

CSC 580 Principles of Machine Learning

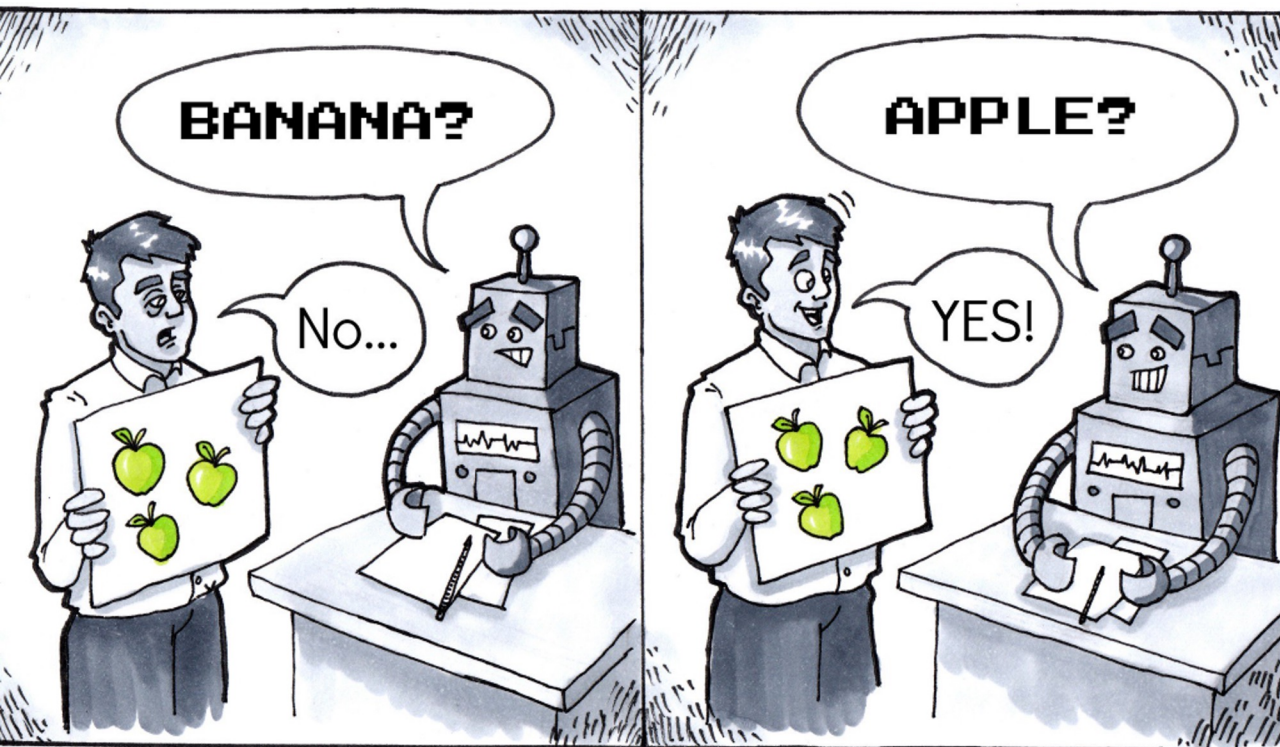
# 09 Unsupervised learning

**Jason Pacheco**

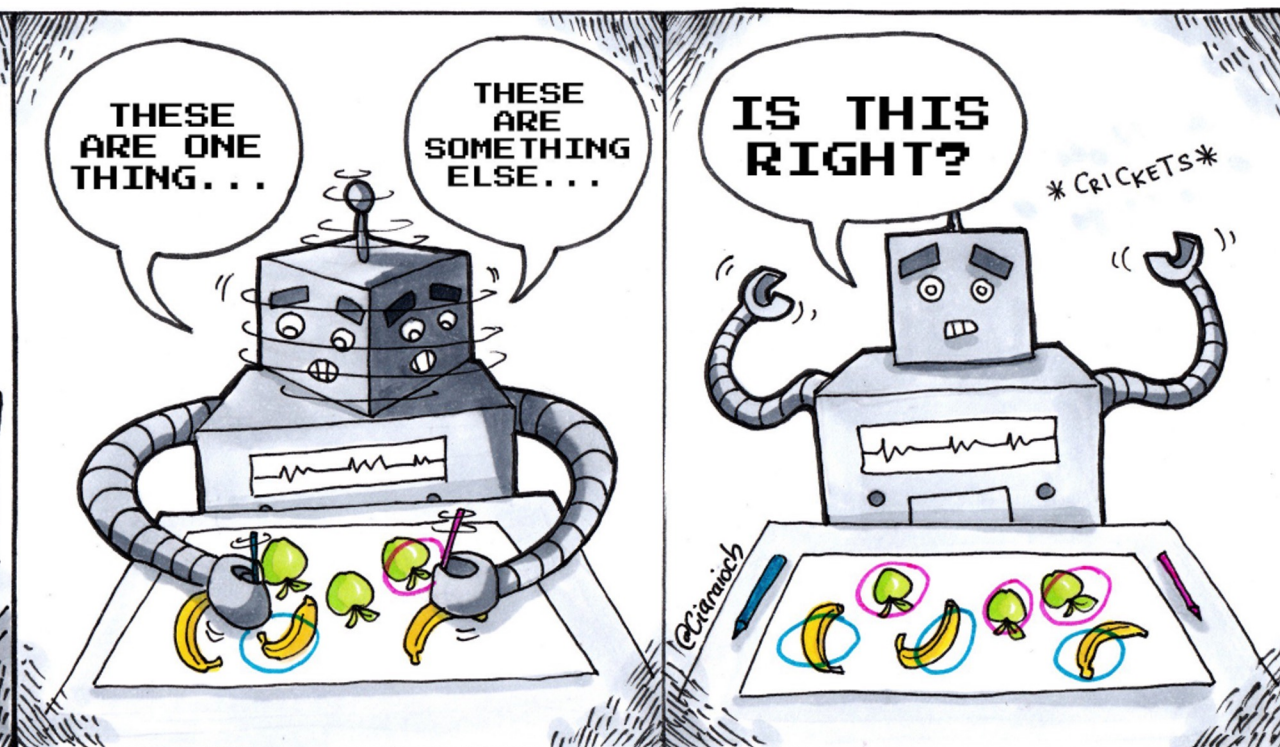
**Department of Computer Science**



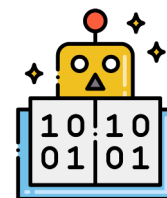
\*slides credit: built upon CSC 580 lecture slides by Chicheng Zhang & Kwang-Sung Jun



## Supervised Learning



## Unsupervised Learning



# Task 1 : Group These Set of Document into 3 Groups based on meaning

Doc1 : Health , Medicine, Doctor

Doc 2 : Machine Learning, Computer

Doc 3 : Environment, Planet

Doc 4 : Pollution, Climate Crisis

Doc 5 : Covid, Health , Doctor



# Task 1 : Group These Set of Document into 3 Groups.

Doc1 : Health , Medicine, Doctor

Doc 2 : Machine Learning, Computer

Doc 3 : Environment, Planet

Doc 4 : Pollution, Climate Crisis

Doc 5 : Covid, Health , Doctor



# Task 1 : Group These Set of Document into 3 Groups.

Doc1 : Health , Medicine, Doctor

Doc 5 : Covid, Health , Doctor

Doc 3 : Environment,  
Planet

Doc 4 : Pollution, Climate  
Crisis

Doc 2 : Machine  
Learning, Computer





# What is unsupervised learning?

- Uncovering structures in unlabeled data
- What can we expect to learn?
  - **Clustering**: obtain partition of the data that are well-separated.
    - can be viewed as a preliminary classification without predefined class labels.
  - **Components**: extract common components that compose data points.
    - e.g., topic modeling given a set of articles: each article talks about a few topics => extract the set of topics that appears frequently.
- Usage
  - As a summary of the data
    - **Exploratory data analysis**: what are the **patterns** we can get even without labels?
  - Often used as ‘preprocessing techniques’
    - e.g., extract useful **features** using “gaussian mixture model” (will be covered later)

# Clustering



# Clustering

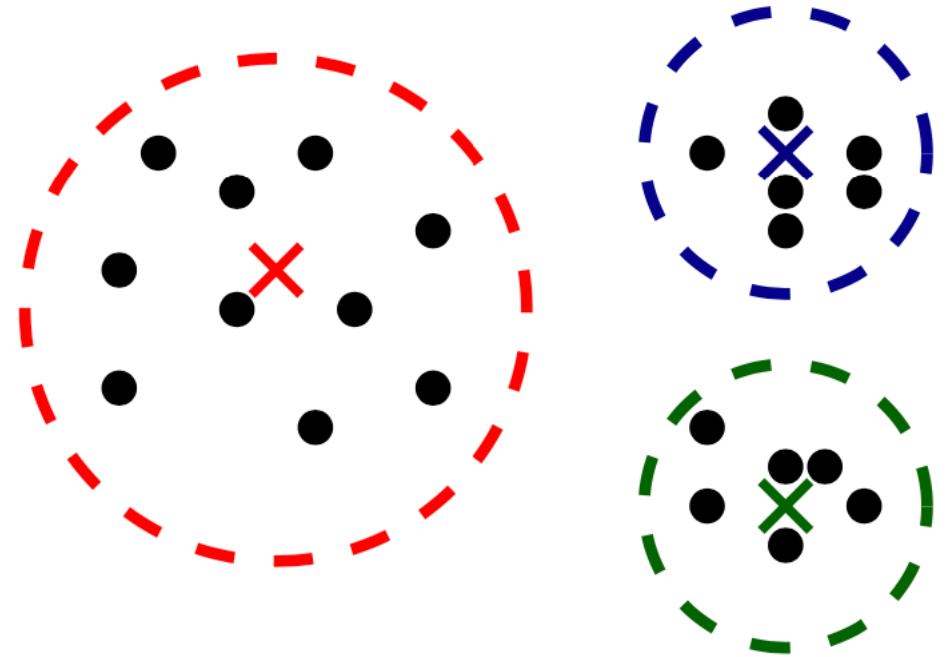
- Input:  $k$ : the number of clusters (hyperparameter)

$$S = \{x_1, \dots, x_n\}$$

- Output

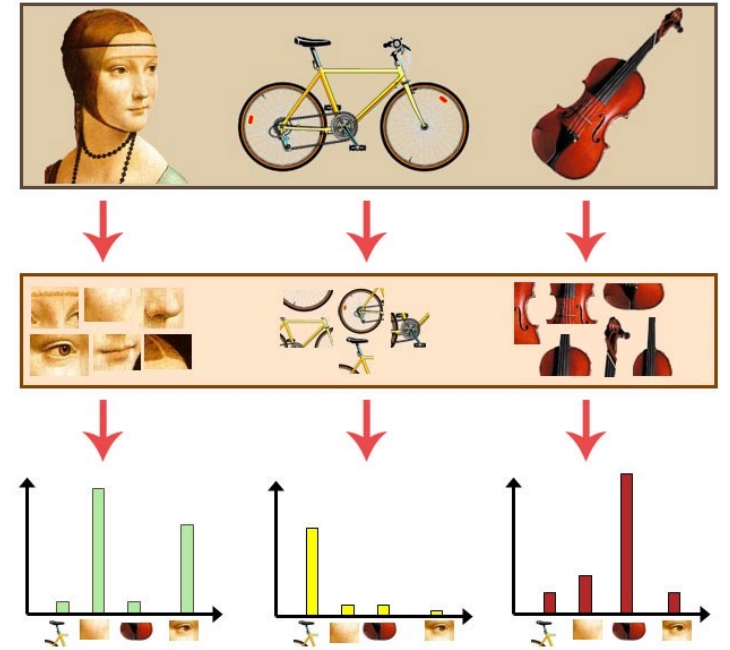
- partition  $\{G_i\}_{i=1}^k$  s.t.  $S = \cup_i G_i$  (disjoint union).
- often, we also obtain 'centroids'

- Q: what would be a reasonable definition of centroids?



# Application: Clustering for feature extraction

- Feature extraction: **histogram features (bag of visual words)**
- A set of images:  $S = \{x_1, \dots, x_n\}$
- Cut up each  $x_i \in \mathbb{R}^d$  into different parts  $x_i^{(1)}, \dots, x_i^{(m)} \in \mathbb{R}^p$ 
  - e.g., small (overlapping) patches of an image
- Notation:  $[n] := \{1, \dots, n\}$
- Pool all the patches together:  $P := \{x_i^{(j)}\}_{i \in [n], j \in [m]}$
- Run clustering on  $P$  with  $\#clusters=k \Rightarrow$  for each  $x_i^{(j)}$ , we have a cluster assignment  $A(x_i^{(j)}) \in [k]$
- Generate the feature vector of  $x_i$  as the histogram of  $\{A(x_i^{(j)})\}_{j \in [m]}$ 
  - i.e.,  $z = (z_1, \dots, z_k)$  where  $z_\ell$  is the count of the cluster  $\ell$



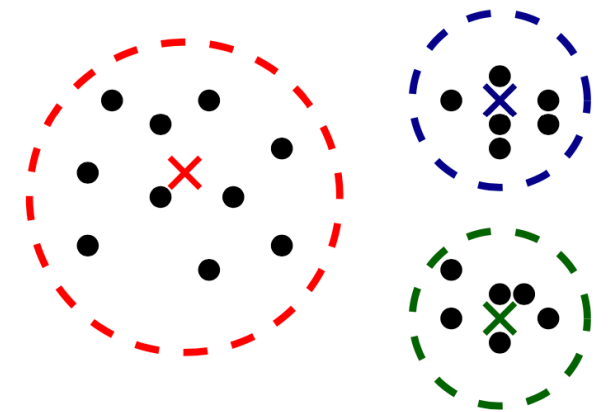
# $k$ -means clustering

- Idea: to partition the data, it would be great if someone gives us  $k$  reasonable centroids  $c_1, \dots, c_k$ , since then we can partition the data with them.

$$A(x) = \arg \min_{j \in [k]} \|x - c_j\|_2$$

- But we don't have those centroids => Let's find them with an optimization formulation.

$$\underset{c_1, \dots, c_k}{\text{minimize}} f(c_1, \dots, c_k), \text{ where } f(c_1, \dots, c_k) = \sum_{i=1}^n \min_{j \in [k]} \|x - c_j\|_2^2$$



# Special case: $k=1$

- $\min_{c_1, \dots, c_k} \sum_{i=1}^n \min_{j \in [k]} \|x_i - c_j\|_2^2 \Rightarrow \min_c \sum_{i=1}^n \|x_i - c\|_2^2$
- Let  $F(c) = \sum_{i=1}^n \|x_i - c\|_2^2$  convex; minimizer  $c^*$  satisfies that  $\nabla F(c^*) = 0$   
 $\Rightarrow \sum_{i=1}^n (x_i - c^*) = 0 \Rightarrow c^* = \frac{1}{n} \sum_{i=1}^n x_i$

# For $k \geq 2$

- minimize  $f(c_1, \dots, c_k)$ , where  $f(c_1, \dots, c_k) = \sum_{i=1}^n \min_{j \in [k]} \|x - c_j\|_2^2 \Rightarrow$  NP-hard even when  $d = 2$

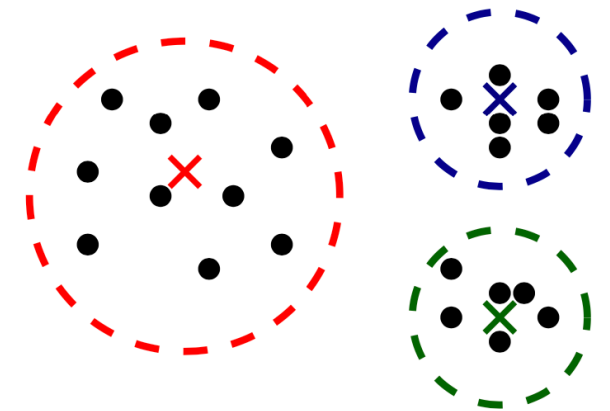
- **K-means algorithm**: solve it approximately (heuristic)

(Also called Lloyd's algorithm)

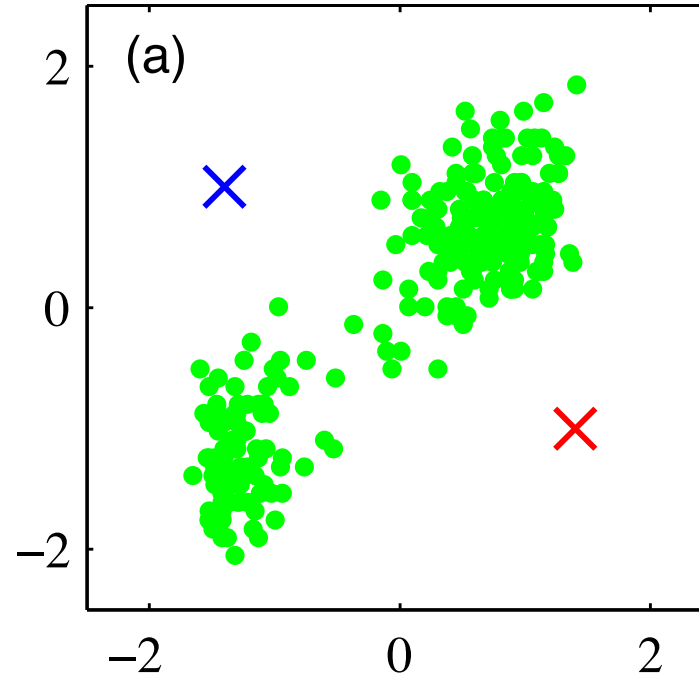
- Observation: The chicken-and-egg problem.

- Cluster center location depends on the cluster assignment
- Cluster assignment depends on cluster location

- Very common heuristic (that may or may not be the best thing to do)

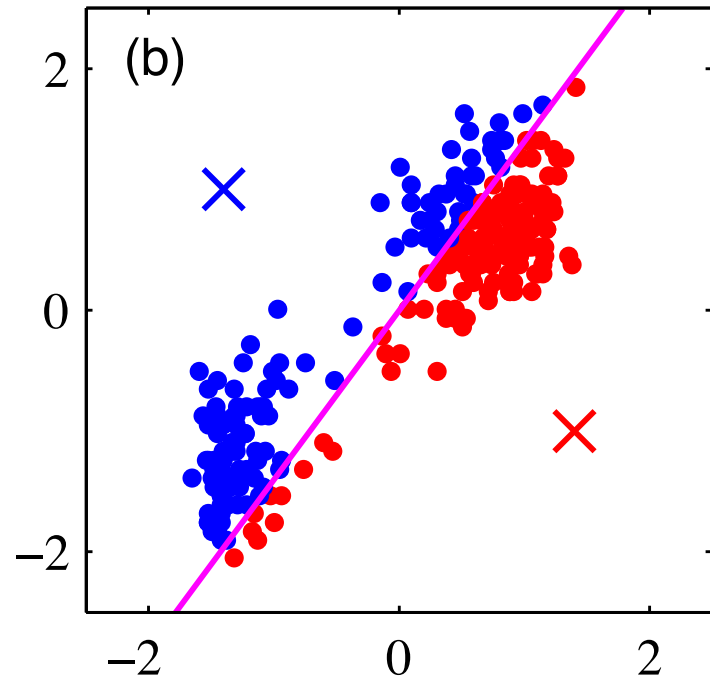


# Initialization

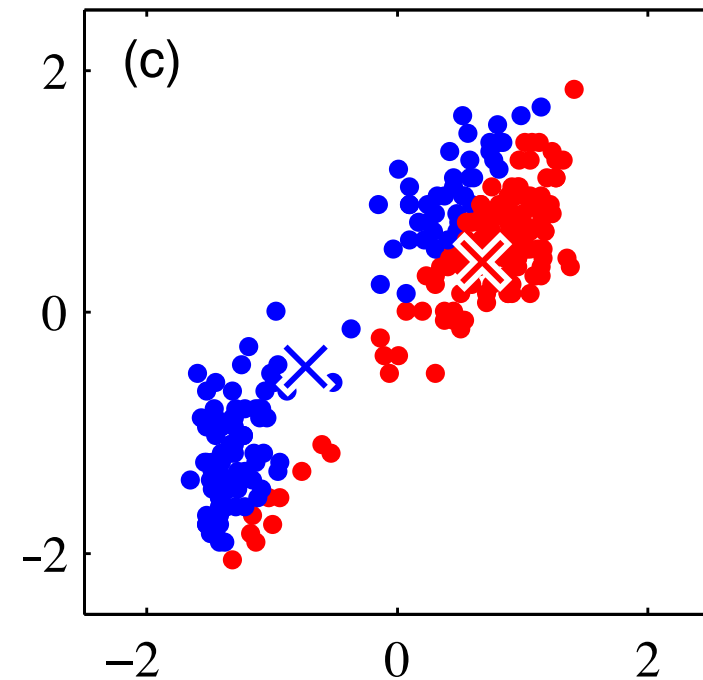


Arbitrary/random initialization of  $c_1$  and  $c_2$

# Iteration 1

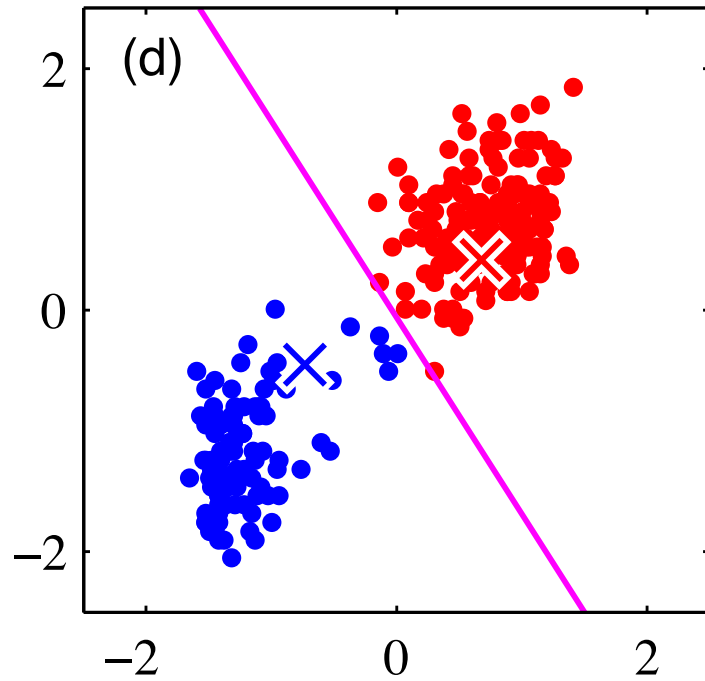


(A) update the cluster assignments.

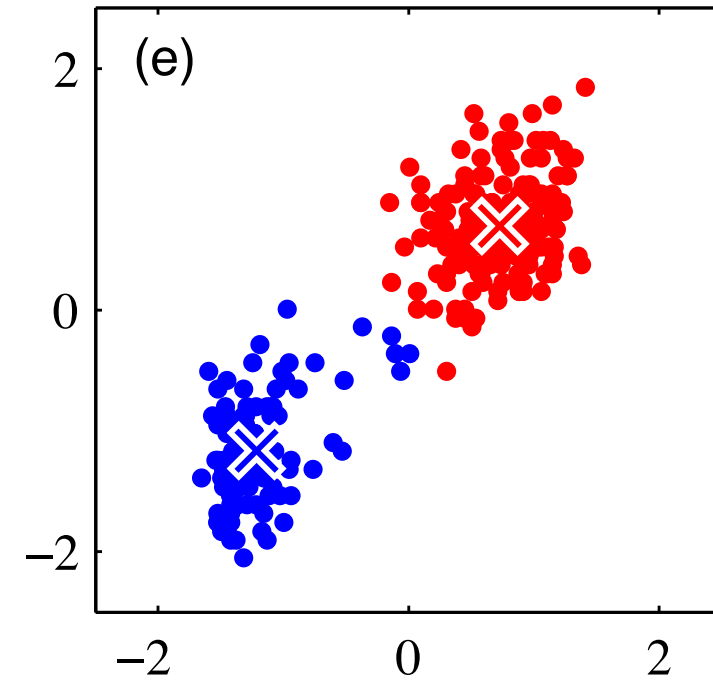


(B) Update the centroids  $\{c_j\}$

# Iteration 2



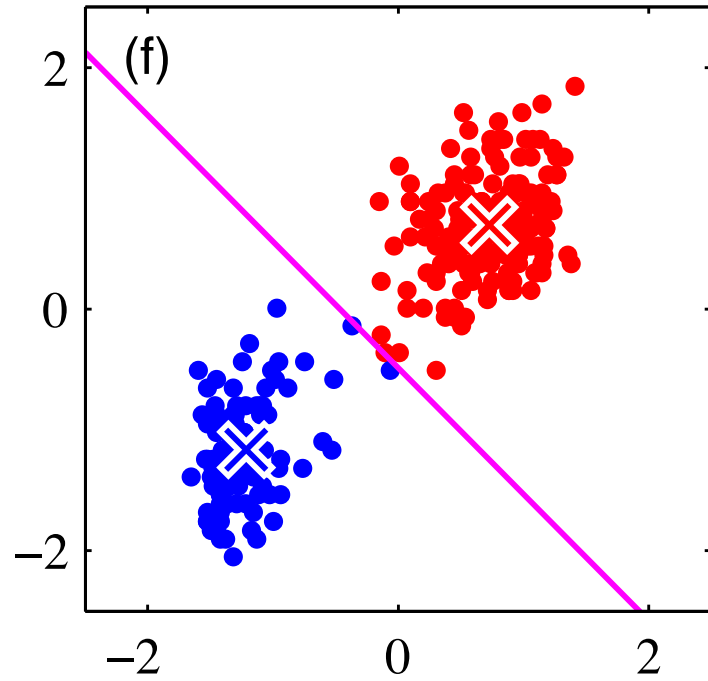
(A) update the cluster assignments.



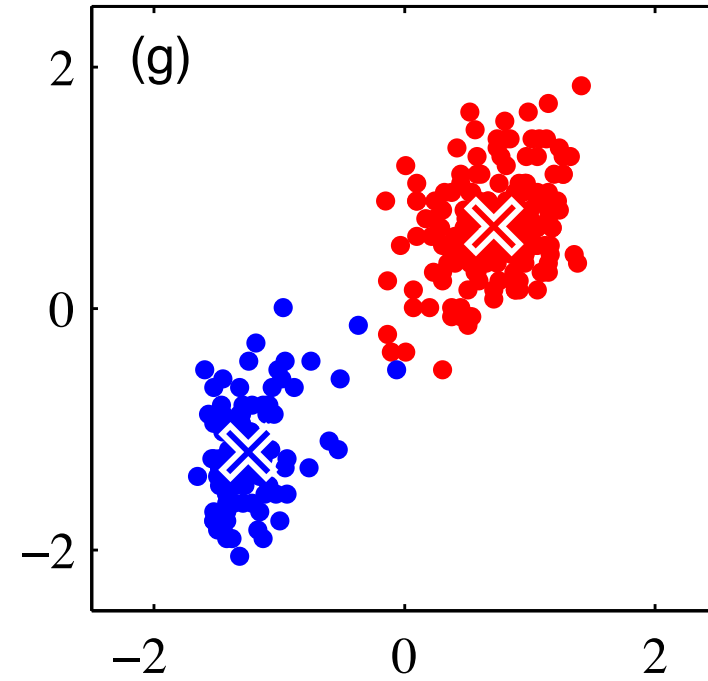
(B) Update the centroids  $\{c_j\}$



# Iteration 3

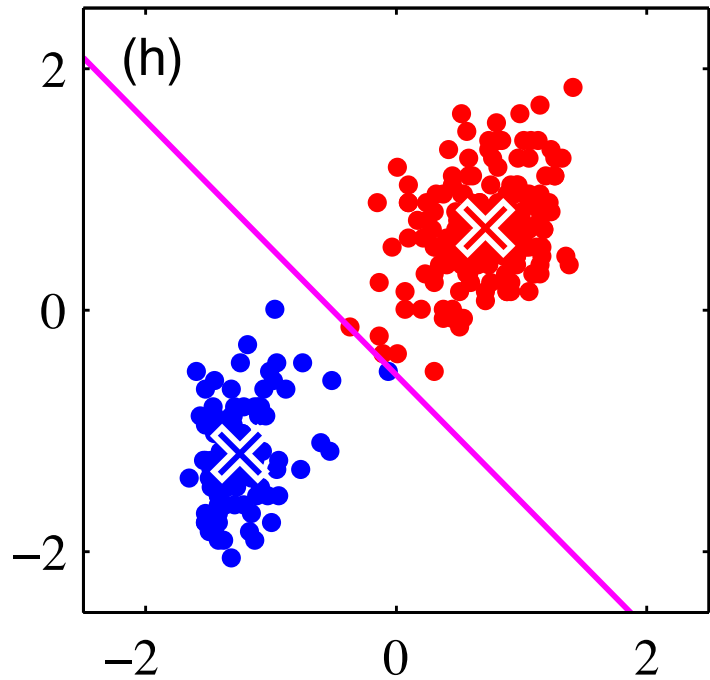


(A) update the cluster assignments.

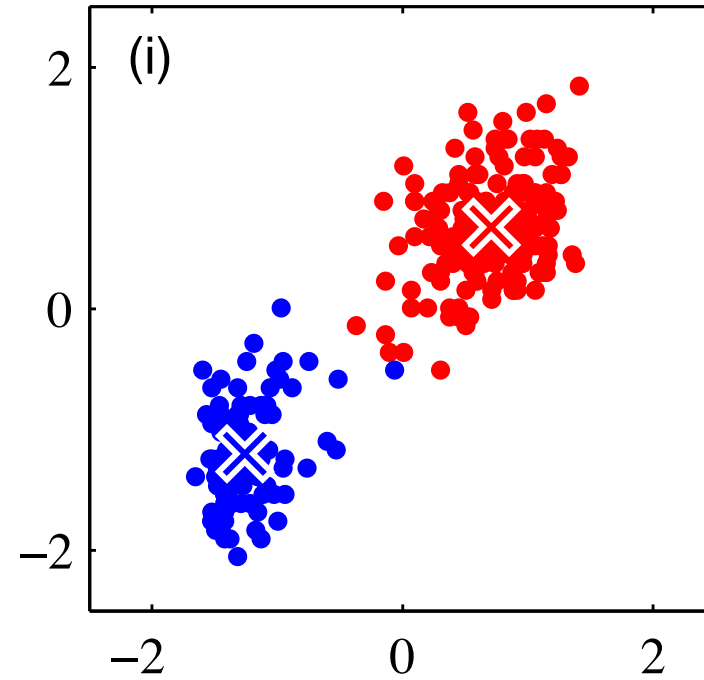


(B) Update the centroids  $\{c_j\}$

# Iteration 4



(A) update the cluster assignments.



(B) Update the centroids  $\{c_j\}$

# K-means clustering

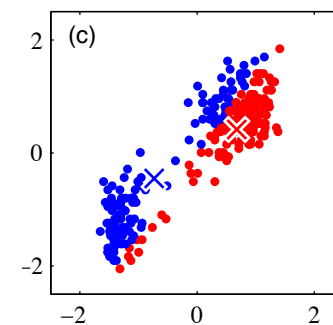
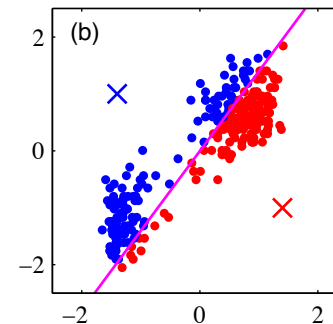
**Input:**  $k$ : num. of clusters,  $S = \{x_1, \dots, x_n\}$

**[Initialize]** Pick  $c_1, \dots, c_k$  as randomly selected points from  $S$  (see next slides for alternatives)

For  $t=1,2,\dots,\text{max\_iter}$

- **[Assignments]**  $\forall x \in S, a_t(x) = \arg \min_{j \in [k]} \|x - c_j\|_2^2$
- If  $t \neq 1$  AND  $a_t(x) = a_{t-1}(x), \forall x \in S$ 
  - break
- **[Centroids]**  $\forall j \in [k], c_j \leftarrow \text{average}(\{x \in S: a_t(x) = j\})$

**Output:**  $c_1, \dots, c_k$  and  $\{a_t(x_i)\}_{i \in [n]}$



# K-means: cost minimization perspective

- Key idea: solving the optimization problem by *reformulation* and *alternating minimization*:

- Reformulation: denote by  $\vec{c} := (c_1, \dots, c_k)$ ,  $\vec{z} := (z_1, \dots, z_n)$ ;

$$f(\vec{c}) = \min_{\vec{z}} g(\vec{c}, \vec{z}), \text{ where } g(\vec{c}, \vec{z}) = \sum_{i=1}^n \|x_i - c_{z_i}\|_2^2$$

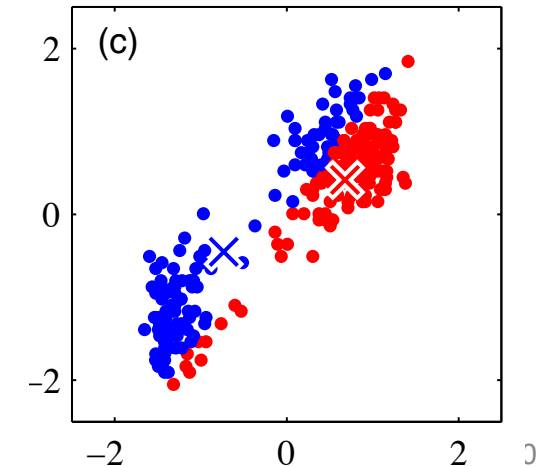
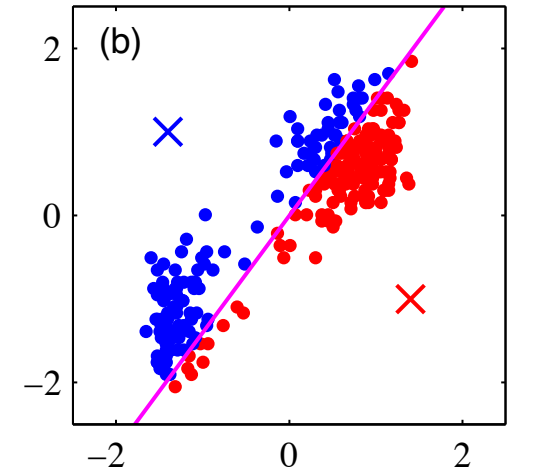
suffices to solve

$$\min_{\vec{c}, \vec{z}} g(\vec{c}, \vec{z})$$

- For  $t = 1, 2, \dots, T$ :

- Update the cluster assignments:  $\vec{z}_t \leftarrow \operatorname{argmin}_{\vec{z}} g(\vec{c}_{t-1}, \vec{z})$
- Update the centroids:  $\vec{c}_t \leftarrow \operatorname{argmin}_{\vec{c}} g(\vec{c}, \vec{z}_t)$

- Observation: objective function  $g(\vec{c}_t, \vec{z}_t)$  decreases *monotonically* in  $t$



# Issue 1: Unreliable solution

- You usually get suboptimal solutions
- You usually get different solutions every time you run.
- **Standard practice**: Run it 50 times and take the one that achieves the smallest objective function
  - Recall: minimize  $f(c_1, \dots, c_k)$ , where  $f(c_1, \dots, c_k) = \sum_{i=1}^n \min_{j \in [k]} \|x - c_j\|_2^2$
- Or, change the initialization (next slide)
  - Idea: ensure that we pick a widespread  $c_1, \dots, c_k$

# Two alternative initializations.

- **Furthest-first traversal**  $\Rightarrow$  Sequentially choose  $c_j$  that are the farthest from the previously-chosen.

- Pick  $c_1 \in \{x_1, \dots, x_n\}$  arbitrarily (or randomly)
- For  $j = 2, \dots, k$ 
  - Pick  $c_j \in \mathbb{R}^d$  as a point in  $\{x_1, \dots, x_n\}$  that maximizes the squared distances to  $c_1, \dots, c_{j-1}$ .

$$c_j = \arg \max_{i \in [n]} \min_{j'=1, \dots, j-1} \|x_i - c_{j'}\|_2^2$$

- **$k$ -means++ (Arthur and Vassilvitskii, 2007)**

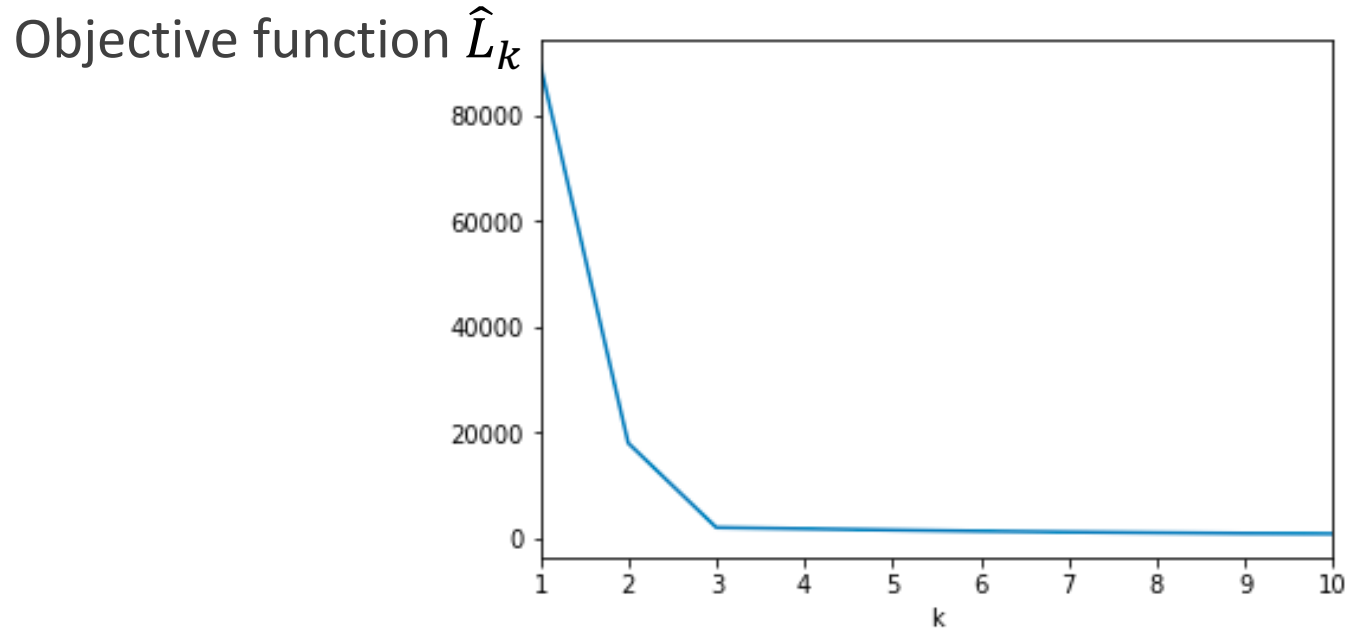
- Pick  $c_1 \in \{x_1, \dots, x_n\}$  uniformly at random
- For  $j = 2, \dots, k$ 
  - Define a distribution  $\forall i \in [n], \mathbb{P}(c_j = x_i) \propto \min_{j'=1, \dots, j-1} \|x_i - c_{j'}\|_2^2$
  - Draw  $c_j$  from the distribution above.

More likely to choose  $x_i$  that is farthest from already-chosen centroids.

$\Rightarrow$  has a mathematical guarantee that it will be better than an arbitrary starting point!

# Issue 2: Choosing k

- $\hat{L}_k = f(c_1, \dots, c_k)$  for  $c_1, \dots, c_k$  obtained by any k-means clustering algorithm



- Elbow method: see where you get saturation.
- Akaike information criterion (AIC):  $\operatorname{argmin}_k (\hat{L}_k + 2kd)$
- Bayesian information criterion (BIC):  $\operatorname{argmin}_k (\hat{L}_k + kd \cdot \log n)$

# Kernelizing K-means algorithm

How to perform clustering with feature transformations  $\phi: \mathcal{X} \rightarrow \mathbb{R}^D$ ?

**Input:**  $k$ : num. of clusters,  $S = \{x_1, \dots, x_n\}$ , kernel function  $K$  with feature map  $\phi$

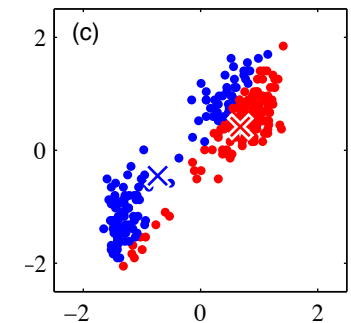
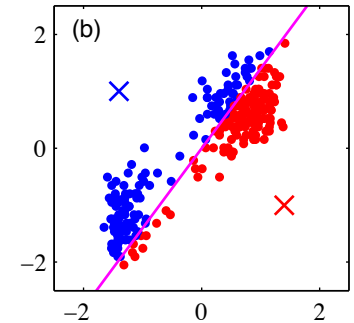
Idea: perform clustering over  $\tilde{S} = \{\phi(x_1), \dots, \phi(x_n)\}$  without explicitly evaluating  $\phi$

**[Initialize]** Pick  $c_1, \dots, c_k$  as randomly selected points from  $\tilde{S}$

For  $t=1, 2, \dots, \text{max\_iter}$

- **[Assignments]**  $\forall x \in S, \quad a_t(x) = \arg \min_{j \in [k]} \|\phi(x) - c_j\|_2^2$
- If  $t \neq 1$  AND  $a_t(x) = a_{t-1}(x), \forall x \in S$ 
  - break
- **[Centroids]**  $\forall j \in [k], \quad c_j \leftarrow \text{average}(\{\phi(x): x \in S, a_t(x) = j\})$

**Output:**  $c_1, \dots, c_k$  and  $\{a_t(x_i)\}_{i \in [n]}$





# Kernelizing K-means algorithm (cont'd)

- How to calculate  $\|\phi(x) - c_j\|_2^2$  without explicitly evaluating  $\phi$ ?
- Key observation:  $c_j$  always takes the form  $c_j = \frac{1}{|S|} \sum_{i \in S} \phi(x_i)$  for some  $S$ , and therefore has the form  $c_j = \sum_{i=1}^n \alpha_i \phi(x_i)$

- Therefore,

$$\begin{aligned} \|\phi(x) - c_j\|_2^2 &= \langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \sum_{i=1}^n \alpha_i \phi(x_i) \rangle + \langle \sum_{i=1}^n \alpha_i \phi(x_i), \sum_{i=1}^n \alpha_i \phi(x_i) \rangle \\ &= K(x, x) - 2 \sum_{i=1}^n \alpha_i K(x, x_i) + \sum_i \sum_j \alpha_i \alpha_j K(x_i, x_j) \end{aligned}$$

- Efficiently computable: only requires evaluating  $K$  now

# Clustering as cost minimization: additional remarks

- k-means objective function is not the only one used in practice

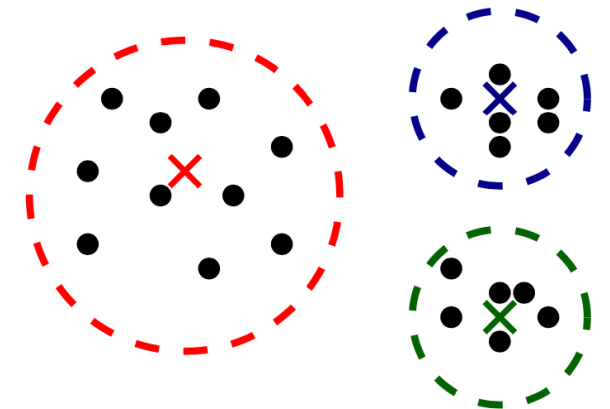
$$f(c_1, \dots, c_k) = \sum_{i=1}^n \min_{j \in [k]} \|x - c_j\|_2^2$$

- Alternative popular cost functions:

$$\text{k-median: } f(c_1, \dots, c_k) = \sum_{i=1}^n \min_{j \in [k]} \|x - c_j\|_2$$

$$\text{k-center: } f(c_1, \dots, c_k) = \max_i \min_{j \in [k]} \|x - c_j\|_2$$

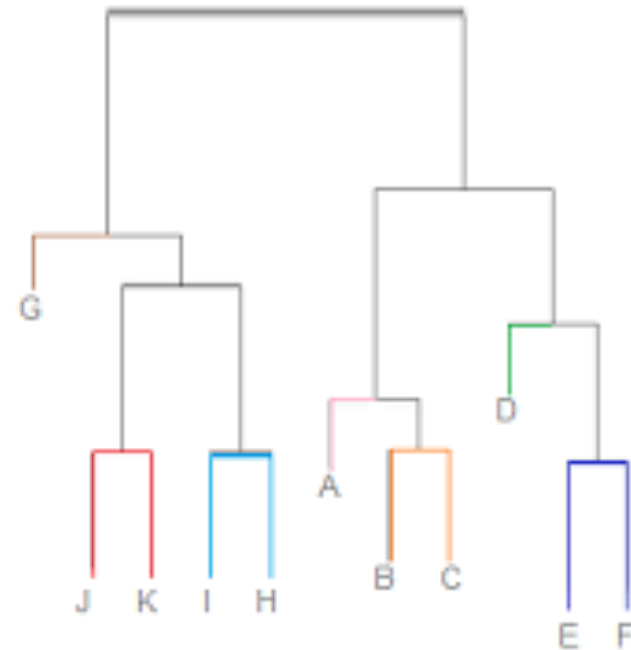
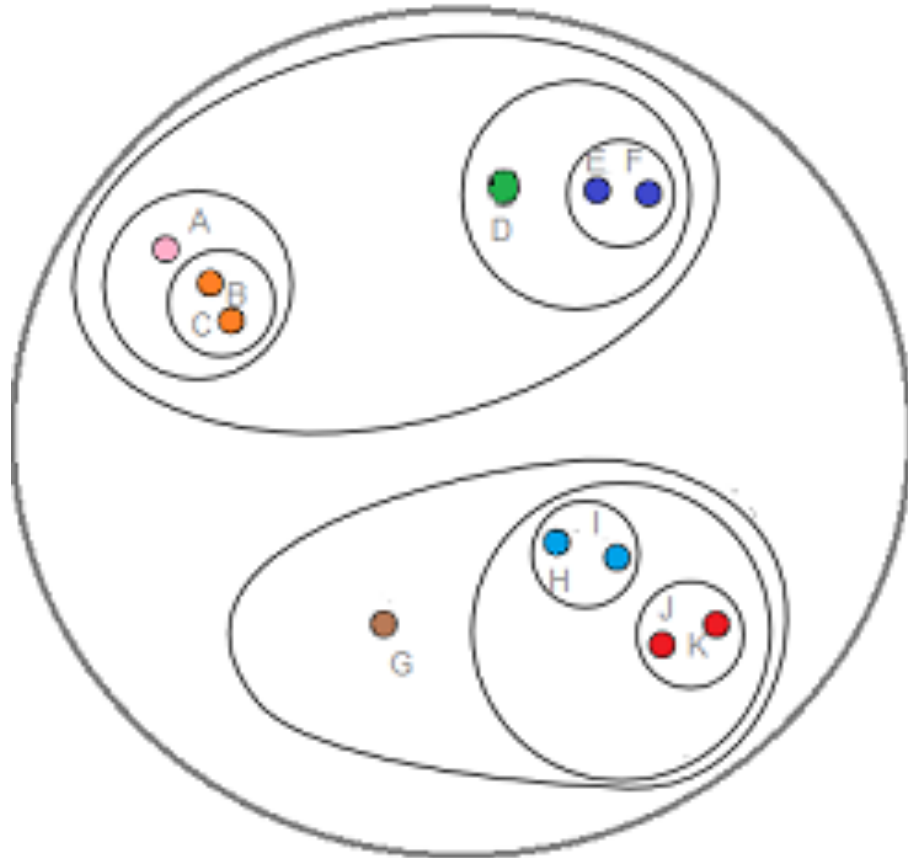
- Furthermore, we don't have to restrict to using the  $\ell_2$  metric



# Hierarchical clustering

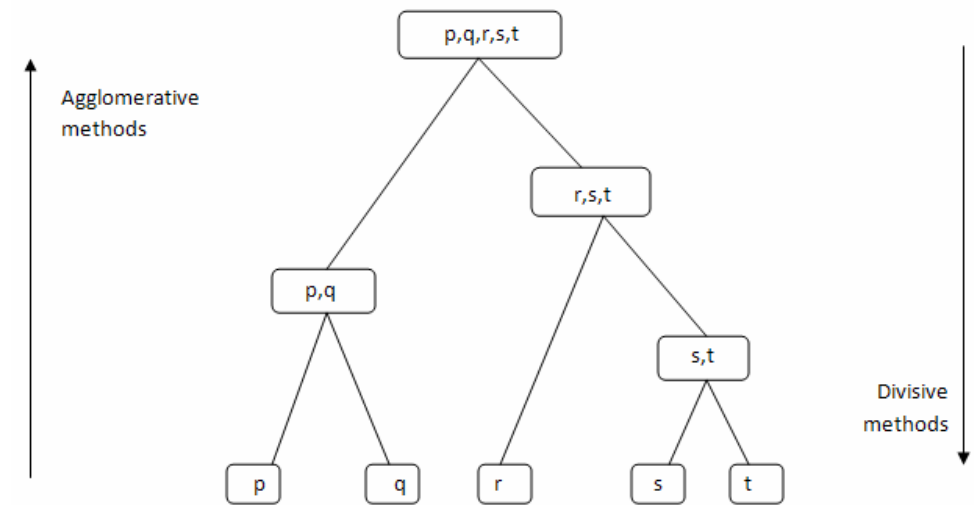
# Hierarchical clustering – getting rid of tuning k

- Idea: produce a tree structure over objects
- Can prune the tree appropriately to fit application needs (e.g. cluster radius / size requirements)



# Hierarchical clustering

- Method 1: Top-down (divisive)
  - $k$ -means clustering with  $k=2$
  - Do this recursively on each resulting cluster (no more recursion when there is only one point in a cluster)
  - You now have a binary tree.
- Method 2: bottom-up (agglomerative, more popular)
  - Start with every point  $x_i$  being a singleton cluster
  - Repeatedly pick a pair of clusters with the smallest 'distance'
  - How do we define a distance between two clusters?



# Agglomerative clustering: Distance between two clusters

- Single linkage

- $\text{dist}(C, C') = \min_{x \in C, x' \in C'} \|x - x'\|_2$

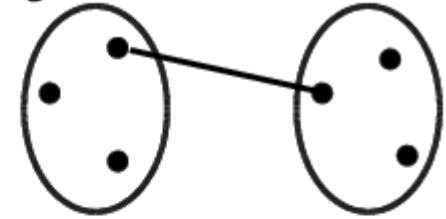
- Complete linkage

- $\text{dist}(C, C') = \max_{x \in C, x' \in C'} \|x - x'\|_2$

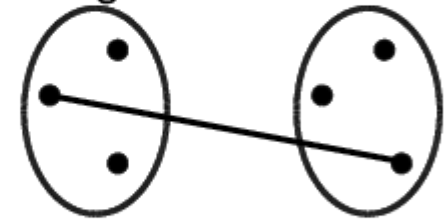
- Average linkage

- $\text{dist}(C, C') = \frac{1}{|C| \cdot |C'|} \sum_{x \in C} \sum_{x' \in C'} \|x - x'\|_2$

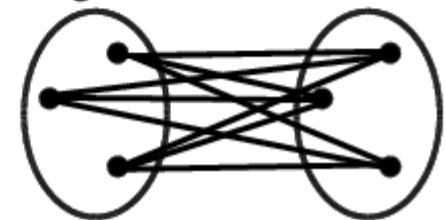
Single Linkage



Complete Linkage



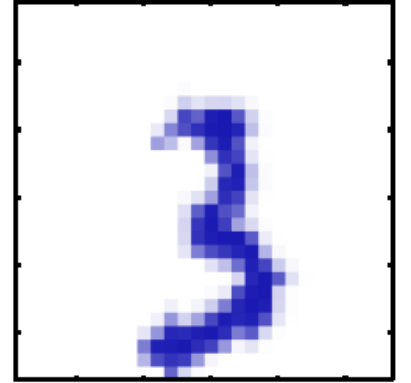
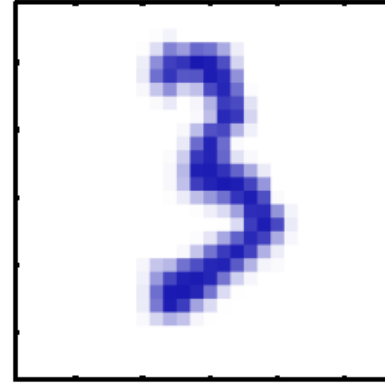
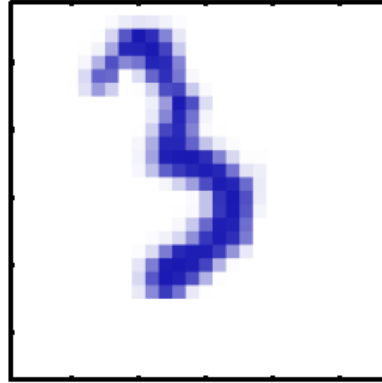
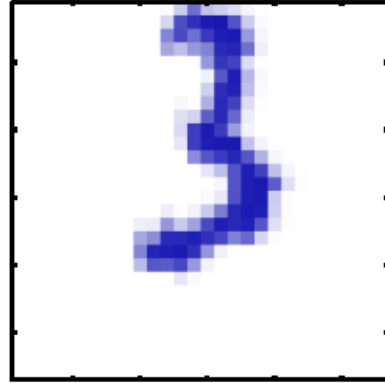
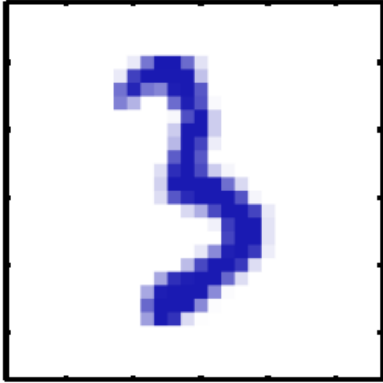
Average Linkage



# Dimensionality Reduction and Principal Component Analysis (PCA)

# Motivation

*Data often have a lot of redundant information...*



**Example** A dataset consisting of a hand-drawn 3 at random locations and rotations in a 100x100 pixel image.

**Data Dimension**  $100 \times 100 = 10,000$

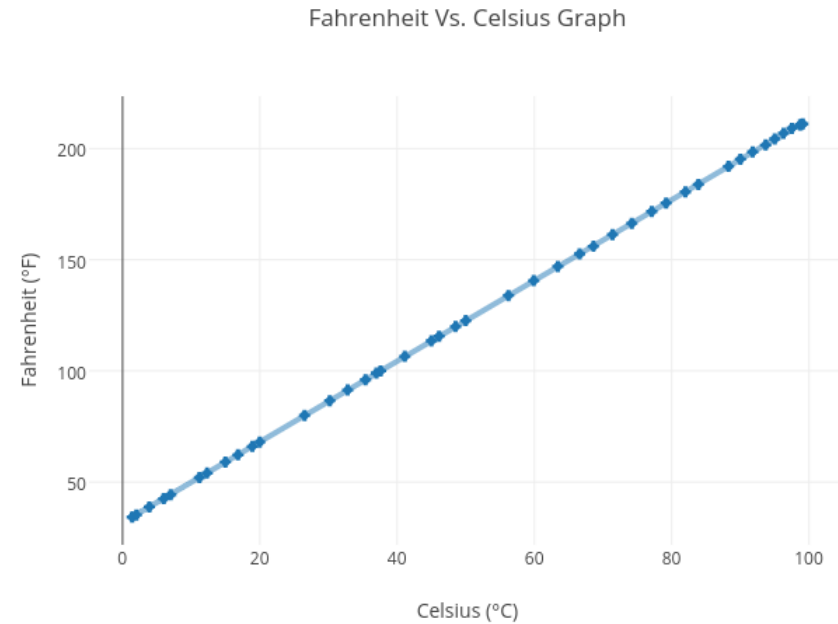
**Intrinsic Dimension** 3 (X-position, Y-position, Rotation)



# Motivation

*...or data have strongly dependent features...*

Fahrenheit	Celsius
3.1	-16.1
100.5	38.1
27.3	-2.6
18.1	-7.7
18.9	-7.3
21.7	-5.7
...	...

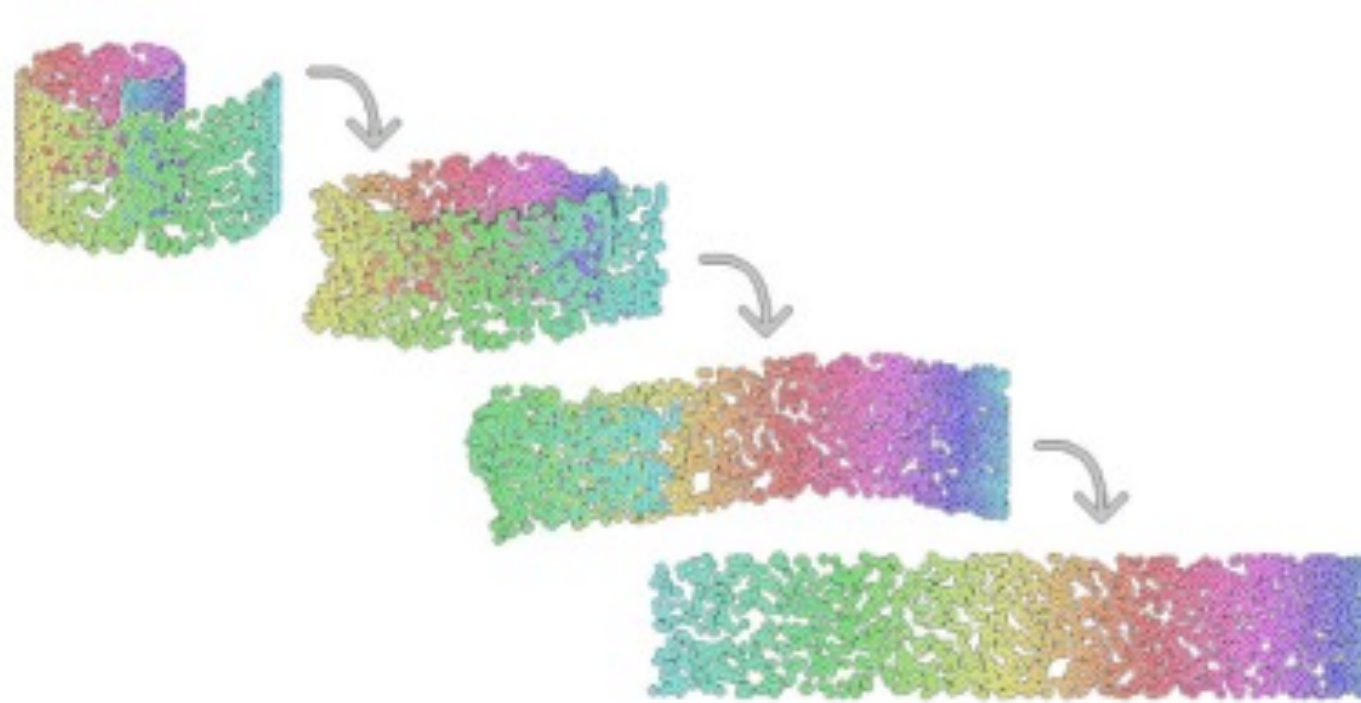


Linear Function

$$F = 1.8C + 32$$

# Motivation

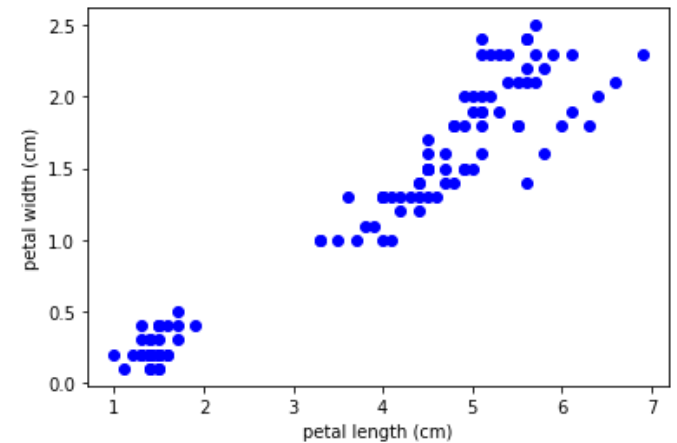
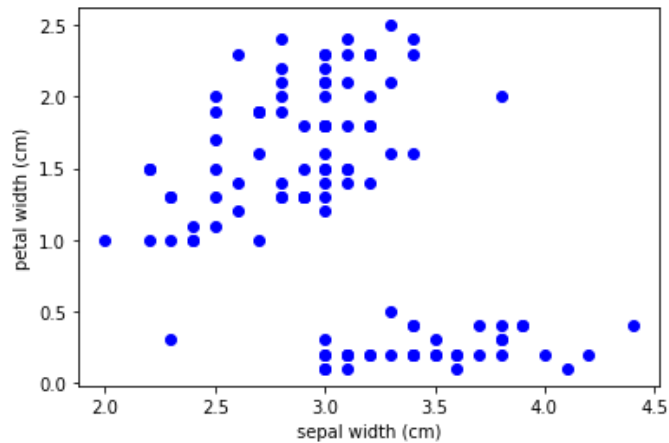
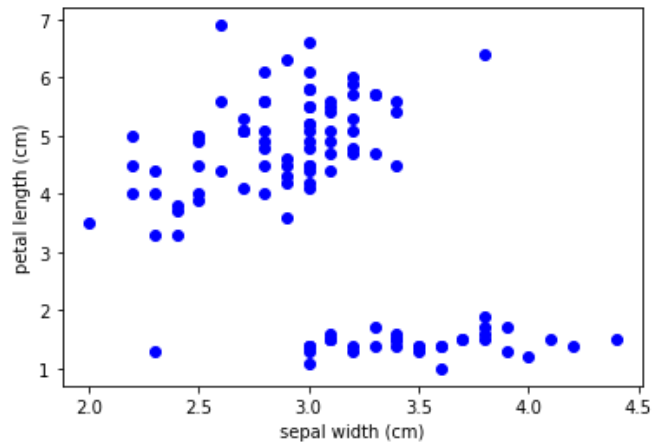
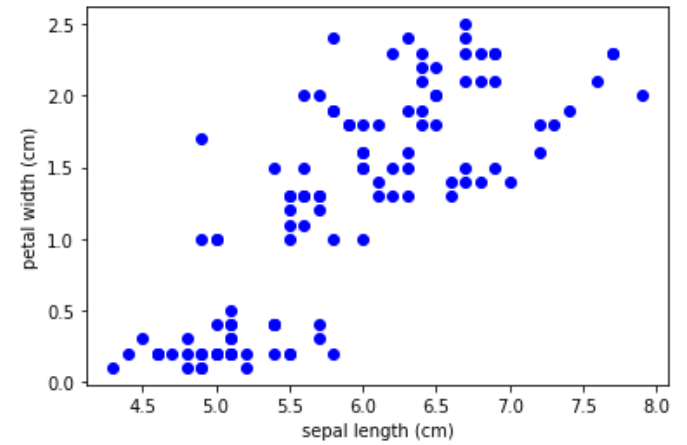
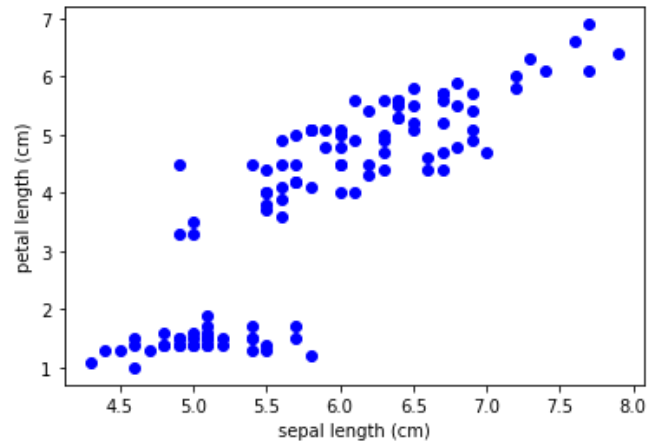
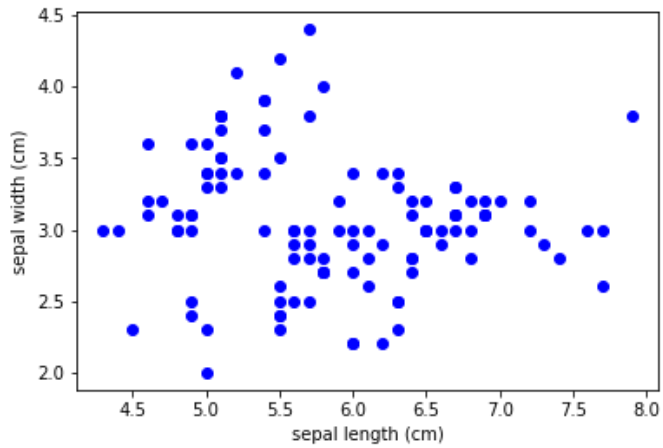
...or data are high-dimensional and hard to visualize...



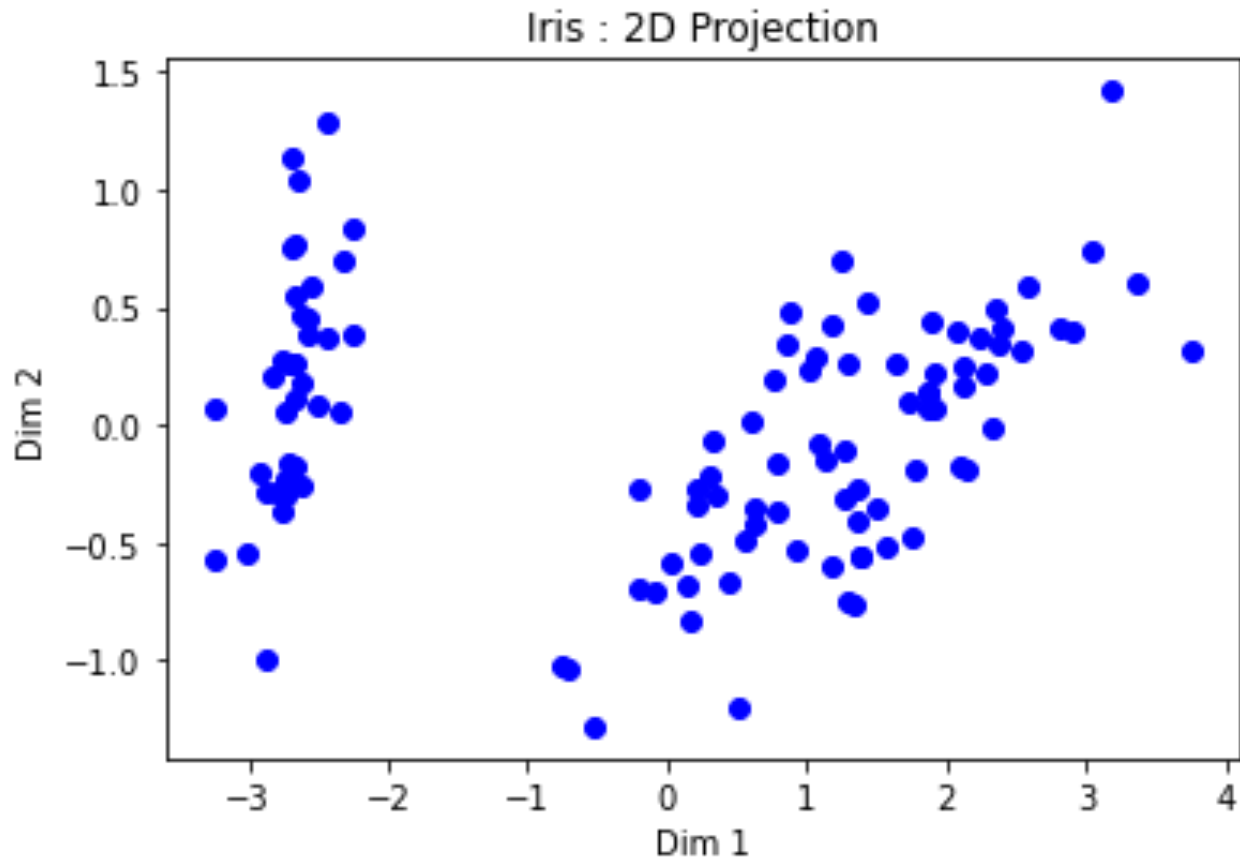
...in all cases finding lower *intrinsic dimension* is useful

# Example : Iris Dataset

*Recall that the Iris dataset has 4 features:  
sepal length / width, petal length / width...*



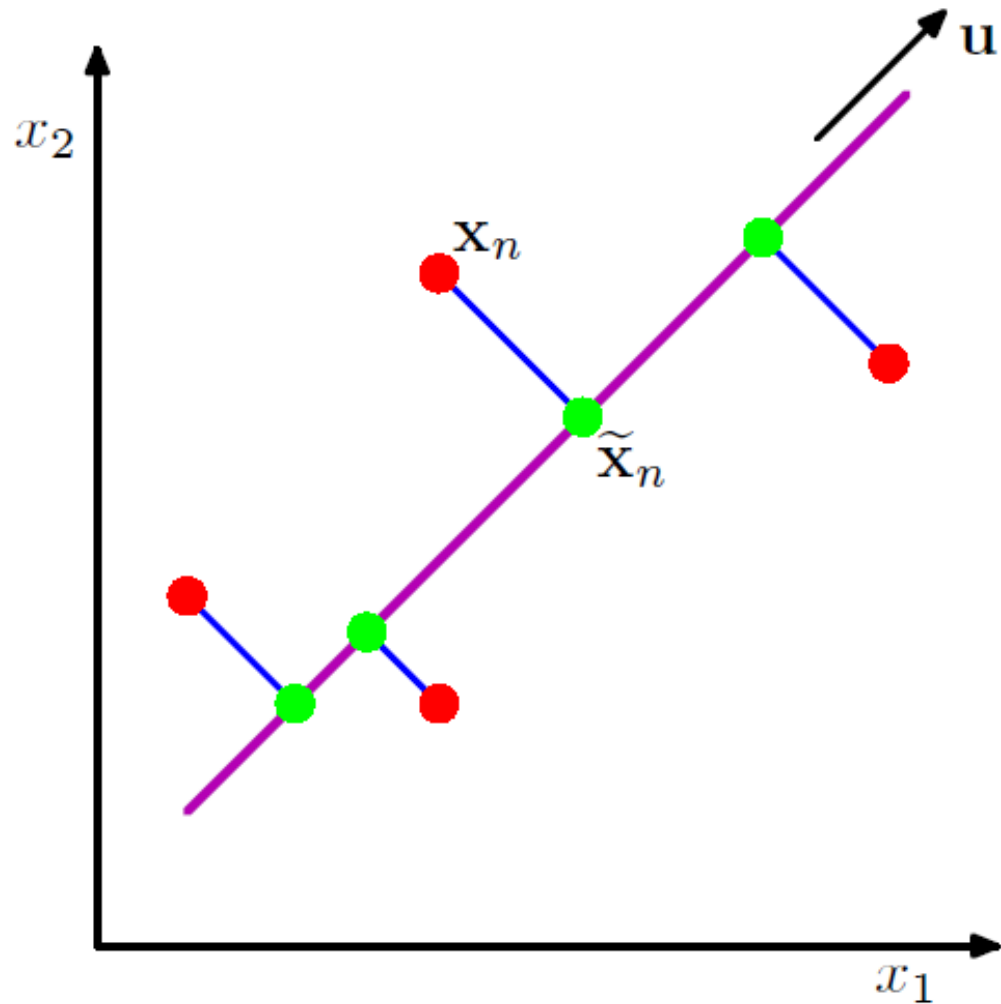
# Example : Iris Dataset



Data still cluster in a two-dimensional subspace

We can fit model in 2D to reduce complexity, visualize results, etc.

# Linear Dimensionality Reduction

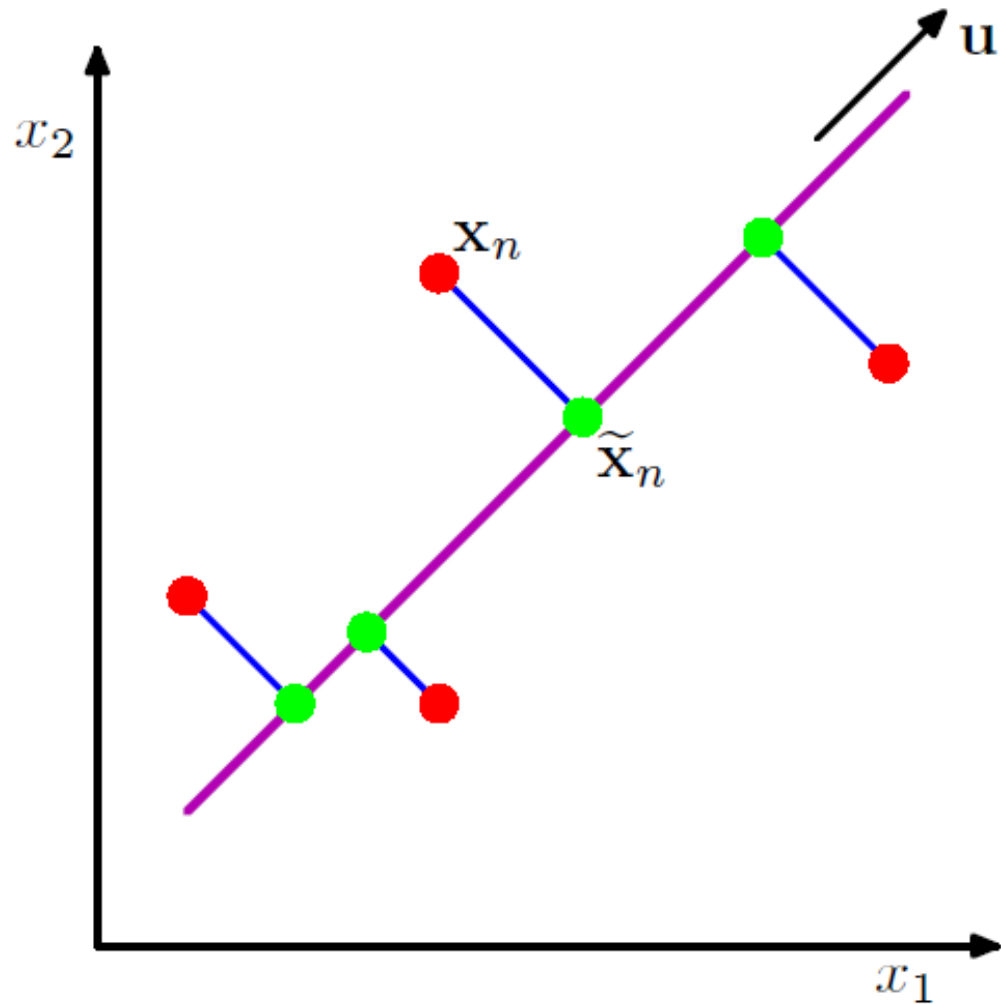


*Project data onto a line or plane...*

*...one of the simplest dimensionality reduction approaches*

**First, let's review some linear algebra...**

# Linear Dimensionality Reduction



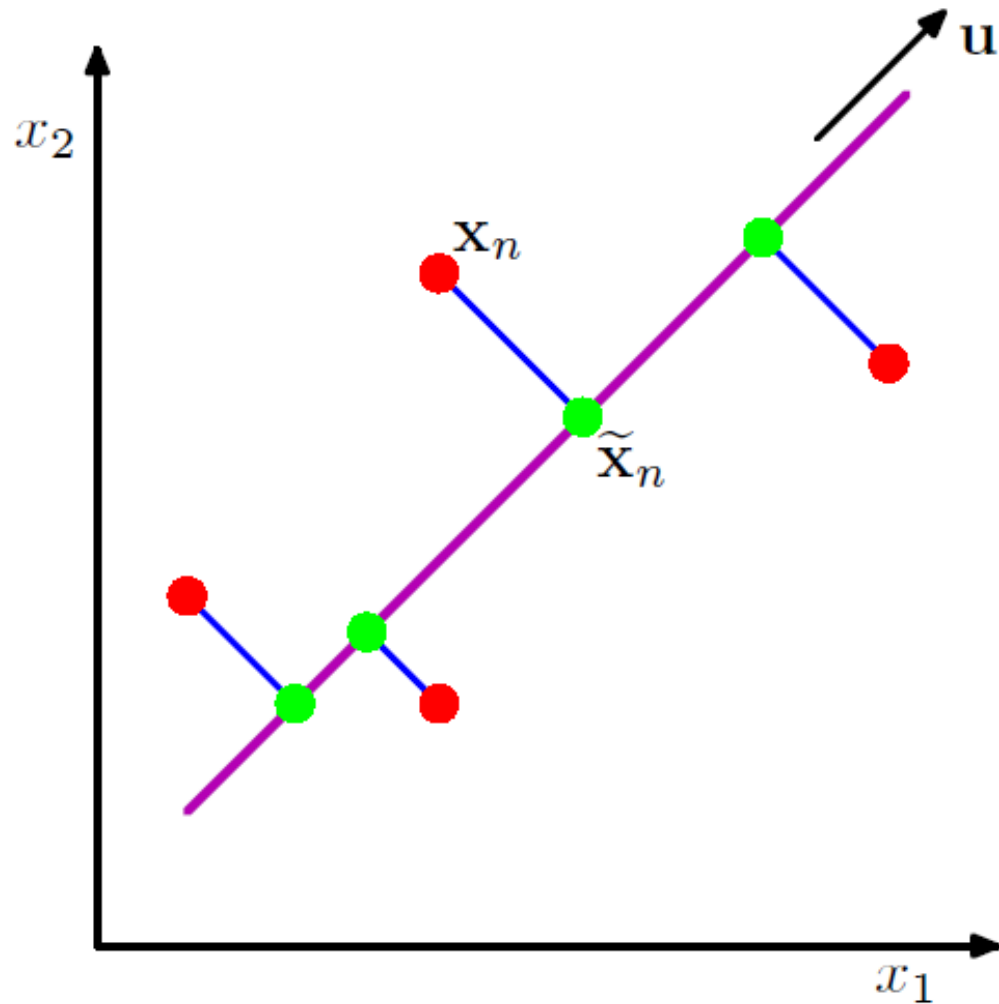
Projecting data onto a vector is a simple inner product,

$$\tilde{x}_n = u^T x_n$$

We call  $u$  the *linear subspace*

**Question** Why would dimensionality reduction be better than feature selection (e.g. choose 1-D features  $X_1$  or  $X_2$ )?

# Linear Dimensionality Reduction



Projecting data onto a vector is a simple inner product,

$$\tilde{x}_n = u^T x_n$$

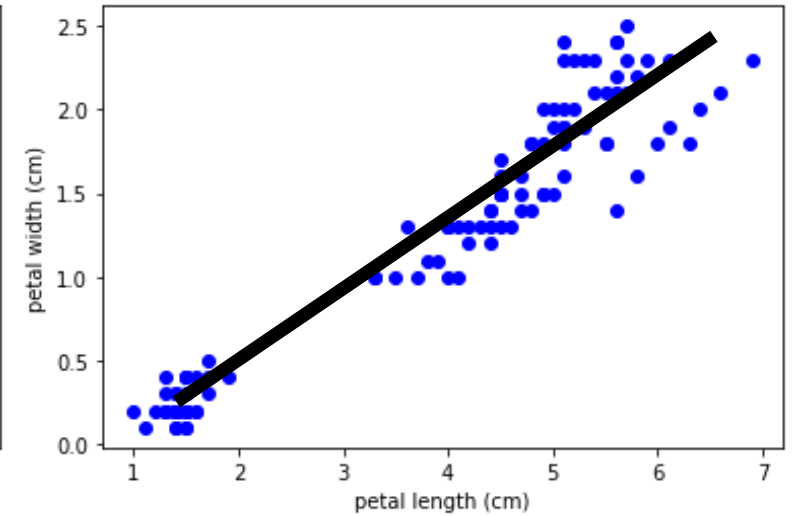
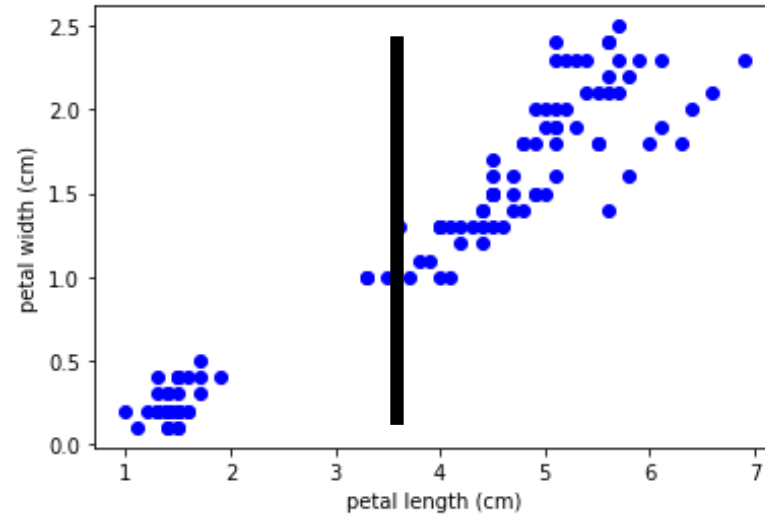
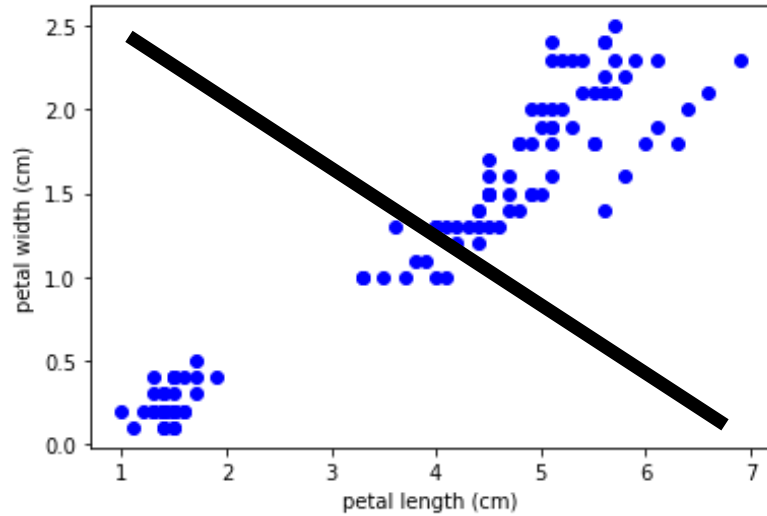
We call  $u$  the *linear subspace*

**Answer** No features are discarded (uses all the data),

$$\tilde{x}_n = u_1 x_{n1} + u_2 x_{n2}$$

# Linear Dimensionality Reduction

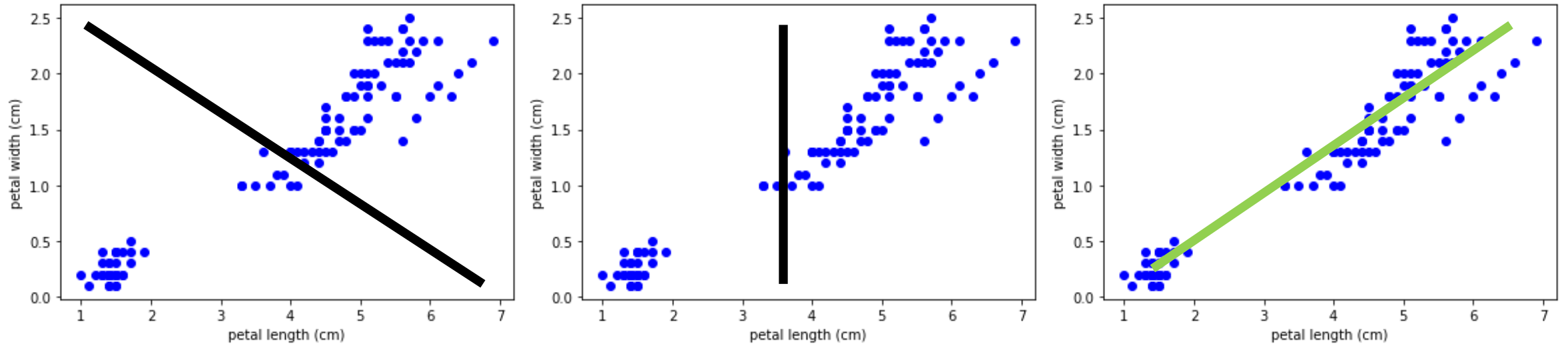
*Which choice of subspace is best? And why?*





# Linear Dimensionality Reduction

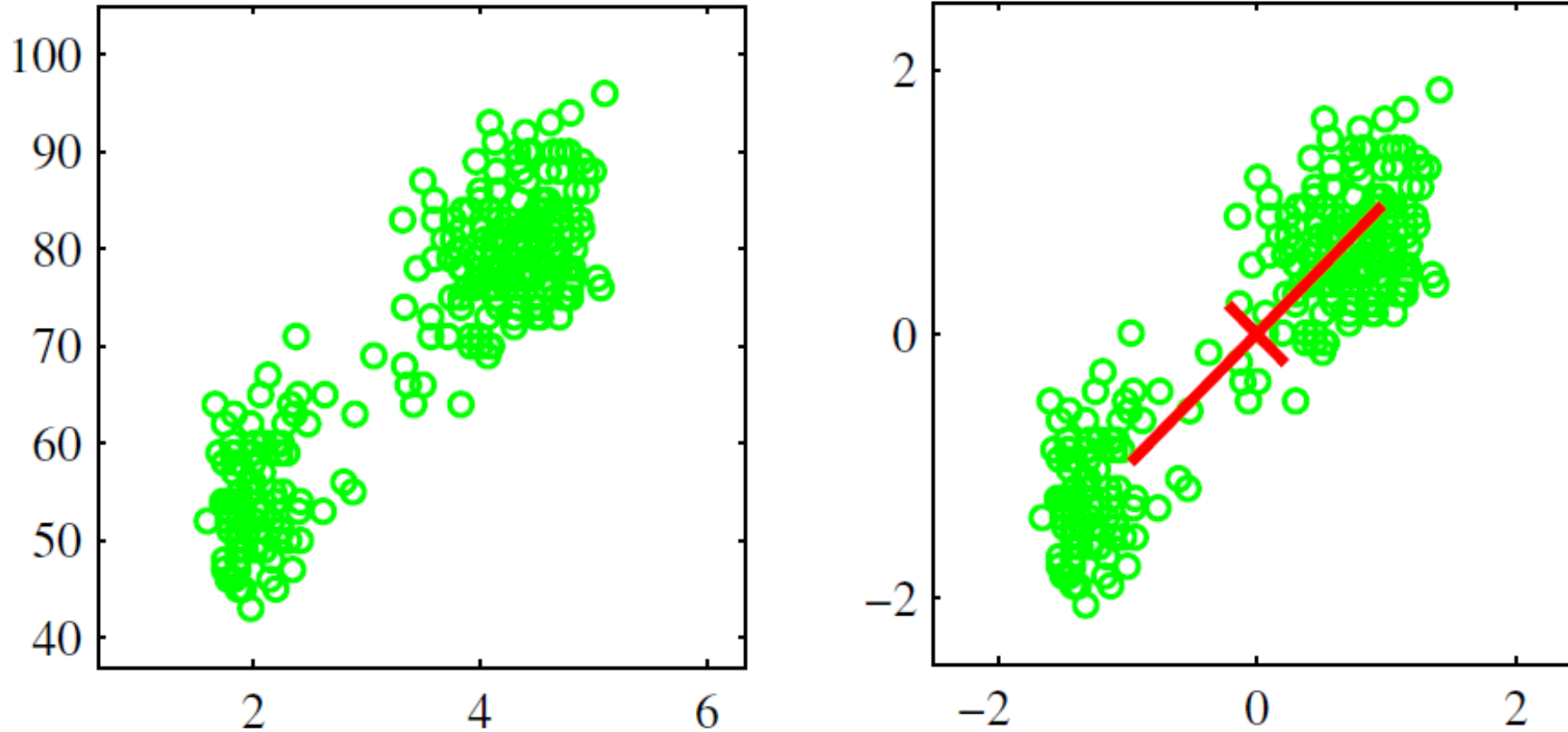
*Which choice of subspace is best? And why?*



**Idea** Choose the subspace that captures the most variation in the original data

# Principal Component Analysis (PCA)

Identify directions of *maximum variation* as subspaces...



...we call each direction a *principal component*

# Principal Component Analysis (PCA)

First, center the data by subtracting the sample mean,

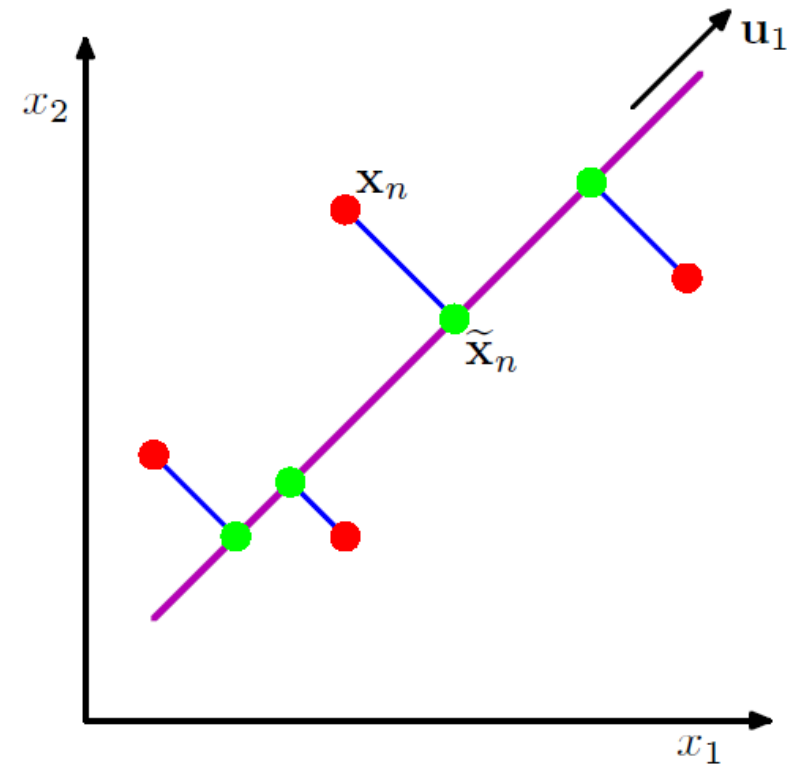
$$\bar{x} = \frac{1}{N} \sum_{n=1}^N x_n$$

Variance of projected subspace,

$$\frac{1}{N} \sum_{n=1}^N \left( u^T x_n - u^T \bar{x} \right)^2$$

Projection of  
n<sup>th</sup> data point

Projection of  
mean




# Minimum Variance Formulation

A little algebra...

$$\frac{1}{N} \sum_{n=1}^N (u^T x_n - u^T \bar{x})^2 = \frac{1}{N} \sum_{n=1}^N \{u^T (x_n - \bar{x})\}^2 \quad \text{Pull out } u$$

$$\text{Quadratic form} = \frac{1}{N} \sum_{n=1}^N u^T (x_n - \bar{x})(x_n - \bar{x})^T u$$

Define:  $S = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})(x_n - \bar{x})^T$

Then:  $\frac{1}{N} \sum_{n=1}^N (u^T x_n - u^T \bar{x})^2 = u^T S u$   **This is what we will optimize over u**

# Minimum Variance Formulation

*Find  $u$  so that projected variance is maximal...*

$$\max_u u^T S u$$

Don't want to *cheat* with large magnitude  $u$ , so we add penalty,

$$\max_u u^T S u - \lambda u^T u$$

Set the derivative (gradient) to zero and solve...

$$S u - \lambda u = 0$$

$$S u = \lambda u$$

What equation is this?

$u$  is an *eigenvector* with  
*eigenvalue*  $\lambda$

# Recap of Concepts

- Learning a reduced *intrinsic dimension* is useful for a bunch of reasons
- The easiest approach is to find a *linear subspace*
- PCA defines the linear subspace as that which maximizes variance of the projected data
- The set of subspaces are defined by the *eigenvectors*,

$$\max_u u^T S u - \lambda u^T u$$

$$S u = \lambda u$$

**But what is an eigenvector?**

# Linear Transformations

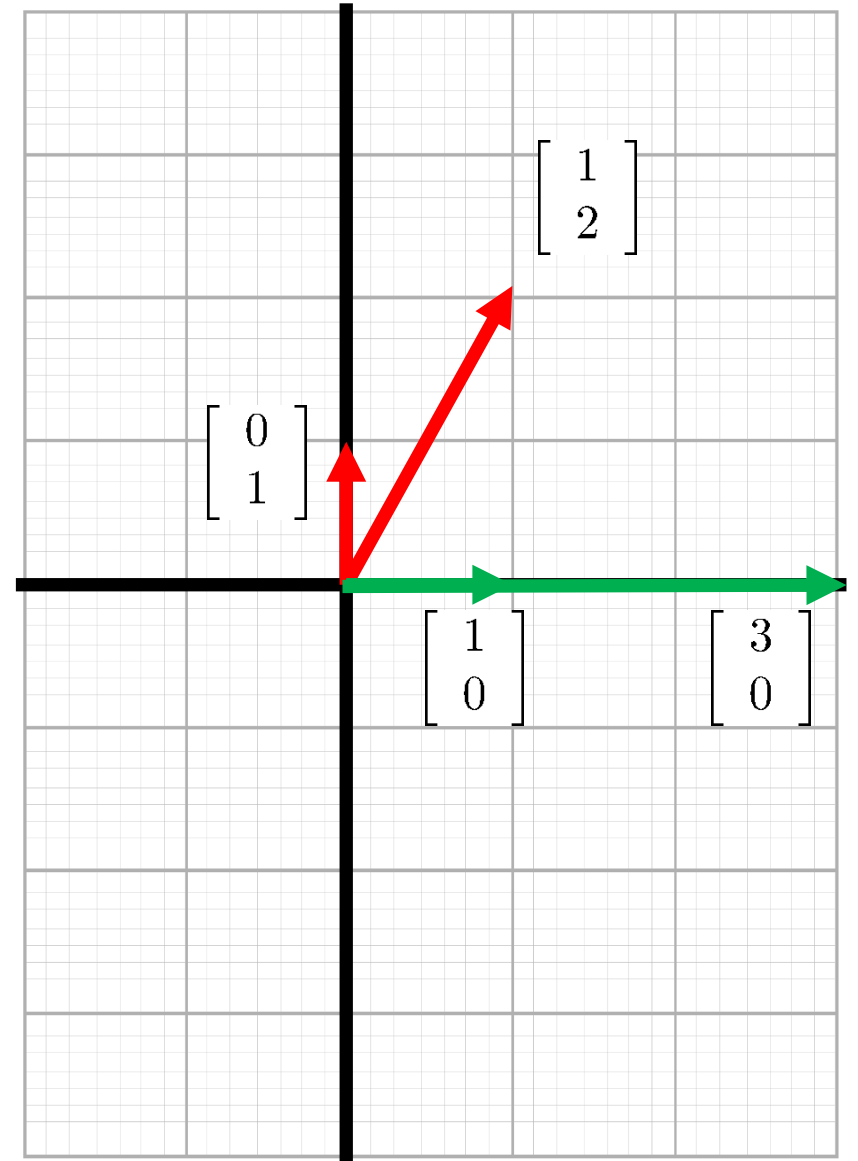
Consider the matrix:  $\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$

Let's multiply it with some vectors...

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \cdot 0 + 1 \cdot 1 \\ 0 \cdot 0 + 2 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 + 1 \cdot 0 \\ 0 \cdot 1 + 2 \cdot 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

- Matrix transforms vectors from one basis to another
- Columns are transformation of standard basis



# Eigenstuff

Observe that the X-axis vector just gets “stretched out”,

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

Factoring out the 3 we have,

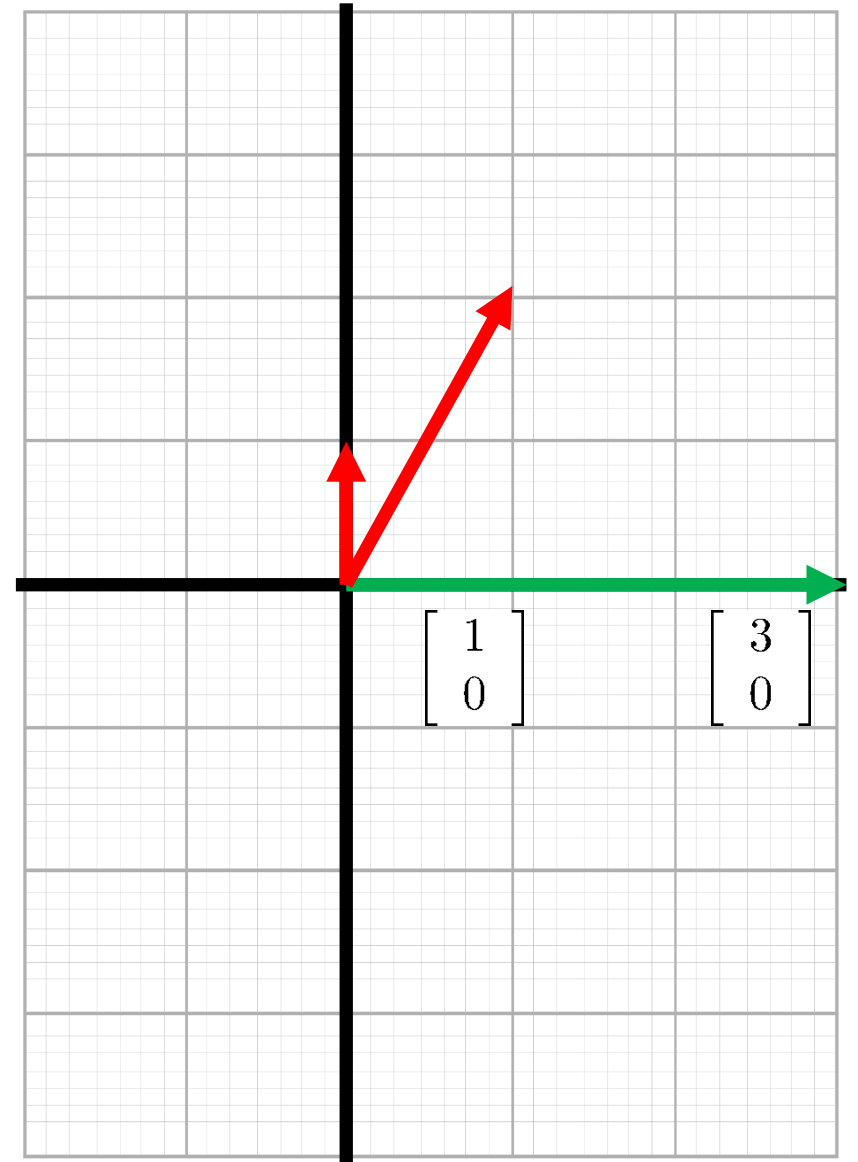
$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$\mathbf{S}$                        $\mathbf{u}$                        $\lambda$                        $\mathbf{u}$

Define some variables and we have the equation,

$$Su = \lambda u$$

So  $(1,0)^T$  is an *eigenvector* of  $S$  with *eigenvalue* 3





# Eigenstuff

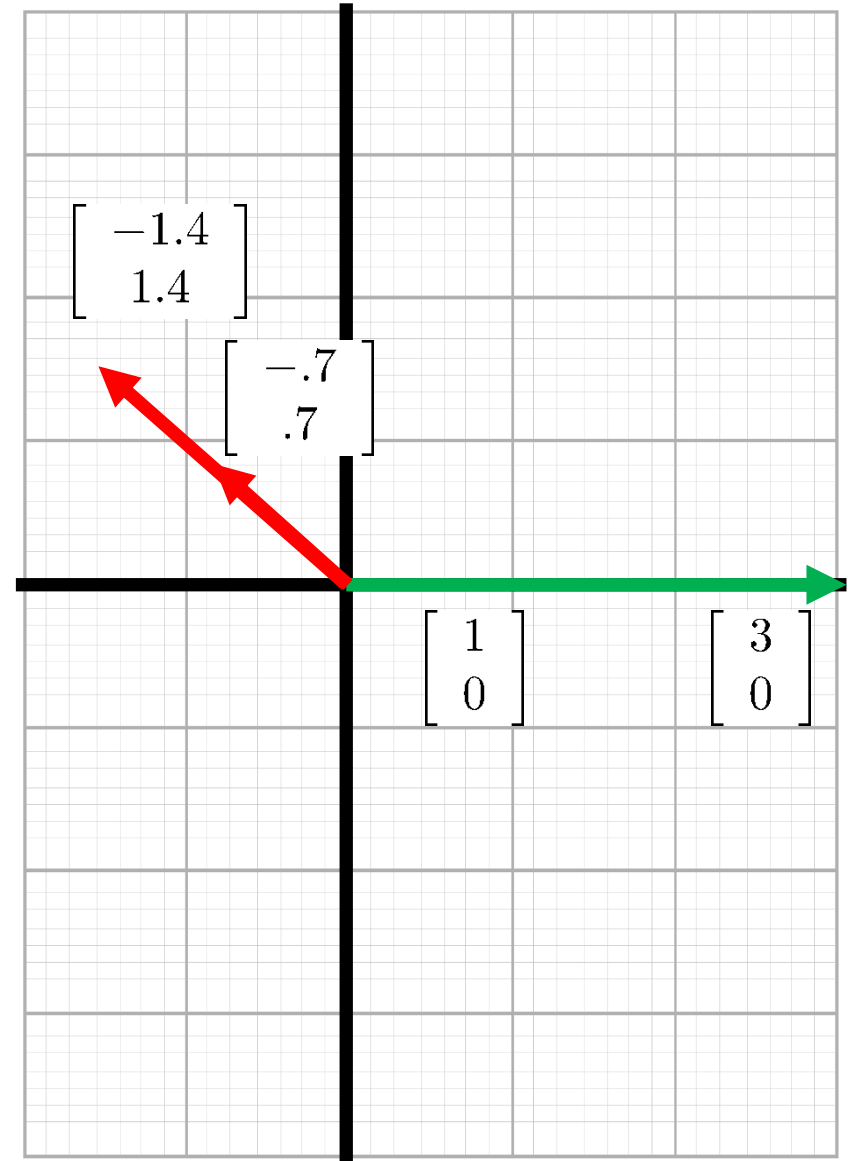
Transformation has one other eigenvector,

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -0.7 \\ 0.7 \end{bmatrix} = \begin{bmatrix} -1.4 \\ 1.4 \end{bmatrix} = 2 \begin{bmatrix} -0.7 \\ 0.7 \end{bmatrix}$$

- Complete eigendecomposition of S

$$\text{Eigenvectors } \begin{bmatrix} 1 & -0.7 \\ 0 & 0.7 \end{bmatrix} \quad \text{Eigenvalues } \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

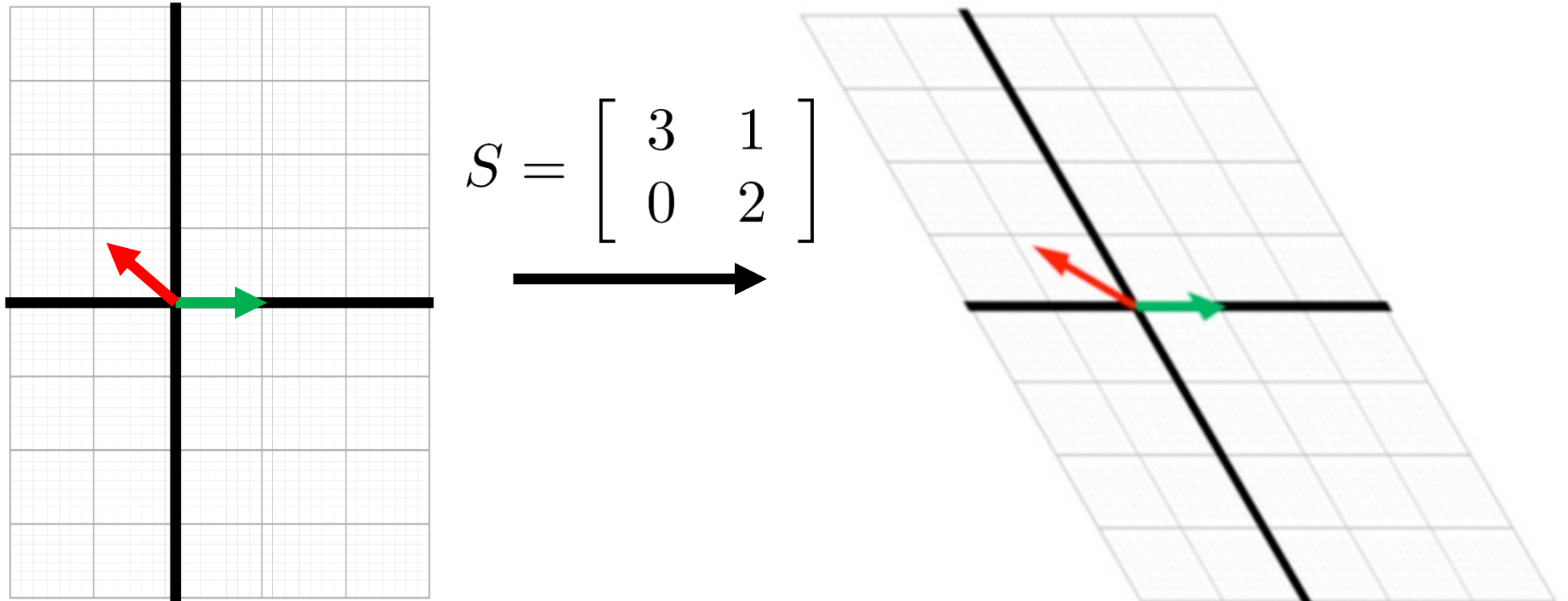
- Eigenvectors of linear transformation S are only stretched / shrunk / flipped
- Eigenvalues tell how much they are stretched / shrunk / flipped



# Eigenstuff

Eigenvectors  $\begin{bmatrix} 1 & -0.7 \\ 0 & 0.7 \end{bmatrix}$  Eigenvalues  $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$

Eigendecomposition highlights what a linear transformation does by identifying directions that are *not* altered



# Eigenstuff

*Eigenvectors / values of a matrix solve the equation*

$$Su = \lambda u$$

- Matrix  $S$  may have *multiple* eigenvectors / values that solve the above equation
- For  $D$ -dimensional  $u$  can find all vectors in  $O(D^3)$  time
- PCA finds  $M < D$  vectors with largest eigenvalues
- Can find  $M < D$  sorted eigenvectors in  $O(MD^2)$  time
- Note that  $D$  can be large!

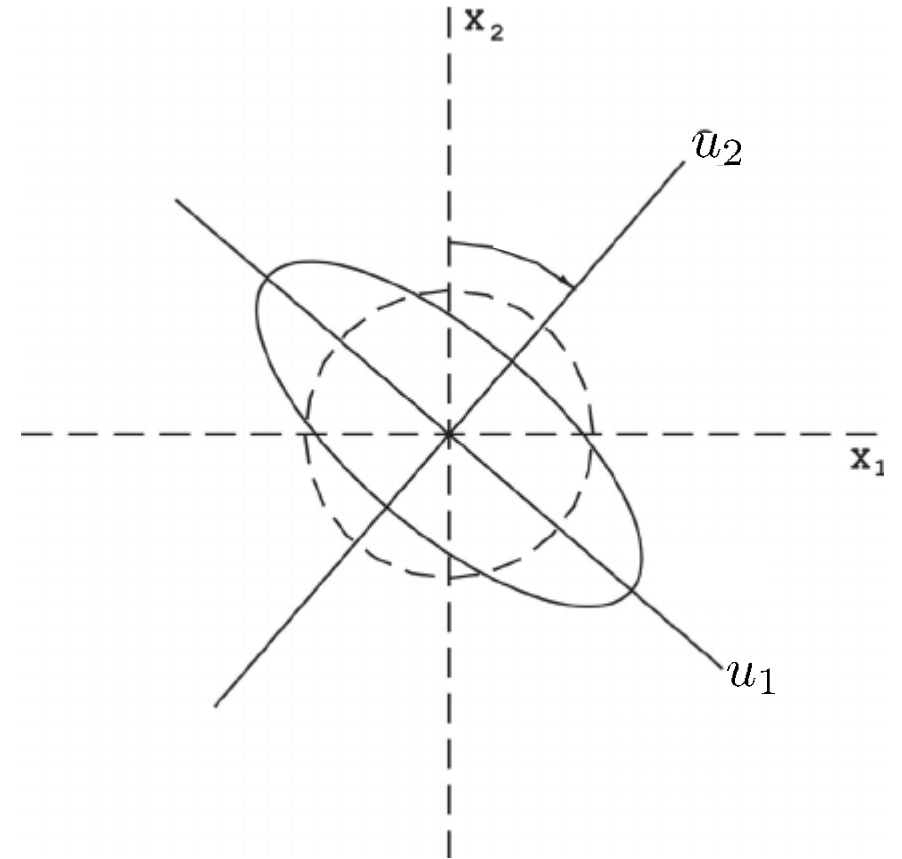
# Eigenvectors and Ellipses

*How does this connect to PCA?*

Take all points on a unit circle and apply the linear transformation  $S$

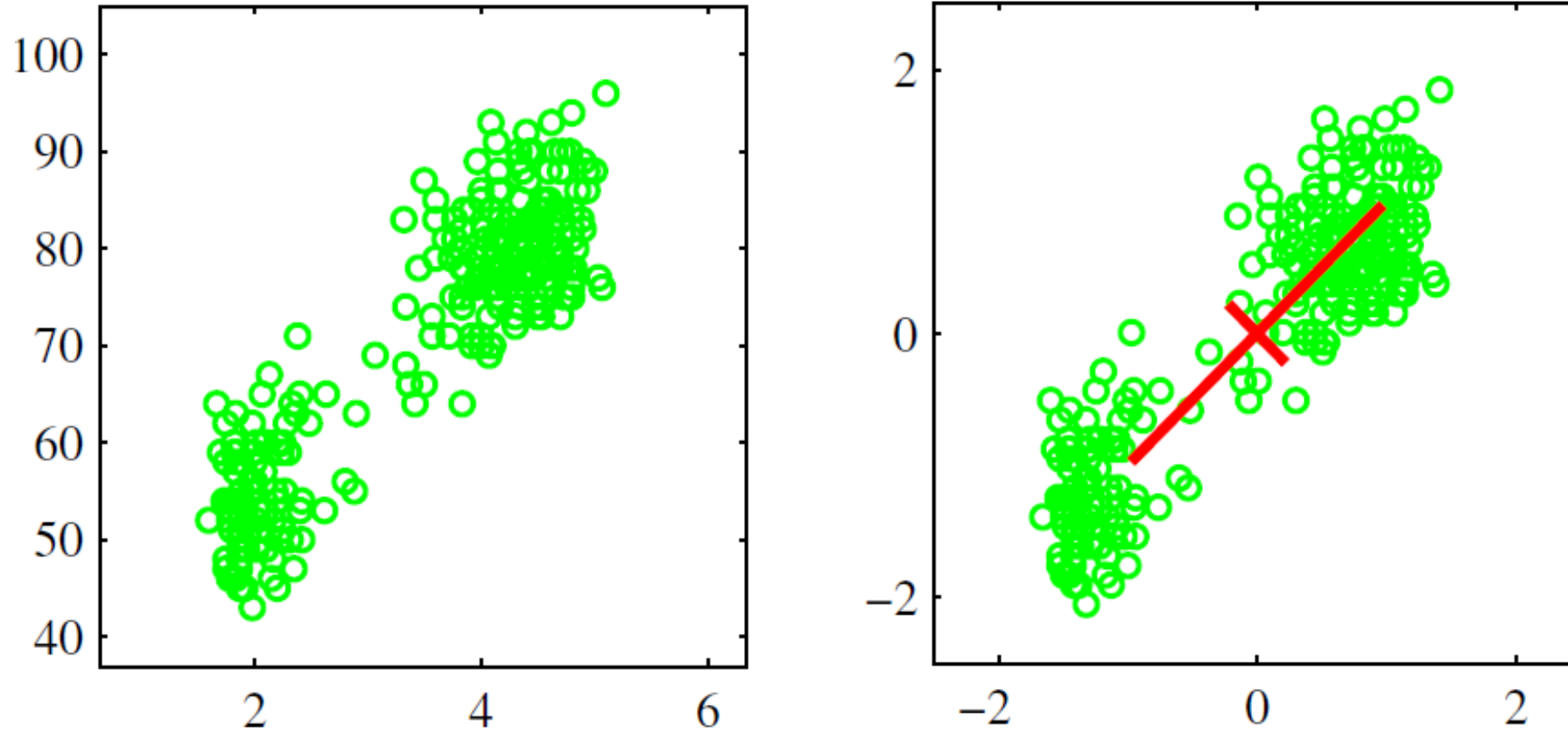
If  $S$  is a *covariance matrix*, then points will be transformed into an ellipse and...

- Eigenvectors are axes of ellipse
- Eigenvalues are length of each axes
- Sort eigenvalues to get major / minor / etc. axes
- In the context of PCA eigenvectors = **principal components**



# Principal Component Analysis (PCA)

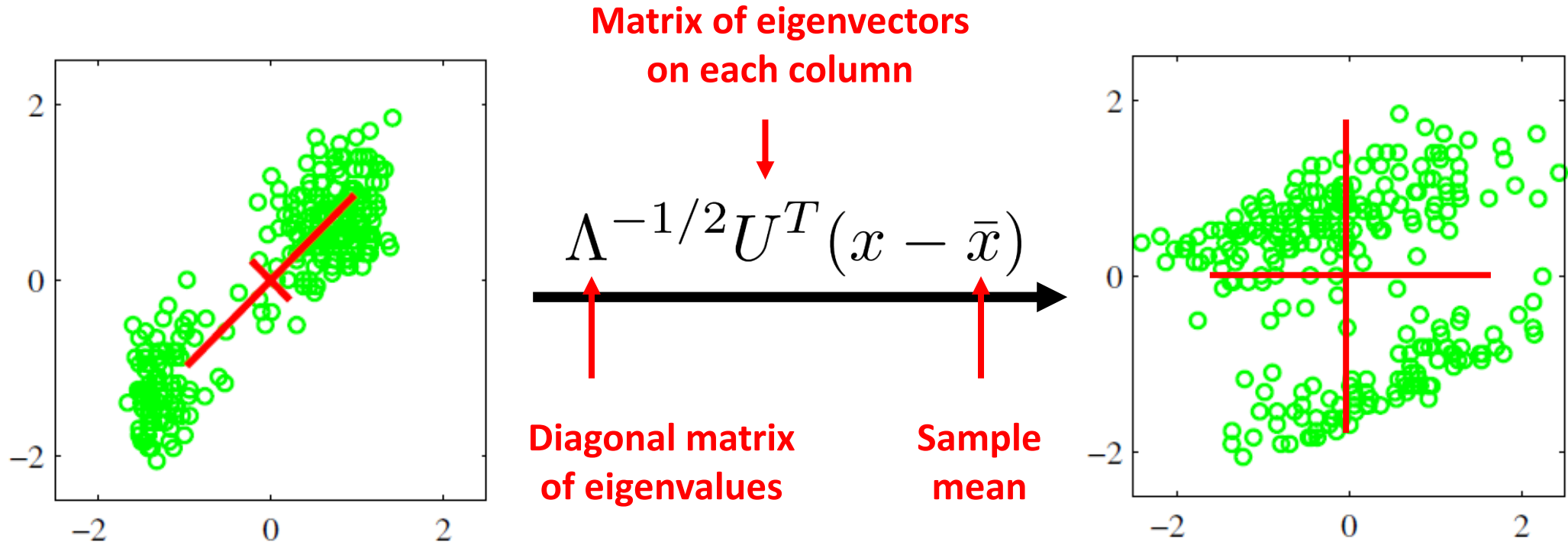
Sort eigenvectors by their eigenvalues...



...amount of variance in each principal component decreases with eigenvalue

# Data “Whitening”

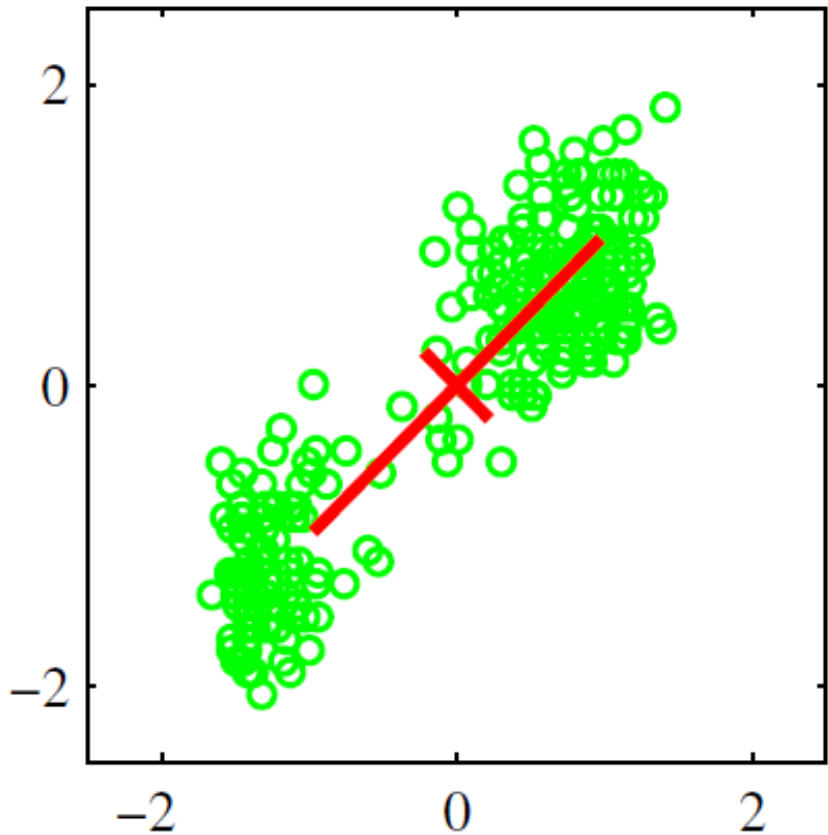
Multiplying data by eigenvectors transforms data so they are zero-mean and uncorrelated



Data whitening can be an important preprocessing step for many data science applications (even if we don't care about dimensionality reduction)

# Principal Component Analysis (PCA)

How much variance is captured by just the first principal component (i.e. eigenvector with largest eigenvalue)?



Let  $u_1$  be the first principal component, then variance of first PC is,

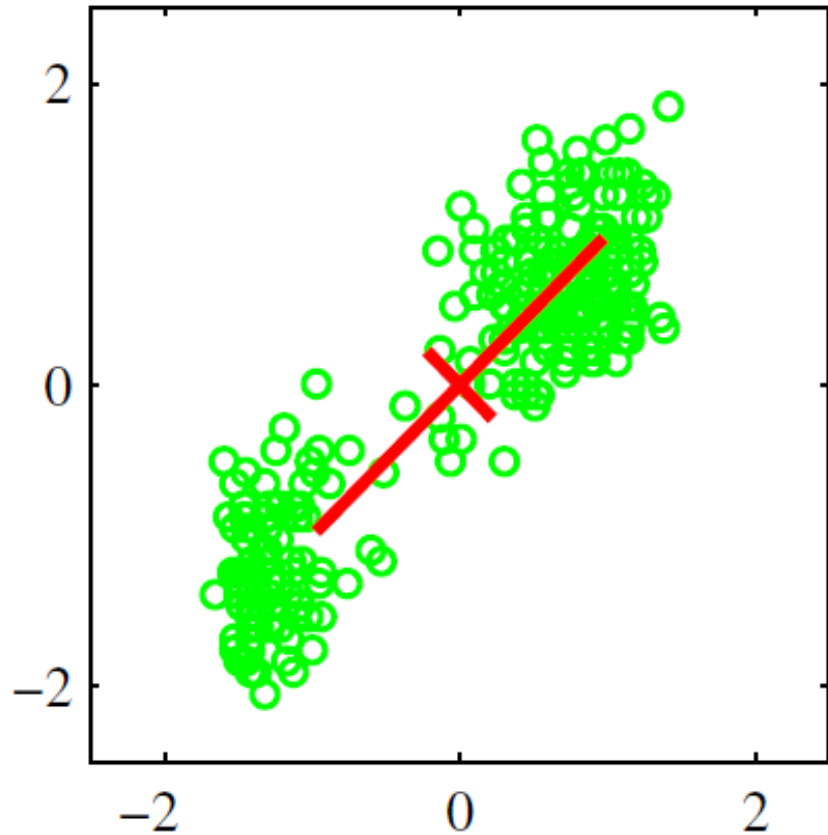
$$\frac{1}{N} \sum_n \{u_1^T (x_n - \bar{x})\}^2$$

How much in the second PC?

$$\frac{1}{N} \sum_n \{u_2^T (x_n - \bar{x})\}^2$$

# Explained Variance

How much variance is captured in  $M < D$  principal components?



$$\frac{1}{N} \sum_{m=1}^M \sum_n \left\{ u_m^T (x_n - \bar{x}) \right\}^2$$

We call this the *explained variance* of the first  $M$  principal components

Divide by total variance to find percentage of the total variance explained by the subspace



# EM for PCA

We can derive an *expectation maximization* (EM) algorithm for PCA...but why would we do this if PCA is closed-form?

## For $N$ data points of $D$ -dimensions

- Computing the first  $M < D$  principal components takes  $O(MD^2)$
- Evaluating the covariance needs  $O(ND^2)$  time
- Most expensive step in EM is  $O(NDM)$  time
- If  $D$  large and  $M \ll D$  then  $O(NDM) \ll O(ND^2)$

# Concept Recap

## Eigenvectors

- For a general linear transform – identify directions that are only stretched / shrunk / flipped
- For a covariance matrix – identify axes of the ellipse that describes covariance

## PCA

- Learns linear subspace as  $M < D$  principal components corresponding to  $M$  eigenvectors with largest eigenvalue
- Can be used to *whiten* (standardize, de-correlate) data
- Explained variance of  $M$  principal components easily calculated as percent of total explained variance in whitened data

# sklearn.decomposition.PCA

## Parameters

**n\_components** : *int, float or 'mle', default=None*

Number of components to keep. if n\_components is not set all components are kept:

**copy** : *bool, default=True*

If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit\_transform(X) instead.

**whiten** : *bool, default=False*

When True (False by default) the `components_` vectors are multiplied by the square root of n\_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

# sklearn.decomposition.PCA

## Attributes

**components\_** : *ndarray of shape (n\_components, n\_features)*

Principal axes in feature space, representing the directions of maximum variance in the data. Equivalently, the right singular vectors of the centered input data, parallel to its eigenvectors. The components are sorted by `explained_variance_`.

**explained\_variance\_** : *ndarray of shape (n\_components,)*

The amount of variance explained by each of the selected components. The variance estimation uses

**explained\_variance\_ratio\_** : *ndarray of shape (n\_components,)*

Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

**singular\_values\_** : *ndarray of shape (n\_components,)*

The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

# Caution

Careful with the following parameter,

**copy** : *bool, default=True*

If False, data passed to fit are overwritten and running `fit(X).transform(X)` will not yield the expected results, use `fit_transform(X)` instead.

**Wrong**

```
pca = PCA(n_components=2, copy=False).fit(X)
X_pca = pca.transform(X) ← X already modified
```

**Right**

```
pca = PCA(n_components=2).fit(X)
X_pca = pca.transform(X)
```

**Why would you prefer one over the other?**

**Right**

```
X_pca = PCA(n_components=2, copy=False).fit_transform(X)
```

# Example : PCA on Iris Data

Load Iris data without labels,

```
iris = datasets.load_iris(as_frame=True)
X = iris.data
```

Find PCA with 2 principal components,

```
pca = PCA(n_components=2).fit(X)
X_pca = pca.transform(X)
```

How much variance did we capture?

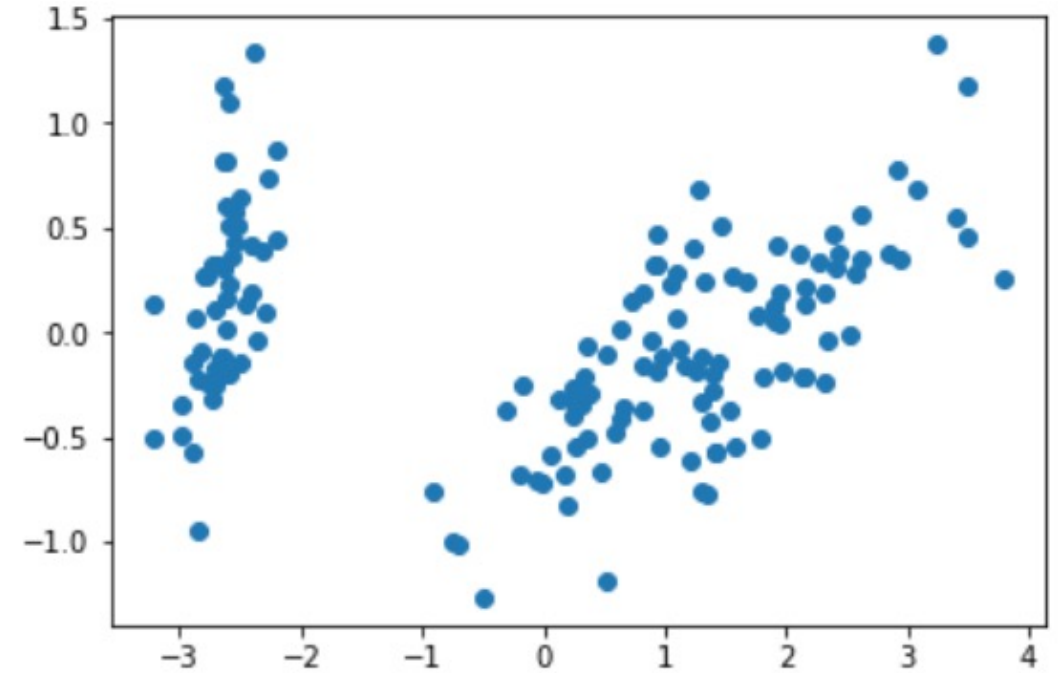
```
expvar = pca.explained_variance_ratio_
print('% Variance in 1st PC: ', expvar[0])
print('% Variance in 2nd PC: ', expvar[1])
print('Total explained variance: ', sum(expvar))
```

```
% Variance in 1st PC: 0.9246187232017271
% Variance in 2nd PC: 0.053066483117067804
Total explained variance: 0.977685206318795
```

# Example : PCA on Iris Data

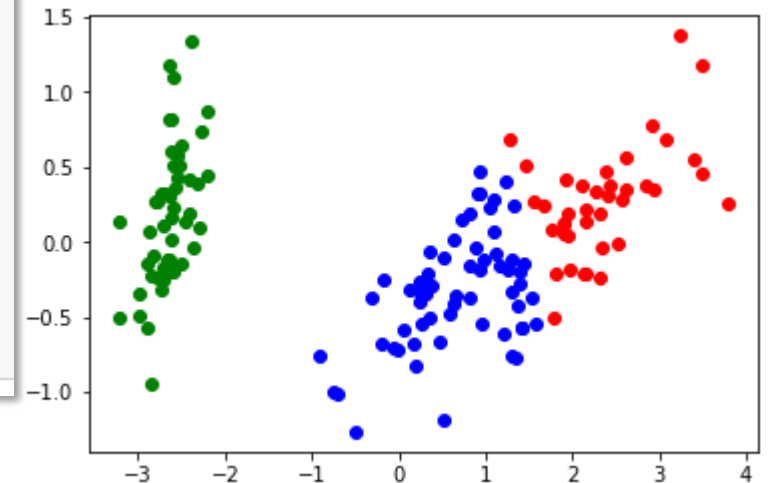
View data in 2-D subspace,

```
plt.scatter(X_pca[:,0], X_pca[:,1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



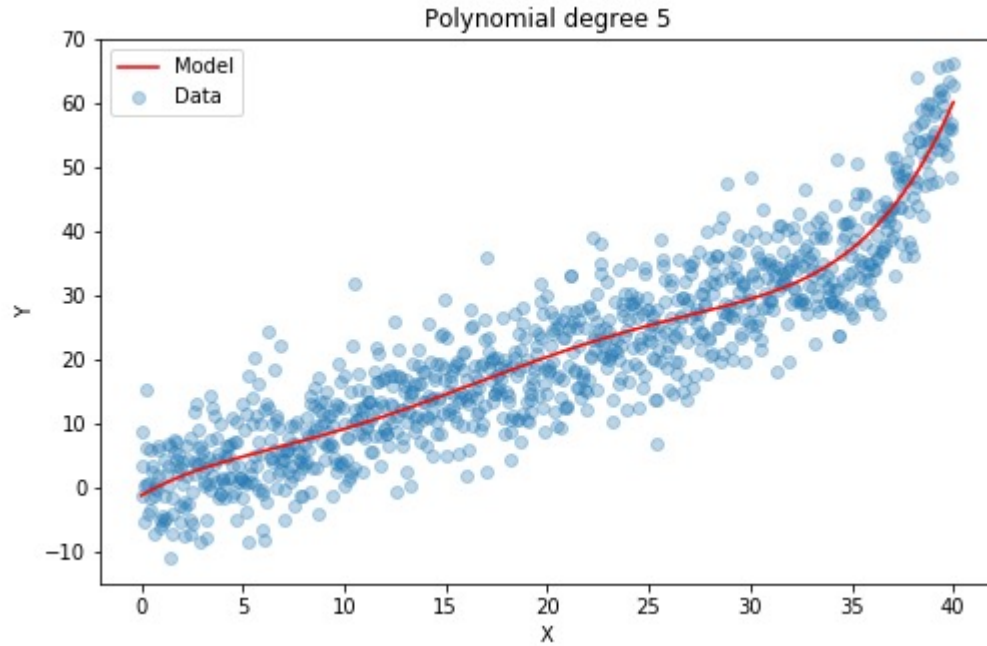
Do K-means clustering in 2-D subspace,

```
kmeans = KMeans(n_clusters=3).fit(X_pca)
labels = kmeans.labels_
fig, ax = plt.subplots()
ax.scatter(X_pca[:,0][labels == 0], X_pca[:,1][labels == 0], c='r')
ax.scatter(X_pca[:,0][labels == 1], X_pca[:,1][labels == 1], c='g')
ax.scatter(X_pca[:,0][labels == 2], X_pca[:,1][labels == 2], c='b')
plt.show()
```

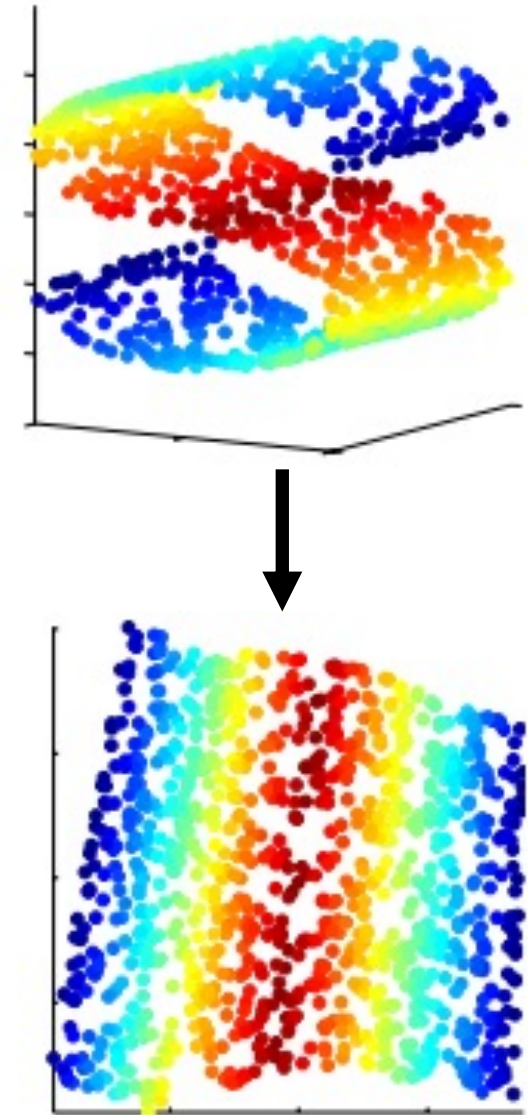


# Nonlinear Dimensionality Reduction

For general data, linear dimensionality reduction is not sufficient...

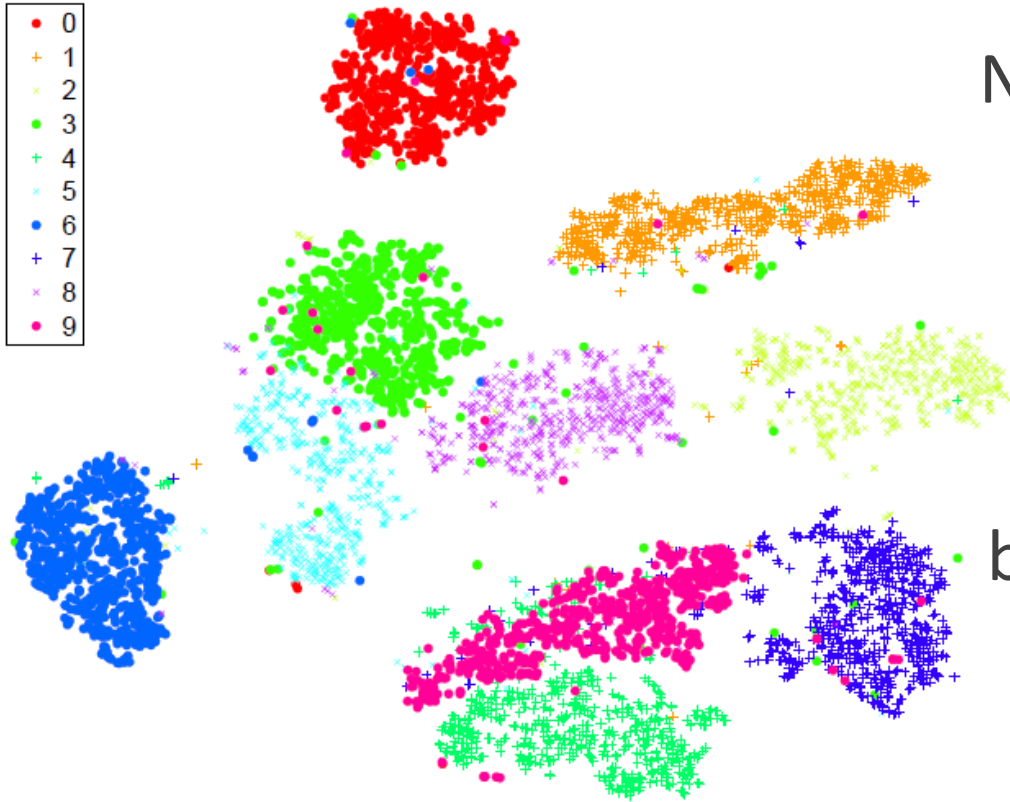


Many methods exist for nonlinear dimensionality reduction





# t-SNE



Nonlinear reduction can (potentially) amplify clustering properties

**t-Distributed Stochastic Neighbor Embedding (t-SNE)** Models similarity between data as a Student's-t distribution in high / low dimensions and optimizes reduction to preserve similarity

Visualization shows MNIST digits (recall from lecture on Neural Nets) projected to 2D and clustered

## Parameters

**n\_components** : *int, default=2*

Dimension of the embedded space.

**perplexity** : *float, default=30.0*

The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results.

**learning\_rate** : *float or 'auto', default=200.0*

The learning rate for t-SNE is usually in the range [10.0, 1000.0].

## Attributes

**embedding\_** : *array-like of shape (n\_samples, n\_components)*

Stores the embedding vectors.

# Example : t-SNE on Iris Dataset

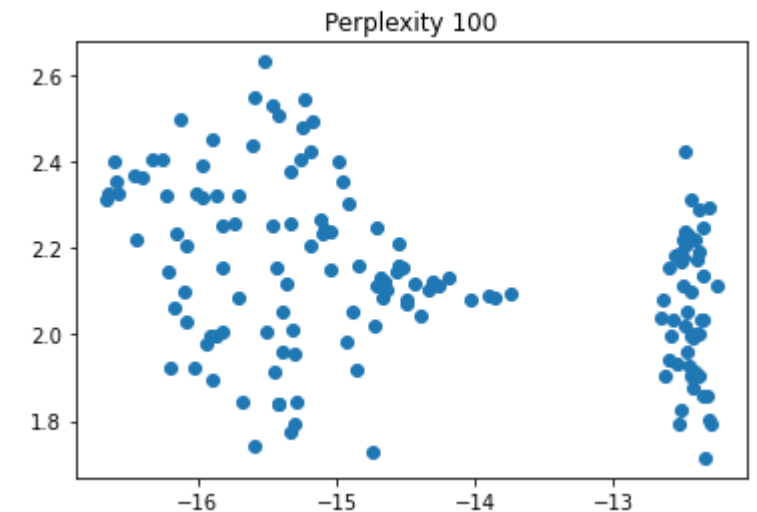
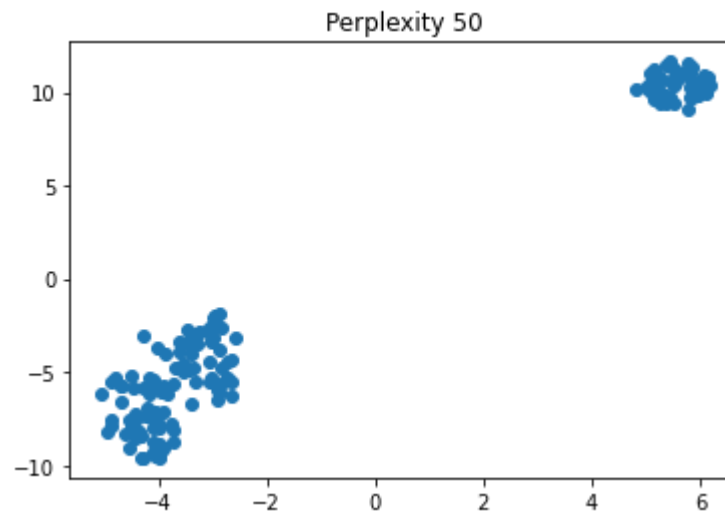
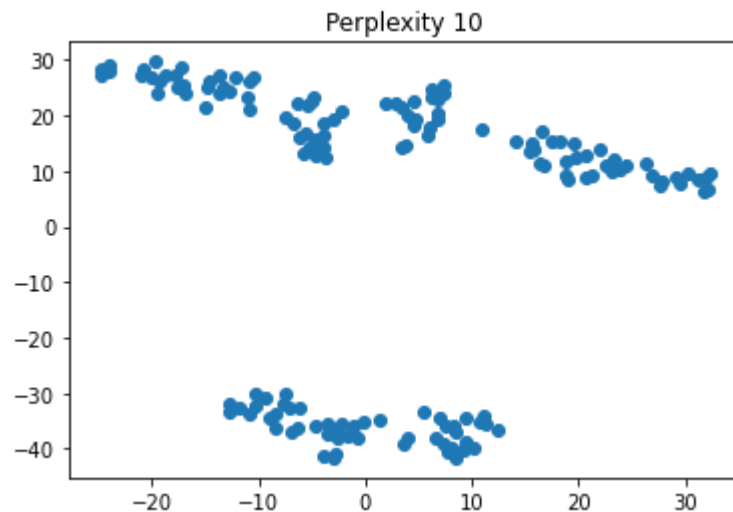
t-SNE can work surprisingly well...

```
from sklearn.manifold import TSNE
perplexity = [20, 50, 100]
for perp in perplexity:
    tsne = TSNE(n_components=2, perplexity=perp)
    X_tsne = tsne.fit_transform(X)
    fig, ax = plt.subplots()
    ax.scatter(X_tsne[:,0], X_tsne[:,1])
    ax.set_title('Perplexity %i' % perp)
plt.show()
```

**One advantage of PCA  
is that it has no parameters  
that need tuning (aside from  
number of PCs)**

**PCA is also much easier  
to interpret**

...but can be a bit fussy about parameters and unreliable

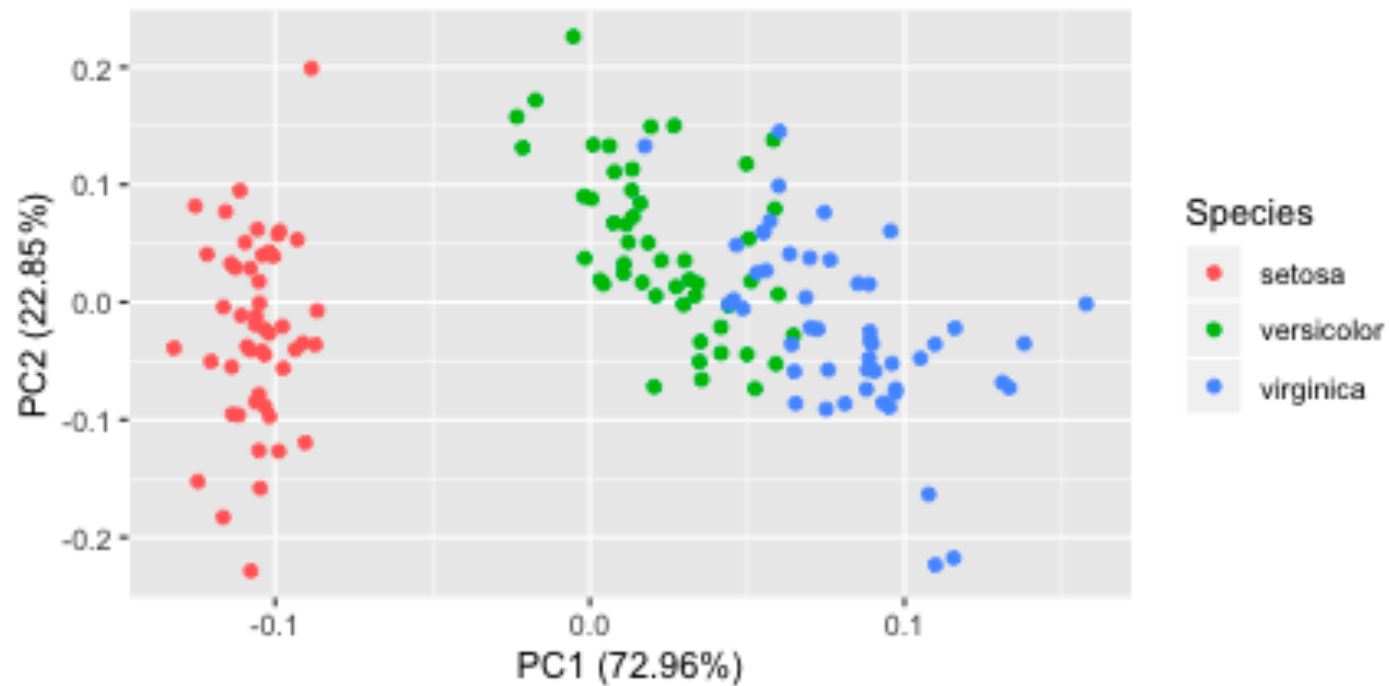


# Closing Comments

- Nonlinear methods in Scikit-Learn categorized under “manifold learning” in the *manifold* sub-package,
  - Isomap, Locally Linear Embedding, Spectral Embedding, Multidimensional scaling, and of course TSNE
- Other methods related to PCA (in *decomposition* sub-pkg):
  - Factor Analysis, Kernel PCA, Incremental PCA
- For multiple data sources, consider cross-decomposition
  - Canonical Correlation Analysis (CCA)
  - Learns same embedding for both spaces
  - Under *cross\_decomposition* sub-package

# Dimensionality reduction: motivation

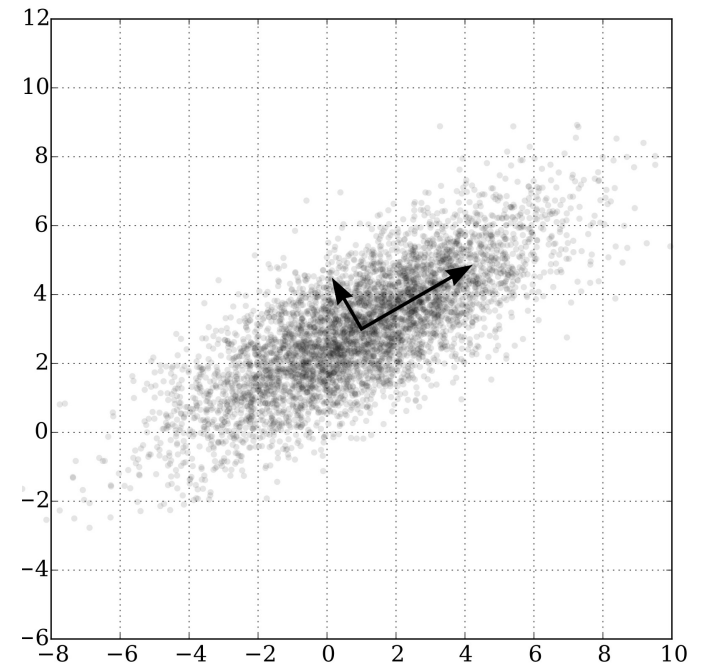
- Data compression: Identifies important components that can reconstruct data points
- Identify informative feature transformations
- Visualization & visual analytics: high-dim data -> 2d => easy to plot



Iris flower dataset (4 features)

# PCA: Introduction

- Task:
  - Given: raw feature vectors  $x_1, \dots, x_n \in \mathbb{R}^d$ , target dimension  $k$
  - Output: a  $k$ -dimensional **subspace** represented by an *orthonormal* basis  $q_1, \dots, q_k \in \mathbb{R}^d$  that the projections of datapoints with it would maximally preserve the “spread”.
- Application: dimensionality reduction
- Closely related to projections



if  $k=1$ , which basis should we choose? 70

# Principal components: usage

- Compressing the data:

- Let  $Q = \begin{pmatrix} - & q_1 & - \\ & \dots & \\ - & q_k & - \end{pmatrix} \in \mathbb{R}^{d \times k}$

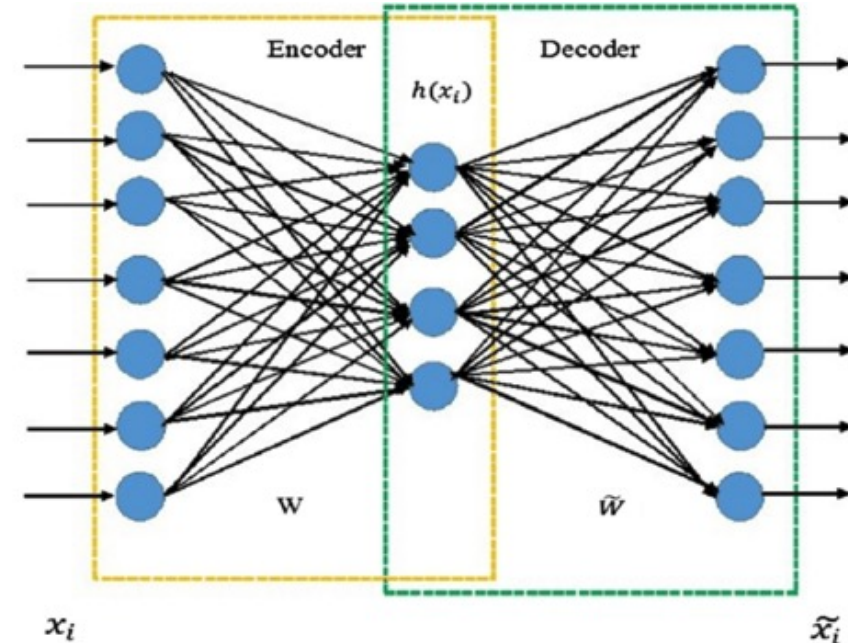
- $x_i \in \mathbb{R}^d$  mapped to 'encoding'  $z_i = Qx_i = \begin{pmatrix} q_1^\top x_i \\ \dots \\ q_k^\top x_i \end{pmatrix} \in \mathbb{R}^k$

- Reconstructing the data ('decoding')

- Given  $z_i$ , reconstruct  $x_i$  with  $\tilde{x}_i = \begin{pmatrix} | & \dots & | \\ q_1 & \dots & q_k \\ | & \dots & | \end{pmatrix} z_i = Q^\top z_i$

- Reconstruction error:  $x_i - \tilde{x}_i = x_i - Q^\top Qx_i$

- If  $k = d$ , then perfect reconstruction ( $\tilde{x}_i = x_i$ )



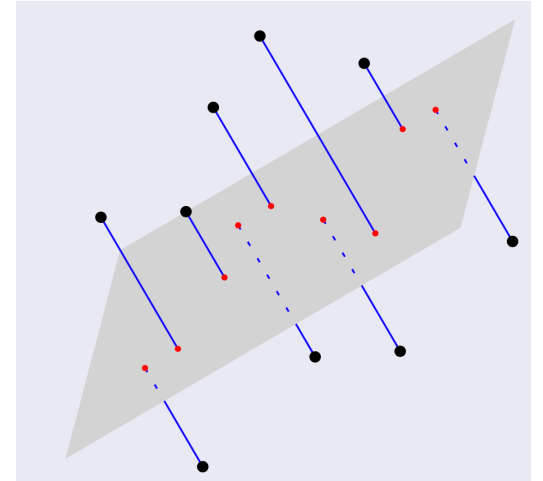
# Projection

- Why reconstructing using  $Q^T z_i$ ?

- Given orthonormal  $Q = \begin{pmatrix} - & q_1 & - \\ & \dots & \\ - & q_k & - \end{pmatrix}$ ,

$$Q^T Qx = \underbrace{\begin{pmatrix} | & \dots & | \\ q_1 & \dots & q_k \\ | & \dots & | \end{pmatrix} \cdot \begin{pmatrix} - & q_1 & - \\ & \dots & \\ - & q_k & - \end{pmatrix}}_{\text{projection matrix } \Pi = \sum_{i=1}^k q_i q_i^T} x = \sum_i (q_i^T x) q_i$$

is also the *projection* of  $x$  to subspace  $\text{span}(q_1, \dots, q_k)$



- **Projection Objective:** find a  $k$ -dimensional projection matrix  $\Pi$  s.t. the average residual squared error (reconstruction error) is minimized:

$$\frac{1}{n} \left( \sum_{i=1}^n \|x_i - \Pi x_i\|_2^2 \right)$$



# Projection when k=1

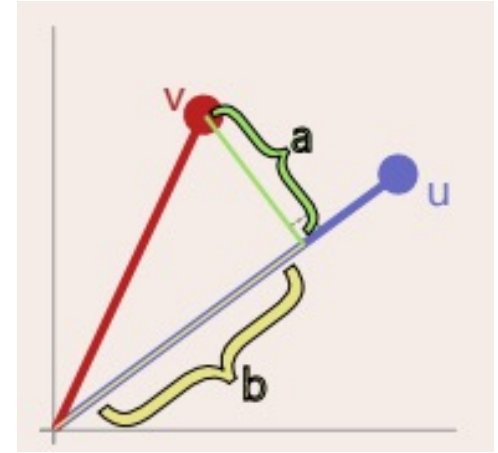
- Objective:

$$\operatorname{argmin}_{q:\|q\|=1} \frac{1}{n} \sum_{i=1}^n \|x_i - qq^T x_i\|_2^2$$

- Observation:  $qq^T x_i$  and  $x_i - qq^T x_i$  are orthogonal, and sum to  $x_i$
- Pythagorean theorem  $\Rightarrow \|x_i - qq^T x_i\|_2^2 = \|x_i\|_2^2 - \|qq^T x_i\|_2^2 = \|x_i\|_2^2 - (q^T x_i)^2$
- PCA optimization problem is thus equivalent to

$$\operatorname{argmax}_{q:\|q\|=1} \frac{1}{n} \sum_{i=1}^n (q^T x_i)^2$$

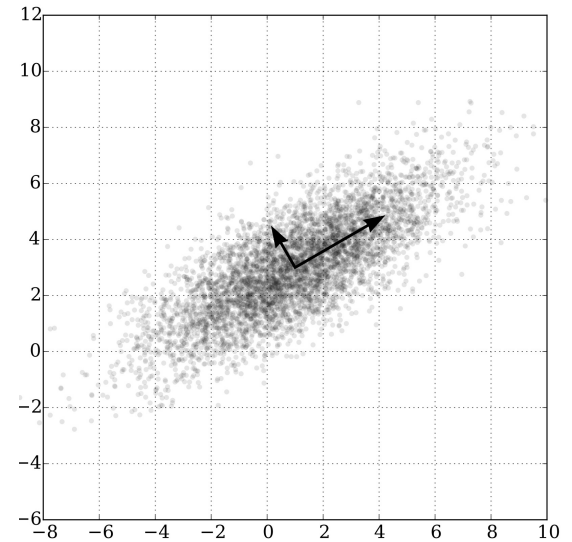
- In matrix form,  $\operatorname{argmax}_{q:\|q\|=1} q^T \left( \frac{1}{n} X^T X \right) q$



# PCA as variance maximization

$$\operatorname{argmax}_{q: \|q\|=1} \frac{1}{n} \sum_{i=1}^n (q^\top x_i)^2$$

- $\frac{1}{n} \sum_{i=1}^n (q^\top x_i)^2 = \mathbb{E}_S[(q^\top x)^2]$
- If data is centered, i.e.,  $\mathbb{E}_S[x] = 0$   
 $\Rightarrow$  the objective =  $\operatorname{var}_S[q^\top x] = \mathbb{E}_S[(q^\top x - \mathbb{E}_S[q^\top x])^2]$
- PCA on centered data  $\Leftrightarrow$  Finding direction  $q$ , such that the projected data  $\{q^\top x\}_{x \in S}$  has the maximum variance



# Eigendecomposition for real symmetric matrices

- Fact: Every **Symmetric real** matrix  $A$  is guaranteed to have eigendecomposition with real eigenvalues:

$$\begin{array}{ccccccc} \square & = & \square & \square & \square & = & \sum_{i=1}^d \lambda_i \mathbf{v}_i \mathbf{v}_i^\top \\ \mathbf{A} & & \mathbf{V} & \mathbf{\Lambda} & \mathbf{V}^\top & & \\ (d \times d) & & (d \times d) & (d \times d) & (d \times d) & & \end{array}$$

- Convention:  $\lambda_1 \geq \dots \geq \lambda_d$
- For positive semi-definite  $A$ ,  $\lambda_i \geq 0$  for all  $i$
- Recall the definition of eigenvectors:  $A\mathbf{v}_i = \lambda_i \mathbf{v}_i \forall i \in [d]$
- Here,  $V = \begin{pmatrix} | & \dots & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_d \\ | & \dots & | \end{pmatrix}$  has orthonormal columns, i.e.  $\mathbf{v}_i^\top \mathbf{v}_j = I(i = j)$

# Variational characterization of the top eigenvector

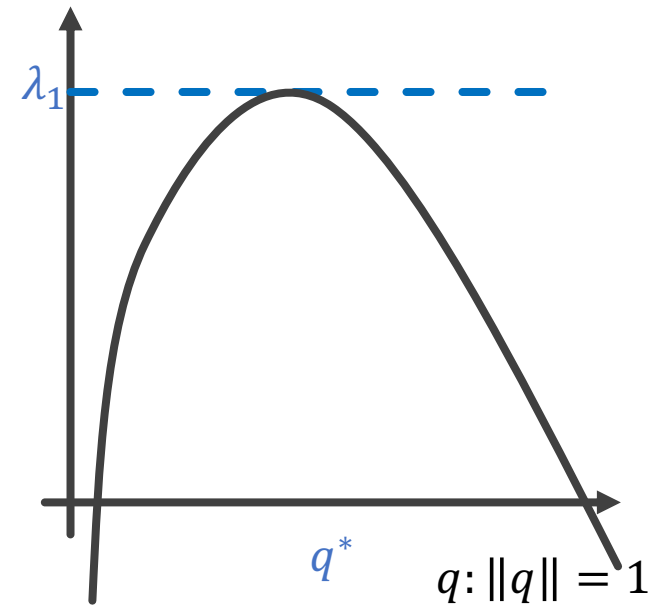
- Claim:  $\max_{q: \|q\|=1} q^\top A q$  has a maximizer  $q^* = v_1$ , with maximum objective value  $\lambda_1$

- Proof: recall  $A = \sum_{i=1}^n \lambda_i v_i v_i^\top$

- (Maximum objective upper bound): For any unit vector  $q$ ,

$$q^\top A q = \sum_{i=1}^d \lambda_i (v_i^\top q)^2 \leq \lambda_1,$$

since  $(a_i = (v_i^\top q)^2)_{i=1}^d$  satisfies  $\sum_{i=1}^d a_i = 1$  and  $a_i \geq 0$  for all  $i$

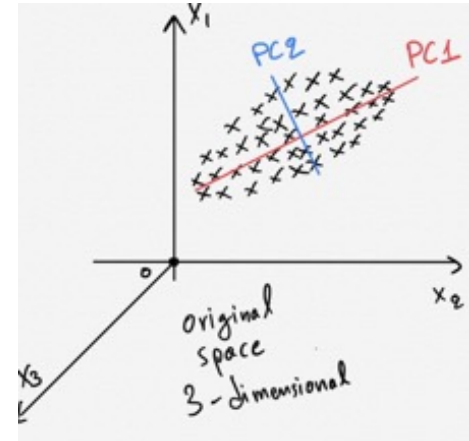


- (The upper bound is achievable)  $q^* = v_1$  satisfies that  $q^{*\top} A q^* = \lambda_1$

# PCA with $k \geq 2$

$$\operatorname{argmin}_{Q \in \mathbb{R}^{d \times k}, Q^T Q = I} \frac{1}{n} \sum_{i=1}^n \|x_i - QQ^T x_i\|_2^2$$

Equivalent to  $\operatorname{argmax}_{Q \in \mathbb{R}^{d \times k}, Q^T Q = I} \frac{1}{n} \sum_{i=1}^n \|Q^T x_i\|_2^2$ , i.e.,  $\operatorname{argmax}_{Q \in \mathbb{R}^{d \times k}, Q^T Q = I} \operatorname{tr} \left( Q^T \left( \frac{1}{n} X^T X \right) Q \right)$ ,



where for  $B \in \mathbb{R}^{d \times d}$ ,  $\operatorname{tr}(B) = \sum_{i=1}^d B_{ii}$  is the *trace* of matrix  $B$  (Important property:  $\operatorname{tr}(AB) = \operatorname{tr}(BA)$ )

- Variance maximization interpretation:

- For centered data,  $Q^T \left( \frac{1}{n} X^T X \right) Q = \frac{1}{n} \sum_{i=1}^n (Q^T x_i)(Q^T x_i)^T$  is the covariance matrix of  $\{Q^T x_i\}$ 's
- PCA chooses  $Q$  with the "largest" variance on projected data

# PCA with $k \geq 2$

$$\operatorname{argmax}_{Q \in \mathbb{R}^{d \times k}, Q^T Q = I} \operatorname{tr}(Q^T A Q)$$

- Fact: optimal  $Q$  has form  $Q^* = \begin{pmatrix} | & \cdots & | \\ v_1 & \cdots & v_k \\ | & \cdots & | \end{pmatrix}$ , where  $A$  has eigendecomposition  $A = \sum_i^d \lambda_i v_i v_i^T$

- In summary,

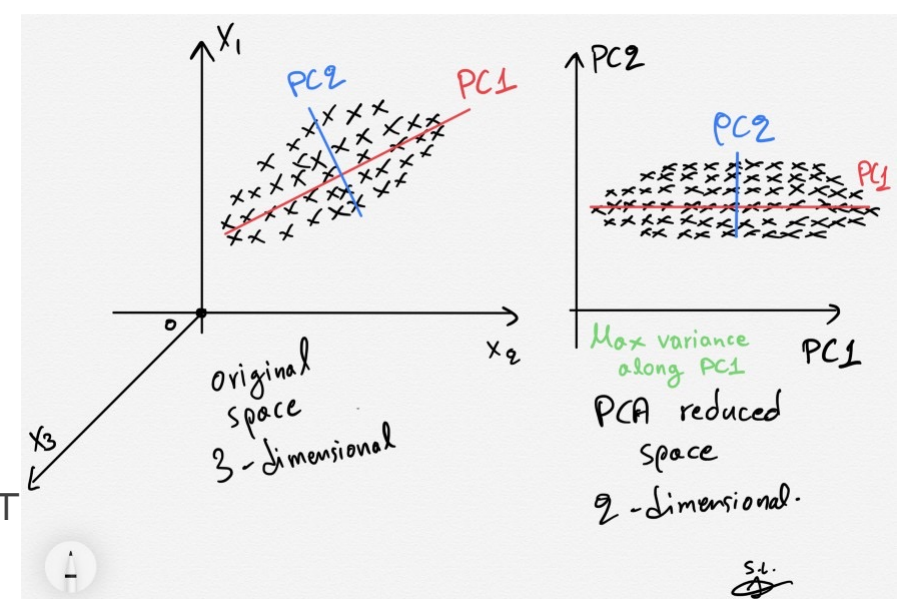
k-dimensional subspace with smallest reconstruction error

= k-dimensional subspace with the maximum total variance

= top-k eigenvectors of  $A = \frac{1}{n} X^T X$

# PCA pseudocode (with centering)

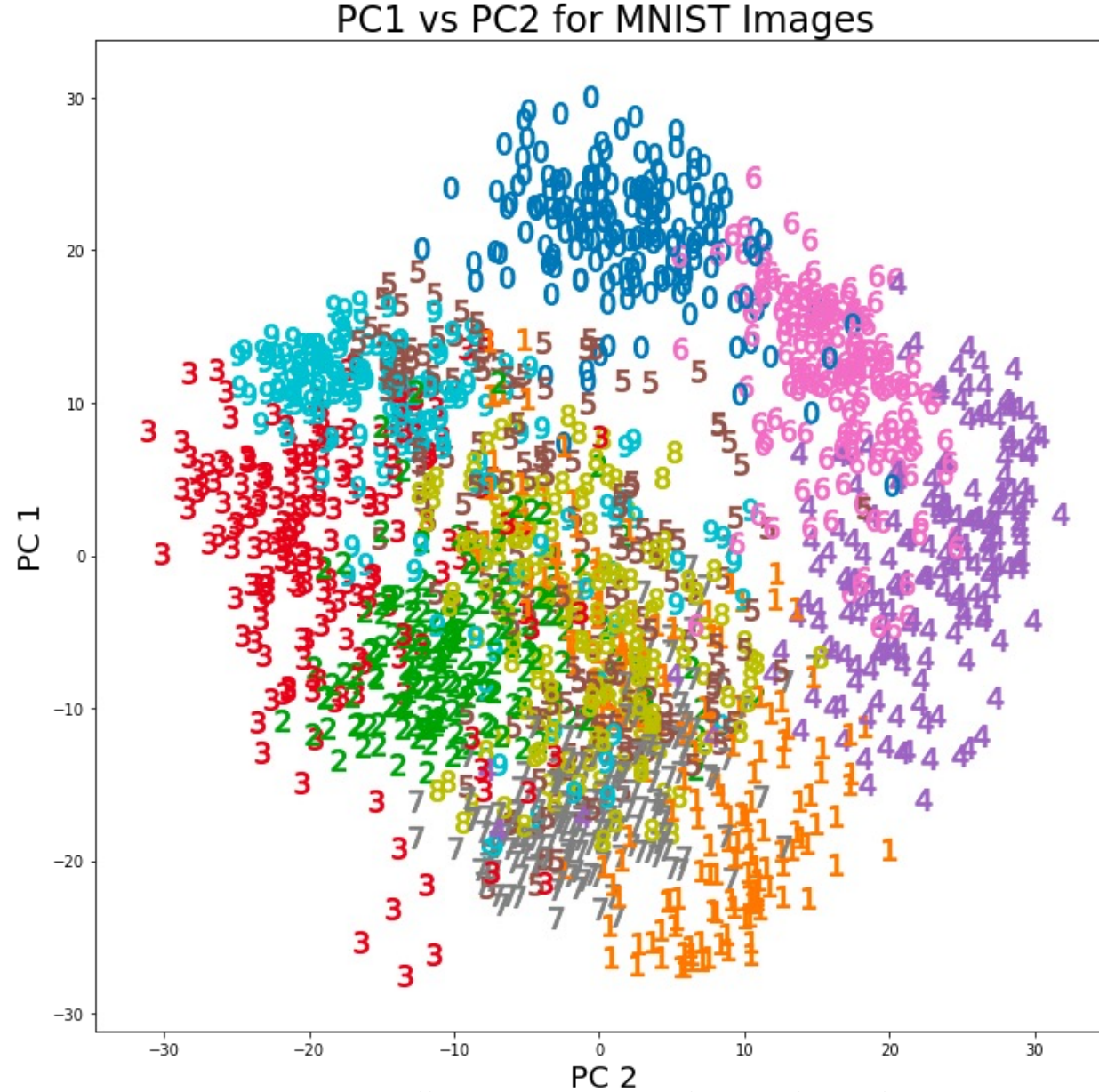
- Input: data matrix  $X \in \mathbb{R}^{n \times d}$
- Centering: Let  $\mu = \frac{1}{n} \sum_{i=1}^n x_i$ . Compute  $x'_i = x_i - \mu, \forall i \in [n]$
- Compute the top  $k$  eigenvectors  $V = [v_1, \dots, v_k]$  of  $\frac{1}{n} \sum_{i=1}^n x'_i (x'_i)^\top$
- Feature map:  $\phi(x) = (v_1^\top (x - \mu), \dots, v_k^\top (x - \mu)) \in \mathbb{R}^k$
- (thm) Decorrelating property (aka “whitening”)
  - $\frac{1}{n} \sum_{i=1}^n \phi(x_i) = 0$
  - $\frac{1}{n} \sum_{i=1}^n \phi(x_i) \phi(x_i)^\top = \text{diag}(\lambda_1, \dots, \lambda_k)$
- (optional) Reconstruction (the actual projection): apply  $\mu + V\phi(x) \in \mathbb{R}^d$ 
  - can be used as a “denoising” procedure.



(k-dimensional embedding)

$\lambda_i$  is the eigen value (paired with  $v_i$ )

# Example: MNIST dataset

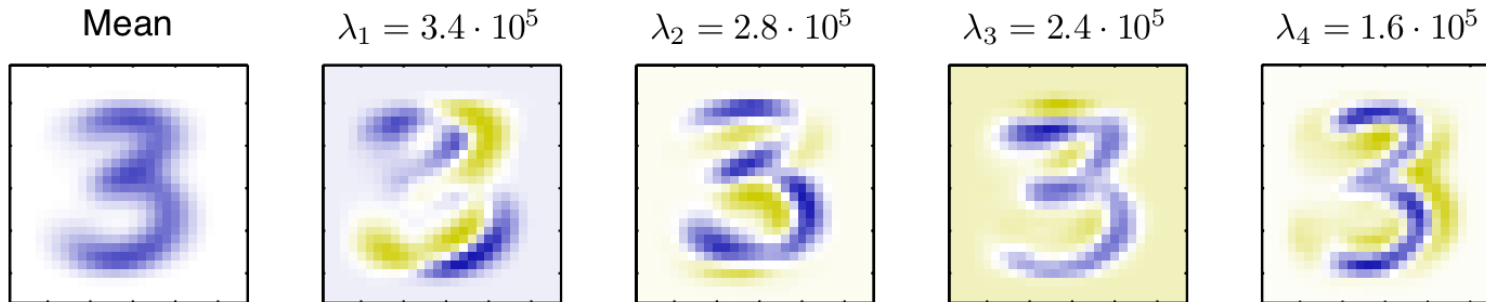




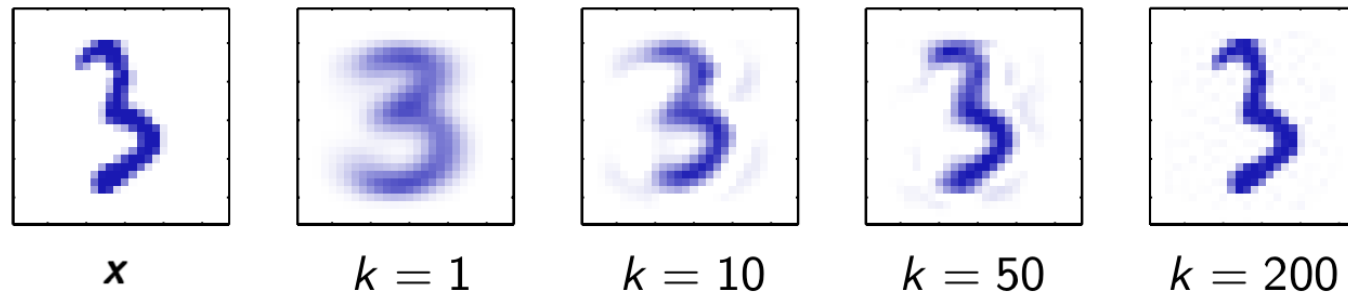
# Example: data compression

16 × 16 pixel images of handwritten 3s (as vectors in  $\mathbb{R}^{256}$ )

**Mean  $\mu$  and eigenvectors  $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$**



**Reconstructions:**



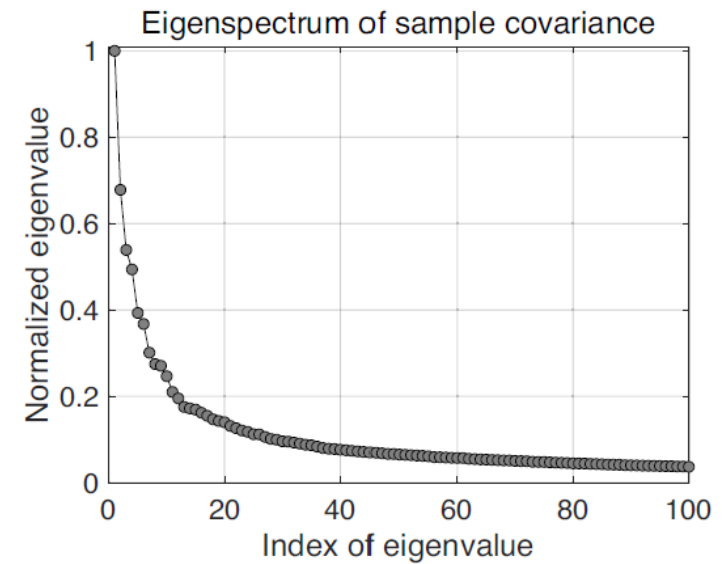
Only have to store  $k$  numbers per image,  
along with the mean  $\mu$  and  $k$  eigenvectors ( $256(k + 1)$  numbers)

# Example: eigenfaces

The Yale Face Dataset;  $n = 165$ ,  $d = 243 \times 320 = 77760$



$$\text{Eigenvalues of } A = \frac{1}{n} \sum_{i=1}^n x_i x_i^T$$

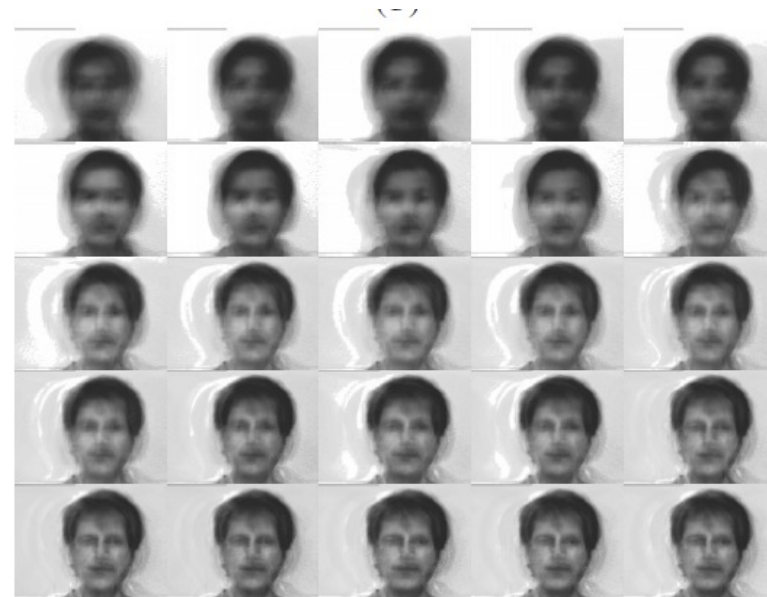


# Example: eigenfaces (cont'd)

The average face, along with the top 24 PCs (eigenfaces)

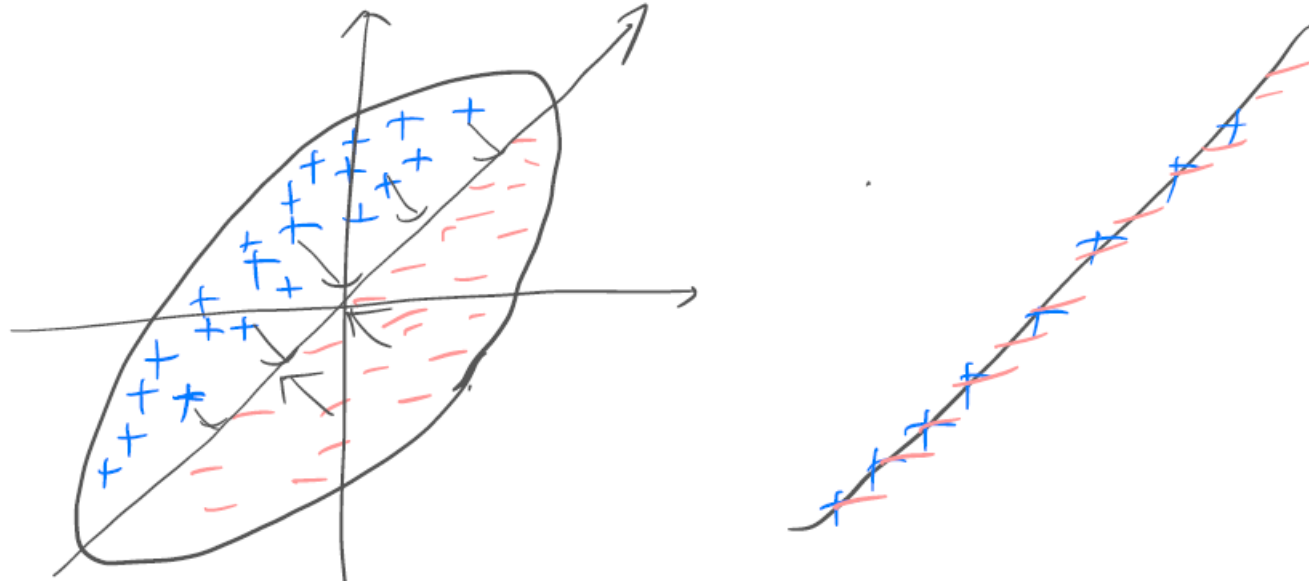


Reconstruction using the average face and the top PCs



# PCA caveat

- The direction of maximizing variance is not necessarily useful for classification!



# Next lecture (10/12)

- Probabilistic machine learning; naïve Bayes algorithm
- Assigned reading: CIML Sections 9.1-9.3