Dask Distributed Analysis of Covid-19 Papers Management and Analysis of Physics dataset (Module B)

Nguyen Xuan Tung
Feb 2022

Università degli Studi di Padova

Overview

1. Introduction
   - Idea of the project
   - Theoretical background

2. Implement
   - Data collection
   - Word counter distributed algorithm
   - The worst and best represented countries universities in the research
   - Get the embedding for the title of the papers
   - Compute the cosine similarity between papers

Summary

With the spread of COVID-19 over last 2 years, a lot of researches are made together with the increase of public paper about the pandemic. In this project, our work is to analysis 1000 papers about COVID-19, SARSCoV-2, and related corona viruses. The dataset can be found in the link: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge.

Introduction: Idea of the project

- The project is divided into $3 + 1$ bonus parts:
- $\longrightarrow$ Word counter distributed algorithm
- $\longrightarrow$ Find the worst and best represented countries in the research
- $\longrightarrow$ Get the embedding for the title of the papers
- $\longrightarrow$ Compute the cosine similarity between papers (bonus point)

What is Dask

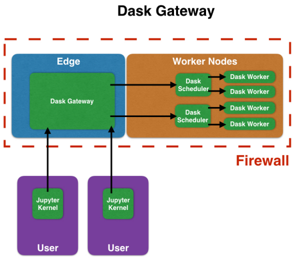
- Dask is a flexible library for parallel computing in Python.
- Dask is composed of two parts:
  $\longrightarrow$ 1 Dynamic task scheduling optimized for computation.
  $\longrightarrow$ 2 “Big Data” collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments.

What is Dask

- Dask uses existing Python APIs and data structures making itself very familiar to Python users.
- Dask allows to parallelize the code on a single machine but also to distribute it to a cluster of hundreds of machines.
- Dask has a variety of use cases and can be run with a single node and scale to thousand node clusters.

Dask Cluster

- ***Client***: is the machine from which a user has submitted a job or has distributed a program.
- ***Scheduler***: is the head/master node of the cluster and decides how to schedule the execution of the processes on the pool of resources.
- ***Worker***: represents a computing node of a cluster that receives the task that should be executed. A pool of resources is a set of workers.

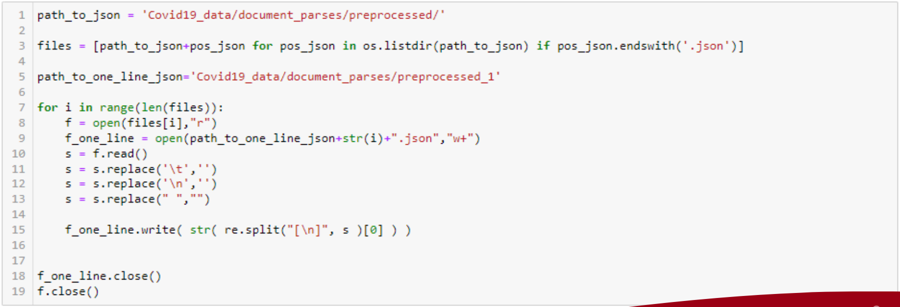


Method: Data collection

- The main dataset can be downloaded in the link: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
- The fasttext model can be downloaded in the link: https://dl.fbaipublicfiles.com/fasttext/vectorswiki/wiki.en.vec

Dataset analysis

- The dataset contain over 1000 papers in json format which is ranndomly chosen from the link.
- A new directory is created, with the content of the original .json files in one single line. Then the data so formatted are loaded.



```python
path_to_json = 'Covid19_data/document_parses/preprocessed/'

files = [path_to_json+pos_json for pos_json in os.listdir(path_to_json) if pos_json.endswith('.json')]

path_to_one_line_json='Covid19_data/document_parses/preprocessed_1'

for i in range(len(files)):
    f = open(files[i],"r")
    f_one_line = open(path_to_one_line_json+str(i)+".json","w+")
    s = f.read()
    s = s.replace('\t','')
    s = s.replace('\n','')
    s = s.replace(" ","")

    f_one_line.write( str( re.split("[\n]", s )[0] ) )


f_one_line.close()
f.close()
```

Implement of word counter distributed algorithm

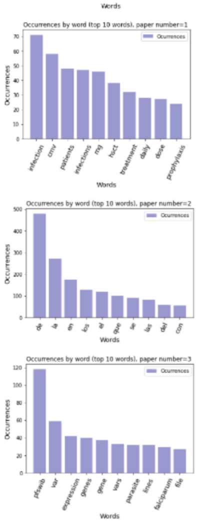
- The resulting items in the bag are strings. We convert each string (that encodes a json) in a dictionary thanks to the function json.loads.
- At this point, dask creates one partition per file by default, in this way indeed, we avoid to group all the files together and then split them in partitions. This operation would require a lot of time and resources, so it is better to read each file as a single partition and to repartition them later.
- We have more than one text for each paper, so that we need to create function *merge* to merge them.

Implement of word counter distributed algorithm

- Once we plucked the body_text field, we decrease the number of partitions in order to save computation time.
- $\longrightarrow$ no need of 1000 partitions since now the dimension of the files is reduced.
- The text of the documents must be sanitized (cit.) before performing analysis, so we create a stop-words function. After that we create a function that performs this cleaning procedure and we map it to the bag that contains the texts of all the papers. The function will:
  - delete special characters;
  - delete numbers;
  - delete single characters;
  - delete stop-words;
  - set all lowercase;
  - split text in strings.

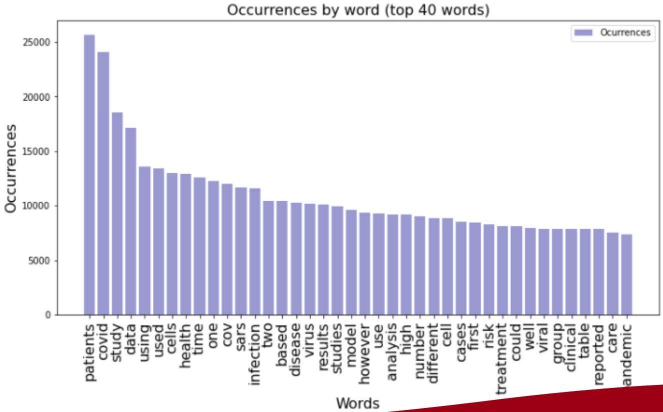
Map phase

- The time need to computed the map-phase: CPU times: user 15.6 s, sys: 1.44 s, total: 17 s — Wall time: 1min 22s

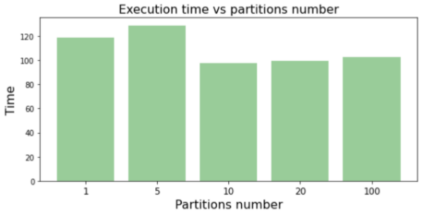


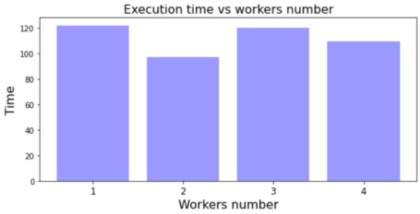
Reduce phase

- The time need to computed the reduce-phase: CPU times: CPU times: user 15.3 s, sys: 1.28 s, total: 16.5 s — Wall time: 1min 20s

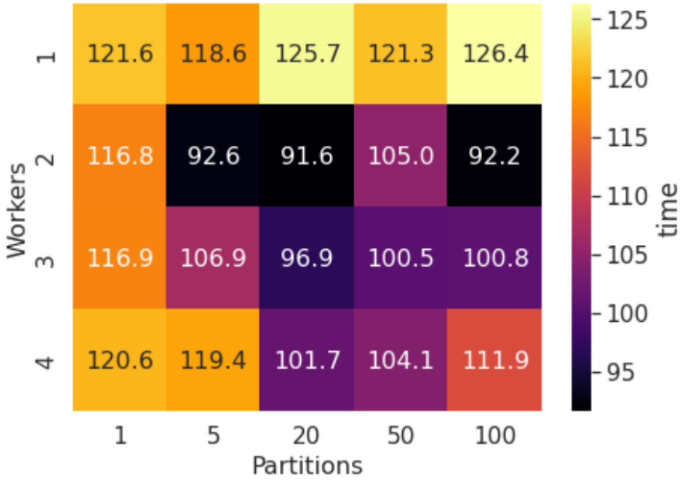


Execution time vs workers and partitions





Grid search



Workers
1
2
3
4
121.6
118.6
125.7
121.3
126.4
116.8
92.6
91.6
105.0
92.2
116.9
106.9
96.9
100.5
100.8
120.6
119.4
101.7
104.1
111.9
1
5
20
50
100
Partitions
125
120
115
110
105
100
95
time

The worst and best represented countries universities in the research

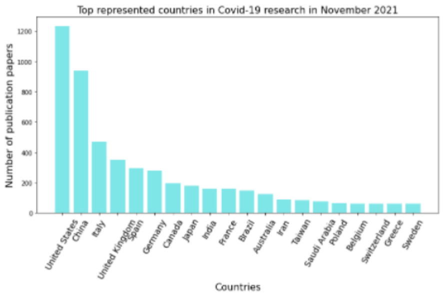
- In this task, we will find which are the worst and best represented countries and universities in the research.
- We load the data in a dask.bag in the very same way as the previous point, but now we pluck the authors field, since it is the one that contains the country and the university the authors belong to.
- The data have to be sanitized: there are cases in which the same country is called in different ways or it is repeated or is mispelled (e.g. UK, U.K., U. K., United Kingdom) or the authors are not all of the same country. Moreover, in some cases the fields are missing or they are empty. We define a function to solve all of those problems and we map it to the bag in which we loaded the data.

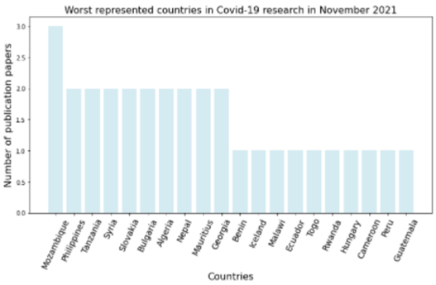
The worst and best represented countries universities in the research

UNIVERSITÀ DEGLI STUDI DI PADOVA

- Once we have pre-processed the data, we can load them in a dataframe structure, that is more performant with structured data with respect to dask bags.
- We select from the dataframe only the column related to the countries, then we use the value_counts() function to compute the occurrence of each of them.

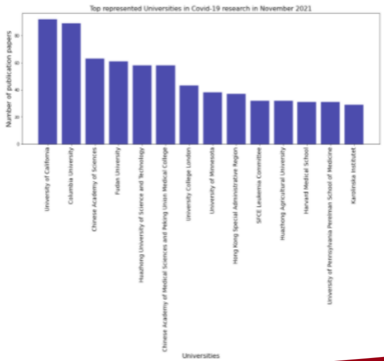
The worst and best represented countries in the research

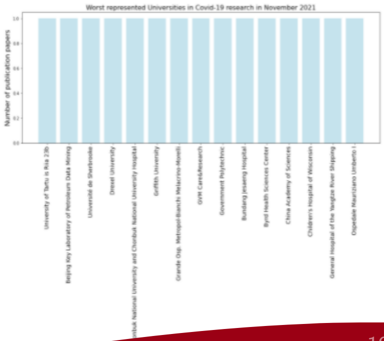




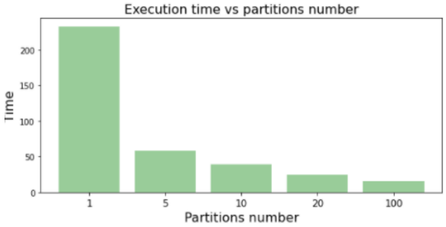
The worst and best represented universities in the research

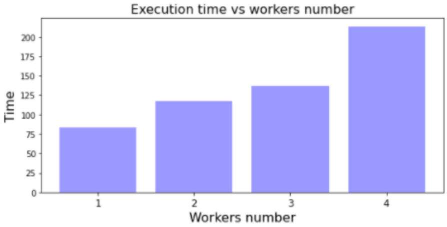
- We select from the dataframe only the column related to the universities, then we use the value_counts() pandas function to compute the occurrence of each of them.



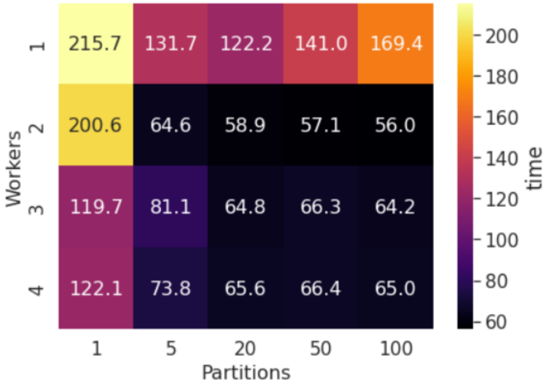


Execution time vs workers and partitions





Grid search



Workers \ Partitions	1	5	20	50	100
1	215.7	131.7	122.2	141.0	169.4
2	200.6	64.6	58.9	57.1	56.0
3	119.7	81.1	64.8	66.3	64.2
4	122.1	73.8	65.6	66.4	65.0

Timing of partitions and workers

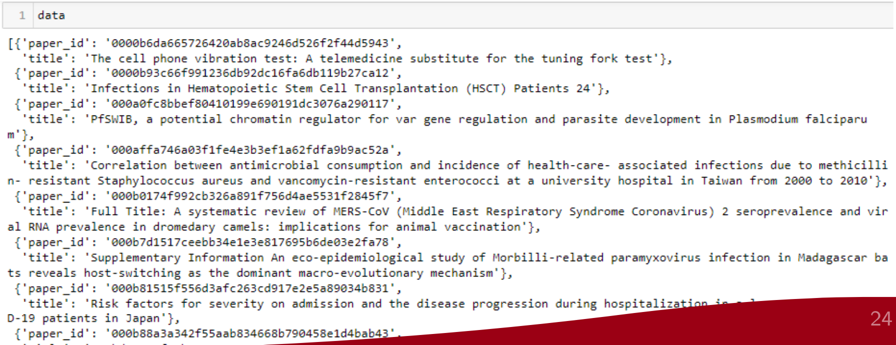
- Partition: their size should fit comfortably in memory but also not be too many, since every operation on every partition takes the central scheduler a few hundred microseconds to process. Moreover, having too many partitions means that a lot of chunks of data has to be transmitted through the cluster and this cause a lot of overhead.
- Worker: having more than one worker is good to distribute the code, but too many of them can cause overhead too, since the workers have to exchange data between each other and this operation takes a lot of time if the workers are too many.
- In conclusion, both the workers and the partitions should not be neither too many nor too few.

Embedding for the title of the papers

- In nature language processing (NLP) a common technique to perform analysis over a set of texts is to transform the text to a set of vectors, each one representing a word inside the document.
- In this task, we will:
  $\longrightarrow$ Load the data plucking only the metadata field of each paper, since it contains the title.
  $\longrightarrow$ Load the english fasttext model.
  $\longrightarrow$ create one dictionary per paper, each one with its title and the embedding for it.
  $\longrightarrow$ save each dictionary in a .json file.

Embedding for the title of the papers

- In order to create a BAG composed by:
  · paper-id
  · title-embedding
  $\longrightarrow$ we create the function flatten(record) which will return the paper_id and title of all the articles and make use of the map function provided by Dask Bag. We also apply a filter in order to remove all the empty title.



```
data
```

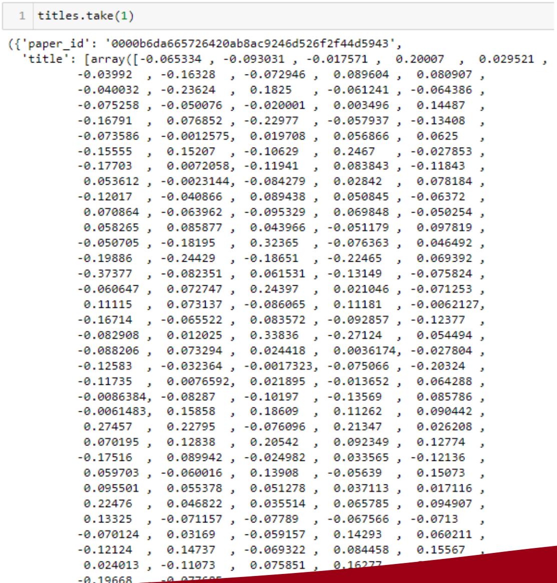
```
[{'paper_id': '0000b6da665726420ab8ac9246d526f2f44d5943',
  'title': 'The cell phone vibration test: A telemedicine substitute for the tuning fork test'},
 {'paper_id': '0000b93c66f991236db92dc16fa6db119b27ca12',
  'title': 'Infections in Hematopoietic Stem Cell Transplantation (HSCT) Patients 24'},
 {'paper_id': '000a0fc8bbef80410199e690191dc3076a290117',
  'title': 'PfSWIB, a potential chromatin regulator for var gene regulation and parasite development in Plasmodium falciparu
m'},
 {'paper_id': '000affa746a03f1fe4e3b3ef1a62fdfa9b9ac52a',
  'title': 'Correlation between antimicrobial consumption and incidence of health-care- associated infections due to methicilli
n- resistant Staphylococcus aureus and vancomycin-resistant enterococci at a university hospital in Taiwan from 2000 to 2010'},
 {'paper_id': '000b0174f992cb326a891f756d4ae5531f2845f7',
  'title': 'Full Title: A systematic review of MERS-CoV (Middle East Respiratory Syndrome Coronavirus) 2 seroprevalence and vir
al RNA prevalence in dromedary camels: implications for animal vaccination'},
 {'paper_id': '000b7d1517ceebb34e1e3e817695b6de03e2fa78',
  'title': 'Supplementary Information An eco-epidemiological study of Morbilli-related paramyxovirus infection in Madagascar ba
ts reveals host-switching as the dominant macro-evolutionary mechanism'},
 {'paper_id': '000b81515f556d3afc263cd917e2e5a89034b831',
  'title': 'Risk factors for severity on admission and the disease progression during hospitalization
D-19 patients in Japan'},
 {'paper_id': '000b88a3a342f55aab834668b790458e1d4bab43',
```

Embedding for the title of the papers

- Now we make use of the pre-trained model and create two functions:
  $\longrightarrow$ 1 create_dict(dict, model): which creates a dictionary consist of the paper_id and the embedding title.
  $\longrightarrow$ 2 embedded(text, wiki): which takes the text of the title as input, lower case it (in the sense of converting all capital letter to small letter) and split it. Then apply the pre-trained model and convert each word into an np.array().

Get the embedding for the title of the papers

- Finally, we get the result

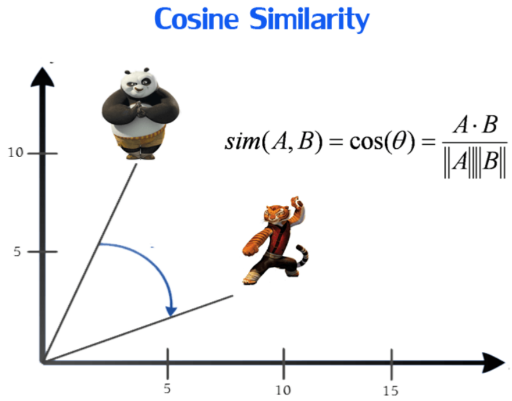


```
titles.take(1)
```

```
({'paper_id': '0000b6da665726420ab8ac9246d526f2f44d5943',
  'title': [array([-0.065334 , -0.093031 , -0.017571 ,  0.20007  ,  0.029521 ,
        -0.03992  , -0.16328  , -0.072946 ,  0.089604 ,  0.080907 ,
        -0.040032 , -0.23624  ,  0.1825   , -0.061241 , -0.064386 ,
        -0.075258 , -0.050076 , -0.020001 ,  0.003496 ,  0.14487  ,
        -0.16791  ,  0.076852 , -0.22977  , -0.057937 , -0.13408  ,
        -0.073586 , -0.0012575,  0.019708 ,  0.056866 ,  0.0625   ,
        -0.15555  ,  0.15207  , -0.10629  ,  0.2467   , -0.027853 ,
        -0.17703  ,  0.0072058, -0.11941  ,  0.083843 , -0.11843  ,
         0.053612 , -0.0023144, -0.084279 ,  0.02842  ,  0.078184 ,
        -0.12017  , -0.040866 ,  0.089438 ,  0.050845 , -0.06372  ,
         0.070864 , -0.063962 , -0.095329 ,  0.069848 , -0.050254 ,
         0.058265 ,  0.085877 ,  0.043966 , -0.051179 ,  0.097819 ,
        -0.050705 , -0.18195  ,  0.32365  , -0.076363 ,  0.046492 ,
        -0.19886  , -0.24429  , -0.18651  , -0.22465  ,  0.069392 ,
        -0.37377  , -0.082351 ,  0.061531 , -0.13149  , -0.075824 ,
        -0.060647 ,  0.072747 ,  0.24397  ,  0.021046 , -0.071253 ,
         0.11115  ,  0.073137 , -0.086065 ,  0.11181  , -0.0062127,
        -0.16714  , -0.065522 ,  0.083572 , -0.092857 , -0.12377  ,
        -0.082908 ,  0.012025 ,  0.33836  , -0.27124  ,  0.054494 ,
        -0.088206 ,  0.073294 ,  0.024418 ,  0.0036174, -0.027804 ,
        -0.12583  , -0.032364 , -0.0017323, -0.075066 , -0.20324  ,
        -0.11735  ,  0.0076592,  0.021895 , -0.013652 ,  0.064288 ,
        -0.0086384, -0.08287  , -0.10197  , -0.13569  ,  0.085786 ,
        -0.0061483,  0.15858  ,  0.18609  ,  0.11262  ,  0.090442 ,
         0.27457  ,  0.22795  , -0.076096 ,  0.21347  ,  0.026208 ,
         0.070195 ,  0.12838  ,  0.20542  ,  0.092349 ,  0.12774  ,
        -0.17516  ,  0.089942 , -0.024982 ,  0.033565 , -0.12136  ,
         0.059703 , -0.060016 ,  0.13908  , -0.05639  ,  0.15073  ,
         0.095501 ,  0.055378 ,  0.051278 ,  0.037113 ,  0.017116 ,
         0.22476  ,  0.046822 ,  0.035514 ,  0.065785 ,  0.094907 ,
         0.13325  , -0.071157 , -0.07789  , -0.067566 , -0.0713   ,
        -0.070124 ,  0.03169  , -0.059157 ,  0.14293  ,  0.060211 ,
        -0.12124  ,  0.14737  , -0.069322 ,  0.084458 ,  0.15567  ,
         0.024013 , -0.11073  ,  0.075851 ,  0.16277
        -0.19668  , -0.0776
```

Cosine similarity

- We will now make used of the previously generated vectors to compute the cosine similarity between each paper and to figure out a couple of papers with the highest cosine similarity score.

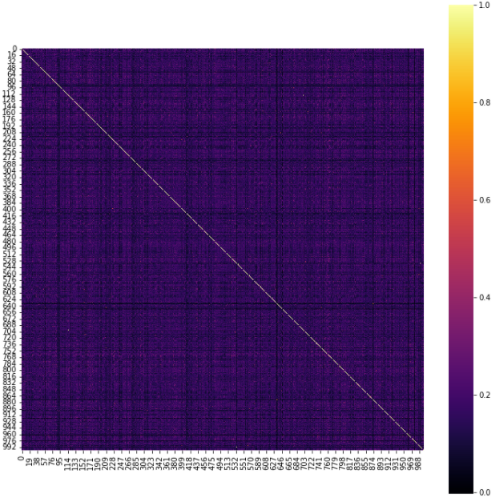


Cosine similarity

- cosine_similarity function which basically computes the similarity between the texts as shown in the image.
- We compute the similarity word by word among the texts. When one of the text end, there's no more comparision.
- The function similarity computes the similarity among two different sentence and return a dictionary. Finally we construct a similarity matrix.

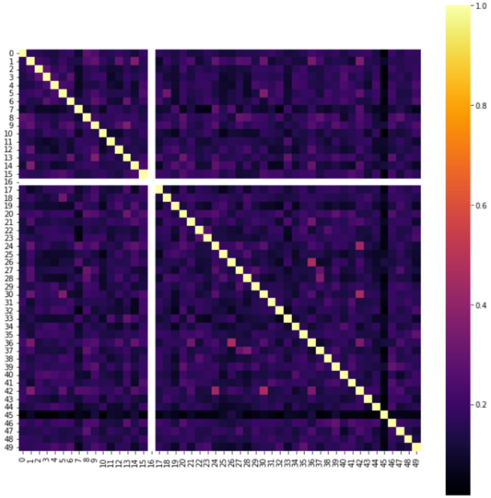
Cosine similarity

- We plot the heat map of first 50 papers and see the similarity between them.



Cosine similarity

- We plot the heat map of first 50 papers and see the similarity between them.



Thank you