Lecture 10 – Unsupervised Learning

Today:

- Unsupervised learning + applications
- Dimensionality reduction (PCA)
- Clustering (k-means and k-medoids)
- Autoencoders

Unsupervised learning + applications

Unsupervised learning: Given data and no labels, learn underlying features of the data

? What are some applications?

Applications:

- Dimensionality reduction – make data simpler/more understandable and extract just the most important features of the data
  - Reduce to 2 or 3 dimensions to visualize
  - Prevent overfitting
  - Eliminate redundancy (correlated features)
- Clustering – identify groups within the data
  - Identifying news topics
  - Recommending "similar" content
- Compression – save space with large text or images
- Generation – generate new data according to the same underlying distribution

Dimensionality Reduction (PCA)

Goal: Reduce the complexity of the data without losing the defining characteristics of the data

→ dimensionality, i.e. size of feature vectors representing data points

$$X = \begin{bmatrix} 3 \\ 5 \\ 7 \\ -2 \\ 4 \end{bmatrix} \rightarrow X = \begin{bmatrix} 5 \\ -2 \end{bmatrix}$$

10-2

Two methods:

1) feature selection - choose which features you think are most important
   ex. for movies maybe just use genre and lead actor/actress

2) feature extraction - build a smaller number of new features which combine parts of the old features
   ex. PCA

$$X = \begin{array}{c|l} \begin{bmatrix}3\\5\\7\\-2\\4\end{bmatrix} & \begin{array}{l}\text{year}\\ \text{genre}\\ \text{budget}\\ \text{lead}\\ \text{nationality}\end{array}\end{array} \xrightarrow{\text{selection}} X = \begin{bmatrix}5\\-2\end{bmatrix} \begin{array}{l}\text{genre}\\ \text{lead}\end{array}$$

$$\xrightarrow{\text{extraction}} X = \begin{bmatrix}6\\0\end{bmatrix} \begin{array}{l}\frac{1}{2}\text{ genre} + \frac{1}{2}\text{ budget}\\ \frac{2}{3}\text{ lead} + \frac{1}{3}\text{ nationality}\end{array}$$

feature selection: pro = intuitive features
con = manual process, limited by original features

feature extraction: pro = automatic, not limited to original features
con = non intuitive features

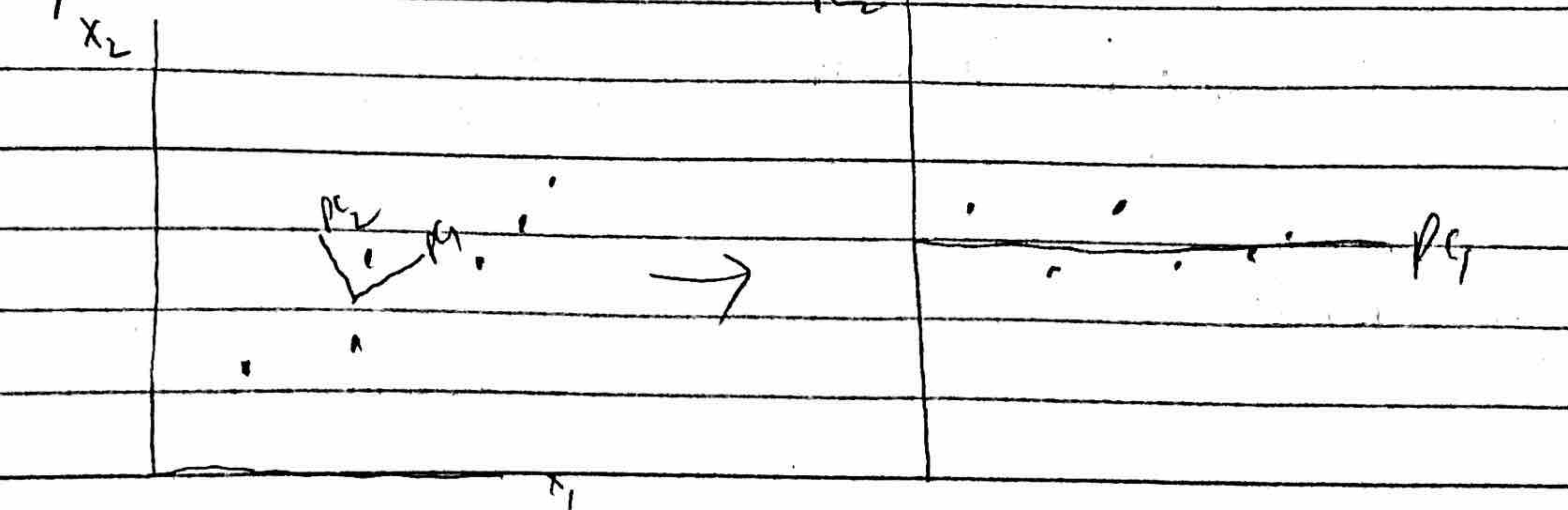
Principal Components Analysis (PCA)

Method for performing feature extraction

Very useful as a preprocessing step for ML algorithms since learning on low dimensional data is faster and prevents overfitting

Idea: Identify largest source of variation in the data, then change axes (features) to reflect largest variation

Example:



Interactive demo: http://setosa.io/ev/principal-component-analysis/

(?) Anyone know what eigenvectors and eigenvalues are

Math background - eigenvectors and eigenvalues

We can multiply a matrix by a vector to get a new vector:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & & & a_{nm} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$Av = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} = \begin{bmatrix} v_1 a_{11} + v_2 a_{12} + \cdots + v_m a_{1m} \\ \vdots \\ v_1 a_{n1} + v_2 a_{n2} + \cdots + v_m a_{nm} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = b$$

$$Av = b$$

Special case: $Av = \lambda v$ where λ is a scalar, i.e. Av = multiple of v

In this case, v is called an eigenvector and λ is called an eigenvalue.

Interactive demo: http://setosa.io/ev/eigenvectors-and-eigenvalues/

Example

$$v = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 2 \\ 8 & 1 \end{pmatrix}$$

(?) compute λ

$$Av = \begin{pmatrix} 1 & 2 \\ 8 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 5 \\ 10 \end{pmatrix} = 5 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 5v \qquad \boxed{\lambda = 5}$$

Eigen decomposition (spectral decomposition)

If A is a square matrix $\underset{(n \times n)}{}$ with linearly independent eigenvectors $q_1, q_2, \ldots, q_n$,

then A can be factorized as

$$\boxed{A = Q \Lambda Q^{-1}}$$

where $Q = \begin{bmatrix} \vert & \vert & & \vert \\ q_1 & q_2 & \cdots & q_n \\ \vert & \vert & & \vert \end{bmatrix}$ is an $\underset{(n \times n)}{\text{matrix}}$ with eigenvectors as columns and →

10-4

$\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & \ddots & \lambda_n \end{bmatrix}$ is an (nxn) matrix with eigenvalues on the diagonal

You can learn the theory behind this in a linear algebra class.

The eigenvectors are orthogonal to each other (right angles) and will form a new set of axes which will be the axes used by PCA.

How PCA works?

Input: $X = n \times d$ matrix with n data points, each with dimension d

PCA steps:

Center and → 1) Standardize columns of X so that all features are at the same scale

- ↳ For each column, subtract mean and divide by standard deviation
- Now every column has mean = 0, std = 1
- Let Z = standardized X $(n \times d)$

2) Compute the covariance matrix of Z to determine how pairs of features vary

- $Cov = Z^T Z$ $(d \times d)$
- Cov_{ij} = covariance between features i and j (up to a constant)
  - ↳ >0 means larger $i \Leftrightarrow$ larger j
  - <0 means larger $i \Leftrightarrow$ smaller j

3) Compute eigen decomposition of $Z^T Z$. → $(d \times d)$ → $(d \times d)$

- $Z^T Z = Q \cdot \Lambda Q^{-1}$ where Q is eigenvector matrix, Λ is diagonal eigenvalue matrix.

4) Sort eigenvectors based on eigenvalues.

- Idea: larger eigenvalue means eigenvector explains more variance in the data
- Q^* = eigenvector matrix sorted by eigenvalue (1^{st} col = largest eigen value) $(d \times d)$

5) Compute $Z^* = ZP^*$ $(n \times d)$.

- Z^* is a centered/standardized version of X, but now features are a combination of original features and are sorted by importance. ←

new features are called principal components

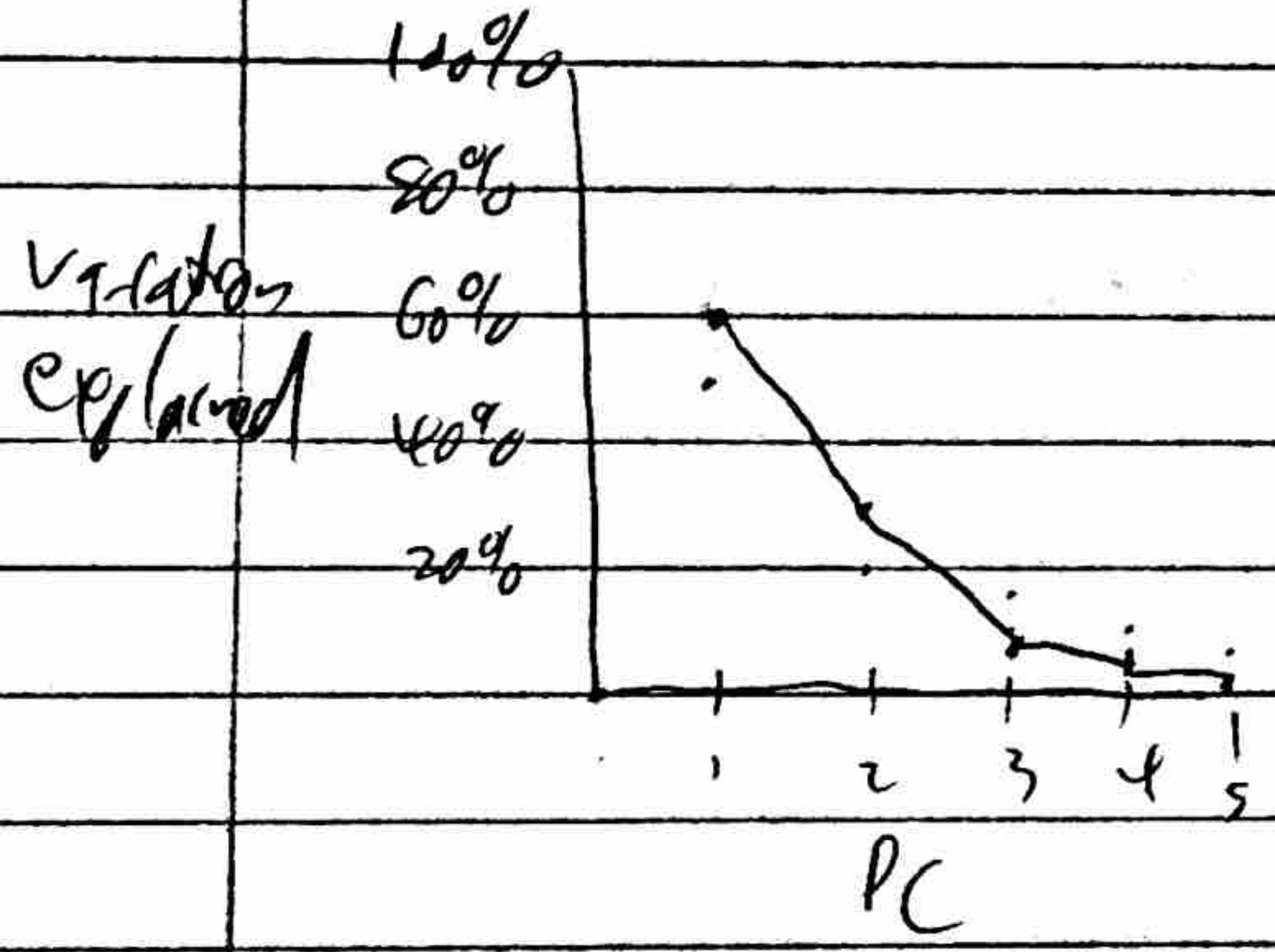
6) Select the first d' columns of Z^* $(d' \leq d)$. $(n \times d')$ Now we have reduced our data from d dimensional to d' dimensional, and those d' dimensions are the most important.

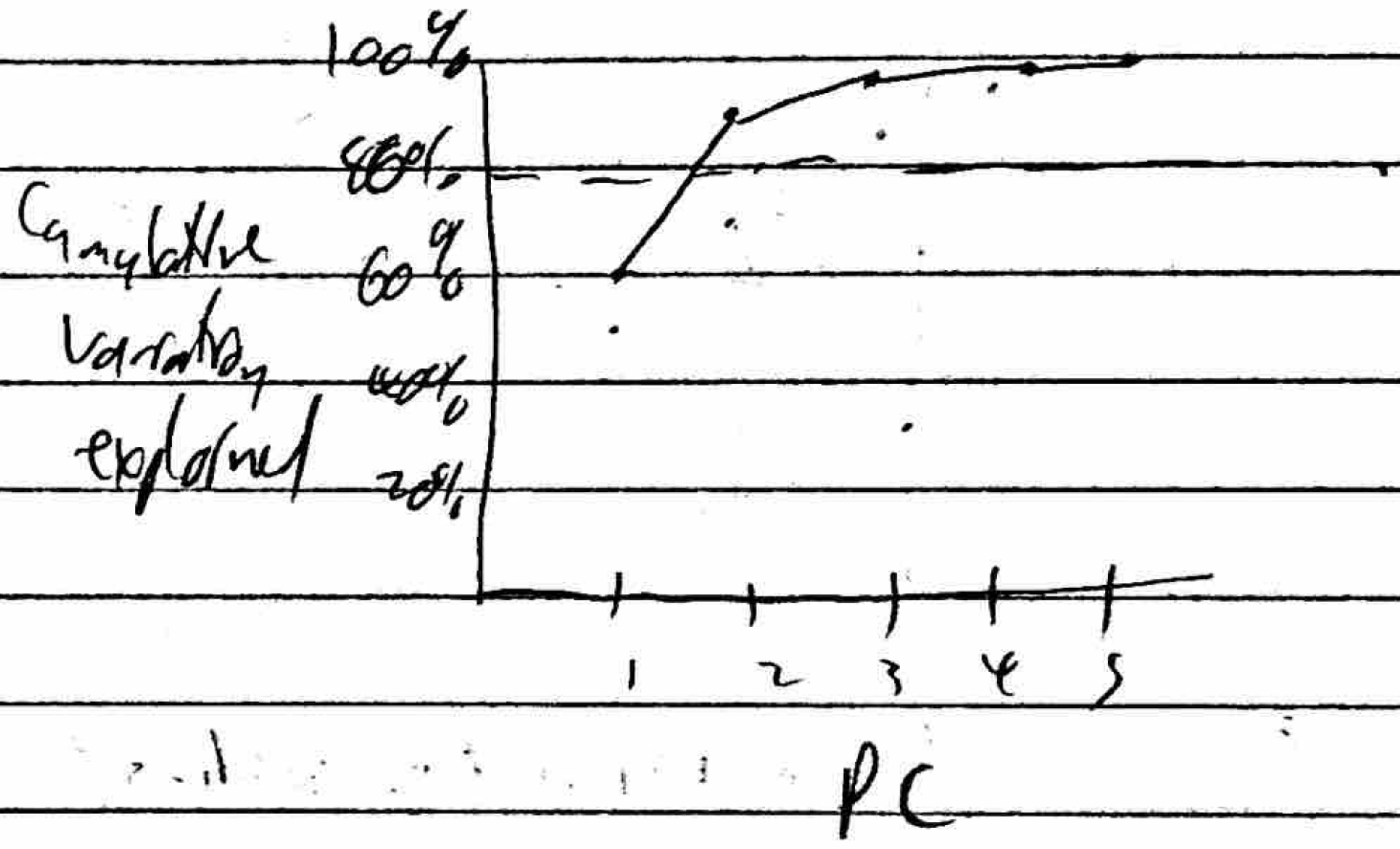
(?) How do we determine d'?

1) Preselect - we want to plot the reduced data so we choose d' = 2 or 3

2) Select based on amount of variation explained

- Choose threshold, ex. want to explain 80% of variation (willing to lose 20%
- Percent of variation explained by principal component i = $\frac{\lambda_i}{\lambda_1+\lambda_2+\dots+\lambda_d}$

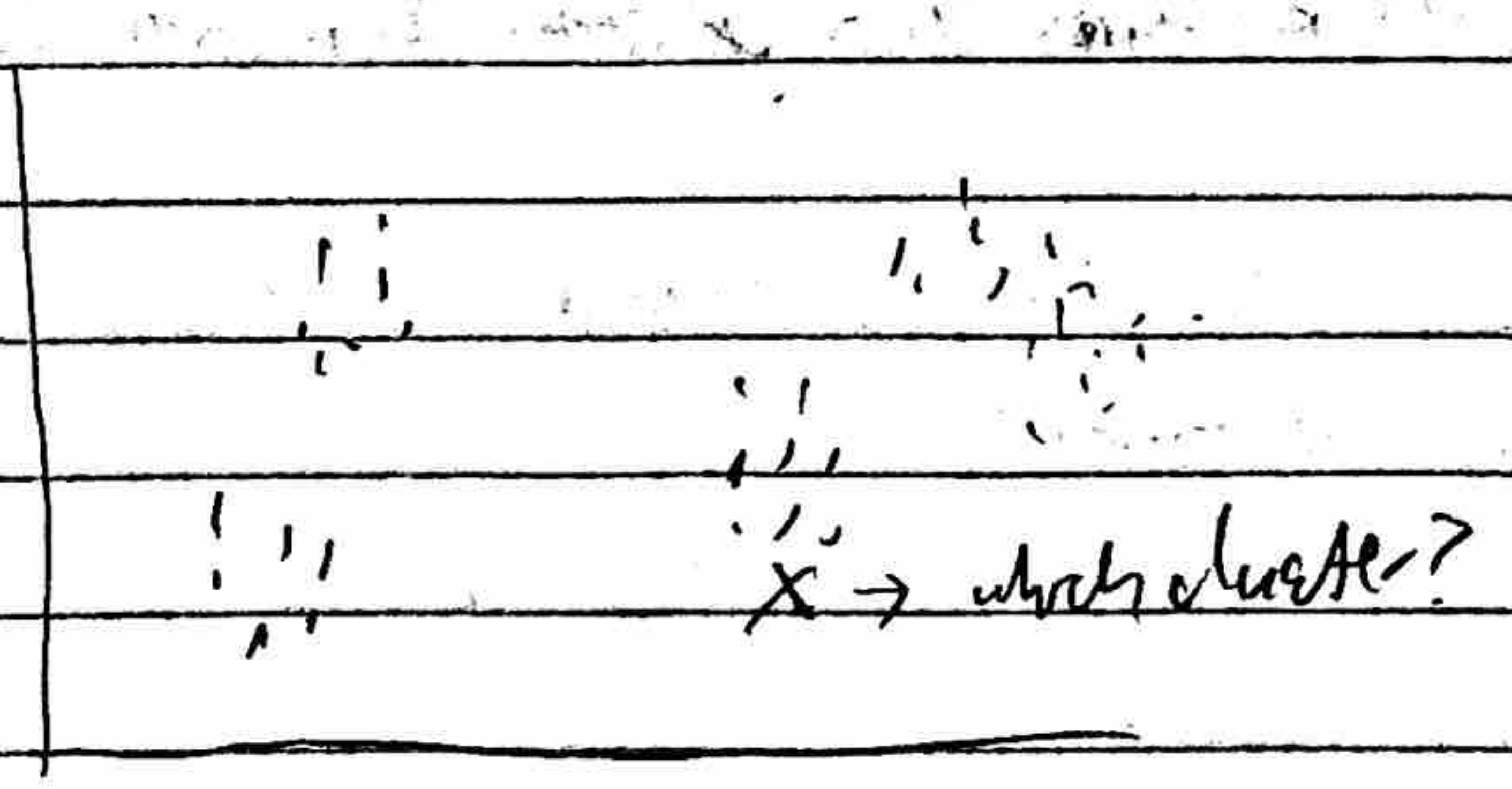




Based on the above analysis, we can reduce the dimensionality from 5 to 2 while still retaining at least 80% of the variance of the data

Clustering

Goal: Identify groups in the data.



Even though we don't have labels, It's clear that there are clusters and we can learn to predict which cluster a new data point belongs to, even if we don't know what the clusters mean

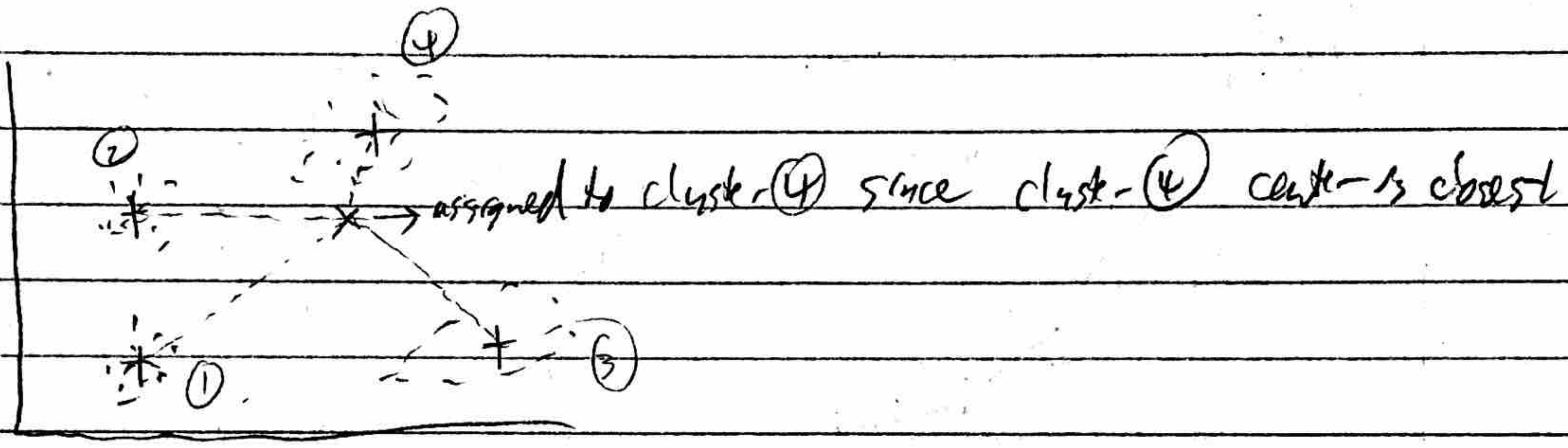
(?) How do we learn to predict clusters?

10-6

k-means

Similar to k-nearest neighbors (KNN) but KNN was supervised (predicted known labels) whereas k-means is unsupervised (predicts clusters, which need to be learned).

Goal: Learn the centers of k clusters. New points will be assigned to clusters based on which cluster center is closest.



? How do we learn the centers?

Define a cost function and try to minimize it

$z^{(1)}, z^{(2)}, \dots, z^{(k)}$ = centers of k clusters

$C^{(1)}, C^{(2)}, \dots, C^{(k)}$ = set of points assigned to each of the k clusters (based on which cluster center is closest)

$$\boxed{\text{cost}(z^{(1)}, z^{(2)}, \dots, z^{(k)}) = \sum_{i=1}^{k} \sum_{j \in C^{(i)}} \|x^{(j)} - z^{(i)}\|^2}$$

Basically cost is sum of distances from cluster center to points assigned to that cluster

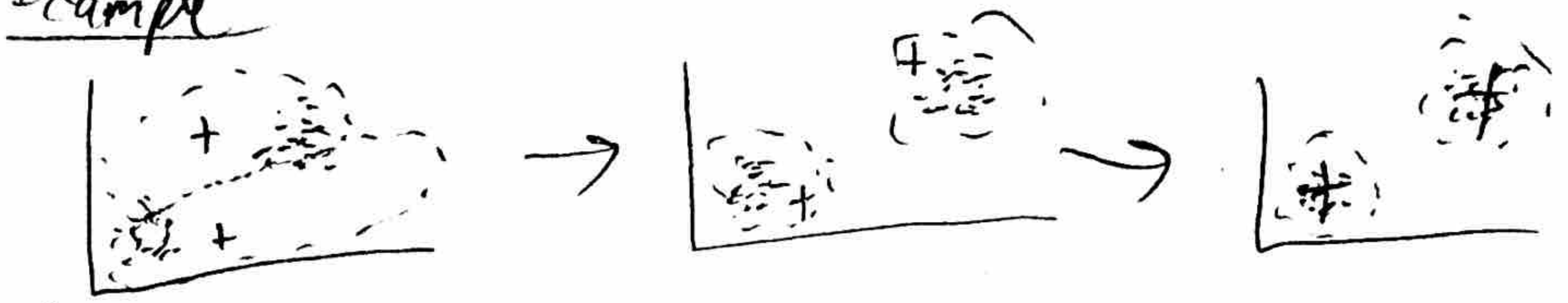
How do we minimize this cost function?

k-means algorithm

```
1) Initialize $z^{(1)}, z^{(2)}, \dots, z^{(k)}$ randomly
2) Repeat until cost doesn't change
    a) · Fix $z^{(1)}, z^{(2)}, \dots, z^{(k)}$ and assign each $x^{(j)}$ to the closest $z^{(i)}$
          i.e. add $x^{(j)}$ to $C^{(i)}$
       · Compute $\text{cost}(z^{(1)}, z^{(2)}, \dots, z^{(k)}) = \sum_{i=1}^{k} \sum_{j \in C^{(i)}} \|x^{(j)} - z^{(i)}\|^2$
    b) · Fix cluster assignments $C^{(1)}, C^{(2)}, \dots, C^{(k)}$
       · Compute new cluster centers $z^{(i)} = \frac{1}{|C^{(i)}|} \sum_{j \in C^{(i)}} x^{(j)}$   (average of points in cluster)
```

cluster centers

Example

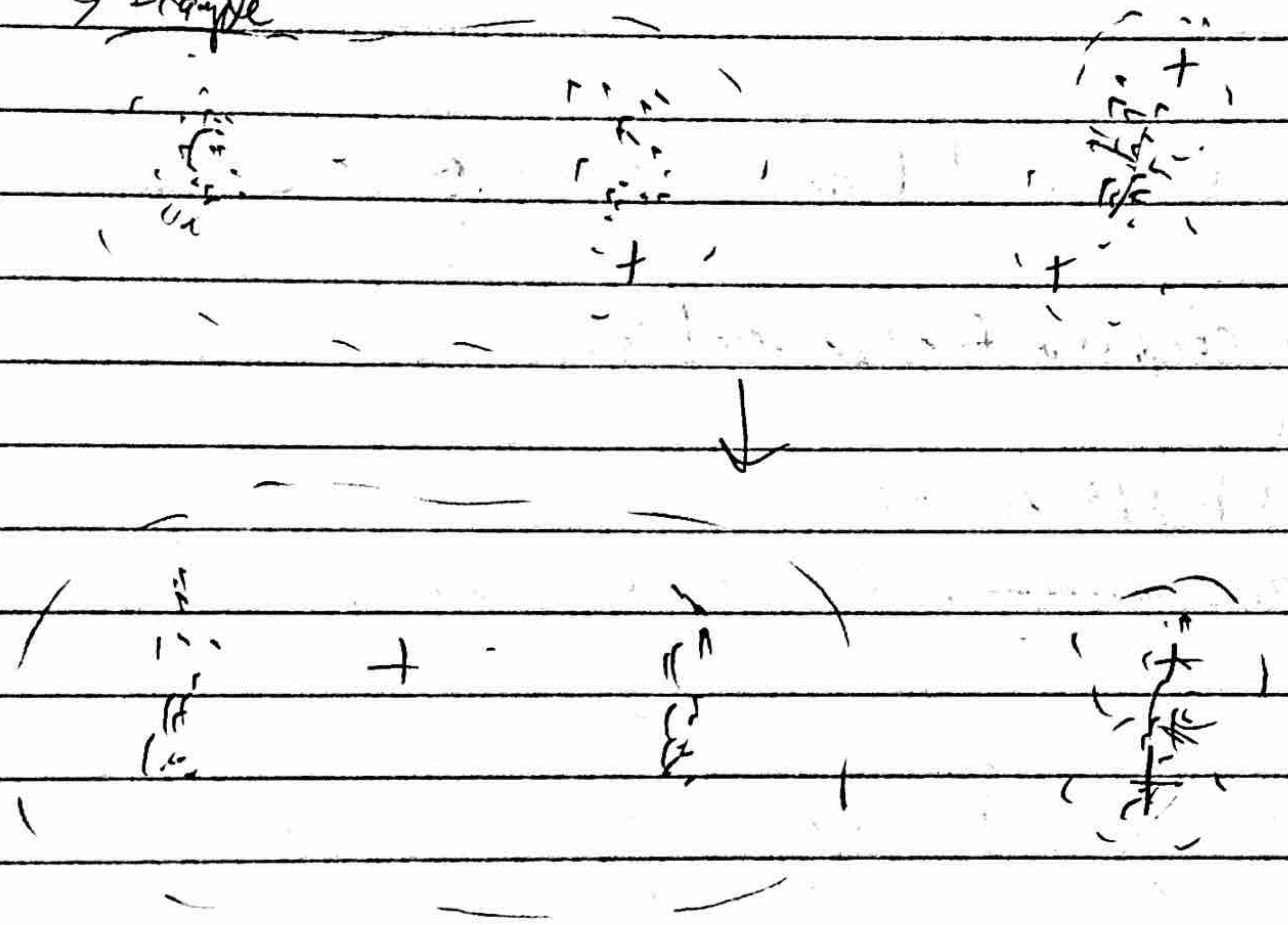


10 ->

Properties

- Can show that new cluster centers minimize cost function given cluster assignments, so cost must go down (or stays same) on every iteration
- Initialization matters. Can get stuck in local minimum

Example



? How can we fix this?

An alternative:

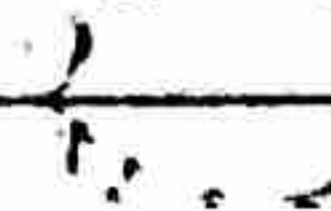
k-medoids

Idea: Use actual data points as cluster centers.

k-medoids algorithm

```
1) Initialize $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\} \subseteq \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$
2) Repeat until centers don't change
    a) Assign points to clusters, i.e. $x^{(i)}$ is assigned to $C^{(j)}$ such that
       $z^{(j)}$ is closest (same as k-means)
    b) Compute new cluster centers $z^{(j)} = x^{(i)} \in C^{(j)}$ s.t. $\sum_{i \in C^{(j)}} ||x^{(i)} - z^{(j)}||^2$
       is minimized
```

Pro: Better initialization because uses actual data points

Con: Doesn't work well for all cluster types, ex.  b/c want center

10-8

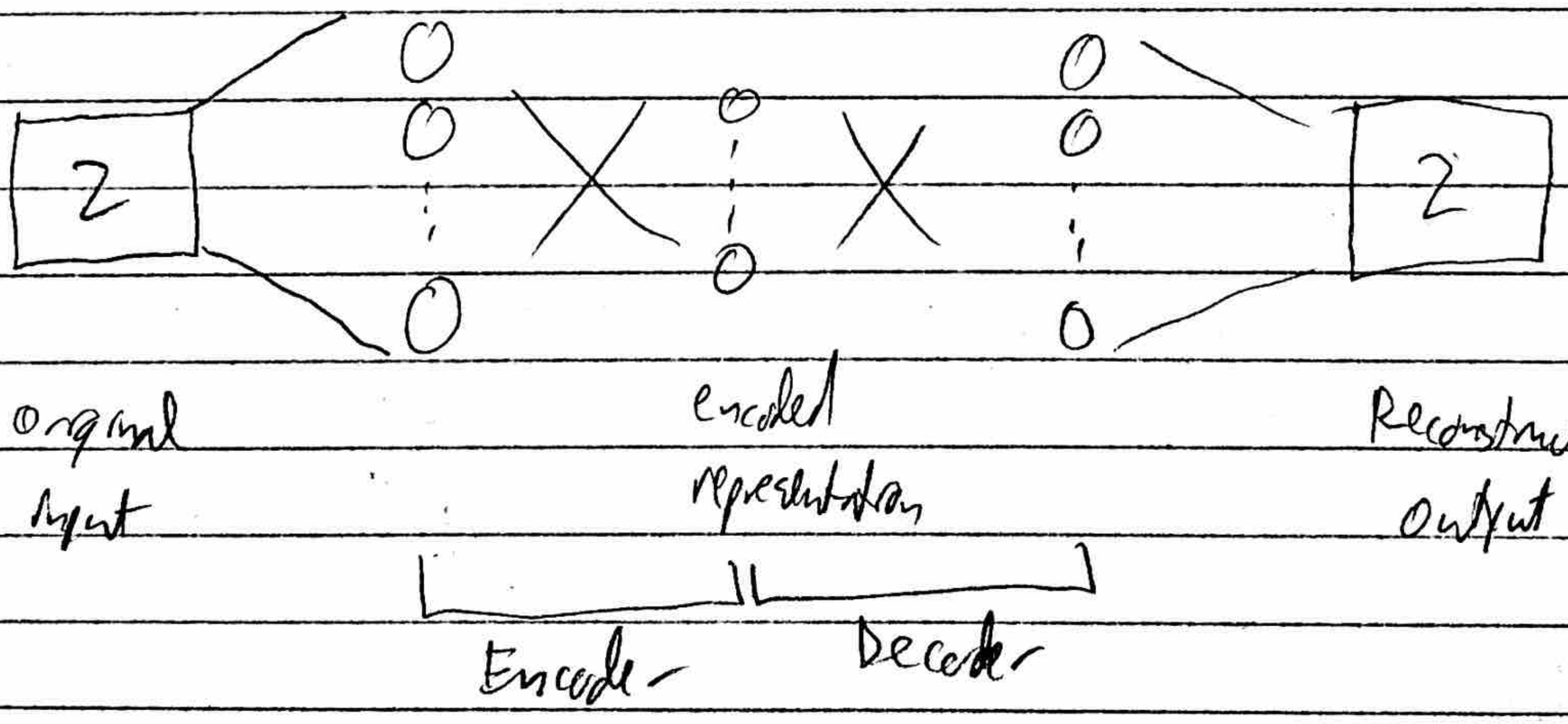
In practice: k-means with a better initialization strategy called "k-means++" which helps guide random initialization toward clusters. Also k-means is typically run multiple times with different initializations and the run with the minimum cost is used to determine final cluster centers.

Autoencoders

Neural networks for unsupervised learning. Sometimes called self-supervised b/c label = input

An autoencoder is a compression function which is:

1) data-specific
2) lossy (like MP3, JPEG)
3) learned automatically rather than engineered by hand



ex. 4/96 we go from 784 → 32 → 784

The encoded representation is smaller than the input, thus introducing a bottleneck. This forces the autoencoder to efficiently compress the input so it can pass through the bottleneck without losing too much information.

Encoder and decoder are typically a single layer neural network but can be deep or convolutional.

The compression is generally not very good compared to other methods, but autoencoders are good for data denoising and dimensionality reduction.
→ train on noisy input, clean output

Variational autoencoder: Learns to model probability distribution of data and can be used to generate new data