
ME 536

— Week 6: Spoiler, Not that inverse, —
SVD, PCA, LDA ...

Spoiler Alert Revisited!!!

You have not and will NOT LEARN directly how to

Design an intelligent machine at the end of the course

Darn it, add drops are over

Many topics deserve a semester on its own

Understanding issues in decision making

Having a start-up background

Intelligent Machines

- Has to be **human-like**?
- **Understand human-intelligence** *before*
building Intelligent Machines?

Types of Intelligence

- Beats you in chess ?
- Can prove a theorem ?
- Can tie shoelaces ?
- Knows that milk will rot when left out ?
- Resolves conflict between 2 people?
- ...

Intelligent Machine(s)

Sense → Decide → Act

Goals

Unknown
situation

Predict

Where is, if any?

Self-Awareness
Self-Expression

Physical body

Context

Conflicts

Perception

Thinking

Ethics

Co-operation
Collaboration
Competition

Attention

Limited time

Other machines

Search

Self-driven

Consciousness

Focus on *things* that are:

- Similar / Related vs *Different / New*
- Changing vs *Not changing*

BUT data is:

- Vast
- high dimensional

Recall:

$$\mathbf{Ax} = \mathbf{b}$$

If A is square and non-singular nice and dandy

What if **NOT**

Note that: $\mathbf{A}_{m \times n}$

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}$$

If \mathbf{A} is full rank, then $\mathbf{A}^T \mathbf{A}$ is **????** hence

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$$

So this is not a generalizable approach :(

Recap-

$$\mathbf{M} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$$

U and V are **Orthogonal**:

→ symmetric

→ transpose is inverse

→ do not scale but just rotate - *check out σ_i of U or V !!!*

Note: Multiplication of 2 orthogonal matrices are orthogonal

Consider: A non-square diagonal matrix

For simplicity, 2×4 matrix is considered w.l.g:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \end{bmatrix}$$

Let

$$\Sigma^+ := \begin{bmatrix} \frac{1}{\sigma_1} & 0 \\ 0 & \frac{1}{\sigma_2} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Then,

$$\Sigma \Sigma^+ = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\Sigma^+ \Sigma = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Finally observe that,

$$\Sigma \Sigma^+ \Sigma = \Sigma \text{ and } \Sigma^+ \Sigma \Sigma^+ = \Sigma^+$$

Consider: A non-square diagonal matrix

This time consider 4x2 matrix:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Let

$$\Sigma^+ := \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & 0 & 0 \end{bmatrix}$$

Then,

$$\Sigma\Sigma^+ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Sigma^+\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Still observe that,

$$\Sigma\Sigma^+\Sigma = \Sigma \text{ and } \Sigma^+\Sigma\Sigma^+ = \Sigma^+$$

Let's call it: *for the special case of a diagonal matrix*

Σ^+ is the **PSEUDO-INVERSE** of Σ

$$\Sigma \Sigma^+ \Sigma = \Sigma \text{ and } \Sigma^+ \Sigma \Sigma^+ = \Sigma^+$$

Recall:

$$\mathbf{Ax} = \mathbf{b}$$

If A is square and non-singular nice and dandy

What if **NOT**

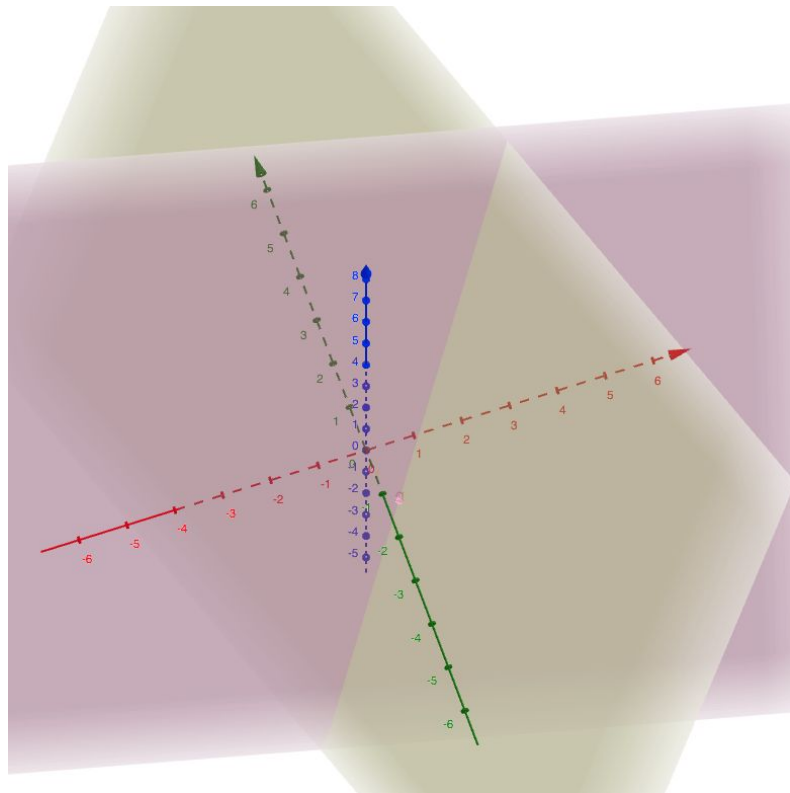
Recall:

$$\mathbf{A}_{m \times n} \mathbf{x}_{n \times 1} = \mathbf{b}_{m \times 1}$$

Underdetermined Case : $n > m$

In general: $\rightarrow \infty$ solutions

$$\left[\begin{array}{c|c|c} \text{purple bar} & \text{yellow bar} & \dots & \text{pink bar} \end{array} \right] \begin{array}{c} \text{red bar} \end{array} = \begin{array}{c} \text{green bar} \end{array}$$



Recall:

$$\mathbf{A}_{m \times n} \mathbf{x}_{n \times 1} = \mathbf{b}_{m \times 1}$$

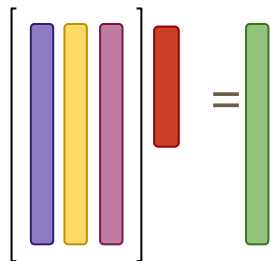
Overdetermined Case : $n < m$

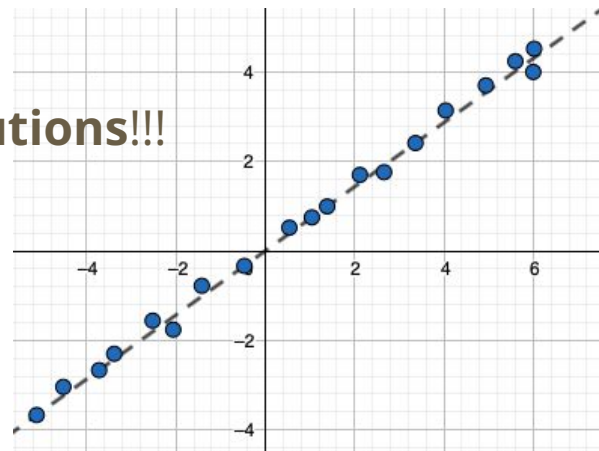
In general: \rightarrow more equations than unknowns

\rightarrow 1 solution if equations are consistent: generally NOT

\rightarrow 0 solution otherwise:

Say hi to **approximate solutions!!!**


$$\begin{bmatrix} \text{purple bar} \\ \text{yellow bar} \\ \text{pink bar} \\ \text{red bar} \end{bmatrix} = \text{green bar}$$

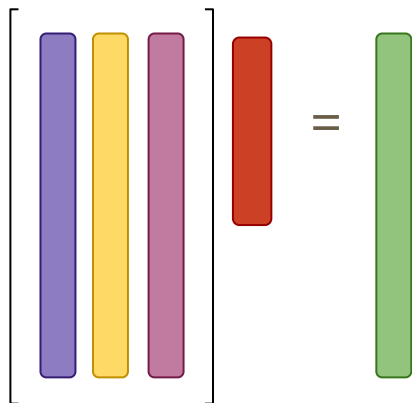


Question:

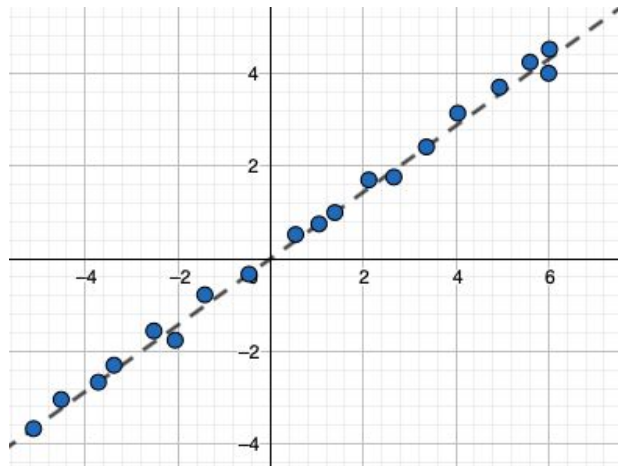
$$\mathbf{A}_{m \times n} \mathbf{x}_{n \times 1} = \mathbf{b}_{m \times 1}$$

Can we get an approximate solution?

But \mathbf{A} is not square \rightarrow not invertible



A diagram illustrating the matrix equation $\mathbf{A} \mathbf{x} = \mathbf{b}$. On the left, a large square bracket contains three vertical bars of different colors (purple, yellow, and pink), representing the columns of matrix \mathbf{A} . To the right of this bracket is a single red vertical bar representing the vector \mathbf{b} . An equals sign follows, and to its right is a single green vertical bar representing the solution vector \mathbf{x} .



the SVD way... $m \neq n$, $\text{rank}(A)$ does not matter

$$\mathbf{A}_{m \times n} \mathbf{x}_{n \times 1} = \mathbf{b}_{m \times 1}$$

SVD think
SVD use
SVD be



SVD it is

Given $\mathbf{Ax} = \mathbf{b}$ and we can use SVD where: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$

Hence,

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \mathbf{x} = \mathbf{b}$$

$$\mathbf{\Sigma}\mathbf{V}^T \mathbf{x} = \mathbf{U}^T \mathbf{b}$$

$$\mathbf{V}^T \mathbf{x} = \mathbf{\Sigma}^+ \mathbf{U}^T \mathbf{b}$$

$$\mathbf{x} = \mathbf{V}\mathbf{\Sigma}^+ \mathbf{U}^T \mathbf{b}$$

Let,

$$\tilde{\mathbf{x}} = \mathbf{A}^+ \mathbf{b}$$

where *left* pseudo-inverse of \mathbf{A} is defined as:

$$\mathbf{A}^+ := \mathbf{V}\mathbf{\Sigma}^+ \mathbf{U}^T$$

and

$\tilde{\mathbf{x}}$ is one of the infinitely many solutions or approximations

Pseudo-inverse: Underdetermined case

For $\mathbf{Ax} = \mathbf{b}$

Let,

$$\tilde{\mathbf{x}} = \mathbf{A}^+ \mathbf{b}$$

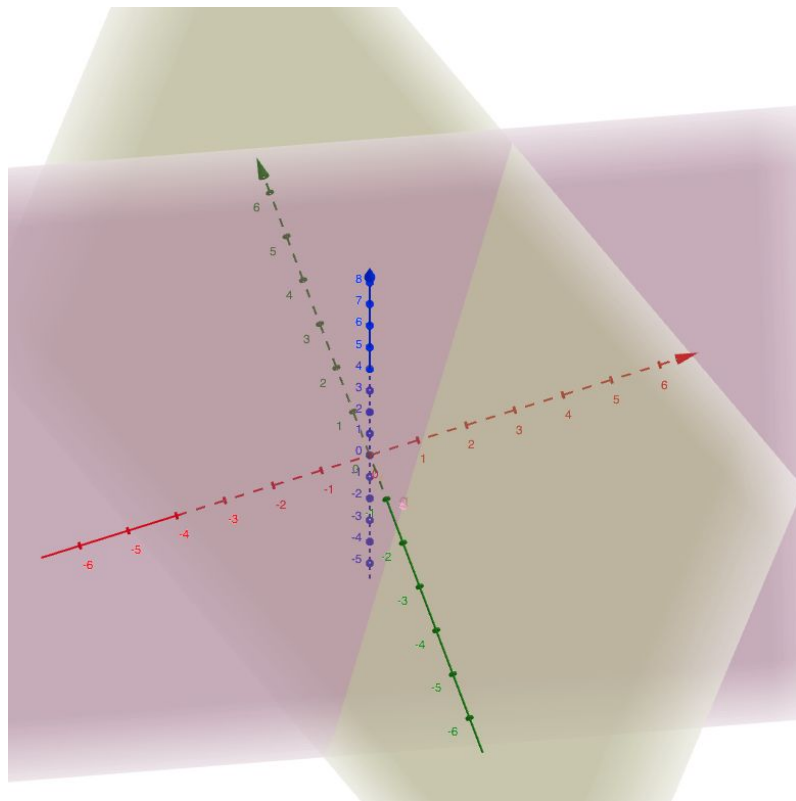
where *left* pseudo-inverse of \mathbf{A} is defined as:

$$\mathbf{A}^+ := \mathbf{V}\mathbf{\Sigma}^+ \mathbf{U}^T$$

$\tilde{\mathbf{x}}$ is the solution with minimal l_2 -norm out of the infinitely many solutions.

In other words, $\hat{\mathbf{x}}$ being one of the infinitely many solutions:

$$\|\tilde{\mathbf{x}}\|_2 \leq \|\hat{\mathbf{x}}\|_2, \forall \hat{\mathbf{x}} \text{ s. t. } \mathbf{A}\hat{\mathbf{x}} = \mathbf{b}$$



Pseudo-inverse: Overdetermined case

For $\mathbf{Ax} = \mathbf{b}$

Let,

$$\tilde{\mathbf{x}} = \mathbf{A}^+ \mathbf{b}$$

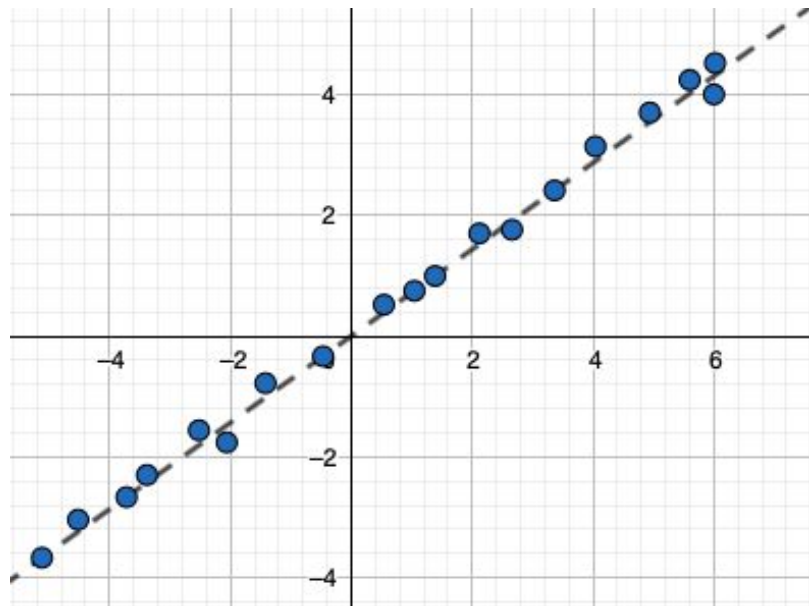
where *left* pseudo-inverse of \mathbf{A} is defined as:

$$\mathbf{A}^+ := \mathbf{V}\mathbf{\Sigma}^+ \mathbf{U}^T$$

$\tilde{\mathbf{x}}$ is the approximate solution with minimal l_2 - error out of the infinitely many possible approximate solutions.

In other words, $\hat{\mathbf{x}}$ being one of the infinitely many possible approximations of \mathbf{x} :

$$\|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{b}\|_2 \leq \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2, \hat{\mathbf{x}} \in \mathbb{R}^n$$



Good old SVD: **U** and **C**

$$\mathbf{M} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$$

$$\mathbf{M} = \mathbf{U} (\mathbf{\Sigma} \mathbf{V}^T)$$

$$\mathbf{M} = \mathbf{U} \mathbf{C}$$

Good old SVD: **U** and **C**

Get a rank-k approximation of **M**

$$\tilde{\mathbf{M}}_{d \times n} = \mathbf{U}_{d \times k} \mathbf{\Sigma}_{k \times k} \mathbf{V}_{k \times n}^T$$

$$\tilde{\mathbf{M}} = \mathbf{U}_{d \times k} \mathbf{C}_{k \times n}$$

SVD → PCA

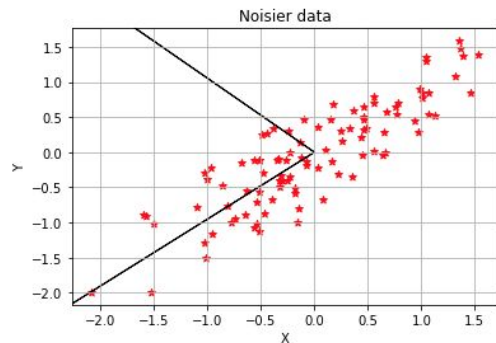
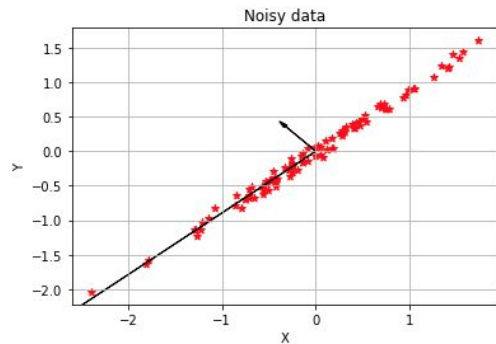
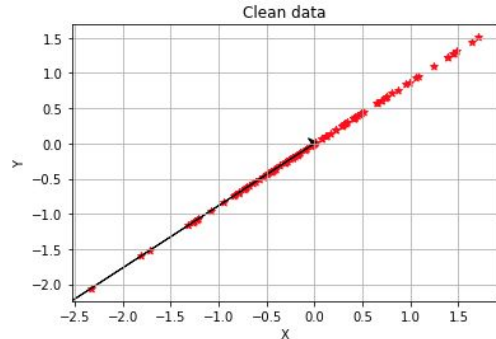
No formal introduction to PCA, but SVD has it all:

- Make sure that your data is zero-mean
- Normalize data if needed
- Columns of \mathbf{U} points along the *Principal Directions*
- Singular values indicate the dominance of *Principal Directions*

Example: SVD \rightarrow PCA

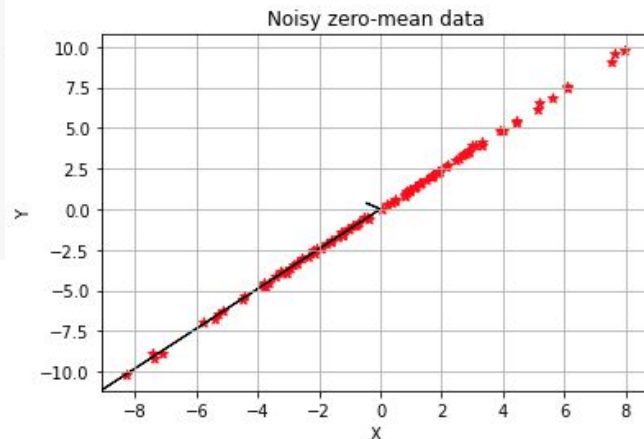
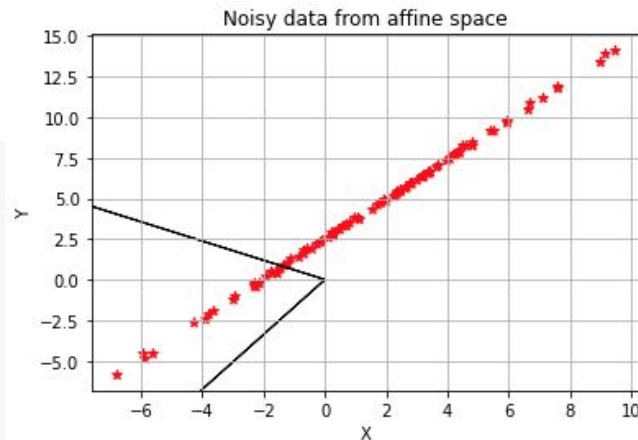
```
1 # let's have datapoints on a line in R2
2 D = DataInSubspace(2, 100, 1)
3 # add a bit of noise
4 Dn = D + np.random.randn(*D.shape)*0.05
5 Dn2 = D + np.random.randn(*D.shape)*0.3
6
7 U,S,Vt = np.linalg.svd(D, full_matrices=False)
8 Un,Sn,Vtn = np.linalg.svd(Dn, full_matrices=False)
9 Un2,Sn2,Vtn2 = np.linalg.svd(Dn2, full_matrices=False)
10
11 # check out the left singular values
12 MatPrint(U @ np.diag(S), 'U of D')
13 MatPrint(Un @ np.diag(Sn), 'U of Dn')
14 MatPrint(Un2 @ np.diag(Sn2), 'U of Dn2')
15
16 CData(D, U @ np.diag(S), title='Clean data')
17 CData(Dn, Un @ np.diag(Sn), title='Noisy data')
18 CData(Dn2, Un2 @ np.diag(Sn2), title='Noisier data')
```

```
U of D
| -7.78  -0.00 |
| -6.88   0.00 |
U of Dn
| -7.82  -0.32 |
| -6.99   0.36 |
U of Dn2
| -7.41  -1.92 |
| -7.05   2.02 |
```



Zero *mean it* if data comes from an affine space

```
1 # let's have datapoints on a line in R2
2 D = DataInSubspace(2, 100, 1)*5
3 # add a bit of noise
4 Dn = D + np.random.randn(*D.shape)*0.05
5 # translate noisy data
6 TM = np.array( [[2],[5]])
7 Dnt = Dn.copy() + TM
8 # Dnt is assumed to be the real data
9 Un,Sn,Vtn = np.linalg.svd(Dnt, full_matrices=False)
10 CData(Dnt, Un @ np.diag(Sn), title='Noisy data from affine space')
11
12 # make data zero-mean
13 Dz = Dnt - Dnt.mean(axis=1).reshape(2,-1)
14 U,S,Vt = np.linalg.svd(Dz, full_matrices=False)
15 CData(Dz, U @ np.diag(S), title='Noisy zero-mean data')
```



Yet another decomposition method

CUR Decomposition

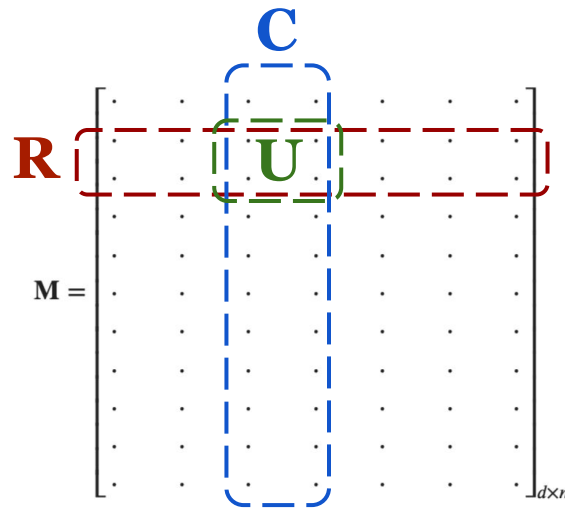
- Compress low rank matrices
- Filter noisy data
- Complete missing data
- Basis from the data itself !!!
- ...

CUR Decomposition: no loss case

- $C = r$ columns
- $R = r$ rows
- U = Intersection of C & R

where $r = \text{rank}(M)$

$$M = C U^{-1} R$$



SVD vs CUR

SVD is unique & forms an orthonormal basis for data

CUR is not unique & comes from data itself

Under certain conditions, denoising via CUR will perform better than SVD

CUR Decomposition: no loss case

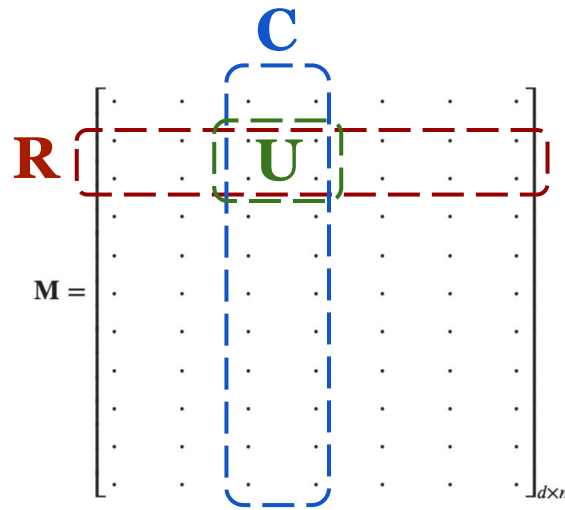
- $C = r$ columns
- $R = r$ rows
- U = Intersection of C & R

Where

actual $\text{rank}(M) = r$

But data is noisy so

$\text{rank}(M) = \min(dxn)$



CUR example: $M = C U^{-1} R$ rank(M) = 2

24.00	-8.00	-15.00	38.00	-9.00	41.00	-9.00	-16.00	-20.00	26.00
7.00	33.00	42.00	-11.00	4.00	-30.00	4.00	13.00	3.00	-41.00
33.00	39.00	45.00	21.00	-3.00	-3.00	-3.00	3.00	-15.00	-33.00
7.00	-15.00	-21.00	19.00	-5.00	27.00	-5.00	-11.00	-9.00	25.00
-9.00	1.00	3.00	-13.00	3.00	-13.00	3.00	5.00	7.00	-7.00
28.00	4.00	0.00	36.00	-8.00	32.00	-8.00	-12.00	-20.00	12.00
4.00	-20.00	-27.00	18.00	-5.00	29.00	-5.00	-12.00	-8.00	30.00
20.00	-4.00	-9.00	30.00	-7.00	31.00	-7.00	-12.00	-16.00	18.00
2.00	-2.00	-3.00	4.00	-1.00	5.00	-1.00	-2.00	-2.00	4.00
17.00	39.00	48.00	-1.00	2.00	-24.00	2.00	11.00	-3.00	-43.00

24.00	-8.00	-15.00	38.00	-9.00	41.00	-9.00	-16.00	-20.00	26.00
7.00	33.00	42.00	-11.00	4.00	-30.00	4.00	13.00	3.00	-41.00
33.00	39.00	45.00	21.00	-3.00	-3.00	-3.00	3.00	-15.00	-33.00
7.00	-15.00	-21.00	19.00	-5.00	27.00	-5.00	-11.00	-9.00	25.00
-9.00	1.00	3.00	-13.00	3.00	-13.00	3.00	5.00	7.00	-7.00
28.00	4.00	0.00	36.00	-8.00	32.00	-8.00	-12.00	-20.00	12.00
4.00	-20.00	-27.00	18.00	-5.00	29.00	-5.00	-12.00	-8.00	30.00
20.00	-4.00	-9.00	30.00	-7.00	31.00	-7.00	-12.00	-16.00	18.00
2.00	-2.00	-3.00	4.00	-1.00	5.00	-1.00	-2.00	-2.00	4.00
17.00	39.00	48.00	-1.00	2.00	-24.00	2.00	11.00	-3.00	-43.00

24.00	-8.00	-15.00	38.00	-9.00	41.00	-9.00	-16.00	-20.00	26.00
7.00	33.00	42.00	-11.00	4.00	-30.00	4.00	13.00	3.00	-41.00
33.00	39.00	45.00	21.00	-3.00	-3.00	-3.00	3.00	-15.00	-33.00
7.00	-15.00	-21.00	19.00	-5.00	27.00	-5.00	-11.00	-9.00	25.00
-9.00	1.00	3.00	-13.00	3.00	-13.00	3.00	5.00	7.00	-7.00
28.00	4.00	0.00	36.00	-8.00	32.00	-8.00	-12.00	-20.00	12.00
4.00	-20.00	-27.00	18.00	-5.00	29.00	-5.00	-12.00	-8.00	30.00
20.00	-4.00	-9.00	30.00	-7.00	31.00	-7.00	-12.00	-16.00	18.00
2.00	-2.00	-3.00	4.00	-1.00	5.00	-1.00	-2.00	-2.00	4.00
17.00	39.00	48.00	-1.00	2.00	-24.00	2.00	11.00	-3.00	-43.00

What if...

Data is not coming from a subspace?

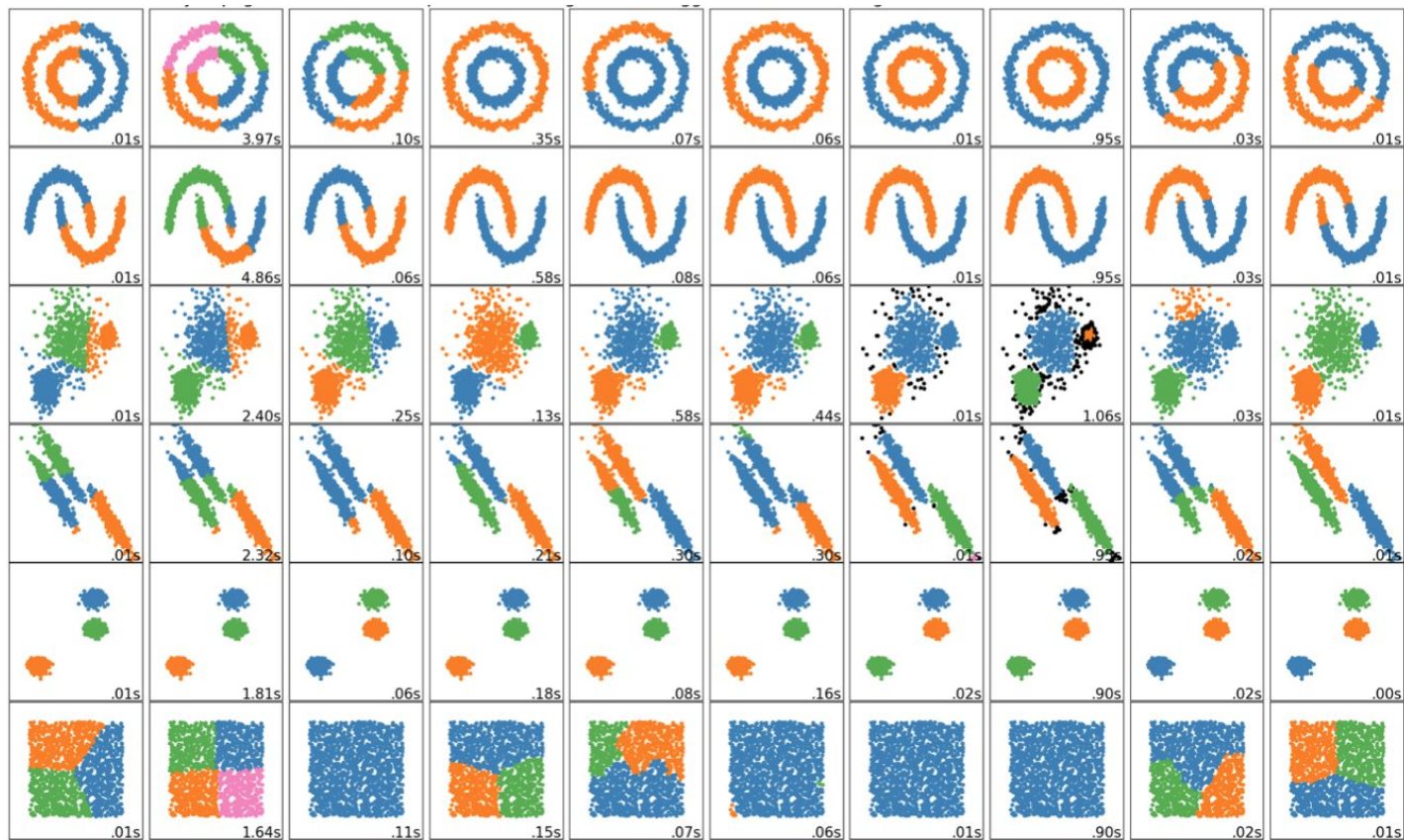
What if multiple subspaces are involved?

What if data is not even coming from subspaces?

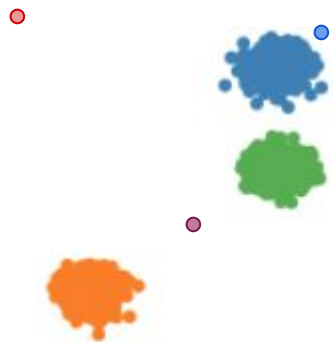
... well that's life...

What if data does not come from a single cluster?

And the fun begins...



Multiple Clusters



How do you handle incoming data?

What if there is

- a conflict?
- an outlier?

What to do with high dimensional data?

- **Cannot visualize** it, *or can we?*
- If not *very* low-rank - good luck
- If *very* low rank (3 or less) visualize via dimension reduction

By the way: What is the point in visualization?

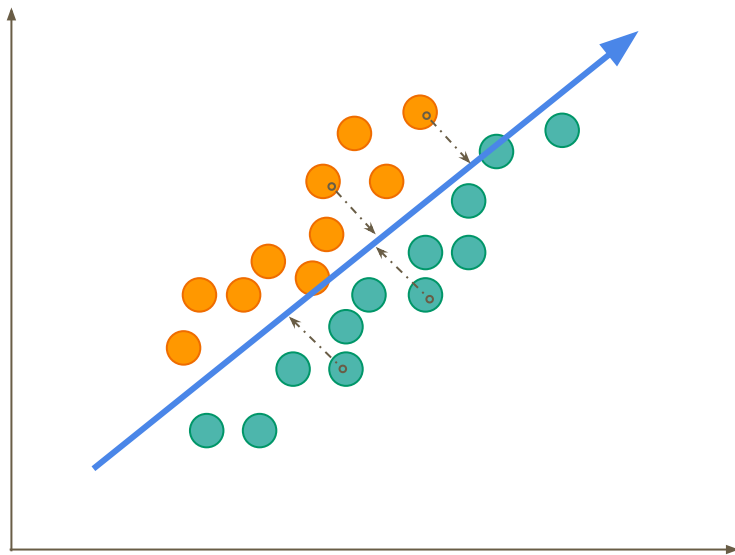
Reducing dimensionality: multiple clusters

What if SVD is used to reduce dimensionality?



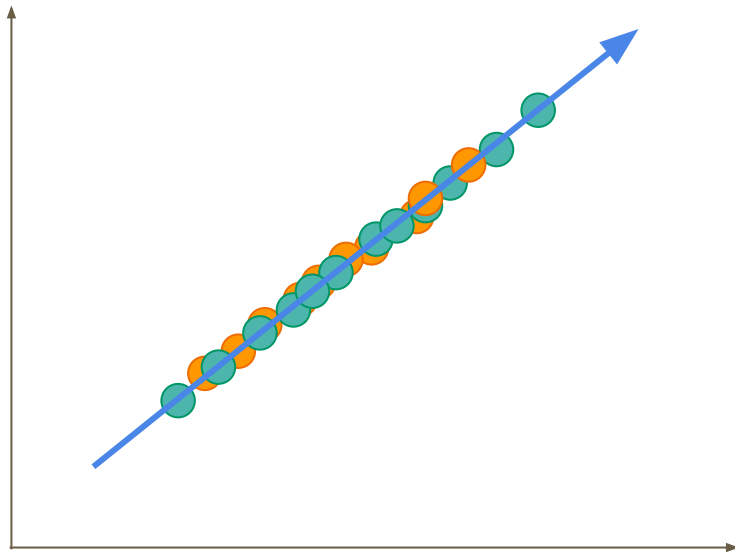
Reducing dimensionality: multiple clusters

SVD way: project data to low dimensions



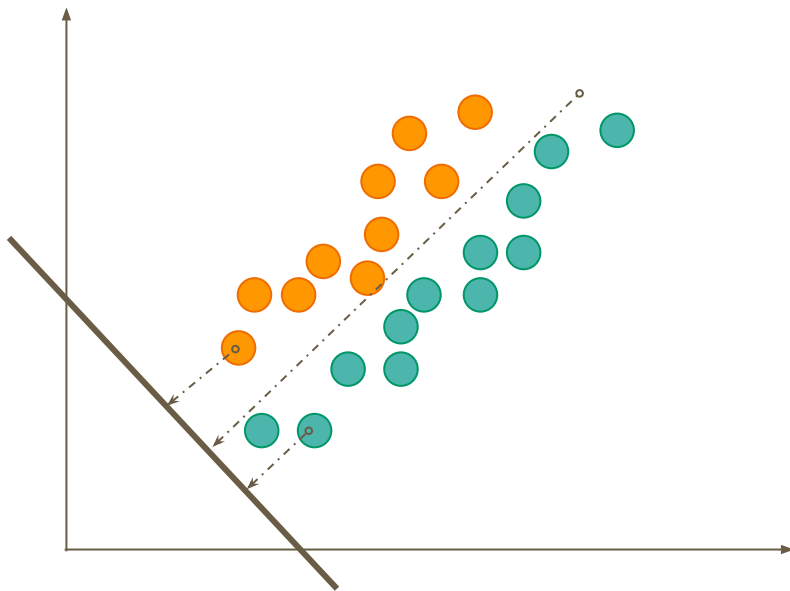
Reducing dimensionality: makes separation harder!

SVD way: project data to low dimensions → did I do something useful?



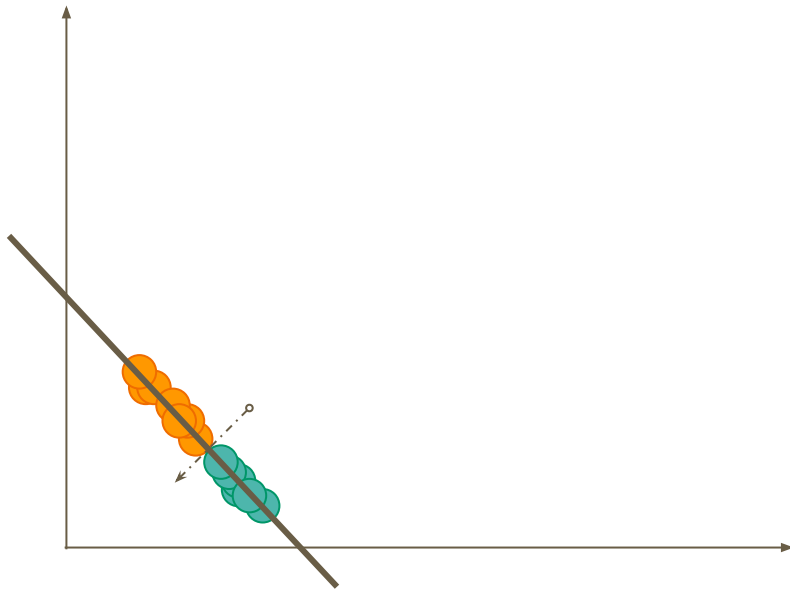
Reducing dimensionality: multiple clusters

LDA: Linear Discriminant Analysis



Reducing dimensionality: multiple clusters

LDA: Linear Discriminant Analysis



PCA vs LDA

- Both reduce dimension
- PCA provides principal directions
- PCA is unsupervised
- LDA provides best directions for discriminating data
- LDA is supervised
- PCA can be used within clusters once clusters are set

Similarity Matrix

Assume that data is in the columns of matrix \mathbf{M}_{dxn} , where m_i are the columns corresponding to data points.

An *ideal* similarity matrix for \mathbf{M} is an $n \times n$ matrix \mathbf{S}_M which is (in most cases) symmetric. If m_i and m_j belong to the same cluster / group / subspace etc., $s_{ij} = 1$ and 0 otherwise.

[Example: Let's start with a small matrix](#)

Complete the IDEAL Similarity Matrix

$S_M =$

1	0		1
			0
0	1		
		0	

Complete the IDEAL Similarity Matrix

$S_N =$

1				1
			1	
	0			
		1		

Complete the IDEAL Similarity Matrix: *with a graph on the side*

$$S_N = \begin{bmatrix} & & 1 & & & \\ 0 & & & & 1 & \\ & & & 1 & & \\ & & & & & \\ & & & & & 1 \\ & & & & & \end{bmatrix}$$

Similarity Matrix

- Good for analyzing high dimensional data
- Can be displayed like an image
- In general, data can be ordered w.l.g.
 - A block diagonal similarity matrix is expected

$$\mathbf{M} = \begin{bmatrix} \text{purple} & \text{purple} & \text{purple} & \text{yellow} & \text{yellow} & \text{yellow} & \text{yellow} & \text{purple} & \text{purple} \end{bmatrix}$$

[illegible]

Heuristics in play:

If data is coming from clusters C_p where $p = 1, \dots, r$

$$f_s(\mathbf{c}_i^{\mathbf{M}}, \mathbf{c}_j^{\mathbf{M}}) = s_{ij} = 1 \text{ if } \mathbf{c}_i^{\mathbf{M}}, \mathbf{c}_j^{\mathbf{M}} \in C_p$$

$$f_s(\mathbf{c}_i^{\mathbf{M}}, \mathbf{c}_j^{\mathbf{M}}) = s_{ij} = 0 \text{ if } \mathbf{c}_i^{\mathbf{M}} \in C_p, \mathbf{c}_j^{\mathbf{M}} \in C_q, p \neq q$$

How do you find f_s given a problem?



Heuristics in play:

If data is coming from clusters C_p where $p = 1, \dots, r$

$$f_s(\mathbf{c}_i^{\mathbf{M}}, \mathbf{c}_j^{\mathbf{M}}) = s_{ij} = 1 \text{ if } \mathbf{c}_i^{\mathbf{M}}, \mathbf{c}_j^{\mathbf{M}} \in C_p$$

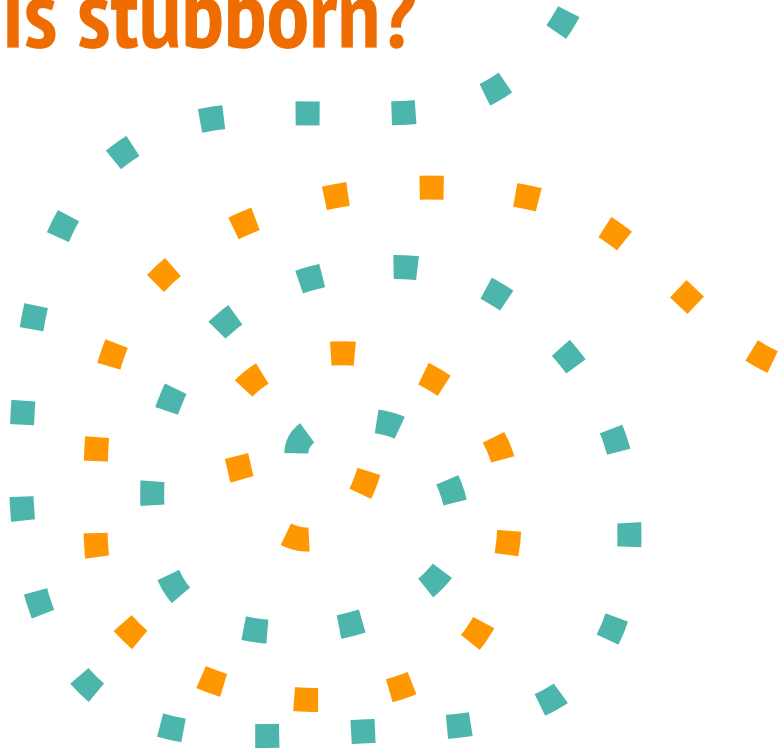
$$f_s(\mathbf{c}_i^{\mathbf{M}}, \mathbf{c}_j^{\mathbf{M}}) = s_{ij} = 0 \text{ if } \mathbf{c}_i^{\mathbf{M}} \in C_p, \mathbf{c}_j^{\mathbf{M}} \in C_q, p \neq q$$

What if you do NOT:

- have any clue about the structure of the data
- know the number of clusters
- Cannot visualize data for cues

Can you still find some f_s given data ?

What if data is stubborn?



See you in another universe brother...

to be continued...

With clustering methods