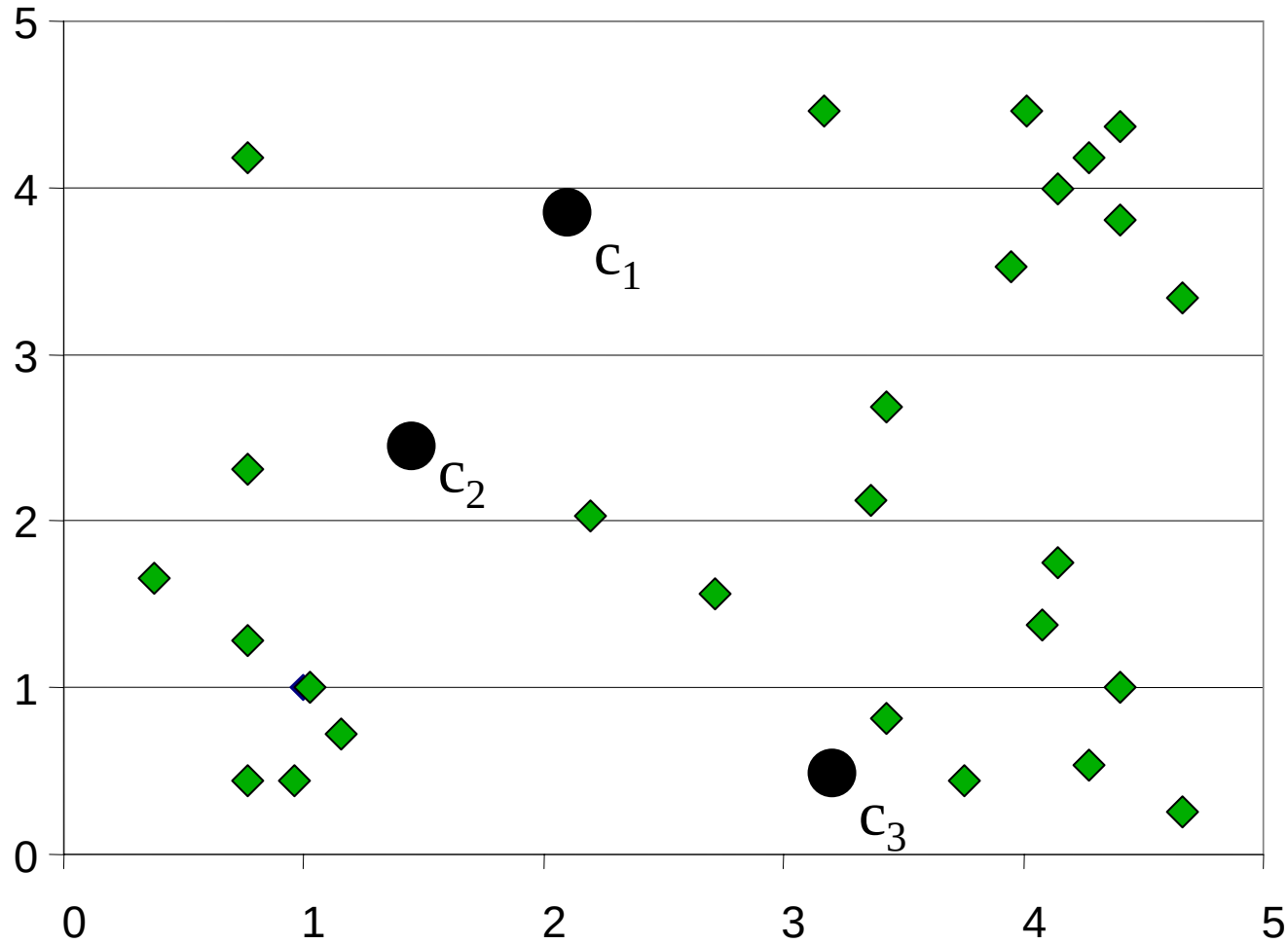
Algorithm

1. Randomly initialize the cluster centers, $c_1, \ldots, c_K$
2. Determine cluster membership
   - For each point p, find the *closest* c_i
   - Put p into cluster i
3. Re-estimate cluster centers
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, go to 2 else done.

Java demo: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

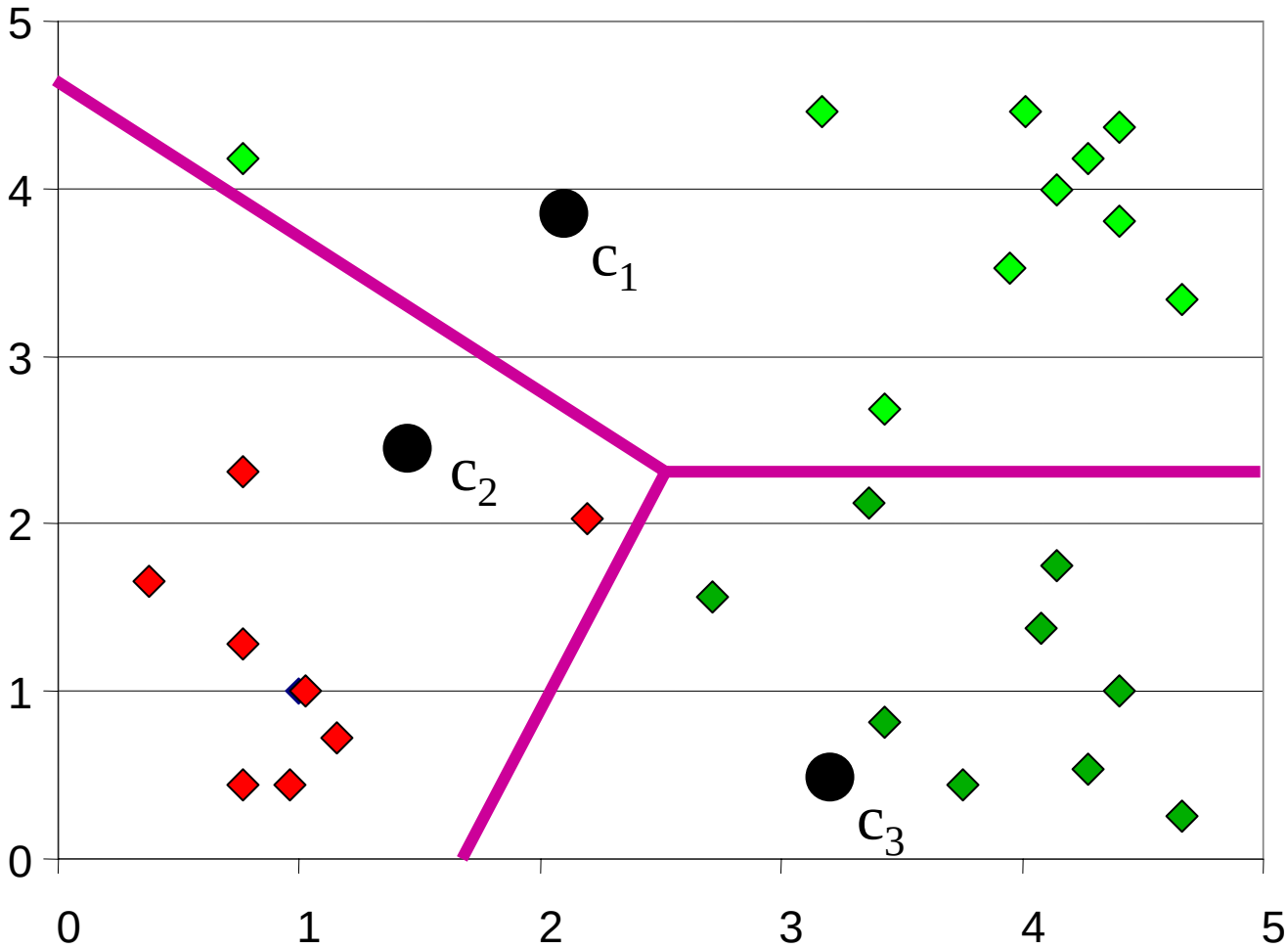
K-means clustering example

Randomly initialize the cluster centers



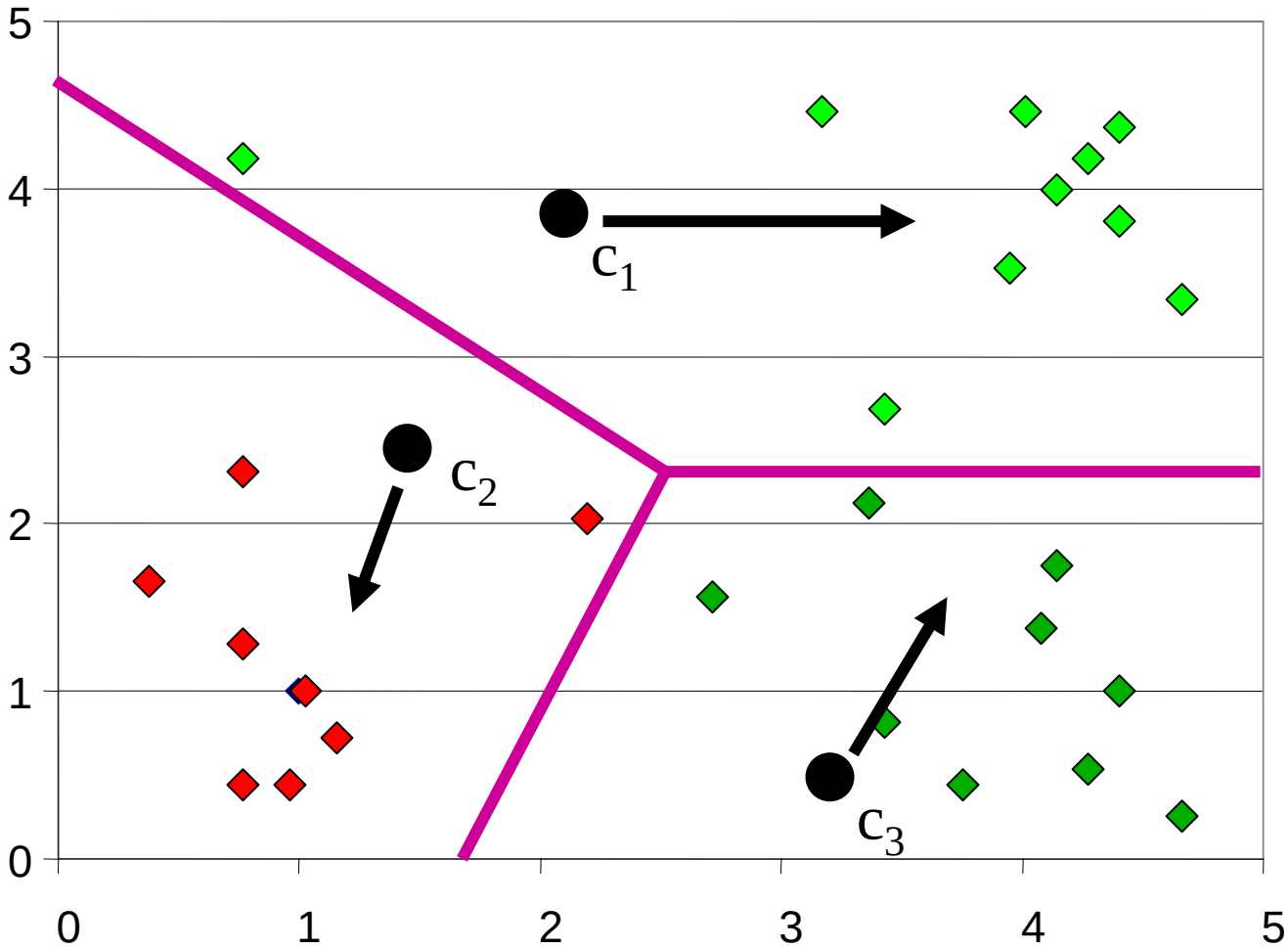
K-means clustering example

Determine cluster membership



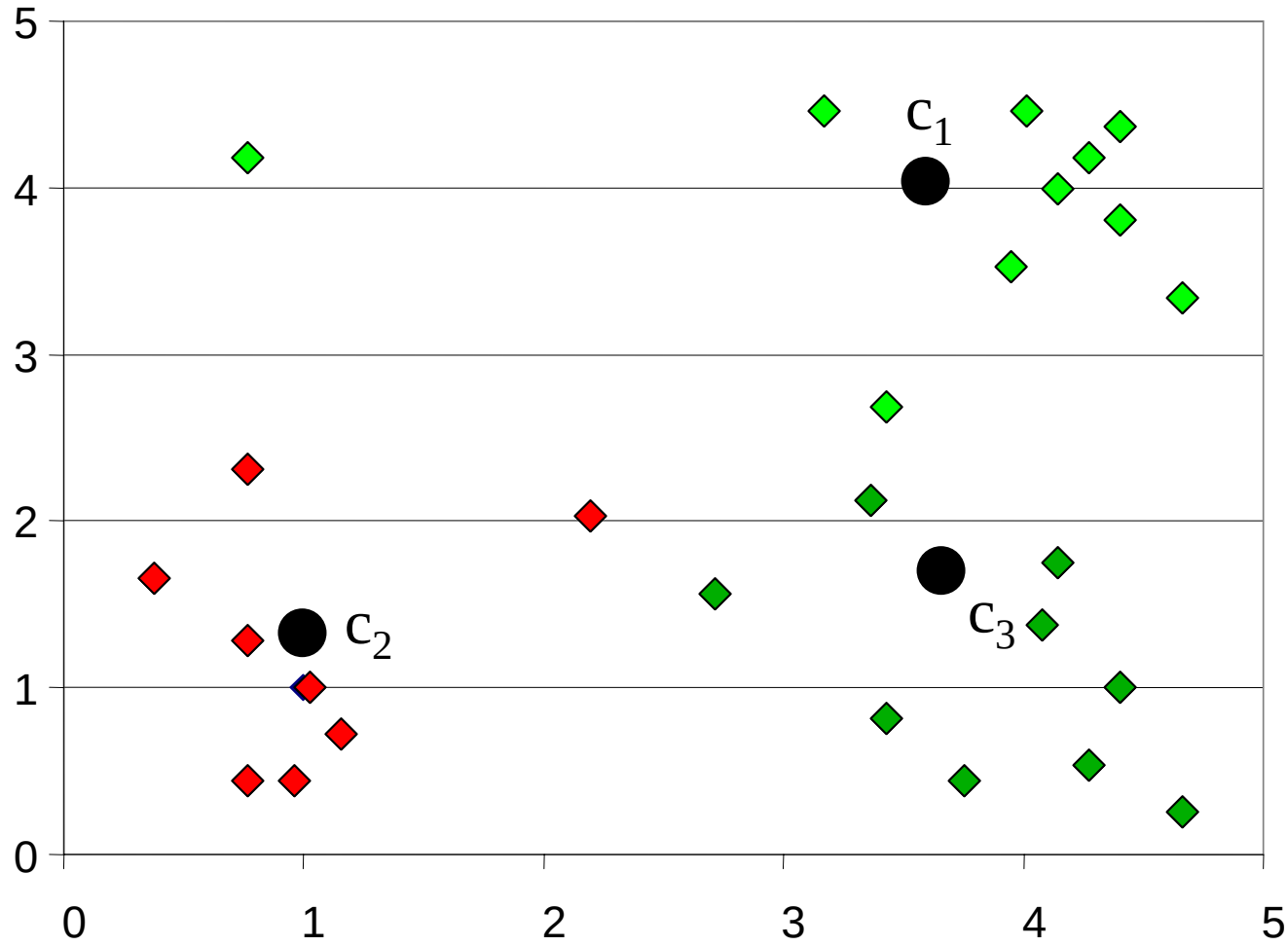
K-means clustering example

Re-estimate cluster centers



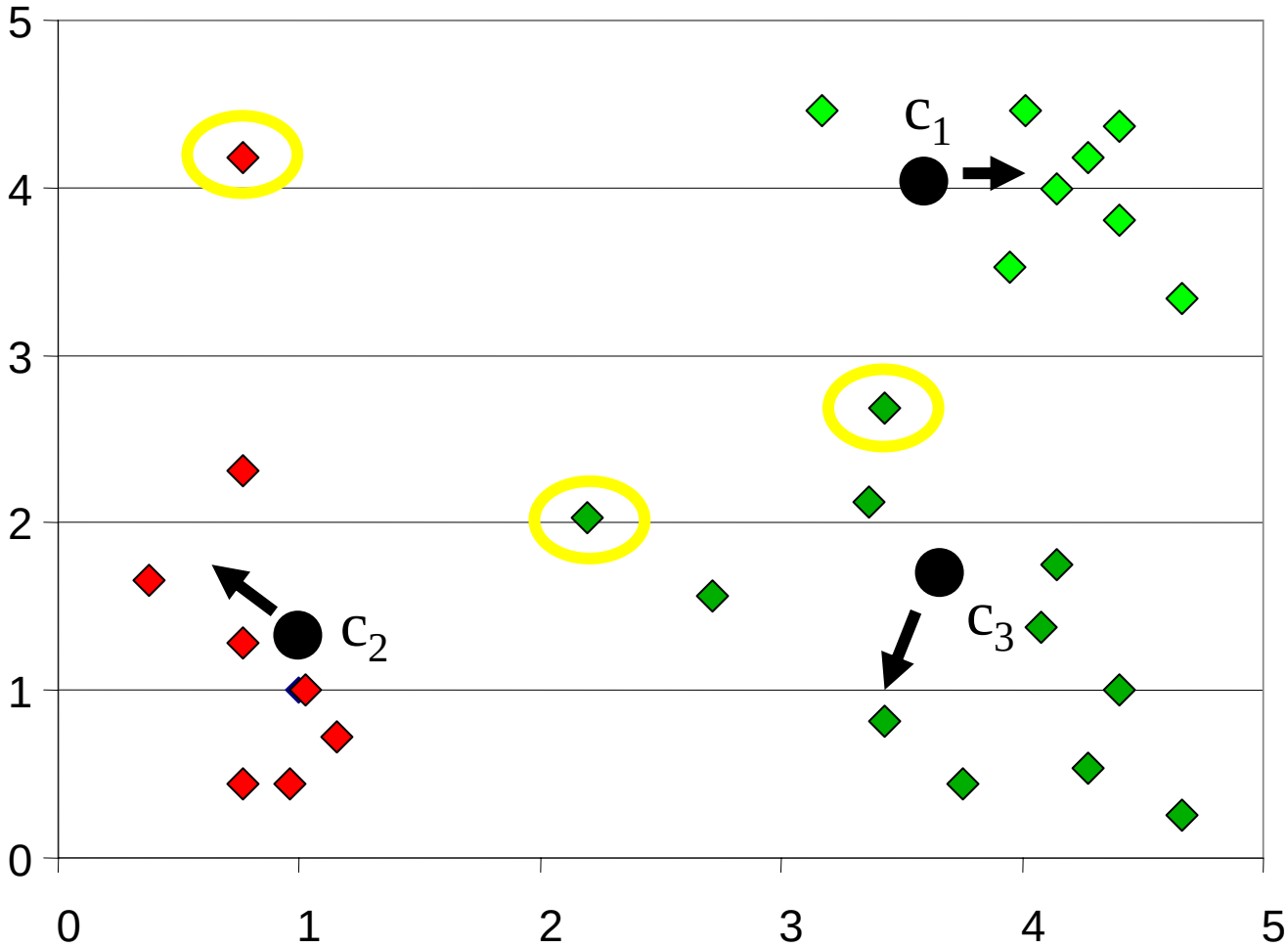
K-means clustering example

Result of first iteration



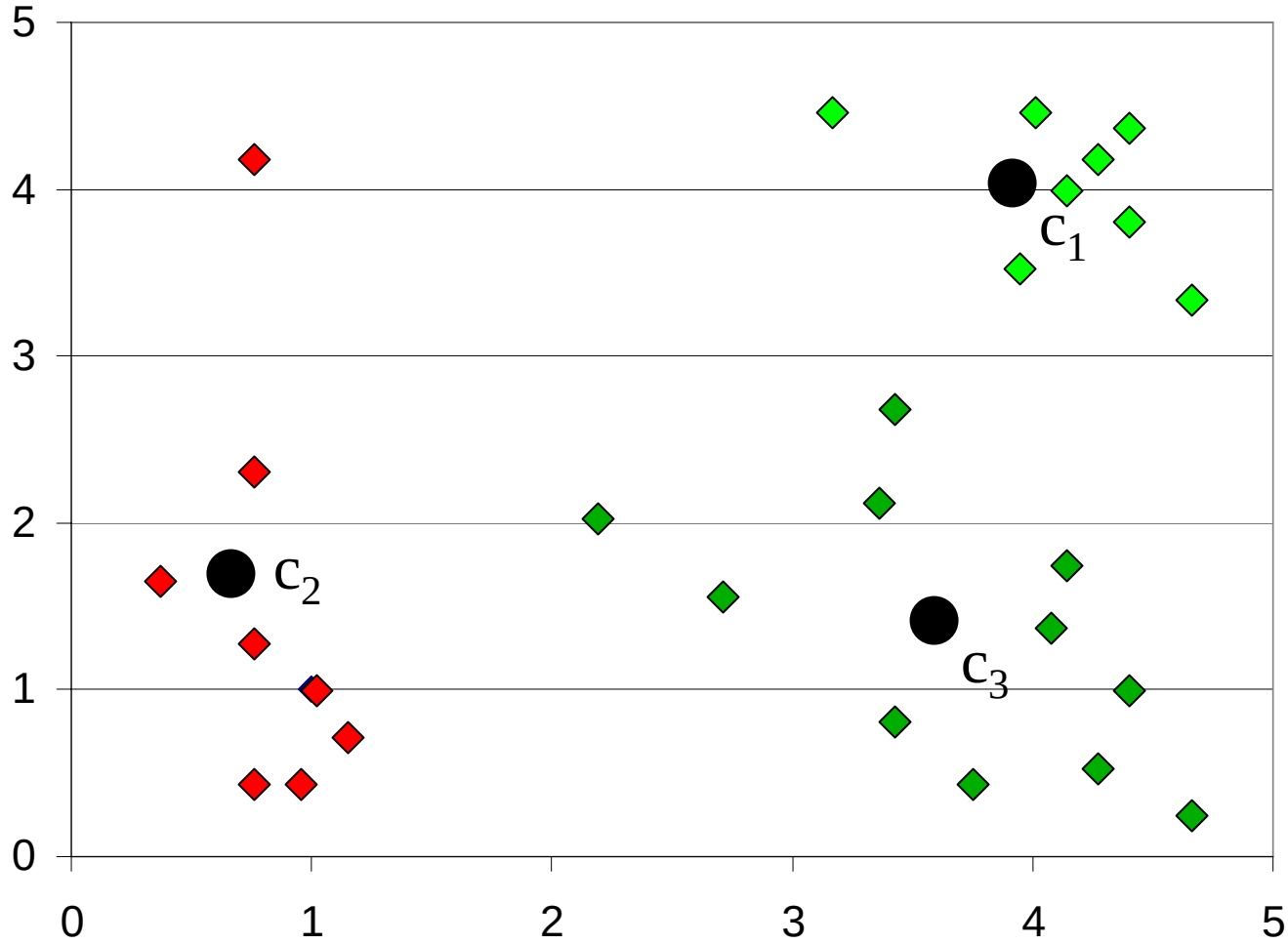
K-means clustering example

Second iteration



K-means clustering example

Result of second iteration



K-means clustering

Properties

- Will always converge to *some* solution
- Can be a "local minimum"
  - does not always find the global minimum of objective function:

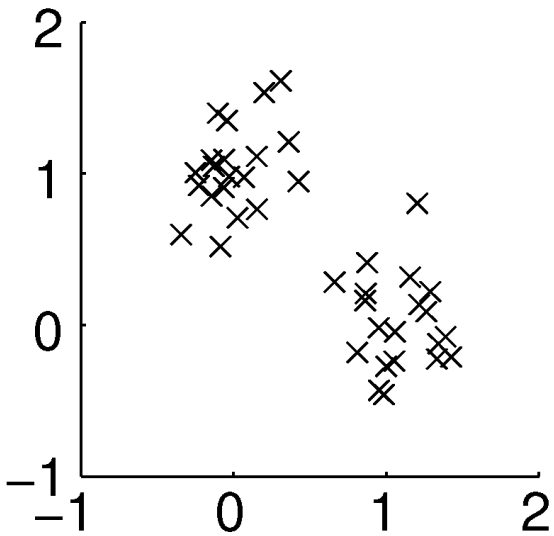
$$\sum_{\text{clusters } i} \quad \sum_{\text{points p in cluster } i} \|p - c_i\|^2$$

K-means as probability density estimation

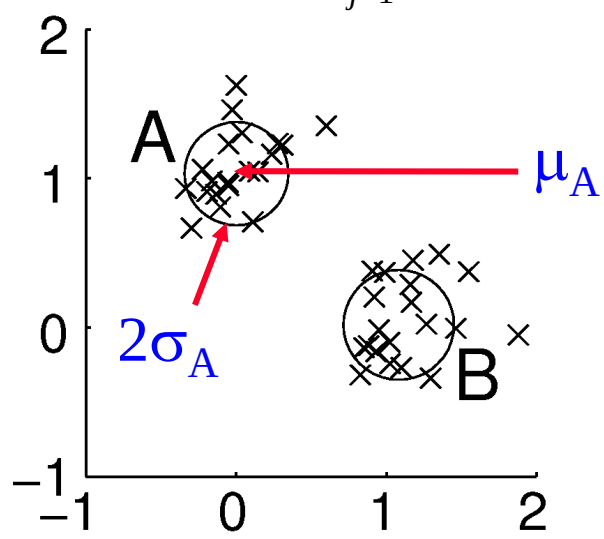
K-means can be formalized as estimating the unknown probability density of a data set

- Model data as a *mixture of K Gaussians*
- Estimate not only means but also (co)variances

Data



Model $p(\mathbf{x}) = \sum_{j=1}^{K} p(C_j) G(\boldsymbol{\mu}_j, \Sigma_j)$



K-Means and the EM Algorithm

- The Expectation Maximization (EM) Algorithm is a general algorithm for unsupervised learning when there are hidden variables (e.g., clusters, non-evidence nodes in Bayesian networks, etc.)
- Like K-means, it involves iterating between 2 steps:
  - E ("expectation") step that estimates posterior probabilities of hidden variables
  - M ("maximization") step that uses the result of E step to update model parameters
- Each iteration improves likelihood of data under the model (or keeps it the same)
- Guaranteed to converge (perhaps to local maximum)

Not to be confused with…

The Expectation Minimization Algorithm

Density estimation using EM

- EM for Gaussian mixtures (similar to K-means):
  - Initialize K clusters: C_1, …, C_K

    (μ_j, Σ_j) and $p(C_j)$ for each cluster j
  - Repeat until convergence:
    - Estimate which cluster each data point belongs to

      $p(C_j \mid x_i)$ ⇒ Expectation step
    - Re-estimate cluster parameters

      $(\mu_j, \Sigma_j), p(C_j)$ ⇒ Maximization step

EM algorithm: The details

E step: Compute probability of membership in cluster based on output of previous M step ($p(x_i|C_j) = \text{Gaussian}(\mu_j, \Sigma_j)$)

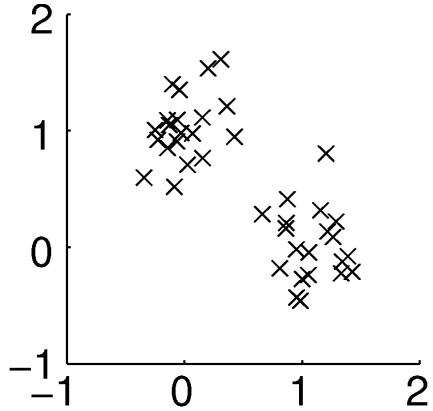
$$p(C_j \mid x_i) = \frac{p(x_i \mid C_j) \cdot p(C_j)}{p(x_i)} = \frac{p(x_i \mid C_j) \cdot p(C_j)}{\sum_j p(x_i \mid C_j) \cdot p(C_j)}$$

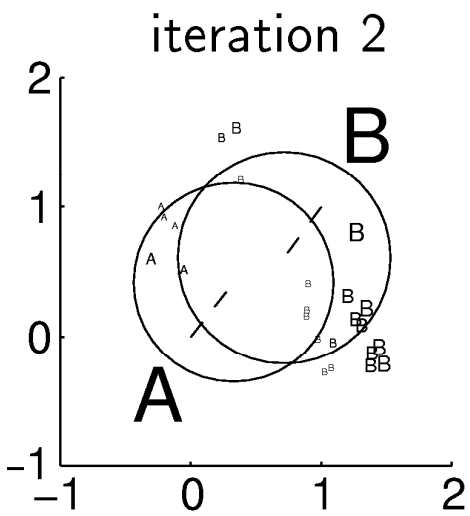
M step: Re-estimate cluster parameters based on output of E step

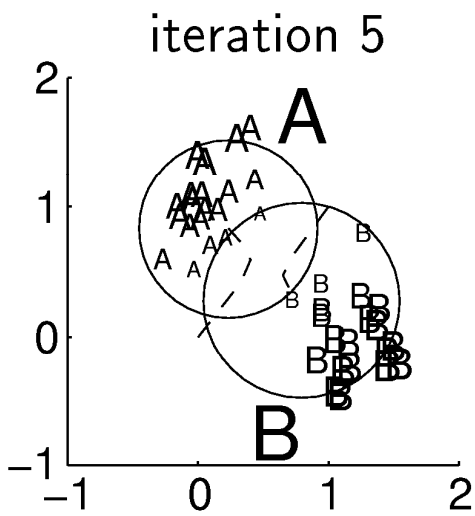
$$\mu_j = \frac{\sum_i p(C_j \mid x_i) \cdot x_i}{\sum_i p(C_j \mid x_i)} \qquad \Sigma_j = \frac{\sum_i p(C_j \mid x_i) \cdot (x_i - \mu_j) \cdot (x_i - \mu_j)^T}{\sum_i p(C_j \mid x_i)} \qquad p(C_j) = \frac{\sum_i p(C_j \mid x_i)}{N}$$

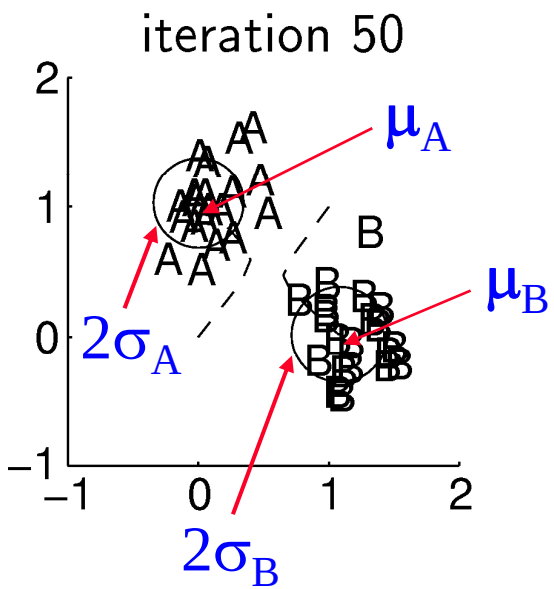
Results from the EM algorithm

Input data:









Suppose we are not interested in density estimation but want to reduce the dimensionality of our data

Example application: Image compression

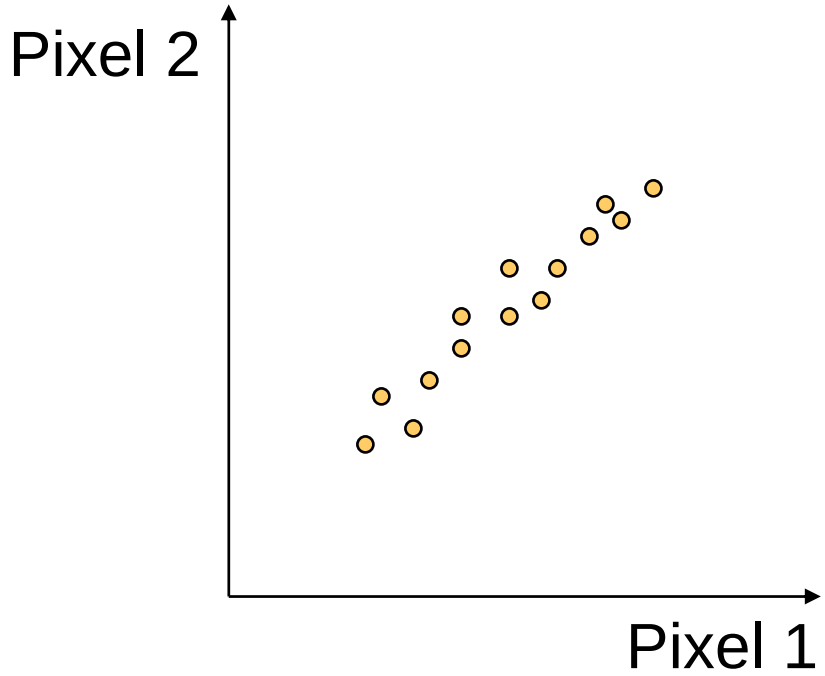
Redundancy in Images

Most natural images (e.g., images of faces) are highly redundant

- Nearby pixels tend to have similar intensities and are therefore highly correlated
- Why?
- Due to physical regularities of natural structures generating the image

Can we use unsupervised learning to capture and reduce this redundancy?

A Simple 2D Example

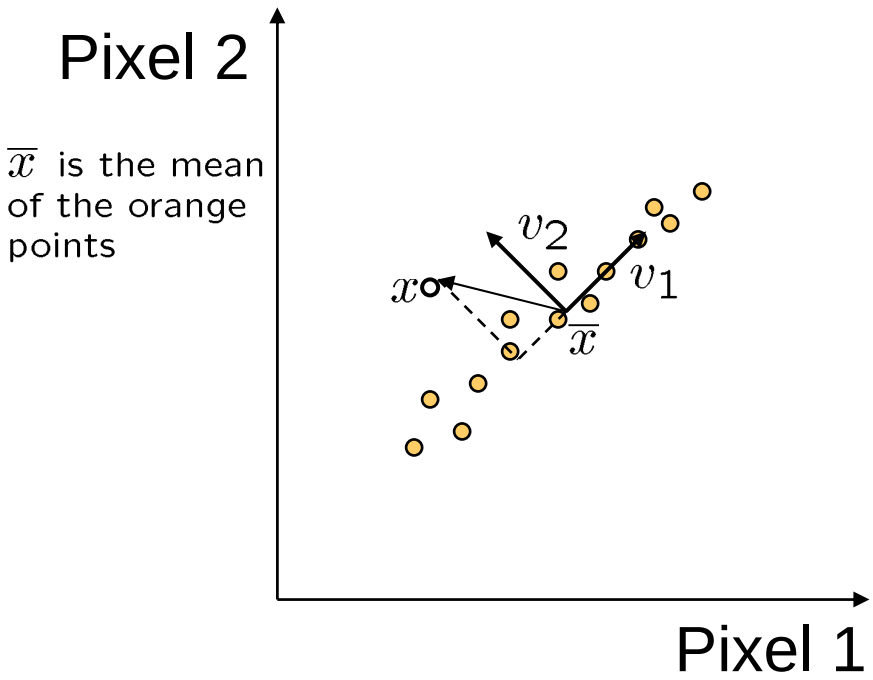


Suppose pixel 1 and pixel 2 are two neighboring pixels

- What does the plot above suggest?
- The two pixels are highly correlated for this data set of images

This and next few slides adapted from Steve Seitz, Linda Shapiro

Linear subspaces



Suppose we fit a line v_1
Let v_2 be orthogonal to v_1

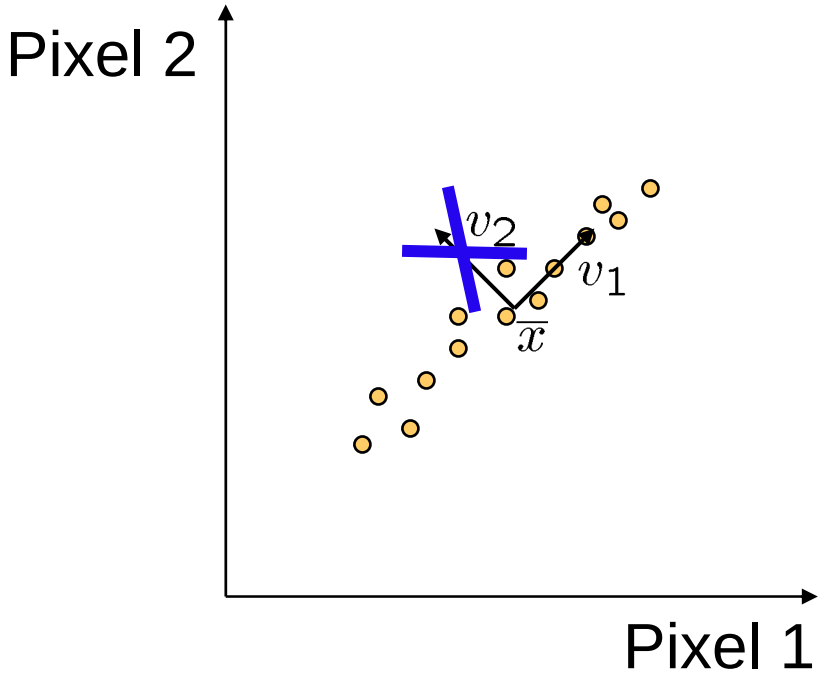
Convert an input **x** into $\mathbf{v_1}$, $\mathbf{v_2}$ coordinates

$$\mathbf{x} \rightarrow ((\mathbf{x} - \overline{x}) \cdot \mathbf{v_1}, (\mathbf{x} - \overline{x}) \cdot \mathbf{v_2})$$

What does the $\mathbf{v_1}$ coordinate measure?
- position along v_1 axis
- use it to specify which point it is

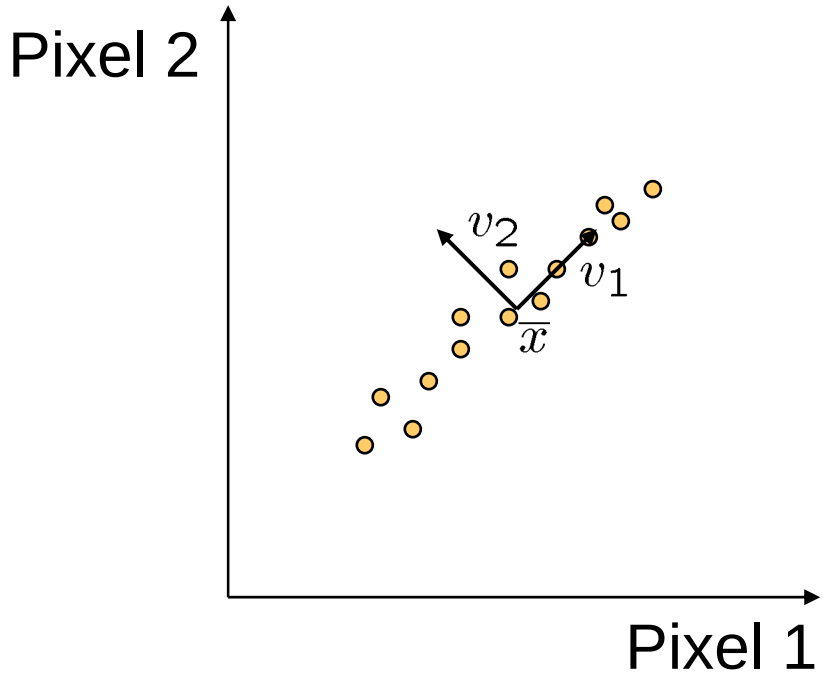
What does the $\mathbf{v_2}$ coordinate measure?
- distance to line (position along v_2 axis)
- near 0 for these pts

Dimensionality reduction



- We can represent the points with *only* their $\mathbf{v_1}$ coordinates
  - since $\mathbf{v_2}$ coordinates are all essentially 0
- Reduce dimensionality of data from 2D to 1D
- This makes it cheaper to store and compare points
- Bigger deal for higher dimensional inputs (like images!)

How do we find v_1, v_2, … ?



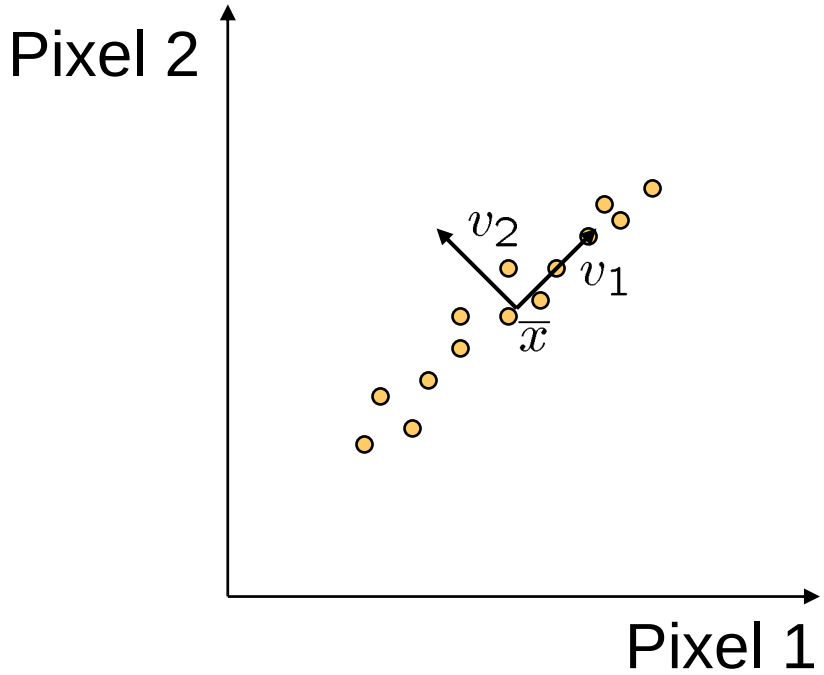
Consider the variation along some direction **v** for all of the *N* points:

$$var(\mathbf{v}) = 1/N \sum_{\text{orange point } \mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \ \mathbf{v}\|^2$$

What unit vector **v** maximizes *var*? $\mathbf{v}_1 = \underset{\mathbf{v}}{arg\,max} \{var(\mathbf{v})\}$

$\mathbf{v}_2$ is then the unit vector orthogonal to $\mathbf{v}_1$

How do we find v_1, v_2, … ?

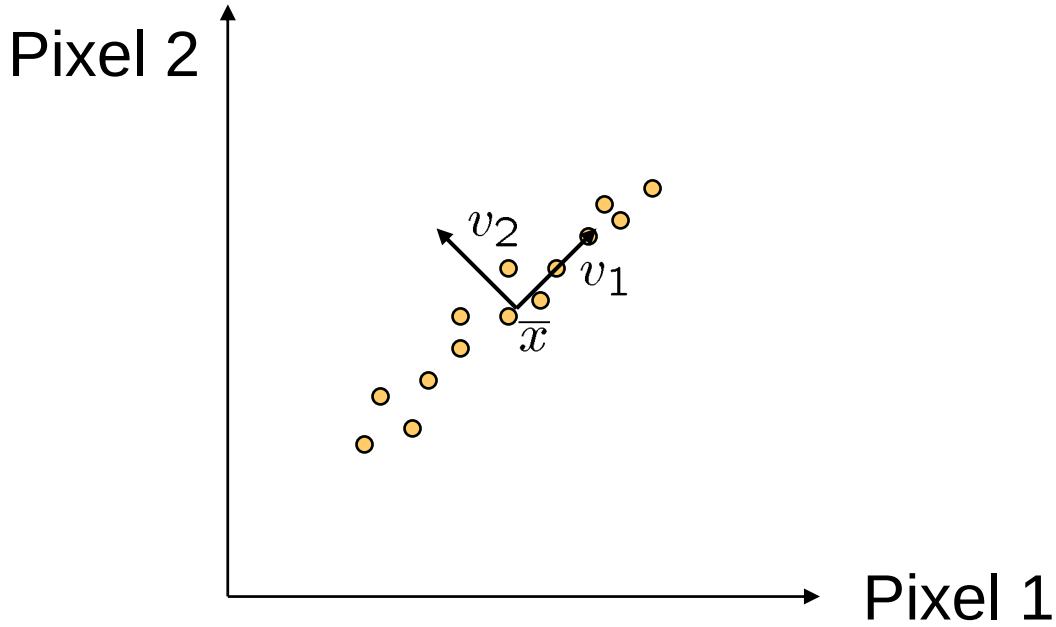


$$
\begin{aligned}
var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x}-\overline{\mathbf{x}})^{\mathrm{T}}\ \mathbf{v}\|^2 /N \\
&= \sum_{\mathbf{x}} \mathbf{v}^{\mathrm{T}}(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathrm{T}}\mathbf{v}\ /N \\
&= \mathbf{v}^{\mathrm{T}}\left[\sum_{\mathbf{x}}(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathrm{T}}\right]\mathbf{v}\ /N \\
&= \mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}}(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathrm{T}}/N
\end{aligned}
$$

A = Covariance matrix of data points

We want to find a unit vector **v** that maximizes $\mathbf{v}^{\mathrm{T}}\mathbf{A}\ \mathbf{v}$

Finding v_1 and v_2: The Math



$\mathbf{v_1} = \text{argmax}_{\mathbf{v}} (\mathbf{v}^T \mathbf{A}\, \mathbf{v})$ subject to $\mathbf{v}^T\mathbf{v} = 1$

Using Lagrange multiplier method,

$\mathbf{v_1} = \text{argmax}_{\mathbf{v}} [\mathbf{v}^T \mathbf{A}\, \mathbf{v} - \lambda(\mathbf{v}^T\mathbf{v} - 1)]$

Setting derivative wrt $\mathbf{v}$ to 0, we get:

$\mathbf{Av} = \lambda\mathbf{v}$ Thus, $\mathbf{v_1}$ is eigenvector of $\mathbf{A}$ with *largest* eigenvalue λ_1

$\mathbf{v_2}$ is eigenvector of $\mathbf{A}$ with *smaller* eigenvalue λ_2

Principal Component Analysis (PCA)

Suppose each of the N data points is L-dimensional

- Form L x L data covariance matrix *A*

$$A = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T$$

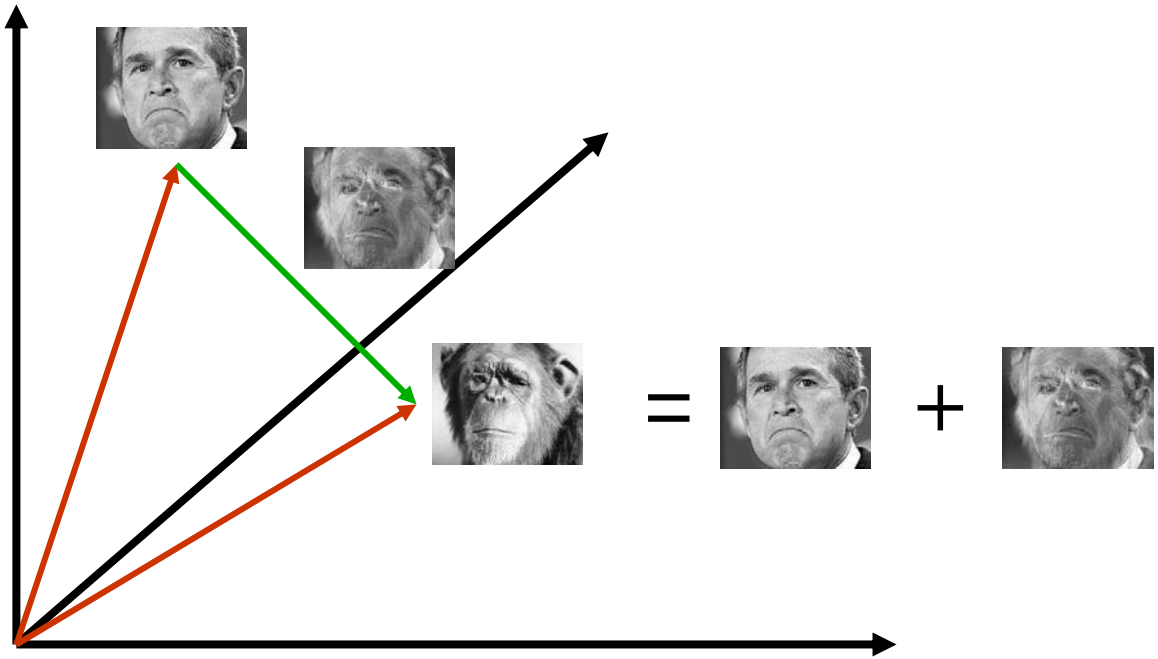
- Compute eigenvectors of *A*
  - Eigenvectors of *A* define a new coordinate system that is a rotation of the original coordinate system
  - Eigenvector with largest eigenvalue captures the most variation among training vectors x
  - Eigenvector with smallest eigenvalue has least variation

Principal Component Analysis (PCA)

We can compress the data by only using the top few eigenvectors with largest eigenvalues

- corresponds to choosing a "linear subspace" of the original data space
- represent points on a line, plane, "hyper-plane"
- these eigenvectors are known as *principal component vectors*
- procedure is known as *Principal Component Analysis (PCA)*

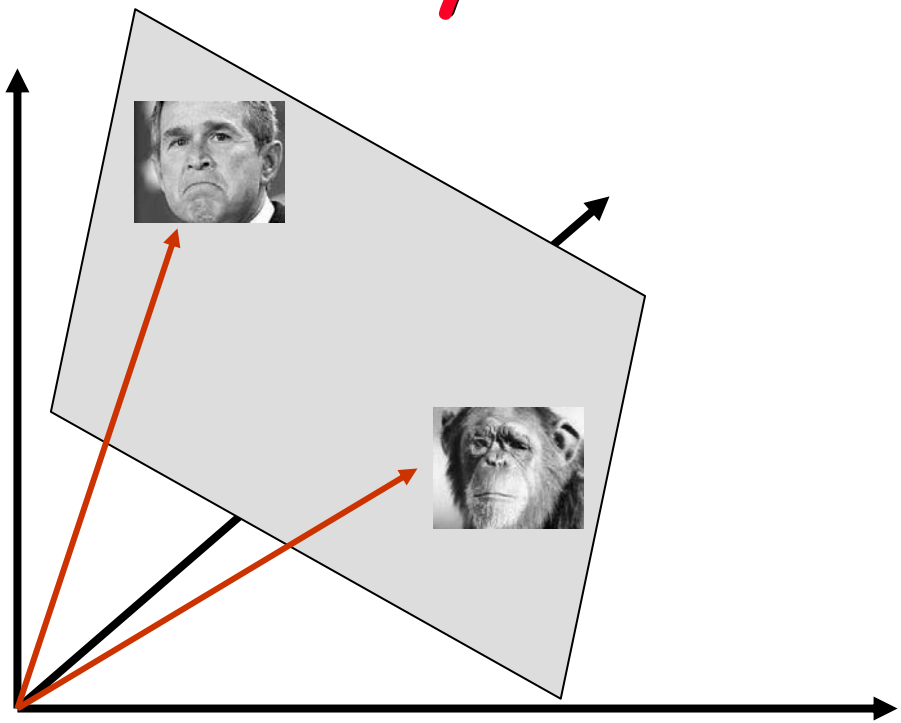
Back in the vector space of faces…



An image is a point in a high dimensional space

- A P x Q pixels image is a point in R^{PQ}
- Vectors in this space behave similarly to the data points in our 2D case

Dimensionality reduction



The space of all faces is a "subspace" of the space of all images

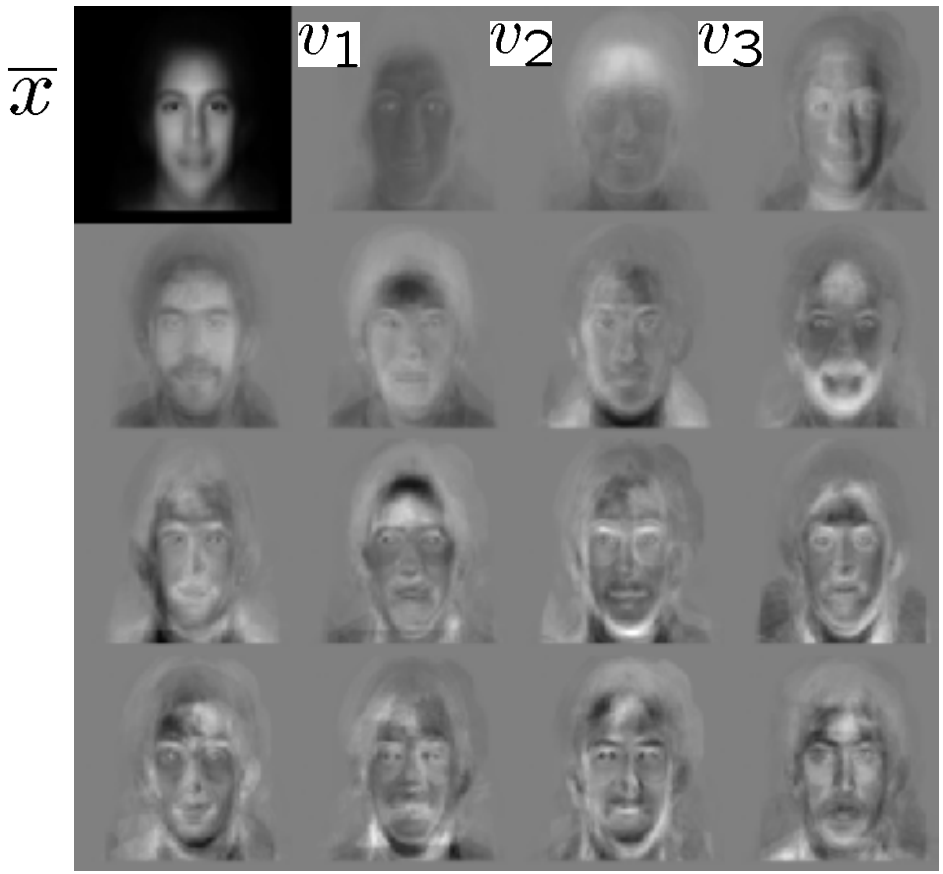
- Suppose this subspace is K dimensional, K << PQ
- We can find such a subspace using PCA
- This is like fitting a "hyper-plane" to the set of faces
  - spanned by eigenvectors $v_1, v_2, \ldots, v_K$
  - any face $\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_k$

Eigenfaces

PCA extracts the eigenvectors v_1, v_2, v_3, ... v_K of covariance matrix A

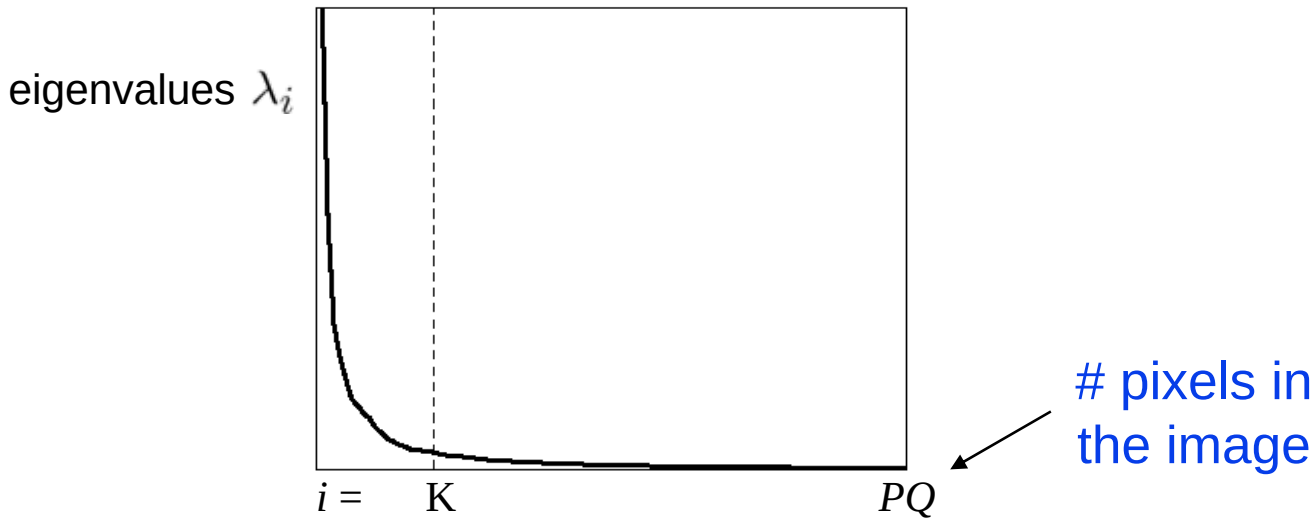
Each one of these vectors is a direction in face space

- what do these look like?



The eigenvectors for face images are called "eigenfaces"

Choosing the reduced dimension K



How many eigenfaces to use?

Look at the decay of the eigenvalues

- the eigenvalue tells you the amount of variance "in the direction" of that eigenface
- ignore eigenfaces with low variance

Application 1: Image Compression

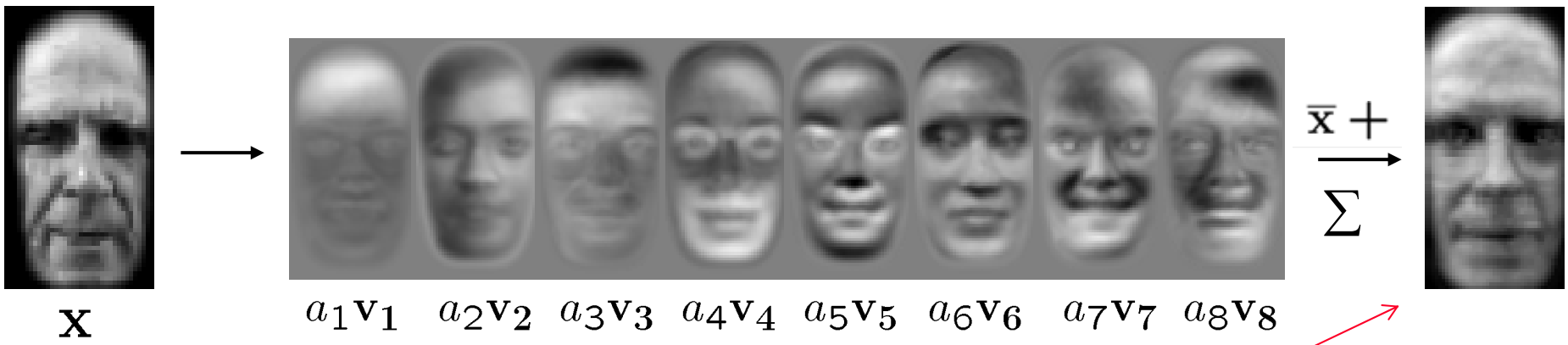
The eigenfaces $v_1, \ldots, v_K$ span the space of faces

- **An image is converted to eigenface coordinates using dot products ("projection"):**

$$\mathbf{x} \rightarrow \big(\underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_1}_{a_1},\ \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_2}_{a_2},\ \ldots,\ \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_K}_{a_K}\big)$$

Input image
(P x Q pixels)

Compressed representation of face
(K usually much smaller than PQ)



Reconstructed face $\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_K\mathbf{v}_K$

Reconstruction using Eigenfaces

- Given image on left, project to Eigenspace, then reconstruct an image (right).

[Turk & Pentland 01]

Application 2: Face Recognition

Algorithm:

1. Given a set of images with labels (e.g., names)
   - Run PCA and compute K eigenfaces
   - Calculate and store the K coefficients for each image with its label
2. Given a new image x, calculate K coefficients

   $$\mathbf{x} \rightarrow (a_1, a_2, \ldots, a_K)$$

3. Verify that x is a face

   $$\|\mathbf{x} - (\overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K})\| < \text{threshold}$$

4. If it is a face, who is it?
   - Find label of closest face in database
   - Nearest-neighbor in K-dimensional space

Break

Next: Applications in Robotic Learning
(Guest Lecture by Dr. Chalodhorn)