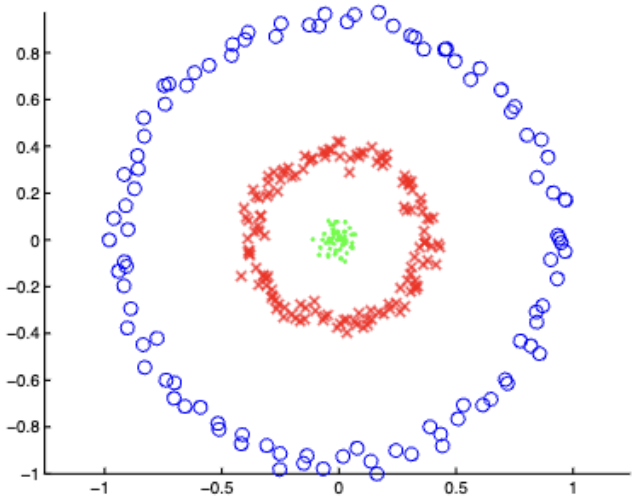
Lesson 12

Final Project, Python

guidelines

- Submission Date: 21/07/2024
- NO EXTENSION!!!
- Automatic tests, manual test

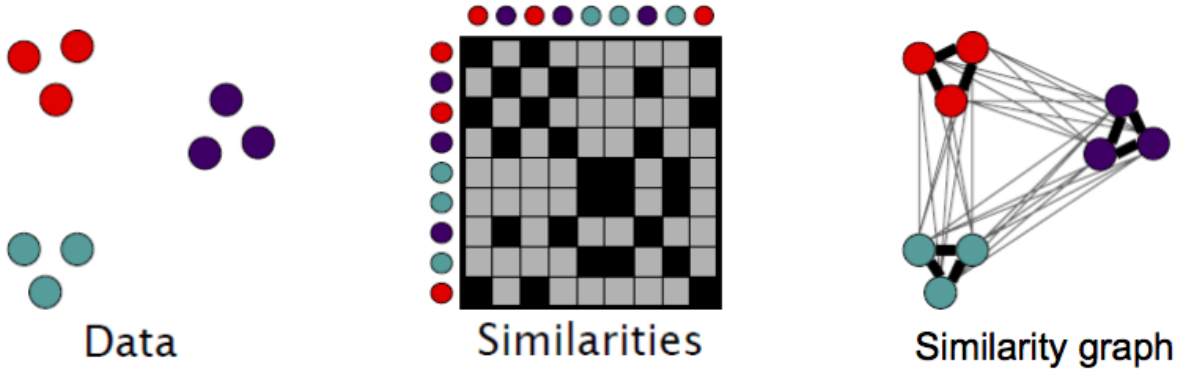
Motivation - Non Linear Data



[Kuang et al.(2012), Simon et al.(2005)]

From Data points to Similarity

- Given: d-dimensional points: x_1, x_2, …, x_n.
- Transform them into a graph (Similarity Graph).
  - G = (V, E; W) - undirected with no self loops.



Weighted Adjacency Matrix

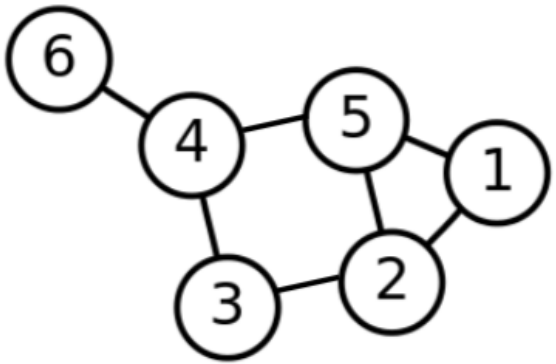
- Gaussian RBF

$$a_{ij} = \exp(\frac{-\|x_i - x_j\|^2}{2})$$

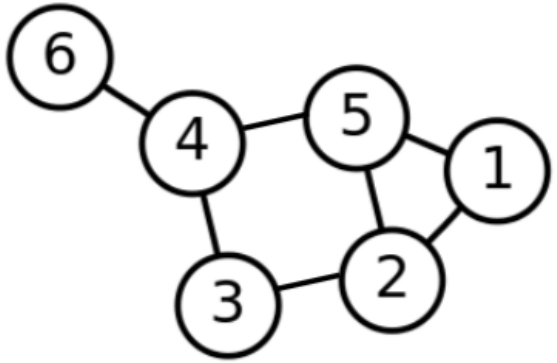
- A is symmetric, non-negative and no self loops $a_{ii} = 0$

Example

- For simplicity, in the next example we will show a non fully connected graph, with all weights set to 1
- We are given d-dimensional data points: x_1 … x_n
- Choose random points and connect them, and we get:



notations



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad W = \begin{bmatrix} 0 & \frac{\sqrt{6}}{6} & 0 & 0 & \frac{\sqrt{6}}{6} & 0 \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{6}}{6} & 0 & \frac{1}{3} & 0 \\ 0 & \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{6}}{6} & 0 & 0 \\ 0 & 0 & \frac{\sqrt{6}}{6} & 0 & \frac{1}{3} & \frac{\sqrt{3}}{3} \\ \frac{\sqrt{6}}{6} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{\sqrt{3}}{3} & 0 & 0 \end{bmatrix}$$

$$d_i = \sum_{j=1}^{n} a_{ij}$$

$$W = D^{-1/2} A D^{-1/2}$$

SymNMF Clustering

- Form the similarity matrix A from X
- Compute the Diagonal Degree Matrix
- Compute the normalized similarity W
- Factorize:

Find $H_{n \times k}$ that solves: $\min_{H \geq 0} \|W - HH^T\|_F^2$

Matrix factorization

Objective:

$$\min_{H \geq 0} \|W - HH^T\|_F^2$$

$H_{n \times k},\ k << n,$

Matrix factorization

- Initialize H randomly, from interval [0,2*sqrt(m/k)], where m is average of W.

- Update H

$$H_{ij}^{(t+1)} \leftarrow H_{ij}^{(t)} \left(1 - \beta + \beta \frac{(WH^{(t)})_{ij}}{(H^{(t)}(H^{(t)})^T H^{(t)})_{ij}}\right)$$

where $\beta = 0.5$

- Convergence

$$\|H^{(t+1)} - H^{(t)}\|_F^2 < \epsilon.$$

Assign clusters

Hard clustering, we choose for each element the cluster with the highest association score.

$$H = \begin{bmatrix} 0.06 & 0.01 \\ 0.01 & 0.05 \\ 0.01 & 0.04 \\ 0.02 & 0.04 \\ 0.05 & 0.02 \end{bmatrix}$$

exam!

structure

- 2 hours
- 1 paper formula sheet (2-sided)
- 4 questions:
  - 2 x open questions in C
  - 2 x open questions Python
- Python Material:
  - Everything we learned in lectures
  - Data science oriented Python questions
  - https://www.practicaldatascience.org/html/class_schedule.html

Example 1

Given table called *production*, which includes the yearly production amount for 5 employees (ascending order). We are interested to calculate the expected production for new employees using table called NEW_EMPS that has columns:ID, years_exp. The expected production is calculated by: exp_prod = years_exp * alpha, where alpha is the average production of employe 3 and 4 in their first 5 years. Complete the below code to solve the question.

```
>>> production.shape #numpy table
(20,5)
# ---------------------------
>>> print(new_emps['exp_prod'])
42105
265455
333008
...
```

Example 1

Given table called *production*, which includes the yearly production amount for 5 employees (ascending order). We are interested to calculate the expected production for new employees using table called NEW_EMPS that has columns:ID, years_exp. The expected production is calculated by: exp_prod = years_exp * alpha, where alpha is the average production of employe 3 and 4 in their first 5 years. Complete the below code to solve the question.

```
>>> production.shape #numpy table
(20,5)
>>> new_emps['exp_prod'] = new_emps['years_exp']*production[:5,[2,3]].mean()
>>> print(new_emps['exp_prod'])
42105
265455
333008
...
```

Example 2

For each **continent** show the **continent** and number of countries with populations of at least 10 million.

```
import numpy as np
import pandas as pd

big_countries = world[world['population']>=10000000]
------
print(big_per_cont)
```

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	78115	3712000000
Angola	Africa	1246700	20609294	100990000000
...				

Example 2

For each **continent** show the **continent** and number of countries with populations of at least 10 million.

```
import numpy as np
import pandas as pd

big_countries = world[world['population']>=10000000]
big_per_cont = big_countries.groupby(['continent']).name.count()
print(big_per_cont)
```

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	78115	3712000000
Angola	Africa	1246700	20609294	100990000000
...				

Example 3

Given a trained classification model by logistic regression, called cls, predict the classification for the point (-0.794, 2.104)

```
from sklearn.linear_model import LogisticRegression

-----------------------------------------------
```

Example 3

Given a trained classification model by logistic regression, called cls, predict the classification for the point (-0.794, 2.104)

```
from sklearn.linear_model import LogisticRegression

Xnew = [[-0.794, 2.104]]
cls.predict(Xnew)
```