Automatic Feature Extraction using Unsupervised Learning

Reading: 19.1, 19.2, 19.3, 19.5, 19.6, 19.7, 19.9

Recollect: Feature extraction

- Recollect: Several steps for Feature extraction
- Operationally, Feature extraction = Dimensionality reduction
- Example: signal waveform (infinite dimensions) → 10-dimensional $\boldsymbol{x}$
- Two ways of doing Feature extraction ...

1. Using Manual methods: Highly domain-specific ✓
2. Using <u>Automatic</u> dimensionality reduction methods
   - Useful if no domain knowledge
   - Need lots of Training data to overcome lack of domain knowledge
   - Deep learning networks increasingly used to extract features now

input → pre-processing → feature selection → feature representation → normalization → $\boldsymbol{x}$

Automatic feature extraction (Unsupervised Learning)

- Linear Discriminant Analysis (LDA) and Kernel LDA
- Principal Component Analysis (PCA) and Kernel PCA
- Latent Semantic Analysis (LSA)
- Independent Component Analysis (ICA)

Linear methods

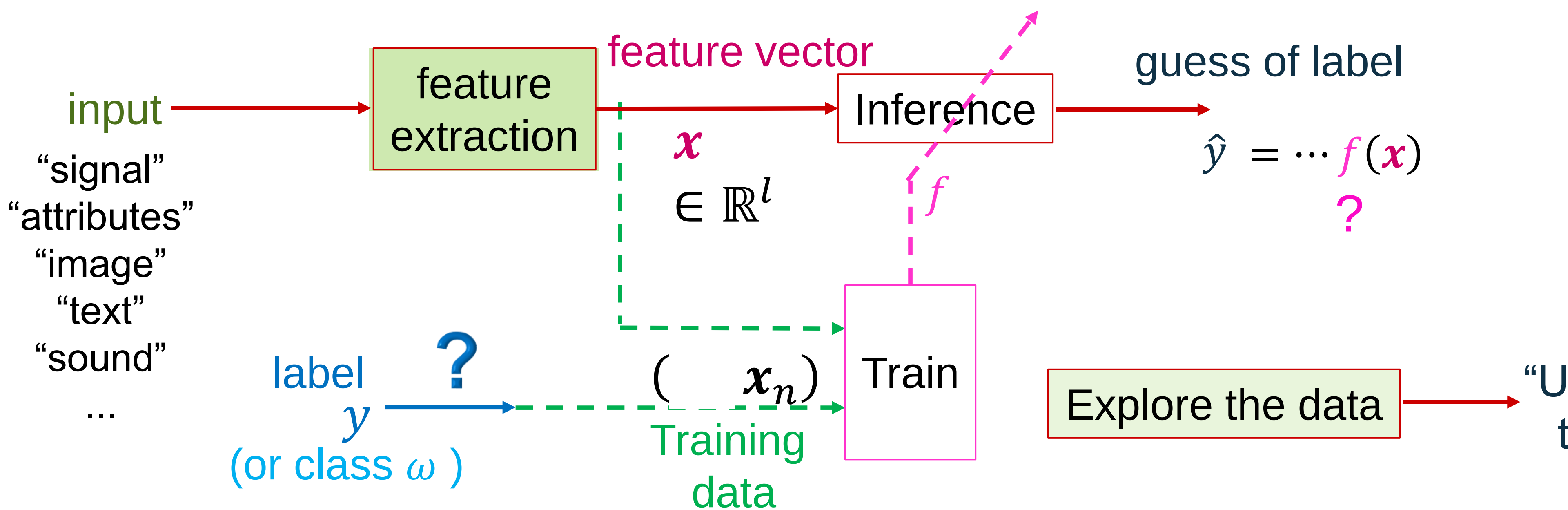
- Multidimensional scaling (MDS), t-SNE
- Autoencoders
- Embeddings
- Clustering (k-Means)
- Except LDA methods, other methods use Unsupervised learning
- Discuss only at high level, for motivation/intuition

Python `sklearn` has most of these

So, this is also a lecture on Unsupervised learning

Recollect: Supervised versus Unsupervised learning

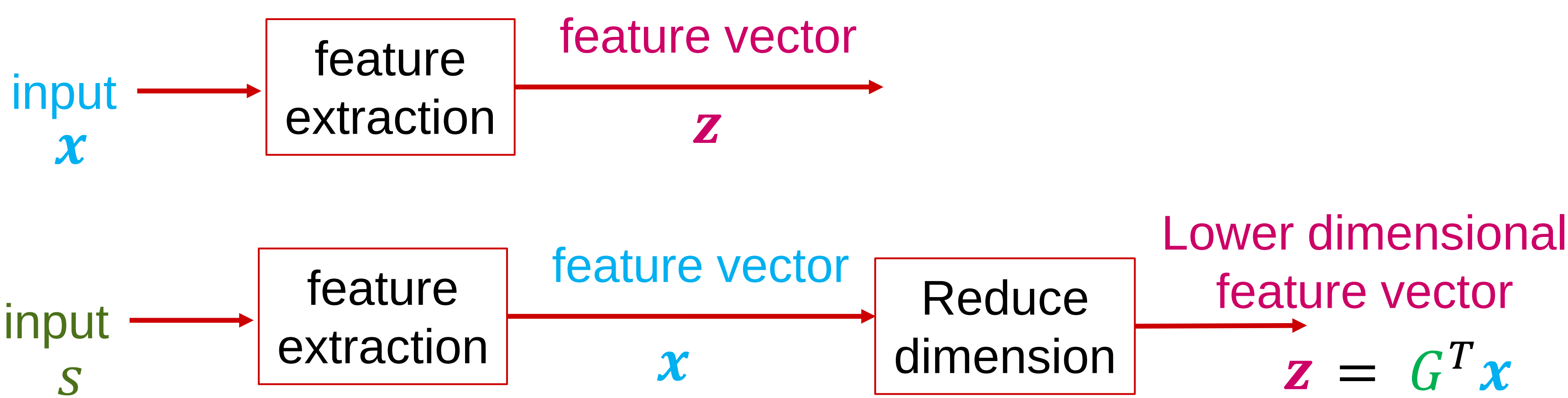
- <u>Unsupervised learning:</u> N Training points are ($\boldsymbol{x}_n$)
  feature vector
- No concept of labels here
- $\Rightarrow$ Cannot do Classification or Regression here
- But can 'Explore data' and do <u>Automatic Feature extraction</u>
- Let's study Feature extraction application of Unsupervised learning



Feature extraction: Linear methods

$$\boldsymbol{z}_n = G^T \boldsymbol{x}_n$$

- First, look at Linear feature extraction
- Map inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ to extracted features $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n$ linearly
- But linear methods require input to already be a vector
- $\Rightarrow$ Methods often used to reduce dimension of feature vector $\boldsymbol{x}$
- (Explains why we called input as $\boldsymbol{x}$ and extracted feature as $\boldsymbol{z}$)



LDA,
Kernel LDA

Recollect: Fisher Linear Discriminant classifier

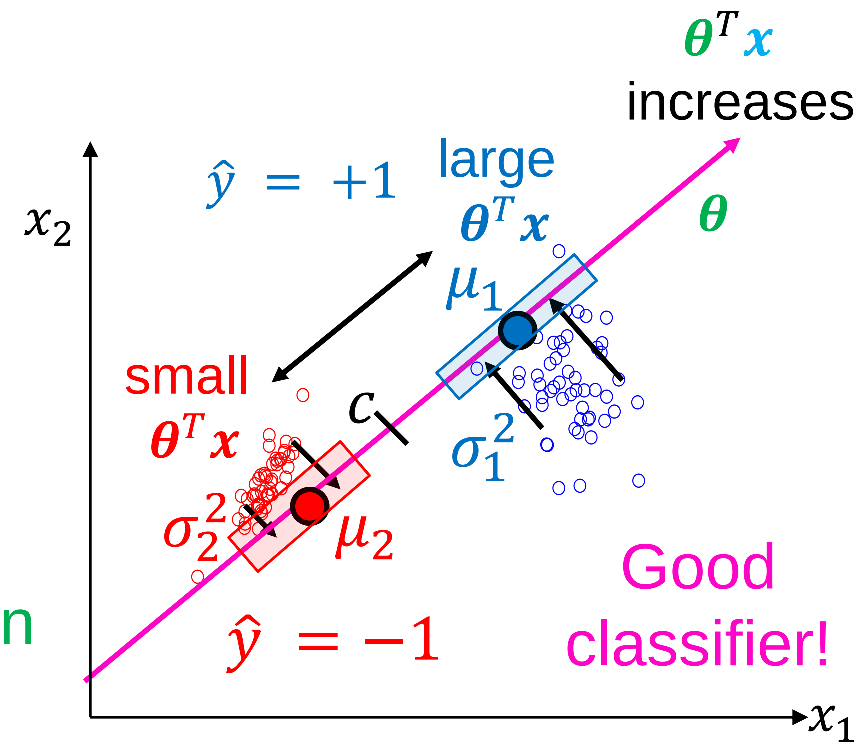
- Suppose feature vector $\boldsymbol{x}$ has high-dimension p $(\boldsymbol{x} \in \mathbb{R}^p)$
- Binary classification problem with Training data
- Earlier saw a simple binary classifier with linear $f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} - c$
- Fisher Linear Discriminant classifier: $\hat{y} = \operatorname{sgn} f(\boldsymbol{x})$
- Projection direction $\boldsymbol{\theta}$?

$$\max \ \text{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

1) Means should be well separated

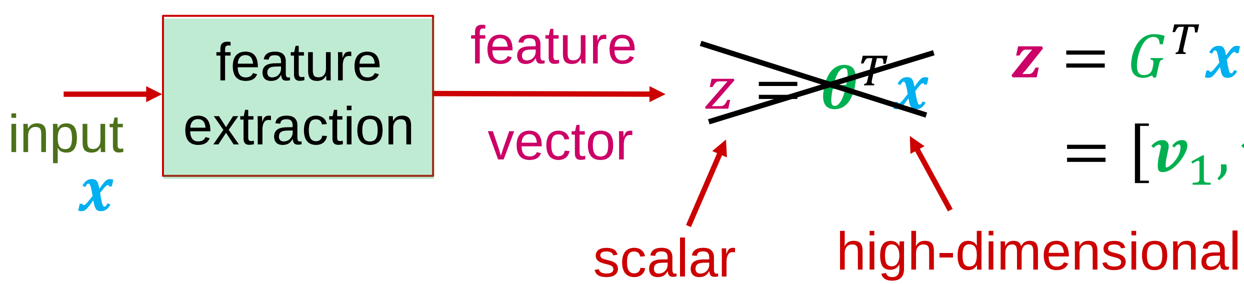
2) Point spreads should be small

Good projection direction



Motivation: LDA

- For the input vector $\boldsymbol{x}$ of high-dimension p $(\boldsymbol{x} \in \mathbb{R}^p)$ ...
- The projection of $\boldsymbol{x}$ onto a line is '<u>Dimensionality reduction</u>' method
  - Scalar (i.e., $1-D$) easily visualized. Less memory to store
  - Also, less parameters $\Rightarrow$ Less computations and less over-fitting in inference
- But projection z can be thought of as an <u>extracted</u> feature vector
- So, any Dimensionality reduction $\Rightarrow$ <u>Automatic Feature extraction</u>
- <u>Vector</u> $\boldsymbol{z} \in \mathbb{R}^k$ possible? Instead of vector $\boldsymbol{\theta}$, use $p \times k$ matrix G
- LDA can be defined now. (Must have #classes $M \geq k+1$)



$$\boldsymbol{z} = G^T \boldsymbol{x}$$
$$= [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]^T \boldsymbol{x}$$

1) Fisher's Linear Discriminant Analysis (LDA)

- Fisher's LDA (vector $\boldsymbol{z}$ case): Optimal matrix G ? Maximize FDR

Binary classes

M −classes ($M \geq 3$)

$$\frac{1}{2}\frac{(\mu_1-\mu_2)^2}{\sigma_1^2+\sigma_2^2}$$

1) Means should be well separated

← FDR →

2) Spreads should be small

$$\frac{\sum_{i=1}^{M}\|\boldsymbol{m}_i-\widehat{\boldsymbol{m}}\|^2}{\sum_{i=1}^{M}\sigma_i^2}$$

$$=\frac{(\mu_1-\hat{\mu})^2+(\mu_2-\hat{\mu})^2}{\sigma_1^2+\sigma_2^2}$$

$$=\frac{\sum_{i=1}^{2}(\mu_i-\hat{\mu})^2}{\sum_{i=1}^{2}\sigma_i^2}$$

$$\hat{\mu} \doteq \frac{1}{2}(\mu_1+\mu_2)$$

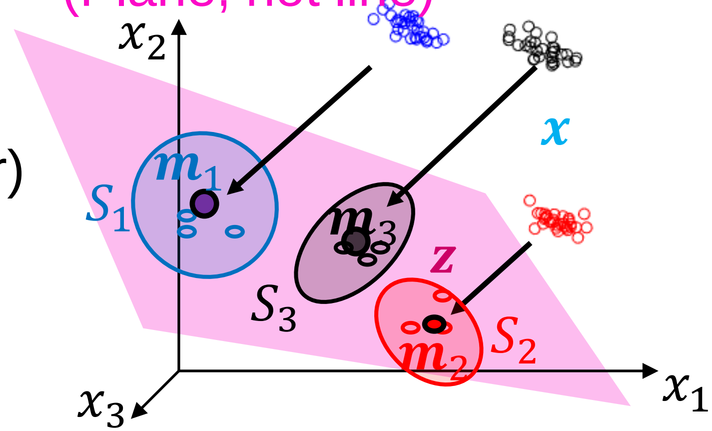
Mean of mean vectors $\widehat{\boldsymbol{m}} = \frac{1}{M}\sum_{j=1}^{M}\boldsymbol{m}_j$

$\sigma_i^2 = \mathrm{Trace}[S_i]$

Projected vector

$\boldsymbol{z} = G^T\boldsymbol{x}$

(Plane, not line)



$\boldsymbol{m}_i$ (mean vector)

S_i (covariance matrix)

Fisher's LDA

`sklearn.discriminant_analysis.LinearDiscriminantAnalysis`

- Fisher's LDA: To max FDR, using input features $\boldsymbol{x}$, calculate

1. Sample means and sample covariance matrices of each class

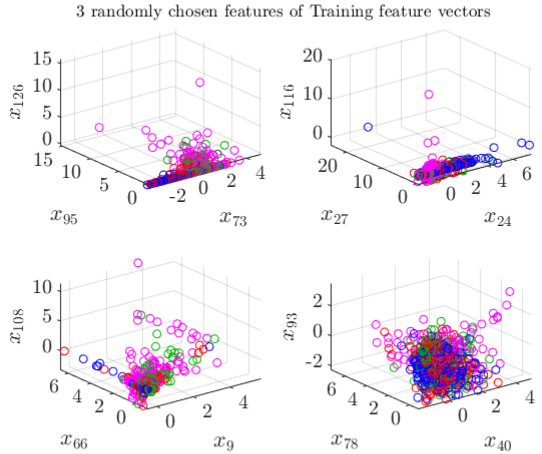
$$\hat{\boldsymbol{\mu}}_i = \frac{1}{N_i}\sum_{n=1}^{N_i} \boldsymbol{x}_n \qquad \hat{\Sigma}_i = \frac{1}{N_i - 1}\sum_{n=1}^{N_i}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_i)(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_i)^T$$

2. 'Within-class' covariance matrix $\hat{\Sigma}_w \doteq \sum_{i=1}^{M} \hat{\Sigma}_i$
3. Average $\hat{\boldsymbol{\mu}}$ of class means $\hat{\boldsymbol{\mu}}_i$: $\hat{\boldsymbol{\mu}} \doteq \frac{1}{M}\sum_{i=1}^{M} \hat{\boldsymbol{\mu}}_i$
4. 'Between-class' covariance matrix $\hat{\Sigma}_b \doteq \frac{1}{M-1}\sum_{i=1}^{M}(\hat{\boldsymbol{\mu}}_i - \hat{\boldsymbol{\mu}})(\hat{\boldsymbol{\mu}}_i - \hat{\boldsymbol{\mu}})^T$
5. Max FDR $\rightarrow$ EVD of matrix $B \doteq \left(\hat{\Sigma}_w + \lambda I\right)^{-1} \hat{\Sigma}_b$ "top-k eigen-vectors"
   Regularize
6. Largest k eigen-values $\rightarrow$ Eigen-vectors collected matrix $G = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k]$
7. Then, Fisher's LDA $\boldsymbol{z}_n = G^T \boldsymbol{x}_n$
8. Ridge-regularize if G is large matrix

$$\text{Reg\_FDR} = \frac{\text{Trace}\left[G^T\left(\widehat{\Sigma_b}\right)G\right]}{\text{Trace}\left[G^T\left(\widehat{\Sigma_w}\right)G\right] + \underline{\lambda\left|\left|G\right|\right|_F^2}}$$

Example: Fisher's LDA

- <u>Crime level data</u>: **Communities and Crime Unnormalized Set ***
  - — Crime levels in different communities



- <u>Attributes</u>: 145 attributes per community
  1. population
  2. household size
  3. % population below poverty
  4. ...

  145. Crime per 100K population

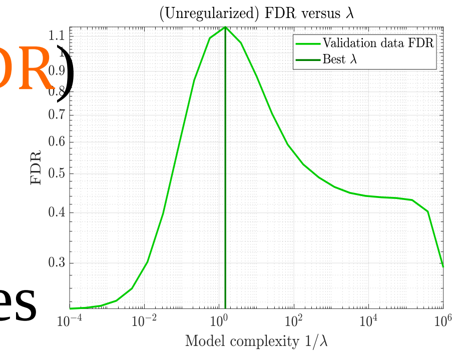
$$\omega = \begin{cases} \omega_1, & Crime < 25\ percentile \\ \omega_2, & 25 \le Crime < 50 \\ \omega_3, & 50 \le Crime < 75 \\ \omega_4, & Crime \ge 75\ percentile \end{cases}$$

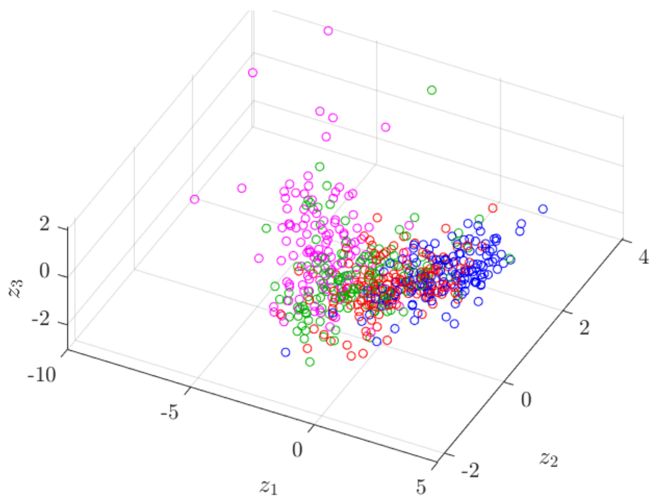
- Convert `Crime` into <u>four classes</u>
- Attributes 1-127 are features $\boldsymbol{x}$. Any 3 features not informative
- So, visualize using <u>Fisher projected vectors</u> $\boldsymbol{z}_n = G^T \boldsymbol{x}_n \in \mathbb{R}^3$

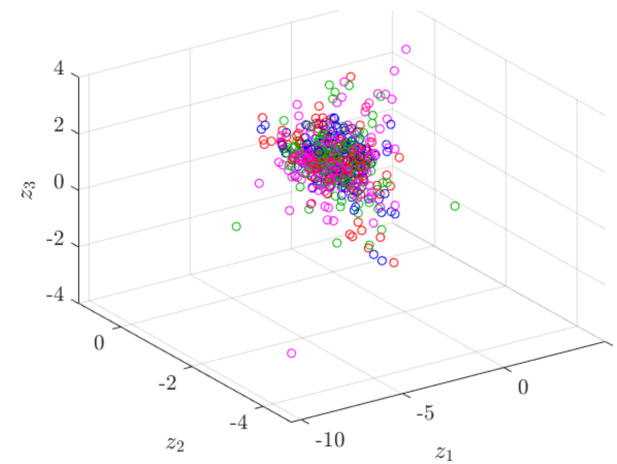
http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unnormalized

Example: Fisher's LDA

- Regularization used. So, normalize by center and scaling each feature
- Validation of penalty λ by plotting FDR (not Reg_FDR)
- Using optimal G ...
- LDA $\boldsymbol{z}_n$ of Testing data shows separation of classes
- Projection using a randomly chosen G $\rightarrow$ No separation
- $\Rightarrow$ LDA does class-aware <u>Automatic feature extraction</u>







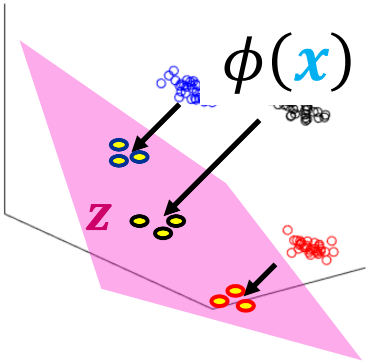
2) Kernel LDA

$$\mathbf{z}_n = G^T \phi(\boldsymbol{x}_n)$$

- New idea: First, map features $\boldsymbol{x}_n$ to high-dimensional vector $\phi(\boldsymbol{x}_n)$

  Then apply LDA to $\phi(\boldsymbol{x}_n)$

- Why? High dimensional mapping $\phi(\boldsymbol{x}_n)$ may fit data better
  - Recollect: Common in GLMs
  - e.g., $\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix} \rightarrow \boldsymbol{\theta}^T \phi(\boldsymbol{x}) = \theta_0 + \theta_1 x + \theta_2 x^2$ (Polynomial)



Kernel LDA

$$\boldsymbol{z}_n = G^T \phi(\boldsymbol{x}_n) = \tilde{G}^T \boldsymbol{\kappa}_n$$

- Problem: For very high-dimensional $\phi(\boldsymbol{x}_n)$, computing G is hard
- Kernel idea: Not hard if mapping $\phi(\cdot)$ is based on a kernel
  - Kernel? A measure of similarity. E.g., $\kappa(\boldsymbol{x}_m, \boldsymbol{x}_n) = e^{-\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2/\sigma^2}$ (RBF kernel)
  - (RKHS theory of Non-parametric inference later)
- So, Kernel LDA: LDA applied to high-dimensional $\phi(\boldsymbol{x}_n)$

1. Each $\boldsymbol{x}_n \rightarrow$ Kernel vector

   Not $\phi(\boldsymbol{x}_n)$ → $\boldsymbol{\kappa}_n = \begin{pmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_n) \end{pmatrix}$

   $\dim(\boldsymbol{\kappa}_n) = N$

   sample mean

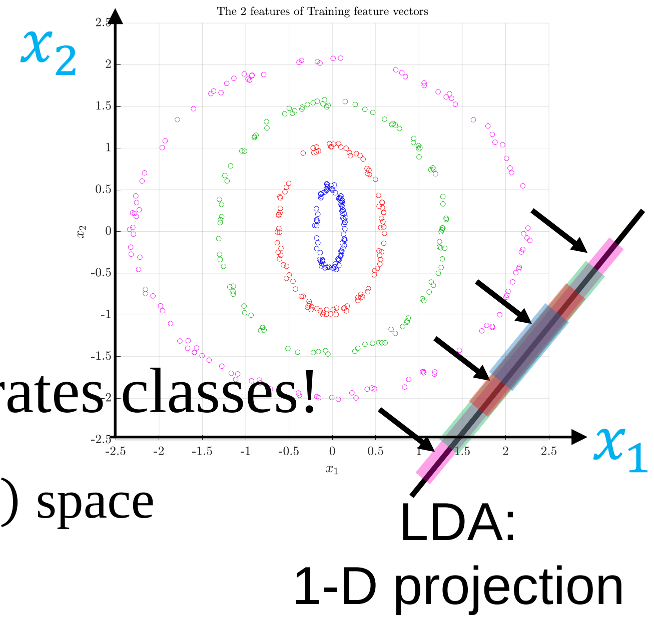
   $$\hat{\boldsymbol{\mu}}_i = \frac{1}{N_i} \sum_{n=1}^{N_i} \boldsymbol{\kappa}_n$$

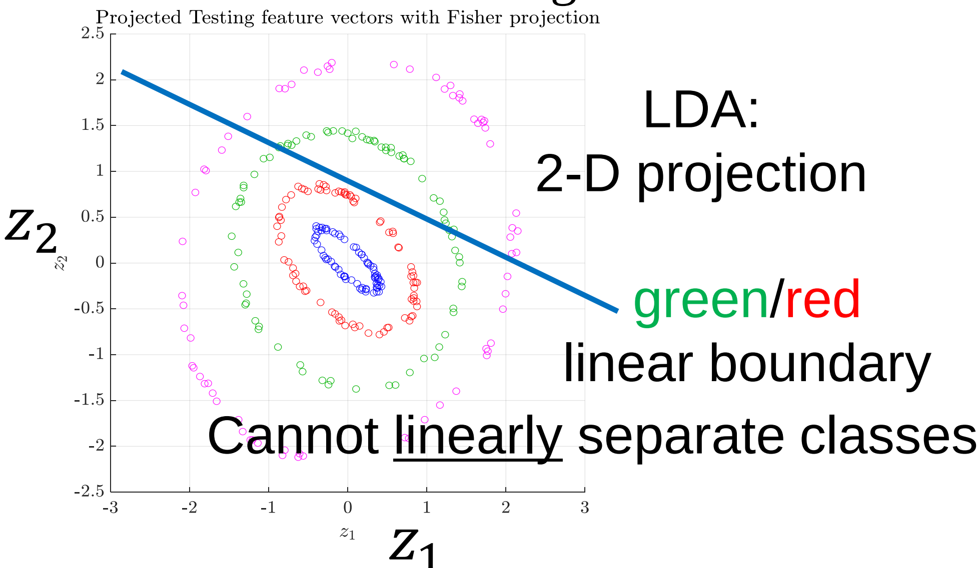
   $\vdots$

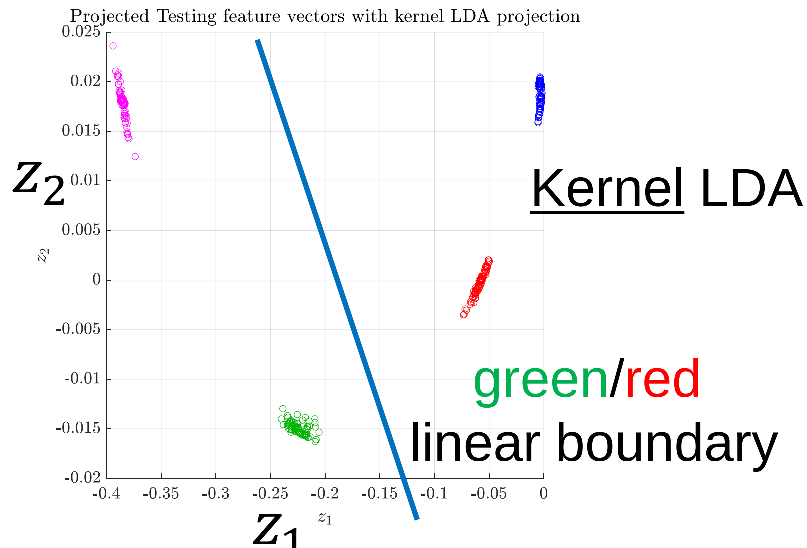
2. Do LDA calculation on $\boldsymbol{\kappa}_n$ (instead of on $\boldsymbol{x}_n$) to get $\tilde{G}$ matrix
3. Projection uses $\tilde{G}$ matrix operating on $\boldsymbol{\kappa}_n$ (instead of on $\phi(\boldsymbol{x}_n)$)

Example: Kernel LDA

- <u>Synthetic</u> data with $\boldsymbol{x} \in \mathbb{R}^2$ of 4 classes
- LDA $\rightarrow$ Cannot <u>linearly</u> separate classes
  - $\Rightarrow$ LDA features are uninteresting
- <u>Kernel</u> LDA projection into $\mathbb{R}^2$ <u>linearly</u> separates classes!
  - Since it is still <u>linear</u>, but in <u>high-dimensional</u> $\phi(\boldsymbol{x})$ space
  - These are interesting features!







PCA,
Kernel PCA,
LSA

3) Principal Component Analysis (PCA)

$\boldsymbol{z}_n = G^T \boldsymbol{x}_n$

Class sample covariance $\hat{\Sigma}_i = \dfrac{1}{N_i - 1} \displaystyle\sum_{n=1}^{N_i \,=1} \underbrace{(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_i)}_{=0} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_i)^T = 0$

- PCA is for unlabeled data. Suppose N points of unlabeled $\boldsymbol{x}_n \in \mathbb{R}^p$
- No classes to separate here. "Best" matrix G for Fisher LDA?
- Note: "No classes" $\leftrightarrow$ Every point $\boldsymbol{x}_n$ is "its own class"
- So LDA calculation reduces to …
- PCA: Uses matrix $G = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k] =$ Top-k eigenvectors of $\hat{\Sigma}$

Sample mean of all points $\hat{\boldsymbol{\mu}} \doteq \dfrac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n$

$\hat{\Sigma} \doteq \dfrac{1}{N-1} \sum_{i=1}^{N} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T$

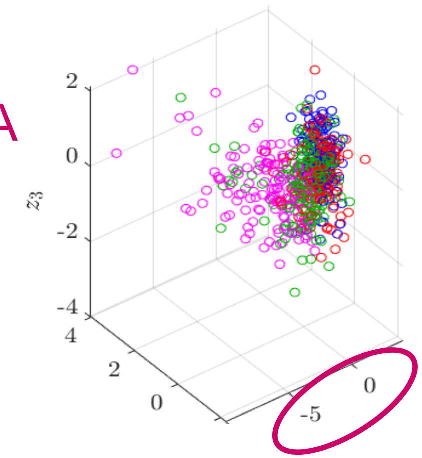
sample covariance matrix of all points

- (Alternative view: PCA maximizes randomness of projected $\boldsymbol{z}_n$)

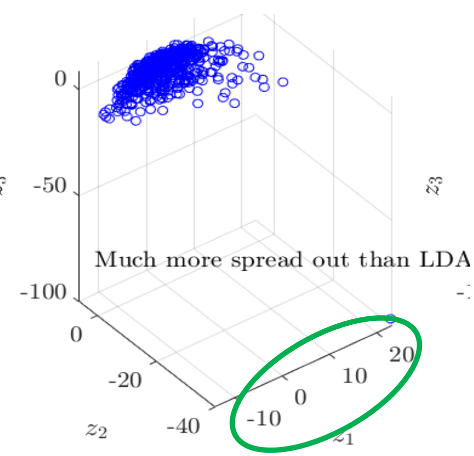
Example: PCA

- Crime level data: **Communities and Crime Unnormalized Set**
  - 127-dimensional feature vectors $\boldsymbol{x}$
  - LDA gives some class separation here
  - PCA? Ignore the classes (label)
- PCA spreads out $\boldsymbol{z}_n$ much more than LDA does
  - Because PCA maximizes sample variance of $\boldsymbol{z}_n$
- PCA does not separate classes, unlike LDA
  - (PCA is for unlabeled data)

LDA

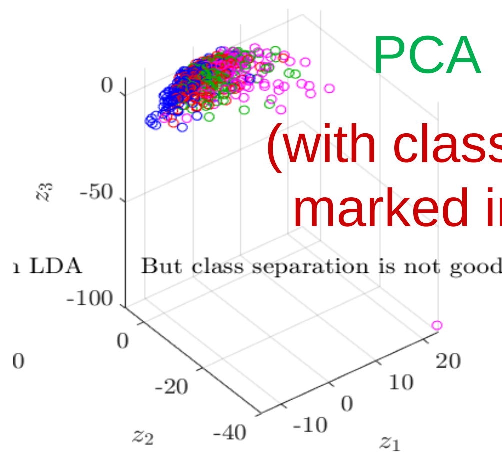


PCA



PCA

(with classes marked in)



Singular Value Decomposition (SVD)

$$\hat{\Sigma} \doteq \frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}})^T \quad = \quad V\,L\,V^T \qquad \textbf{EVD}$$

Columns $\boldsymbol{v}_i$ are eigen-vectors

Diagonal L_{ii} are eigen-values

- PCA matrix G calculated using EVD of data covariance matrix
- Covariance matrix $\hat{\Sigma} = \left(\frac{1}{\sqrt{N-1}}X_c\right)^T\left(\frac{1}{\sqrt{N-1}}X_c\right)$
  $= A^T A$
- Matrix of form such as $\hat{\Sigma}$ is called a 'Gram matrix'

Matrix of <u>centered</u> feature vectors

$$X_c = \begin{bmatrix} \boldsymbol{x}_1^T - \hat{\boldsymbol{\mu}}^T \\ \boldsymbol{x}_2^T - \hat{\boldsymbol{\mu}}^T \\ \vdots \\ \boldsymbol{x}_N^T - \hat{\boldsymbol{\mu}}^T \end{bmatrix}$$

Singular Value Decomposition (SVD)

$A = U\,D\,V^T$ **SVD**

Diagonal elements D_{ii} are 'singular-values'

Columns $\boldsymbol{v}_i$ are 'right singular vectors'

- For any Gram matrix $\hat{\Sigma} = A^T A$, EVD can also be calculated as …

```
[U,D,V] = svd(A)        U,D,Vt = numpy.linalg.svd(A)
```

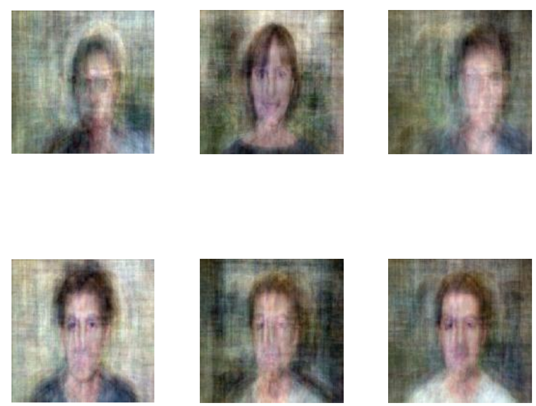
- Singular Value Decomposition (SVD) of matrix $A = \frac{1}{\sqrt{N-1}} X_c$
  - Right singular vectors of A are eigen-vectors of $\hat{\Sigma}$
  - Singular values D_{ii} of A relate to eigen-values L_{ii} of $\hat{\Sigma}$ as $D_{ii} = \sqrt{L_{ii}}$
  - So, for PCA, find largest k ~~singular~~ values D_{ii} ~~eigen~~-values L_{ii}

    PCA matrix ✓

    eigen-vectors $\boldsymbol{v}_i$
  - And collect their right singular-vectors $\boldsymbol{v}_i$ into $G = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k]$
- Advantage? Potentially large matrix $\hat{\Sigma}$ not computed anywhere

Example: PCA using SVD

- Example: **Caltech vision images data set (use 435 face images)***
  - Image dimensions = $227 \times 227 \times 3 = 154587$
  - ($\hat{\Sigma}$ is 154587×154587, while A is 435×154587) (quite low dimensional)
- So for PCA G, use SVD instead of EVD $\rightarrow$ $\boldsymbol{z}_n = G^T \boldsymbol{x}_n \in \mathbb{R}^{50}$
- To visualize $\boldsymbol{z}_n$, map it back to original space $\hat{\boldsymbol{x}}_n = G\boldsymbol{z}_n$ "Eigen-faces"
- Even 50 dimensions in $\boldsymbol{z}_n$ captures quite a bit of image structure



http://www.vision.caltech.edu/Image_Datasets/Caltech101/101_ObjectCategories.tar.gz

Example: PCA using SVD

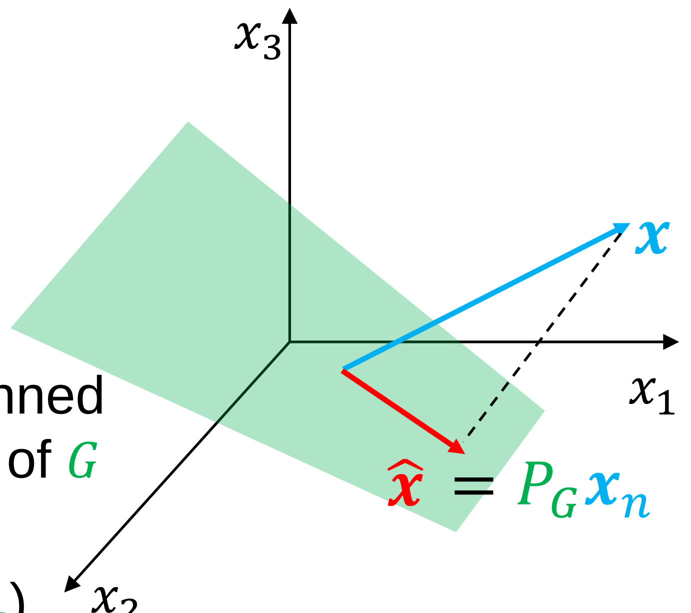
- Eigen-faces $\hat{\boldsymbol{x}}_n$: $\boldsymbol{z}_n$ mapped back to original space
- Essentially, applying a "Projection matrix" to original $\boldsymbol{x}_n$

$$\hat{\boldsymbol{x}}_n = G\boldsymbol{z}_n$$

$$= GG^T\boldsymbol{x}_n$$

$$= P_G\boldsymbol{x}_n$$

Projection matrix of orthonormal matrix G



Plane spanned by columns of G

(vectors like $\alpha_1\boldsymbol{g}_1 + \cdots + \alpha_k\boldsymbol{g}_k$)

4) Kernel PCA

$$\boldsymbol{z}_n = G^T \phi(\boldsymbol{x}_n) = \tilde{G}^T \boldsymbol{\kappa}_n$$

- Same motivation as Kernel LDA $\rightarrow$ PCA using transformed $\phi(\boldsymbol{x}_n)$
- As earlier, computations difficult, unless $\phi(\cdot)$ is based on a kernel

Example (RBF kernel)

`sklearn.decomposition.KernelPCA`

$$e^{-\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2/\sigma^2}$$

- Kernel PCA: To maximize randomness in $\boldsymbol{z}_n$,

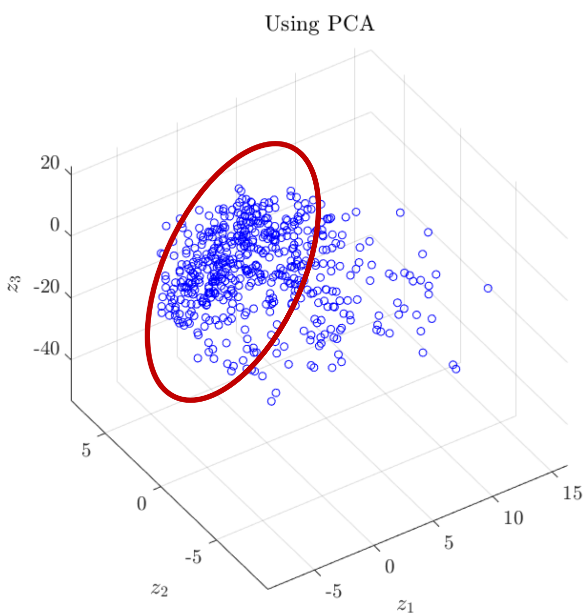
1. Each $\boldsymbol{x}_n$ $\rightarrow$ Kernel vector

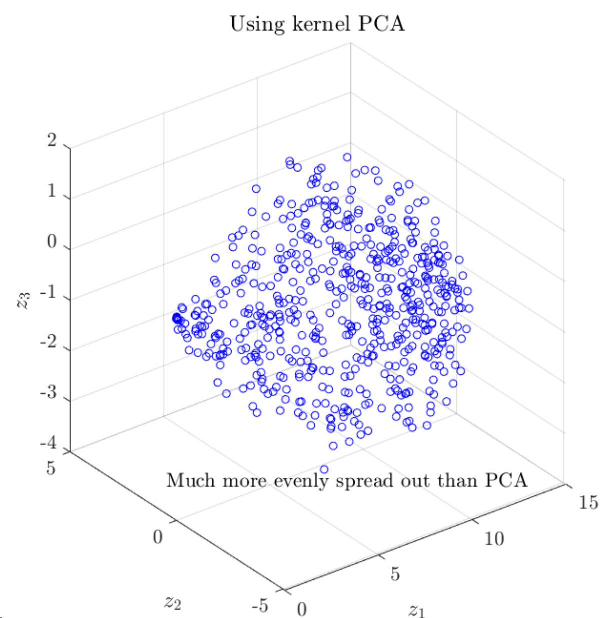
$$\boldsymbol{\kappa}_n = \begin{pmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_n) \end{pmatrix}$$

2. Do PCA calculation on $\boldsymbol{\kappa}_n$ (instead of on $\boldsymbol{x}_n$) to get $\tilde{G}$ matrix
3. Projection $\boldsymbol{z}_n$ uses $\tilde{G}$ matrix operating on $\boldsymbol{\kappa}_n$ (instead of on $\phi(\boldsymbol{x}_n)$)

Example: Kernel PCA

- <u>Crime level data</u>: **Communities and Crime Unnormalized Set**
- Map Input vector $\boldsymbol{x}_n \in \mathbb{R}^{127}$ to feature vector $\boldsymbol{z}_n \in \mathbb{R}^3$
- PCA does spread out points (it is variance maximizing)
  - But notice large cluster of points close together
- Kernel PCA spreads out points much more evenly





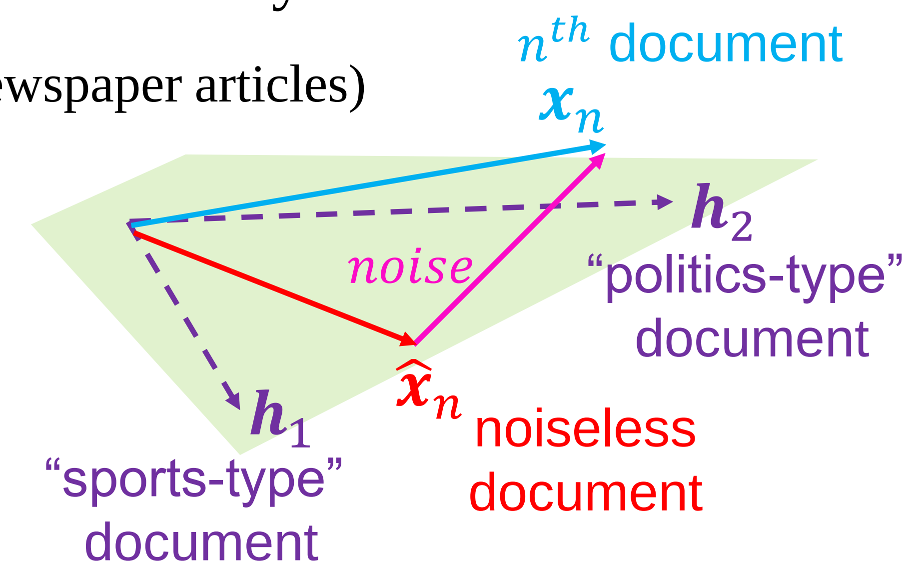
5) Latent Semantic Analysis (LSA)

- Example: Application of LSA to Text Analysis
  - Dataset of N text documents (e.g., newspaper articles)
  - Each document → 'Bag of words' $\boldsymbol{x}$

$$\boldsymbol{x} = \begin{bmatrix} 3 \\ \vdots \\ 0 \\ 2 \end{bmatrix}$$

← 'prize' occurs 3 times

← 'party' is absent



- Latent factor Model: Assume each document combines k 'prototypes'

$$\boldsymbol{x}_n = \hat{\boldsymbol{x}}_n + \textit{Gaussian noise} \qquad \text{where}$$

$$\hat{\boldsymbol{x}}_n = \sum_{i=1}^{k} \boldsymbol{h}_i \, c_{i,n}$$

$c_{i,n}$: i^{th} coefficient of n^{th} document $\boldsymbol{x}_n$

$\boldsymbol{h}_i$: i^{th} latent factor vector ("prototype document")

Latent Semantic Analysis (LSA)

$$\boldsymbol{x}_n = \hat{\boldsymbol{x}}_n + \mathit{Gaussian\ noise} \quad \text{where}$$

$$\hat{\boldsymbol{x}}_n = \sum_{i=1}^{k} \boldsymbol{h}_i \, c_{i,n}$$

i^{th} coefficient of n^{th} document $\boldsymbol{x}_n$

i^{th} latent factor vector

- Orthonormal Latent factors $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_k$ <u>common</u> to all documents
- But, coefficient vector $\boldsymbol{c}_n$ makes document $\boldsymbol{x}_n$ unique
- <u>LSA problem</u>: Using only Training data $\boldsymbol{x}_1 \ldots, \boldsymbol{x}_N$ (documents)

  Find MLE of all factors $(\boldsymbol{h}_i)$ and coefficients $(\boldsymbol{c}_n)$

$$\widehat{\boldsymbol{h}_i}, \widehat{\boldsymbol{c}_n} = \underset{\boldsymbol{h}_i, \boldsymbol{c}_n}{\arg\min} \underbrace{\sum_{n=1}^{N} \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|^2}$$

<u>Quadratic</u> NLL (*Gaussian noise*)

coefficient vector $\boldsymbol{c}_n$ of $\boldsymbol{x}_n$ $\doteq \begin{bmatrix} c_{1,n} \\ c_{2,n} \\ \vdots \\ c_{k,n} \end{bmatrix}$

Latent Semantic Analysis (LSA)

$$\checkmark \quad \widehat{\boldsymbol{h}_i}, \widehat{\boldsymbol{c}_n} = \arg\min_{\boldsymbol{h}_i, \boldsymbol{c}_n} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \widehat{\boldsymbol{x}}_n\|^2$$

- Closed form solution is possible here. Called ...
- Latent Semantic Analysis (LSA):
  - "Document-Term" feature matrix $X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} \text{document 1 bag-of-words} \\ \text{document 2 bag-of-words} \\ \vdots \\ \text{document } N \text{ bag-of-words} \end{bmatrix}$

    $= U\, D\, V^T$ (SVD)
  - PCA matrix using SVD method $G = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k]$ (top$-k$ right-singular vectors)
  - Projection of $\boldsymbol{x}_n \rightarrow \boldsymbol{z}_n = G^T \boldsymbol{x}_n$ (feature vectors)
  - Then, a) MLE of coefficient vector $\widehat{\boldsymbol{c}_n} = \boldsymbol{z}_n$
  - Then, b) MLE of latent factor $\widehat{\boldsymbol{h}_i} = \boldsymbol{v}_i$ (right-singular vector)

Example: LSA for Text Documents

- Example: **BBC news articles dataset***
  - 1490 BBC news articles in 5 categories
- Document-Term matrix using tf-idf
- Compute LSA G, then project $z_n = G^T x_n \in \mathbb{R}^k$
  - (For Text, generally choose large k, say 50)
  - Here, choose $k = 3$ for easy visualization

	ArticleId	Text	Category
0	1833	worldcom ex-boss launches defence lawyers defe...	business
1	154	german business confidence slides german busin...	business
2	1101	bbc poll indicates economic gloom citizens in ...	business
3	1976	lifestyle governs mobile choice faster bett...	tech
4	917	enron bosses in $168m payout eighteen former e...	business
...	...	...	...
1485	857	double eviction from big brother model caprice...	entertainment
1486	325	dj double act revamp chart show dj duo jk and ...	entertainment
1487	1590	weak dollar hits reuters revenues at media gro...	business
1488	1587	apple ipod family expands market apple has exp...	tech
1489	538	santy worm makes unwelcome visit thousands of ...	tech

3078 dictionary terms

	000	10	100	100m	11	12	120	13	14	15	...	year	years	yen	yes	yet	york	young	younger	yukos	zealand
0	0.000000	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.000000	0.033765	0.0	0.0	0.000000	0.057611	0.000000	0.000000	0.0	0.0
1	0.000000	0.041944	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.025758	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
2	0.028581	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.042893	0.044012	0.080704	...	0.000000	0.025569	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
3	0.023337	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.071872	0.000000	...	0.000000	0.020877	0.0	0.0	0.000000	0.000000	0.000000	0.094044	0.0	0.0
4	0.000000	0.038268	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.000000	0.032200	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1485	0.056301	0.000000	0.0	0.0	0.00000	0.071659	0.0	0.000000	0.000000	0.000000	...	0.036759	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1486	0.000000	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.016108	0.022071	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1487	0.000000	0.049054	0.0	0.0	0.12762	0.058725	0.0	0.000000	0.000000	0.000000	...	0.210869	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1488	0.000000	0.053066	0.0	0.0	0.00000	0.031764	0.0	0.000000	0.000000	0.000000	...	0.016294	0.000000	0.0	0.0	0.032494	0.000000	0.036562	0.000000	0.0	0.0
1489	0.035532	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0

1490 documents

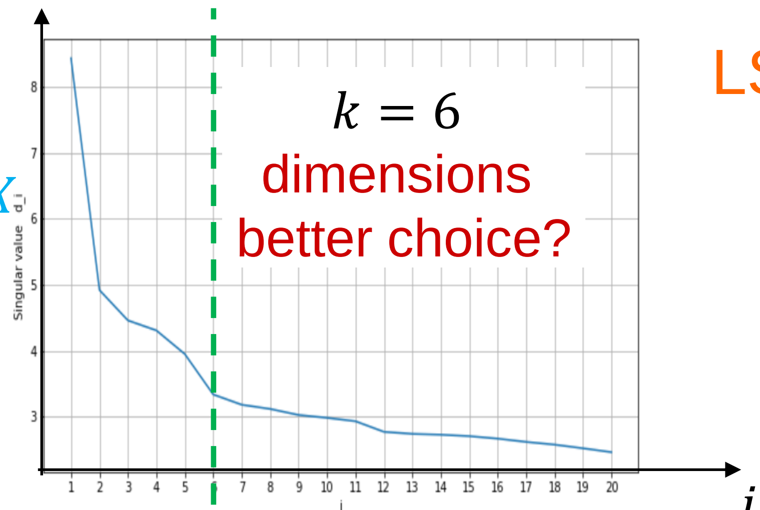
Document-Term matrix X

https://www.kaggle.com/c/learn-ai-bbc

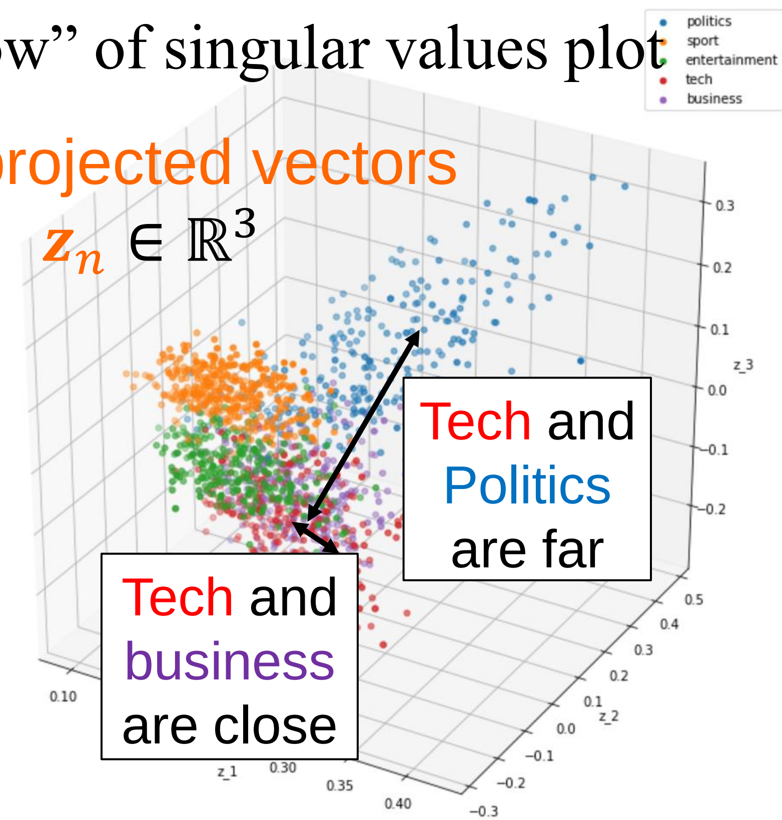
Example: LSA for Text Documents

- Note: LSA did not use labels. Yet, documents get clustered nicely!
- $\Rightarrow$ LSA "discovered" underlying 'prototype documents'
- PCA or LSA $\rightarrow$ k selected using "elbow" of singular values plot

Singular value D_{ii} of matrix X



LSA-projected vectors $z_n \in \mathbb{R}^3$



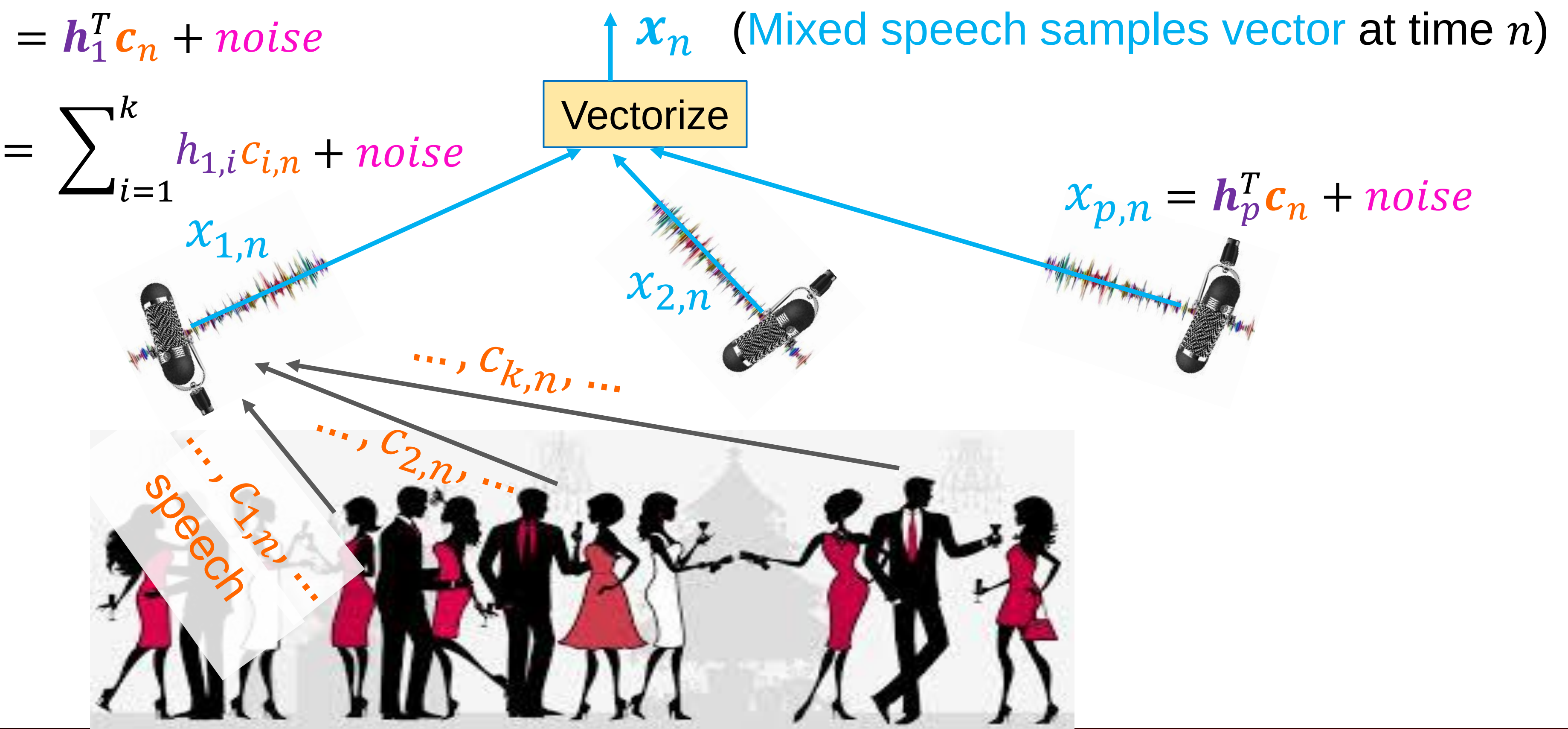
	000	10	100	100m	11	12	120	13	14	15	...	year	years	yen	yes	yet	york	young	younger	yukos	zealand
0	0.000000	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.000000	0.033765	0.0	0.0	0.000000	0.057611	0.000000	0.000000	0.0	0.0
1	0.000000	0.041944	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.025758	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
2	0.028581	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.042893	0.044012	0.080704	...	0.000000	0.025569	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
3	0.023337	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.071872	0.000000	...	0.000000	0.020877	0.0	0.0	0.000000	0.000000	0.000000	0.094044	0.0	0.0
4	0.000000	0.038268	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.000000	0.032200	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1485	0.056301	0.000000	0.0	0.0	0.00000	0.071659	0.0	0.000000	0.000000	0.000000	...	0.036759	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1486	0.000000	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.016108	0.022071	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1487	0.000000	0.049054	0.0	0.0	0.12762	0.058725	0.0	0.000000	0.000000	0.000000	...	0.210869	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1488	0.000000	0.053066	0.0	0.0	0.00000	0.031764	0.0	0.000000	0.000000	0.000000	...	0.016294	0.000000	0.0	0.0	0.032494	0.000000	0.036562	0.000000	0.0	0.0
1489	0.035532	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0

Document-Term matrix X

Independent Component Analysis (ICA)

ICA motivation: Cocktail party problem

- <u>Cocktail party problem</u> : Illustrates ICA
- k independent audio signals $c_{i,n}$ $\rightarrow$ Assume at least k microphones
- <u>Goal</u>: From mixed samples vector $\boldsymbol{x}_n$, <u>estimate</u> separate speeches $\boldsymbol{c}_n$



ICA motivation: Cocktail party problem

$$x_{i,n} = \boldsymbol{h}_i^T \boldsymbol{c}_n + noise, \qquad i = 1,2,\dots,p$$

$p \geq k$ microphones

vectorize

$$H = \begin{bmatrix} \boldsymbol{h}_1^T \\ \vdots \\ \boldsymbol{h}_p^T \end{bmatrix}$$ mixing matrix

$$\boldsymbol{x}_n = H\boldsymbol{c}_n + Gaussian\ noise$$

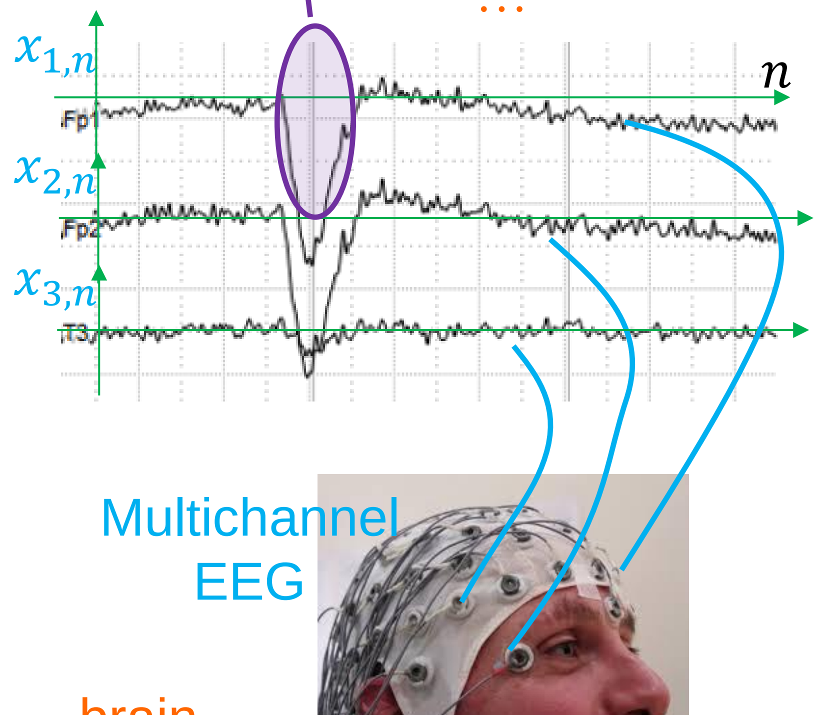
- This is usual Normal Discriminative model (with vector label $\boldsymbol{c}$)
- $\Rightarrow$ Solved by Linear Statistical Regression if mixing matrix H known
- But mixing matrix H is unknown, so use ICA

Practical application: EEG signal separation

- Each EEG signal $x_{i,n}$ is linear mixture of several signals $c_{m,n}$
- Goal: Isolate brain waves from other artifacts
- (i.e., Pre-processing step in Feature extraction)

brain waves,
eye-blinking,
heart contractions,
…

EEG signals



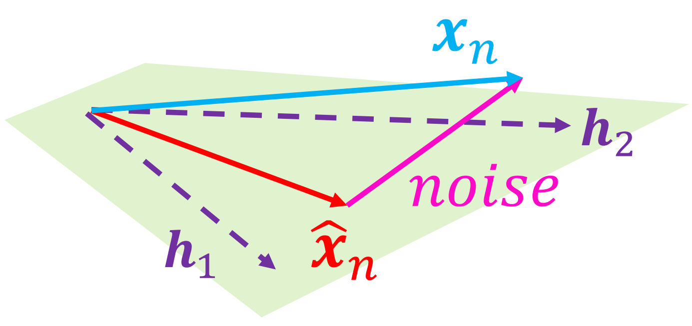
EEG signals input → **ICA** pre-processing → … → x brain waves

6) Independent Component Analysis (ICA)

- ICA model: $\boldsymbol{x}_n = H\boldsymbol{c}_n + \textit{Gaussian noise}$

  $= \hat{\boldsymbol{x}}_n + \textit{Gaussian noise}$

- But this is just LSA model!
- But LSA assumed ~~factor matrix H is orthonormal~~ No. Instead ...
- <u>Independent Component Analysis (ICA)</u>: Assume
  - Elements $c_{i,n}$ of $\boldsymbol{c}_n$ are <u>independent</u> signals "components"
  - $c_{i,n}$ are <u>non-Gaussian</u> variables
- <u>Goal of ICA</u>: Estimate H and $\boldsymbol{c}_n$



H unknown $c_{i,n}$

- ICA is example of "Blind source separation"

ICA algorithms

$$\boldsymbol{x}_n = H\boldsymbol{c}_n + noise$$

- There are many heuristic ICA algorithms
- General idea: Estimate $\boldsymbol{c}_n$ from $\boldsymbol{x}_n$ linearly as $\boldsymbol{z}_n = G^T\boldsymbol{x}_n$
- Optimal "unmixing matrix" $G \approx H^{-1}$ ($\approx c_{i,n}$ wanted)
- But H unknown. So, find G that makes elements z_i of $\boldsymbol{z}_n$

  "as independent and non-Gaussian" as possible, just like $c_{i,n}$
- Example (FastICA algorithm): Minimizes "Entropy"

```
sklearn.decomposition.FastICA()
```

Example: FastICA

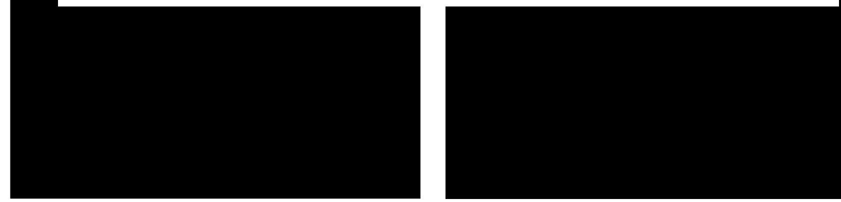
Two Mixed images

$\boldsymbol{x}_n = \left[x_{1,n}, x_{2,n}\right]^T$ is n^{th} pixel pair

Unmixed and then scaled

$D\,\boldsymbol{z}_n = \left[\lambda_1 z_{1,n}, \quad \lambda_2\; z_{2,n}\right]^T$

ICA's optimal G matrix always has a scaling ambiguity



Optimally unmixed images (FastICA)

$\boldsymbol{z}_n = \left[z_{1,n}, z_{2,n}\right]^T$

Underlying true images

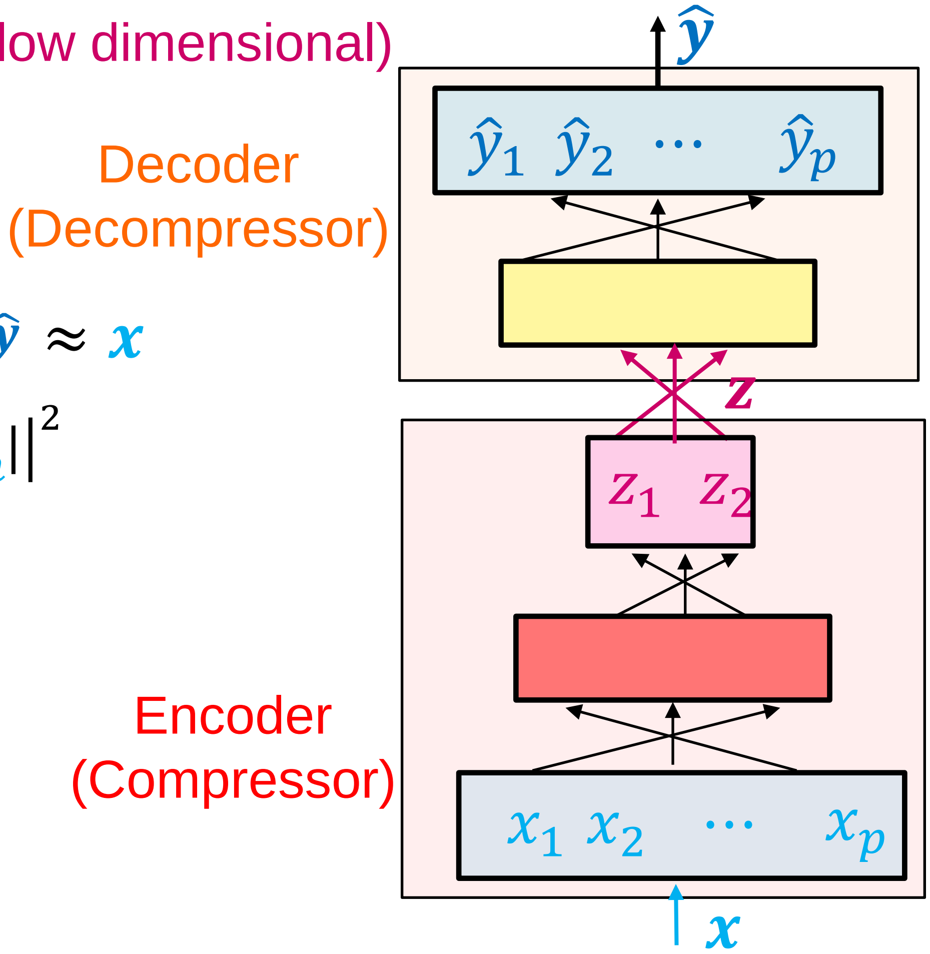
$\boldsymbol{c}_n = \left[c_{1,n}, c_{2,n}\right]^T$

Non-linear methods

7) Auto-encoder

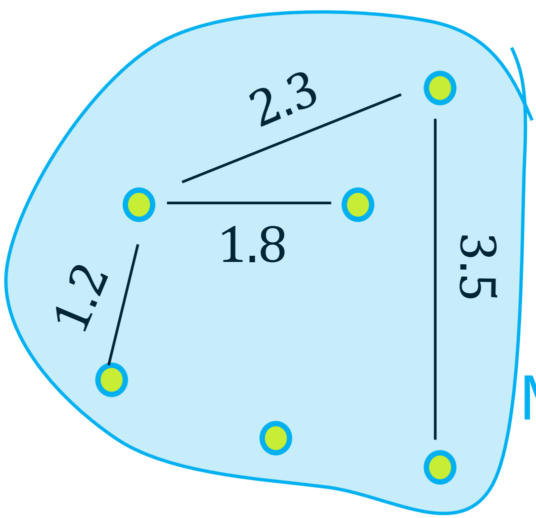
- We have explored Linear feature extraction $\boldsymbol{z}_n = \cancel{G^T \boldsymbol{x}_n}\ g(\boldsymbol{x}_n)$
- Auto-encoder: Non-linear Feature extraction using self-prediction
  - Has an Encoder: $\boldsymbol{x} \rightarrow \boldsymbol{z} \in \mathbb{R}^k$ (low dimensional)
  - Has a Decoder: $\boldsymbol{z} \rightarrow \hat{\boldsymbol{y}}$
  - Train Encoder and Decoder to make $\hat{\boldsymbol{y}} \approx \boldsymbol{x}$

    minimize $\sum_{n=1}^{N} || \hat{\boldsymbol{y}}_n - \boldsymbol{x}_n ||^2$
  - i.e., Self-prediction of input $\boldsymbol{x}$
  - Nowadays uses Deep neural network
  - (Why useful? Details later)



Metric space

- Until now, "input" was vector $\boldsymbol{x}$.   Now, more general inputs …
- Set $S = \{s_1, s_2, \dots, s_N\}$ of N inputs (Abstract ideas, not vectors)
  - Set of images, Set of documents, Set of signals, ...
- Suppose distance function $d(s_i, s_j)$ measures distance between s_i, s_j
- Metric space (S, d): Set S with $d(s_i, s_j)$ assigned to each pair s_i, s_j
  - Example: Euclidean space $\mathbb{R}^p$ is a Metric space with $d(\boldsymbol{s}_1, \boldsymbol{s}_2) \doteq ||\boldsymbol{s}_1 - \boldsymbol{s}_2||$



Metric space (S, d)

Distance function $d(\cdot,\cdot)$

measures 'dissimilarity' of points

$$d(s_1, s_2) = 0 \quad \Leftrightarrow \quad s_1 = s_2$$

$$d(s_1, s_2) = d(s_2, s_1)$$

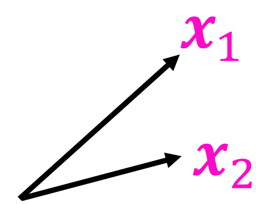
$$d(s_1, s_3) \le d(s_1, s_2) + d(s_2, s_3)$$

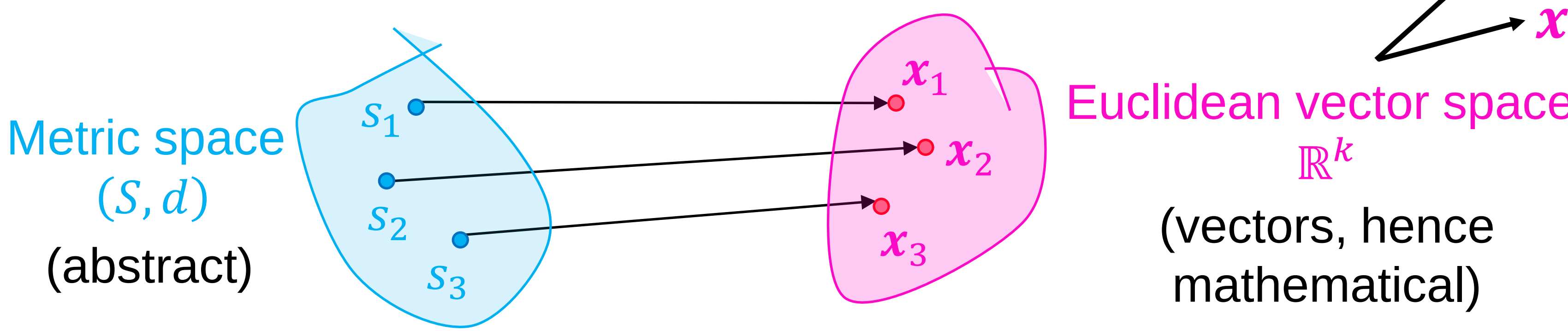
triangle-inequality

Embedding

$$\boldsymbol{x}_n = g(s_n)$$

- Until now, Feature extraction assumed vector input $\boldsymbol{x}$
- Now, from (possibly abstract) input s, get feature vector $\boldsymbol{x}$
- Embedding of Metric space (S, d) : For each abstract input s_n in it map it to feature vector $\boldsymbol{x}_n \in \mathbb{R}^k$
  - Why? For Automatic feature extraction of abstract ideas s_n, or
  - Visualization (if dimension $k = 1,2,3$)
  - Extracted feature vector $\boldsymbol{x}_n$ is called "Embedding" of s_n





8(a) Multi-Dimensional Scaling (MDS)

- Multi-Dimensional Scaling (MDS): Embedding $\boldsymbol{x}$ found by

  retaining the distance geometry between pairs of points

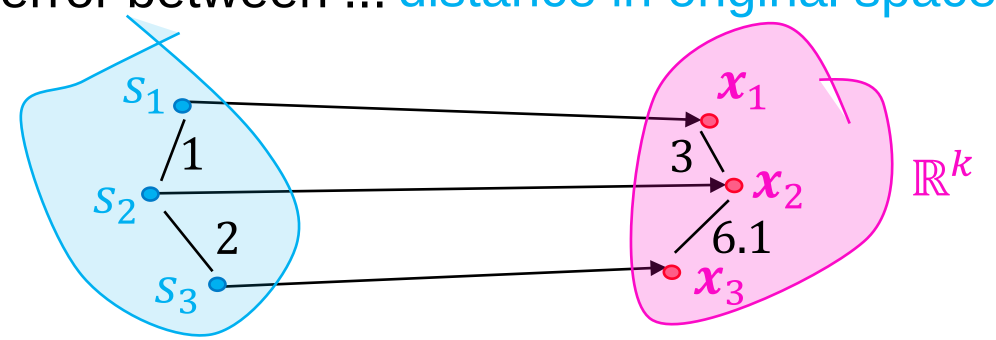
- Heuristic algorithms to solve. e.g., Metric MDS (see handout)

  Python: `sklearn.manifold.MDS`

$$\min_{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(d(s_i, s_j) - \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\| \right)^2$$

Choose embeddings that minimize … the total squared error between ... distance in original space … and distance in new space

Metric space (S, d)



Metric MDS algorithm

- Metric MDS algorithm: Heuristic to embed (S, d) in $\mathbb{R}^k$,

1. Matrix of squared-distances
2. 'Double-centered' matrix

$$D_2 \doteq \begin{bmatrix} \left(d(s_1, s_1)\right)^2 & \cdots & \left(d(s_1, s_N)\right)^2 \\ \vdots & \ddots & \vdots \\ \left(d(s_N, s_1)\right)^2 & \cdots & \left(d(s_N, s_N)\right)^2 \end{bmatrix}$$

$$B = -\frac{1}{2} C\, D_2\, C$$

3. EVD of matrix $B = V \Lambda V^T$

$$C = I_N - \frac{1}{N} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

4. Choose largest k eigen-values $\lambda_1, \lambda_2, \dots, \lambda_k$ and eigen-vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k$
5. Calculate feature vector matrix $\rightarrow$ The N rows of X are the embeddings $\boldsymbol{x}_n$

$$X = [\lambda_1 \boldsymbol{v}_1 \quad \lambda_2 \boldsymbol{v}_2 \quad \dots \quad \lambda_k \boldsymbol{v}_k] = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix}$$

(k columns in X)

- Heuristic exactly solves MDS if (S, d) is already Euclidean space

8(b) t-SNE

- Another popular Embedding is …
- t-distributed Stochastic Neighbor Embedding (t-SNE):

  Similar to MDS, but uses probability models to define distances

  — Algorithm tries to make $q_{ij} \approx p_{ij} \quad \Rightarrow \quad \|\boldsymbol{x}_i - \boldsymbol{x}_j\| \approx d(s_i, s_j)$

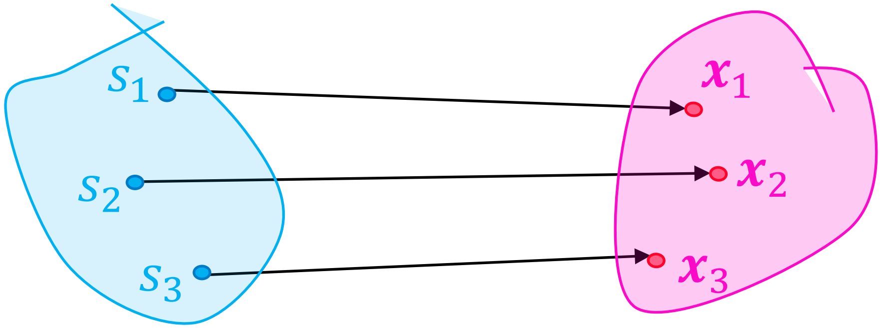
  — See handout for some details

  Python: `sklearn.manifold.TSNE`

- (Later: Embeddings using Deep neural networks becoming popular)

Metric space (S, d)

$d(s_i, s_j) \rightarrow p_{ij}$



$\mathbb{R}^k$

$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \rightarrow q_{ij}$

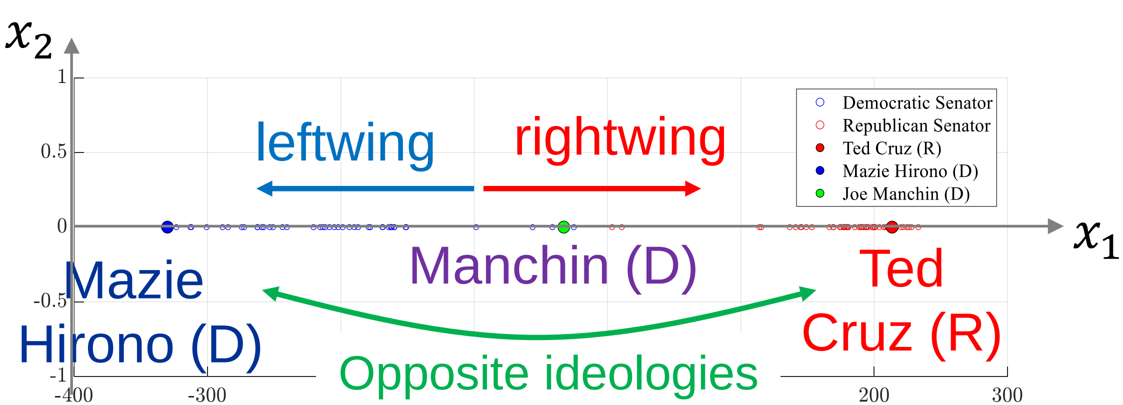
t-SNE

- t-distributed Stochastic Neighbor Embedding (t-SNE):
  - Define conditional probability $p_{j|i} \doteq \begin{cases} \frac{1}{Z_i} \exp\left(-\frac{1}{2\sigma^2}\left(d(s_i, s_j)\right)^2\right), & j \neq i \\ 0, & j = i \end{cases}$
  - Intuition: $p_{j|i}$ is Gaussian-based probability of "point s_i picking point s_j"
  - Define $p_{ij} = \frac{1}{2N}(p_{j|i} + p_{i|j})$
  - In Euclidean space $\mathbb{R}^k$, define $q_{ij} \doteq \begin{cases} \frac{1}{Z}\left(1 + \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right)^{-1}, & j \neq i \\ 0, & j = i \end{cases}$
  - Intuition: Like p_{ij}, but using a heavy-tailed "Student-t pdf"
  - Find the embeddings $\boldsymbol{x}_i$ that minimize Kullback-Leibler distance

$$\min_{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N} \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Example: MDS

- **Senators in 116th Senate***: Voting record of 102 Senators
- Task: Embed Senators into $\mathbb{R}^2$ based on voting record (ideology)
- Metric space (S, d): $S = \{s_1, s_2, \dots, s_{102}\}$ is set of Senators
  - Distance d? Define $d(s_i, s_j)$ as number of times Senators s_i, s_j voted differently
  - $d(s_i, s_j)$ is called 'Hamming distance' $\rightarrow$ So, (S, d) is indeed Metric space
- Metric MDS to embed this metric space into $\mathbb{R}^2$ $(k = 2)$
- $\Rightarrow$ Each Senator is now vector $\boldsymbol{x}_n \in \mathbb{R}^2$. Plot them!



Roll call	Senator	Vote
1	Collins	Nay
1	Manchin	Yea
⋮	⋮	⋮
2	Collins	Yea
2	Manchin	Yea
⋮	⋮	⋮

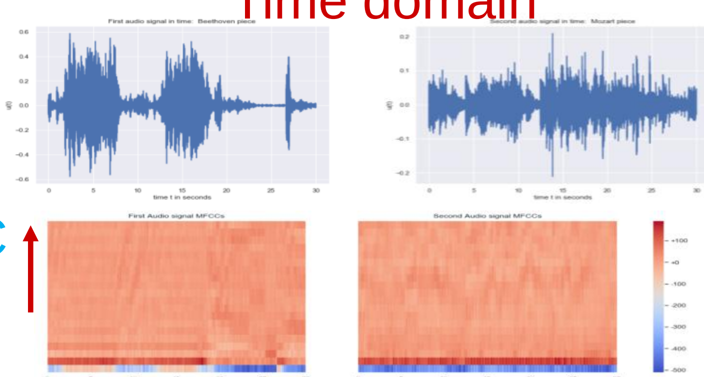
https://voteview.com/data

Example: t-SNE for Visualization

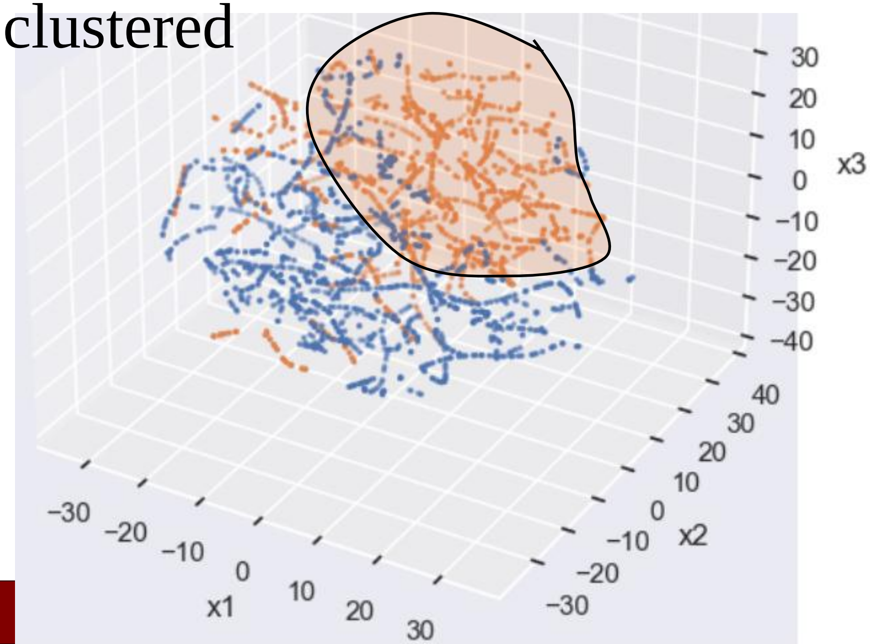
- t-SNE popular alternative to PCA to visualize unlabeled vectors $\boldsymbol{s}_n$ …
- Here, input Metric space is Euclidean space $\rightarrow$ $d(\boldsymbol{s}_i, \boldsymbol{s}_j) = \|\boldsymbol{s}_i - \boldsymbol{s}_j\|$
- Recollect audio demo : Beethoven piece and Mozart piece
- 20-dim MFCC feature vectors $\boldsymbol{s}_n$
- Ignore $\boldsymbol{s}_n$ labels now $\rightarrow$ Get t-SNE embeddings $\boldsymbol{x}_n \in \mathbb{R}^3$
- Mozart and Beethoven embeddings clustered

Time domain

MFCC $\boldsymbol{s}_n$



segment n

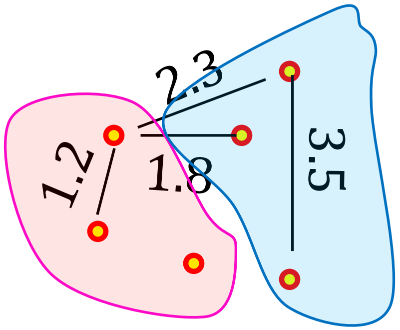


Clustering

9) k-Means clustering

- Suppose unlabeled Training data $s_1, s_2, \dots, s_N$
- Clustering: Group data into k clusters of nearby points
- Application?
  - E.g., Pre-processing step in Feature extraction
  - (Recollect Bag-of-Visual-Words: Dictionary creation using clustering)
- k-Means clustering: A low-complexity clustering algorithm
- Assumption: Data s_n lives in a Metric space (S, d)
  - Example: Euclidean space $\mathbb{R}^p$ with $d(\boldsymbol{s}_i, \boldsymbol{s}_j) = \|\boldsymbol{s}_i - \boldsymbol{s}_j\|$

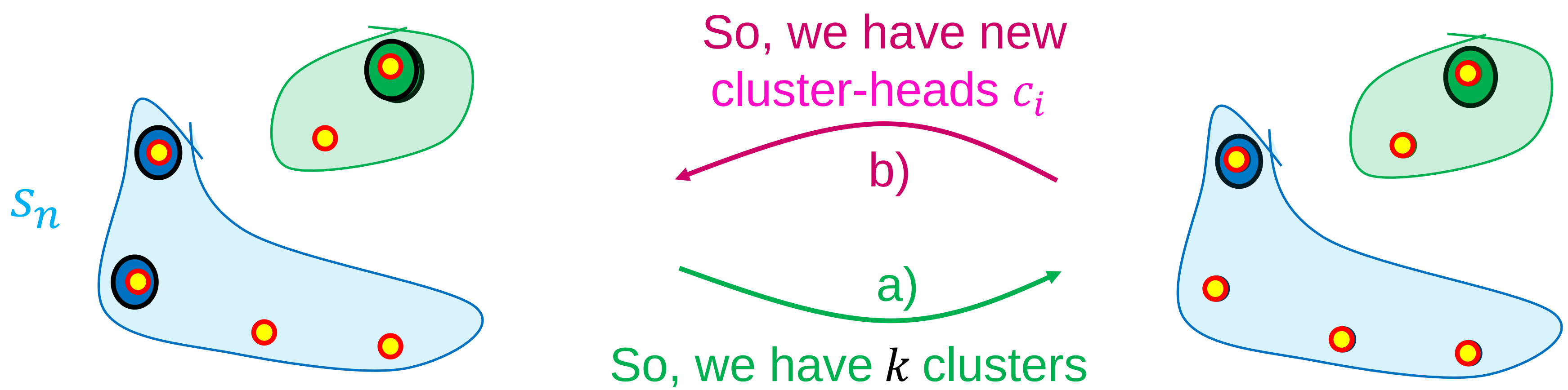
$k = 2$



k-Means clustering

- <u>k-Means clustering</u>: Choose number of clusters k (say 2)

1. Initialize: Randomly select k of the s_n as cluster-heads c (set $\mathcal{C}$)
2. Iterate: Run these two steps
   a) Cluster: Assign each data point s_n to <u>closest</u> cluster head c
   b) New cluster-heads: Each cluster $\rightarrow$ Choose point <u>closest</u> to its points as c_i
3. Terminate: On convergence (cluster-heads don't change)



k-Means clustering

- k-Means clustering: Choose number of clusters k (say 2)

1. Initialize: Randomly select k of the s_n as cluster-heads c (set $\mathcal{C}$)
2. Iterate: Run these two steps

a) Cluster: Assign each data point s_n to closest cluster head c

$$\widehat{c_n} = \underset{c \in \mathcal{C}}{\operatorname{argmin}} \; d(s_n, c)$$

b) New cluster-heads: Each cluster $\rightarrow$ Choose point closest to its points as c_i

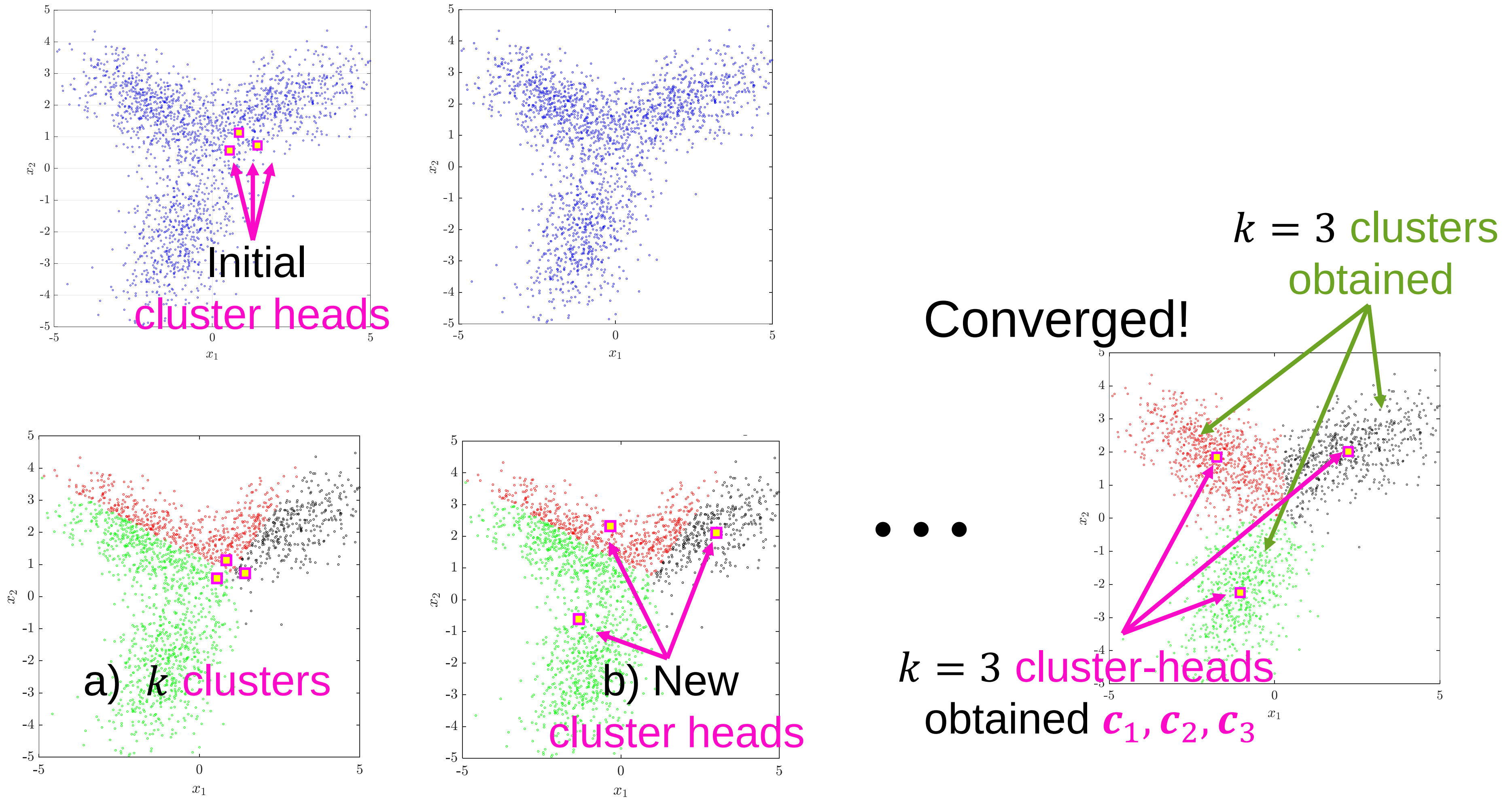
$$c_i = \underset{s \in S}{\operatorname{argmin}} \sum_{\substack{s_n: \\ s_n \text{ is in Cluster } i}} d(s_n, s)$$

In Euclidean space

$$\boldsymbol{c}_i = \frac{1}{n_i}\left(\boldsymbol{s}_{k_1} + \cdots + \boldsymbol{s}_{k_{n_i}}\right)$$

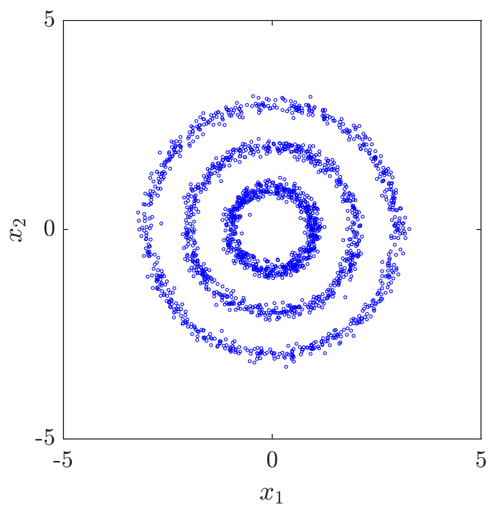
3. Terminate: On convergence (cluster-heads don't change)

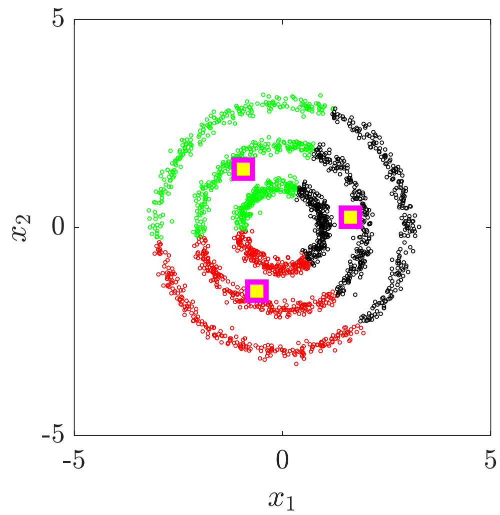
- Choose Euclidean distance $\|\boldsymbol{s}_1 - \boldsymbol{s}_2\|_2$ and cluster with $k = 3$



Example: k-Means clustering

- Metric space choice is important to get "good" clustering
- E.g., Clustering below is "bad" if Metric space is Euclidean space $\mathbb{R}^2$
  - (Need metric space that better captures idea of 'distance' here)
  - (e.g., 'Graph metric space' perhaps)



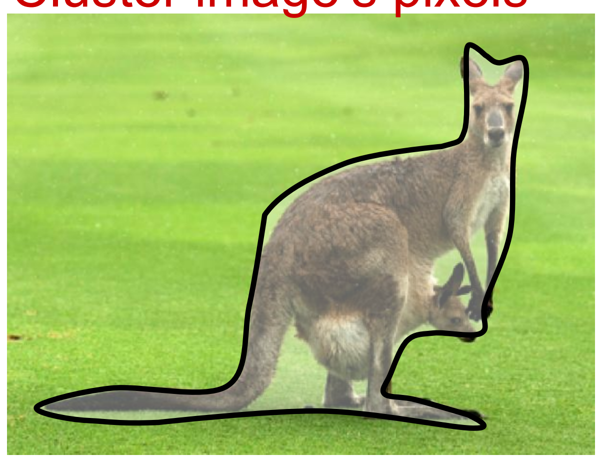


Clustering for Feature extraction

- Clustering algorithm $\rightarrow$ k clusters and cluster heads $\boldsymbol{c}_i$. Use?

1. Dictionary creation: Bag-of-Visual-Words for images (recollect)
2. Segmentation: Partition image into 'objects' (Pre-processing)
   - Each pixel is vector $\boldsymbol{x}_n = \begin{bmatrix} \mathit{Red\ value} \\ \mathit{Green\ value} \\ \mathit{Blue\ value} \end{bmatrix}$
   - Cluster the pixels (vectors $\boldsymbol{x}_n$)
   - Each cluster gives 'mask' $\rightarrow$ Object to be investigated/classified

Cluster image's pixels



$k = 3$ clusters