## Project 1: Mining information from Text Data

https://colab.research.google.com/drive/1XUit1prSgzaePkPHaD5sdq1QJ3EXdO8k?usp=sharing

**MLDP-Project1-1**

First of all, I read data with abstract from https://aclanthology.org/anthology+abstracts.bib.gz and check the data is fetched correctly. The result was:

url = "https://aclanthology.org/2023.yrrsds-1.0",

}

@inproceedings{chiyah-garcia-2023-processing,

title = "Processing Referential Ambiguities in Situated Dialogue Systems",

author = "Chiyah-Garcia, Javier",

editor = "Hudecek, Vojtech and

Schmidtova, Patricia and

Dinkar, Tanvi and

Chiyah-Garcia, Javier and

Sieinska, Weronika",

booktitle = "Proceedings of the 19th Annual Meeting of the Young Reseachers' Roundtable on Spoken Dialogue Systems",

month = sep,

year = "2023",

address = "Prague, Czechia",

publisher = "Association for Computational Linguistics",

url = "https://aclanthology.org/2023.yrrsds-1.1",

pages = "1--4",

abstract = "Position paper for YRRSDS 2023",

}

After that I wrote for every record including title, abstract, editor, author, publisher, url in a txt file and insert the files in the folder project1, selected 1000 random records and consequently randoms data fin txt files.

## Task 1: Finding Similar Items

**1. Compare the performance in time and the results for k-shingles = 3, 5 and 10, for the three methods and similarity thresholds s=0.1 and 0.2. Use 50 hashing functions. Comment your results.**

I ran the code for k-shingles = 3, 5 and 10, and similarity thresholds s=0.1 and 0.2 **without 50 hash functions** and the result was with numerous similarities, in this case is not seen any exact similarity in finding pairs. (you can see in the output of code)

Before that, I had result of performing of the code for k-shingles = 3, 5 and 10, and similarity thresholds s=0.46, 0,90 and 0.95 and it was the output:

S= 0.46 k =3

Pair ('./project1/388.txt', './project1/829.txt') is similar (Jaccard similarity score: 0.4660714285714286)

Pair ('./project1/388.txt', './project1/788.txt') is similar (Jaccard similarity score: 0.4671340929009641)

Pair ('./project1/90.txt', './project1/75.txt') is similar (Jaccard similarity score: 0.46446700507614214)

Pair ('./project1/829.txt', './project1/182.txt') is similar (Jaccard similarity score: 0.46343612334801765)

Pair ('./project1/544.txt', './project1/549.txt') is similar (Jaccard similarity score: 0.46088957055214724)

Pair ('./project1/544.txt', './project1/788.txt') is similar (Jaccard similarity score: 0.46862265688671556)

Pair ('./project1/549.txt', './project1/788.txt') is similar (Jaccard similarity score: 0.46811945117029863)

Pair ('./project1/945.txt', './project1/905.txt') is similar (Jaccard similarity score: 0.47572815533980584)

Pair ('./project1/533.txt', './project1/469.txt') is similar (Jaccard similarity score: 0.49019607843137253)

Pair ('./project1/533.txt', './project1/411.txt') is similar (Jaccard similarity score: 0.4733606557377049)

Pair ('./project1/533.txt', './project1/846.txt') is similar (Jaccard similarity score: 0.4766260162601626)

Pair ('./project1/997.txt', './project1/538.txt') is similar (Jaccard similarity score: 0.47692307692307695)

Pair ('./project1/997.txt', './project1/121.txt') is similar (Jaccard similarity score: 0.5121951219512195)

Pair ('./project1/469.txt', './project1/411.txt') is similar (Jaccard similarity score: 0.4735772357723577)

K-Shingles: 3, Threshold: 0.46, Number of similar items: 14, Execution Time: 800.36 seconds

S=0.9 k=3

K-Shingles: 3, Threshold: 0.9, Number of similar items: 0, Execution Time: 794.07 seconds

S=0.95 k=3

K-Shingles: 3, Threshold: 0.95, Number of similar items: 0, Execution Time: 789.97 seconds

S=0.46 k=5

K-Shingles: 5, Threshold: 0.46, Number of similar items: 0, Execution Time: 824.45 seconds

S=0.9 k=5

K-Shingles: 5, Threshold: 0.9, Number of similar items: 0, Execution Time: 825.14 seconds

S=0.95 k=5

K-Shingles: 5, Threshold: 0.95, Number of similar items: 0, Execution Time: 815.84 seconds

S=0.46 k=10

K-Shingles: 10, Threshold: 0.46, Number of similar items: 0, Execution Time: 852.75 seconds

S=0.9 k=10

K-Shingles: 10, Threshold: 0.9, Number of similar items: 0, Execution Time: 854.78 seconds

S=0.95 k=10

K-Shingles: 10, Threshold: 0.95, Number of similar items: 0, Execution Time: 854.40 seconds

print(read_file('./project1/533.txt'))

This paper presents the results of the newstranslation task……..

print(read_file('./project1/469.txt'))

This paper presents the University of Edinburgh{'}s …….

**Comment: Certainly, in high threshold, we can have more similar pairs but in s=0.1, 0,2, we cannot have acceptable achievement.**

**With hashing functions = 50**

I changed the code of function get_shingles() to have 50 hashing function. There was same result for S=0.1, 0.2 and 0.25 means not exact similarity but as you see with k=3,5,10 and S =0.46, 0.90 and 0.95, we can have some similar pairs.

k=3 S=0.46

Pair ('./project1/388.txt', './project1/829.txt') is similar (Jaccard similarity score: 0.4660714285714286)

Pair ('./project1/388.txt', './project1/788.txt') is similar (Jaccard similarity score: 0.4671340929009641)

Pair ('./project1/90.txt', './project1/75.txt') is similar (Jaccard similarity score: 0.46446700507614214)

Pair ('./project1/829.txt', './project1/182.txt') is similar (Jaccard similarity score: 0.46343612334801765)

Pair ('./project1/544.txt', './project1/549.txt') is similar (Jaccard similarity score: 0.46088957055214724)

Pair ('./project1/544.txt', './project1/788.txt') is similar (Jaccard similarity score: 0.46862265688671556)

Pair ('./project1/549.txt', './project1/788.txt') is similar (Jaccard similarity score: 0.46811945117029863)

Pair ('./project1/945.txt', './project1/905.txt') is similar (Jaccard similarity score: 0.47572815533980584)

Pair ('./project1/533.txt', './project1/469.txt') is similar (Jaccard similarity score: 0.49019607843137253)

Pair ('./project1/533.txt', './project1/411.txt') is similar (Jaccard similarity score: 0.4733606557377049)

Pair ('./project1/533.txt', './project1/846.txt') is similar (Jaccard similarity score: 0.4766260162601626)

Pair ('./project1/997.txt', './project1/538.txt') is similar (Jaccard similarity score: 0.47692307692307695)

Pair ('./project1/997.txt', './project1/121.txt') is similar (Jaccard similarity score: 0.5121951219512195)

Pair ('./project1/469.txt', './project1/411.txt') is similar (Jaccard similarity score: 0.4735772357723577)

K-Shingles: 3, Threshold: 0.46, Number of similar items: 14, Execution Time: 1619.09 seconds

K=3 S=0.9

K-Shingles: 3, Threshold: 0.9, Number of similar items: 0, Execution Time: 1569.91 seconds

K=3 S=0.95

K-Shingles: 3, Threshold: 0.95, Number of similar items: 0, Execution Time: 1591.84 seconds

K=5 S=0.46

K-Shingles: 5, Threshold: 0.46, Number of similar items: 0, Execution Time: 1643.75 seconds

K=5 S=0.9

K-Shingles: 5, Threshold: 0.9, Number of similar items: 0, Execution Time: 1660.76 seconds

K=5 S=0.95

K-Shingles: 5, Threshold: 0.95, Number of similar items: 0, Execution Time: 1663.77 seconds

K=10 S=0.46

K-Shingles: 10, Threshold: 0.46, Number of similar items: 0, Execution Time: 1666.26 seconds

K=10 S=0.9

K-Shingles: 10, Threshold: 0.9, Number of similar items: 0, Execution Time: 1686.18 seconds

K=10 S=0.95

K-Shingles: 10, Threshold: 0.95, Number of similar items: 0, Execution Time: 1704.33 seconds

print(read_file('./project1/90.txt'))

This document describes the participation of Webinterpret in the shared …….

print(read_file('./project1/75.txt'))

This document describes a hybrid pipeline comprising rules and machine learning …….

**Comment: Again using 50 hash functions, in high threshold, we can have more almost similar pairs.**

**2. Compare the results obtained for MinHash and LSH for different similarity thresholds s = 0.1, 0.2 and 0.25 and 50, 100 and 200 hashing functions. Comment your results.**

About comparing the results obtained for MinHash and LSH for different similarity thresholds s = 0.1, 0.2 and 0.25 and 50, 100 and 200 hashing functions:

In **Minhash** the result was not acceptable for s = 0.1, 0.2 and 0.25 (it can be seen in the output of code) and in other setting to check thresholds s = 0.5, 0.9 and 0.95

Similarity threshold: 0.5, Number of Hash Functions: 50

Finding candidates took 4.566491603851318 seconds

Found 31 candidates

./project1/111.txt ./project1/373.txt

./project1/111.txt ./project1/478.txt

./project1/680.txt ./project1/280.txt

.

.

./project1/259.txt ./project1/787.txt

./project1/844.txt ./project1/787.txt

./project1/844.txt ./project1/642.txt

./project1/578.txt ./project1/642.txt

Similarity threshold: 0.5, Number of Hash Functions: 100

Finding candidates took 4.313330888748169 seconds

Found 31 candidates

./project1/111.txt ./project1/373.txt

./project1/111.txt ./project1/478.txt

./project1/680.txt ./project1/280.txt

./project1/746.txt ./project1/578.txt

.

.

./project1/746.txt ./project1/184.txt

./project1/746.txt ./project1/642.txt

./project1/259.txt ./project1/787.txt

./project1/844.txt ./project1/787.txt

./project1/844.txt ./project1/642.txt

./project1/578.txt ./project1/642.txt

Similarity threshold: 0.5, Number of Hash Functions: 200

Finding candidates took 5.167374849319458 seconds

Found 31 candidates

./project1/111.txt ./project1/373.txt

./project1/111.txt ./project1/478.txt

./project1/680.txt ./project1/280.txt

./project1/680.txt ./project1/82.txt

./project1/341.txt ./project1/707.txt

.

.

./project1/746.txt ./project1/184.txt

./project1/746.txt ./project1/642.txt

./project1/259.txt ./project1/787.txt

./project1/844.txt ./project1/787.txt

./project1/844.txt ./project1/642.txt

./project1/578.txt ./project1/642.txt

Similarity threshold: 0.9, Number of Hash Functions: 50

Finding candidates took 4.157390356063843 seconds

Found 0 candidates

Similarity threshold: 0.9, Number of Hash Functions: 100

Finding candidates took 4.34530234336853 seconds

Found 0 candidates

Similarity threshold: 0.9, Number of Hash Functions: 200

Finding candidates took 5.346006631851196 seconds

Found 0 candidates

Similarity threshold: 0.95, Number of Hash Functions: 50

Finding candidates took 4.184524774551392 seconds

Found 0 candidates

Similarity threshold: 0.95, Number of Hash Functions: 100

Finding candidates took 4.769109725952148 seconds

Found 0 candidates

Similarity threshold: 0.95, Number of Hash Functions: 200

Finding candidates took 4.814048528671265 seconds

Found 0 candidates

print(read_file('./project1/111.txt'))

Editor: Lavelli, Alberto and Carroll, John and Berwick, ……

print(read_file('./project1/373.txt'))

Editor: Bunt, Harry and Berwick, Robert and Church, Ken …..

**Comment: In Minhash and different hash functions, again just in high threshold, we can have more almost similar pairs but in less time than k_shingle.**

And for **LSH**, as we know the S does not have meaning because bands = 10, rows = 10 as b , r.

nsig = bands*rows  are the number of elements in signature, or the number of different random hash functions, means s(threshold) almost equal (1/b)**(1/r) so we do not need the thresholds.

finding candidates took 0.10364532470703125 seconds

found 6 candidates

candidate similar pairs of files are:

./project1/746.txt ./project1/970.txt

./project1/196.txt ./project1/107.txt

./project1/662.txt ./project1/839.txt

./project1/437.txt ./project1/298.txt

./project1/242.txt ./project1/787.txt

./project1/751.txt ./project1/366.txt

print(read_file('./project1/751.txt'))

Despite end-to-end neural systems making significant progress in the last decade for task-oriented …

print(read_file('./project1/366.txt'))

Despite end-to-end neural systems has been applied successfully to a range of tasks…….

For 50 hashing functions:

Finding candidates took 0.011356830596923828 seconds

Found 0 candidates

For 100 hashing functions:

Finding candidates took 0.023818492889404297 seconds

Found 6 candidates

For 200 hashing functions:

Finding candidates took 0.09367775917053223 seconds

Found 6 candidates

**Comment: In LSH, we can have almost similar pairs but in the least time.**

## Task 2: Mining information from Text Data

https://colab.research.google.com/drive/1Q_iCzmTbVEQZhmObWcI10Qmvvj28I5F3?usp=sharing

**MLDP-Project1-2**

I wrote code of this task in separated program, so 6 cells at the beginning are the same as first part .I tried to create a data.txt file including 2 separate basket Author and Editor for each txt file (1000 files) and cleaned the data, the rows in this txt file will be 2000, the result look likes:

Li,Sheng,Sun,Maosong,Liu,Yang,Wu,Hua,Liu,Kang,Che,Wanxiang,He,Shizhu,Rao,Gaoqi

Tang,Xuemei,Su,Qi,Wang,Jun,Chen,Yuhang,Yang,Hao

Bouamor,Houda,Pino,Juan,Bali,Kalika

Wang,Zhecan,Chen,Long,You,Haoxuan,Xu,Keyang,He,Yicheng,Li,Wenhao,Codella,Noel,Chang,KaiWei,Chang,ShihFu

Fiumara,James,Cieri,Christopher,Liberman,Mark,CallisonBurch,Chris

Chamberlain,Jon,Kruschwitz,Udo,Poesio,Massimo

Habash,Nizar,Diab,Mona,Darwish,Kareem,ElHajj,Wassim,AlKhalifa,Hend,Bouamor,Houda,Tomeh,Nadi,ElHaj,Mahmoud,Zaghouani,Wajdi

**I also put Author and Editor in one basket but I did not get any suitable result.**

**1. Find the frequent pair of items (2-tuples) using the naïve, A-priori and PCY algorithms. For each of these compare the time of execution and results for supports s=10, 50, 100. Comment your results.**

The frequent pair of items (2-tuples) using the naïve, A-priori and PCY algorithms and for each of these compare the time of execution and results for supports s=10, 50, 100 are:

**Naïve**

Len 1-tuple is **1292** and 2-tuples **834113**

CPU times: user 5 µs, sys: 0 ns, total: 5 µs

Wall time: 7.63 µs

For s = 10:

35995 items with >10 occurrences

For s = 50:

3777 items with >50 occurrences

For s = 100:

778 items with >100 occurrences

The interest is so different:

ohn,Trevor,He,Yulan,Liu,Yang --> Calzolari,Nicoletta,Choukri,Khalid,Declerck,Thierry,Loftsson,Hrafn,Maegaard,Bente,Mariani,Joseph,Moreno,Asuncion,Odijk,Jan,Piperidis,Stelios with interest 14.992500

Moens,MarieFrancine,Huang,Xuanjing,Specia,Lucia,Yih,ScottWentau --> Cohn,Trevor,He,Yulan,Liu,Yang with interest 17.991000

Calzolari,Nicoletta,Bechet,Frederic,Blache,Philippe,Choukri,Khalid,Cieri,Christopher,Declerck,Thierry,Goggi,Sara,Isahara,Hitoshi,Maegaard,Bente,Mariani,Joseph,Mazo,Helene,Moreno,Asuncion,Odijk,Jan,Piperidis,Stelios --> Cohn,Trevor,He,Yulan,Liu,Yang with interest 17.991000

Cohn,Trevor,He,Yulan,Liu,Yang --> Calzolari,Nicoletta,Choukri,Khalid,Maegaard,Bente,Mariani,Joseph,Odijk,Jan,Piperidis,Stelios,Tapias,Daniel with interest 10.994500

Cohn,Trevor,He,Yulan,Liu,Yang --> Matsumoto,Yuji,Prasad,Rashmi with interest 10.994500

Cohn,Trevor,He,Yulan,Liu,Yang --> Scott,Donia,Bel,Nuria,Zong,Chengqing with interest 10.994500

Cohn,Trevor,He,Yulan,Liu,Yang --> Calzolari,Nicoletta,Choukri,Khalid,Maegaard,Bente,Mariani,Joseph,Odijk,Jan,Piperidis,Stelios,Rosner,Mike,Tapias,Daniel with interest 14.992500

Cohn,Trevor,He,Yulan,Liu,Yang --> Zong,Chengqing,Xia,Fei,Li,Wenjie,Navigli,Roberto with interest 17.991000,

**A-priori**

Len(L1): **25**, Len(L2): **325**

CPU times: user 5 µs, sys: 0 ns, total: 4 µs

Wall time: 9.06 µs

A-priori: 325 items with >10 occurrences

A-priori: 325 items with >50 occurrences

A-priori: 317 items with >100 occurrences

**PCY**

Len(L1) = **25** , Len(L2) = **368**

CPU times: user 3 µs, sys: 0 ns, total: 3 µs

Wall time: 6.91 µs

PCY: 317 items with >10 occurrences

PCY: 317 items with >50 occurrences

PCY: 317 items with >100 occurrences

**Comment: The number of items decrease and also time.**

**2. For the PCY algorithm, create up to 5 compact hash tables. What is the difference in results and time of execution for 1,2,3,4 and 5 tables? Comment your results.**

Execution time for s = 10: 32.10754728317261 seconds

5 hash tables PCY 325 items

Execution time for s = 50: 29.710164308547974 seconds

5 hash tables PCY 325 items

Execution time for s = 100: 30.751627922058105 seconds

5 hash tables PCY 317 items

**Comment: the times are different but items in s= 100 is less than s=10 and 50.**

**3. Find the final list of k-frequent items (k-tuples) for k=3 and 4. Experiment a bit and describe the best value for the support in each case. Warning: You can use any of the three algorithms, but be careful because the algorithm can take too long if you don't choose it properly.**

3-tuples

CPU times: user 4 µs, sys: 0 ns, total: 4 µs

Wall time: 8.11 µs

Time is running out – it takes too times, with code was controlled.

578 items ……. 2625 items

4-tuples

CPU times: user 5 µs, sys: 0 ns, total: 5 µs

Wall time: 8.34 µs

Time is running out- it takes too times, with code was controlled.

277 items, I did not get result in exact time

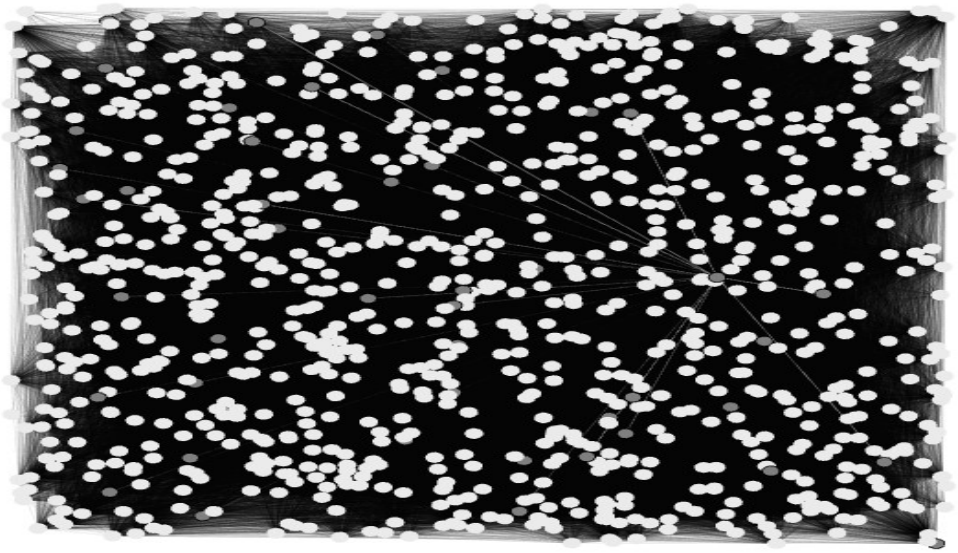
**Comment: It takes too time to run and I used the A-Priori method and anyway in the huge data, we should employ some techniques to have k-tuples and decrease time.**

## Task 3: Graphs and Social Networks

**1. From the abstracts part, using two different similarity thresholds (e.g. 0.05, 0.1), create an unweighted network from the obtained similar pairs (use MinHashing or LSH) for each threshold.**

Because of refering the task to part 1 ,2 , the codes was put in 2 programs MLDP-Project1-1 and MLDP-Project1-2.

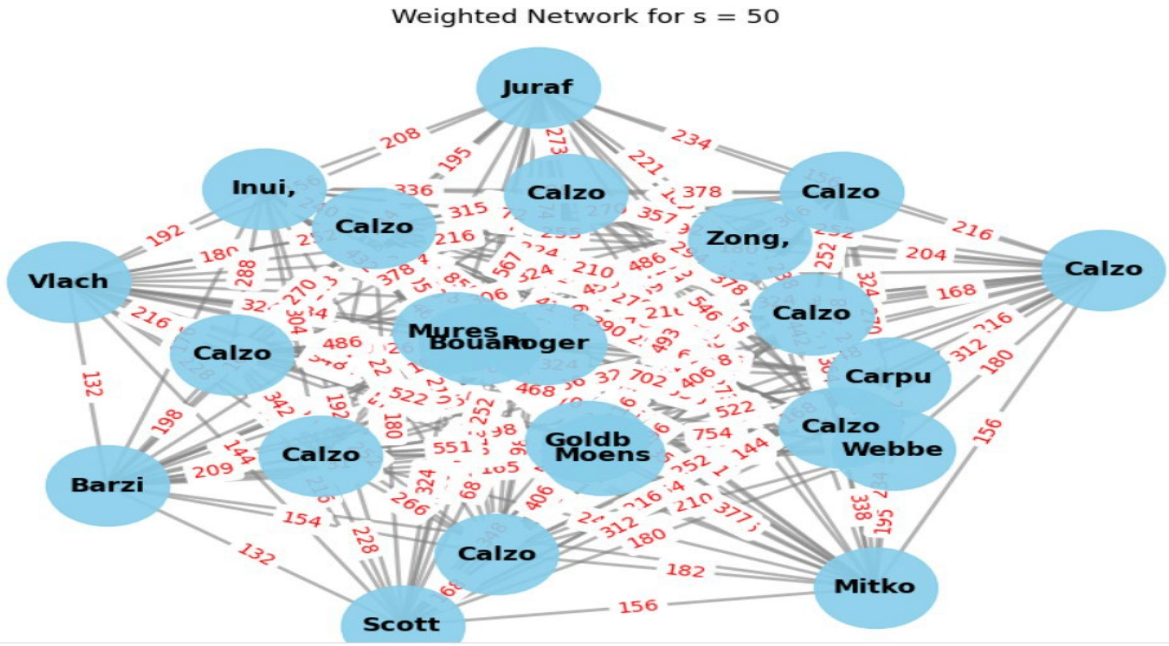
I used Minhash and in s=0.1 or s=0.05, there was a huge data and the image turned to **black**, so I used s=0.5 and hash function 50, 100 and 200 and tried to use Gephi software and I had this image almost:



I changed the threshold and hash function and plot it.


**2. From the basket list of authors/editors, using A-Priori or PCY, create a *weighted network*, using the *support threshold* as weight.**

I used A-priori and for a weighted network considering item1, item2, weight=count and tried to plot it with matplotlib for different s_values = [10, 50, 100]




**3. Find the communities for the networks obtained in 1. and 2. Comment about the results with respect to the different thresholds and methods. How does it compare with the same analysis using 1-NN done in previous tasks?**

For part 1,

number of communities: 9 Components: [{'./project1/208.txt', './project1/927.txt', './project1/767.txt', './project1/626.txt', './project1/299.txt', './project1/162.txt', './project1/422.txt', './project1/113.txt', './project1/823.txt', './project1/11.txt', './project1/548.txt', './project1/585.txt', './project1/186.txt', './project1/614.txt', './project1/344.txt', './project1/728.txt', './project1/512.txt', './project1/792.txt', './project1/492.txt', './project1/453.txt', './project1/579.txt', './project1/53.txt', './project1/811.txt', './project1/769.txt', './project1/4.txt', './project1/270.txt', './project1/479.txt', './project1/908.txt', './project1/188.txt', './project1/827.txt', './project1/483.txt', './project1/427.txt', './project1/146.txt', './project1/32.txt', './project1/955.txt', './project1/947.txt', './project1/741.txt', './project1/443.txt', './project1/992.txt', './project1/597.txt', './project1/930.txt'}, {'./project1/643.txt', './project1/452.txt'}, {'./project1/441.txt', './project1/172.txt'}, {'./project1/630.txt', './project1/165.txt'}, {'./project1/309.txt', './project1/472.txt'}, {'./project1/331.txt', './project1/803.txt'}, {'./project1/341.txt', './project1/237.txt'}, {'./project1/410.txt', './project1/409.txt'}, {'./project1/431.txt', './project1/310.txt'}]

For part 2.

Communities for network with threshold s = 10

Communities for k = 2:

Community 1 : frozenset({'Bouamor,Houda,Pino,Juan,Bali,Kalika', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue'})

---

Communities for k = 3:

Community 1 : frozenset({'Bouamor,Houda,Pino,Juan,Bali,Kalika', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue'})

---

Communities for k = 4:

Community 1 : frozenset({'Muresan,Smaranda,Nakov,Preslav,Villavicencio,Aline', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Bouamor,Houda,Pino,Juan,Bali,Kalika'})

---

Communities for network with threshold s = 50

Communities for k = 2:

Community 1 : frozenset({'Bouamor,Houda,Pino,Juan,Bali,Kalika', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue'})

---

Communities for k = 3:

Community 1 : frozenset({'Bouamor,Houda,Pino,Juan,Bali,Kalika', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue'})

---

Communities for k = 4:

Community 1 : frozenset({'Muresan,Smaranda,Nakov,Preslav,Villavicencio,Aline', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Bouamor,Houda,Pino,Juan,Bali,Kalika'})

---

Communities for network with threshold s = 100

Communities for k = 2:

Community 1 : frozenset({'Bouamor,Houda,Pino,Juan,Bali,Kalika',
'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue'})

---

Communities for k = 3:

Community 1 : frozenset({'Bouamor,Houda,Pino,Juan,Bali,Kalika',
'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki', 'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue'})

---

Communities for k = 4:

Community 1 : frozenset({'Muresan,Smaranda,Nakov,Preslav,Villavicencio,Aline',
'Goldberg,Yoav,Kozareva,Zornitsa,Zhang,Yue', 'Rogers,Anna,BoydGraber,Jordan,Okazaki,Naoaki',
'Bouamor,Houda,Pino,Juan,Bali,Kalika'})

Comparing with the same analysis using 1-NN done in previous tasks:

- In the previous tasks using 1-NN (1-Nearest Neighbour), communities were identified based on the similarity between data points (in this case, files). Each data point was assigned to the community of its nearest neighbour.
- The communities identified in the previous tasks might differ from those obtained using the methods described above. The differences could arise due to the different algorithms and criteria used for community detection.
- For example, the threshold-based approach considers the similarity between data points, while the k-means algorithm partitions the data into clusters based on their features without considering the notion of similarity directly.
- Additionally, the number and composition of communities may vary depending on the parameters chosen for each method, such as the threshold value or the number of clusters (k).

**4. Use a ready-made software, library or service (eg. gephi) for visualizing one the network in 1. or 2.**

I described in parts 1, 2.