

Analysis of COVID 19 papers

Group 4:

Noemi Manara (2022909)

Giuseppe Simionato (2029013)

Nicholas Sinigaglia (2029059)

Francesco Zambelli (2029014)

The project

Database of papers about COVID 19 → not only title and body text, also other metadata

Goal: pre process and analyze the dataset in NLP fashion,

extract summary variables:

- most frequent words
- top contributing countries and institutions
- most similar titles (through embedding)



The problem

Dataset

A big collection of articles in json format (>20 GB)

- 1) Need a subset
- 2) Formatting issues

Cluster

Local Area Network
Cluster with our own resources

Analysis

Preprocess the files and extract summary information from them

The approach

Dataset

A big collection of articles in json format (>20 GB)

- 1) Need a subset
- 2) Formatting issues

→ **bash script**

Cluster

Local Area Network
Cluster with our own resources

→ **Network File System**

Analysis

Preprocess the files and extract summary information from them

→ **dask**

The workflow

Pre processing

Clean the text

Find the most frequent words

Find the most active countries and institutions

Title embedding

Load a dictionary

translate each title word as a 300-dim vector

Cosine similarity

Find most similar titles

average the embedding of each word of the title



Dataset

Dataset preparation

The dataset available on kaggle is a 4 GB zipped folder that unzipped weights **20.6 GB** and contains more than **229k .json files**

1) Sub-sampling

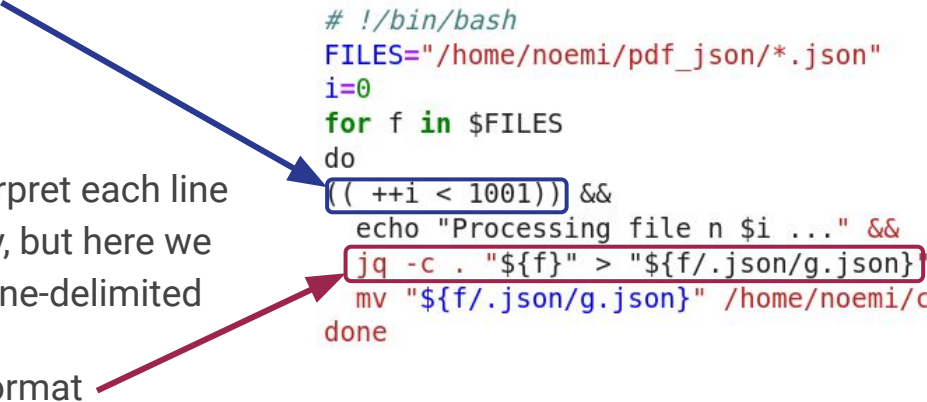
only 1000 samples needed

2) Format

dask reading functions interpret each line of a .json file as a new entry, but here we have one entry per file not line-delimited

→ recast in *line-delimited* format

```
# !/bin/bash
FILES="/home/noemi/pdf_json/*.json"
i=0
for f in $FILES
do
  (( ++i < 1001 )) &&
  echo "Processing file n $i ..." &&
  jq -c . "${f}" > "${f/.json/g.json}" &&
  mv "${f/.json/g.json}" /home/noemi/covid_distributed/
done
```



Network File System

NFS setup

Valid protocol for all task reading functions, exploits our Local Area Network setup

Server

- package: `nfs-kernel-server`
- has locally the directory with dataset:
`/home/noemi/covid_distributed`
- directory shared through setting permissions in `/etc/exports`

```
12 /home/noemi/covid_distributed (*) (ro,async,no_subtree_check)
```

*: any IP address of the Network

ro: Read-Only

async: changes not committed to storage before requests (no risks: ro)

no_subtree_check: do not check permissions of mounted directory

Client

- package: `nfs-commons`
- directory mounted in common path

```
(base) noemi@hal-2001:~$ sudo mount 192.168.1.125:/home/noemi/covid_distributed  
/mnt/nfs/covid_distributed
```


- proper paths on notebook, e.g.
`file:///.../mnt/nfs/covid_distributed/`

Cluster setup

Through command lines on
each laptop

- scheduler dask-scheduler
- workers dask-worker
tcp://192.168.1.125:8786
--nprocs 2 --nthreads 4
--memory-limit="4GiB"
- client

```
[3]: client = Client("tcp://192.168.1.125:8786")
      client
```

[3]:  **Client**
Client-f7710621-e2ed-11eb-9d91-9d6f9ad79f13
Connection method: Direct
Dashboard: <http://192.168.1.125:8787/status>
► Scheduler Info



Loading data

Dask bag

- Compute file names list locally (easy)

```
path_to_json = '../mnt/nfs/covid_distributed/'
json_files = [pos_json for pos_json in os.listdir(path_to_json) if pos_json.endswith('.json')]
print('Example of filename:', json_files[3])
```

Example of filename: 0098190ff26a17c7046fccba55196195fac5281g.json

```
n_files=len(json_files)
print('Number of files:', n_files)
```

Number of files: 1000

- Load from shared folder all 1000 files through NFS protocol
- Can access data in json format

```
b = db.read_text(path_to_json + json_files[3]).map(json.loads)
b.take(1)[0]["paper_id"]
```

'0098190ff26a17c7046fccba55196195fac5281'

- Can change **npartitions**

Option 1: single bag

List of file paths with the right extension

```
[7]: l = []
    for j in range(0,n_files,1):
        l += [path_to_json + json_files[j]]

%time b = db.read_text(l).map(json.loads)
b

CPU times: user 77.3 ms, sys: 20.8 ms, total: 98.1 ms
Wall time: 97.8 ms
[7]: dask.bag<loads, npartitions=1000>
```

Dask can build a single dask bag loading a list of file paths

- best performances, fast execution
- memory is automatically managed

Dask bag

- Compute file names list locally (easy)

```
path_to_json = '../mnt/nfs/covid_distributed/'
json_files = [pos_json for pos_json in os.listdir(path_to_json) if pos_json.endswith('.json')]
print('Example of filename:', json_files[3])
```

Example of filename: 0098190ff26a17c7046fccba55196195fac5281g.json

```
n_files=len(json_files)
print('Number of files:', n_files)
```

Number of files: 1000

- Load from shared folder all 1000 files through NFS protocol
- Can access data in json format

```
[7]: b = db.read_text(path_to_json + json_files[3]).map(json.loads)
     b.take(1)[0]["paper_id"]
```

```
[7]: '0098190ff26a17c7046fccba55196195fac5281'
```

- Can change **npartitions**

Option 2: multiple bags

Create one bag per file, append them together, compute from_delayed

```
%%time
futures = [] # this variable will not be computed now (lazy computation)

for j in range(0,n_files,1):
    bb = db.read_text(path_to_json + json_files[j]).map(json.loads)
    futures.append(client.scatter(bb)) # scatter moves data from the local client
```

CPU times: user 2.78 s, sys: 279 ms, total: 3.06 s
Wall time: 15.5 s

```
%%time b2 = db.from_delayed(futures) # creates a dask bag
```

CPU times: user 62.8 ms, sys: 8.31 ms, total: 71.1 ms
Wall time: 70 ms

→ worse performances, slow execution
→ memory is explicitly managed through `.scatter()`



Pre processing

Pre processing: introduction

Before going through the analysis we must clean up our data since we are working with texts.

In the pre processing we:

- remove punctuation
- remove numeric characters
- set all letters to lowercase
- remove “stop words”

```
# Loading stopwords list
stop_words = stopwords.words('english')
more_stopwords = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
                  'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'u', 'im', 'c', 'et', 'al']
stop_words = stop_words + more_stopwords

# Function which returns True if a word is not a stopword
def isvalidword(x, stop_words = stop_words):
    return not x in stop_words

# Function which returns True if x is made only of letters
def isalpha(x):
    return x.isalpha()
```

Pre processing: code

The following function pre processes the dask bag putting together every text of each article and returns a bag which contains every word of the obtained “big text”.

```
def pre_process(words, bag=True):
    out = None

    if(bag):
        out = ( words
                # In general every article has several "body text" (different sections)
                .pluck('body_text')
                # De-nest, creating one big element
                .flatten()
                # Taking the text of the element
                .pluck('text') )
    else:
        # Creating a Dask bag from the input string
        out = db.from_sequence(words)

    return (out
            # Dividing our text in "sub-strings" related to words.
            # Spaces are used as separator, words are made only by alphanumeric characters or only by punctuation.
            .map(WordPunctTokenizer().tokenize)
            .flatten()
            # All characters to lower, must be done before filtering!
            .map(lambda x: x.lower())
            # Filtering stop words and numeric character
            .filter(isvalidword)
            .filter(isalpha) )
```


Pre processing: example

The **COVID-19 pandemic in Italy** is part of the pandemic of coronavirus disease 2019 (COVID-19) caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The virus was first confirmed to have spread to Italy on 31 January 2020, when two Chinese tourists in Rome tested positive for the virus.^[1]

```
bag_of_words = pre_process(["The COVID-19 pandemic in Italy is part of the pandemic of coronavirus disease 2019 (COV",  
np.array( bag_of_words.take(100) )
```

```
array(['covid', 'pandemic', 'italy', 'part', 'pandemic', 'coronavirus',  
      'disease', 'covid', 'caused', 'severe', 'acute', 'respiratory',  
      'syndrome', 'coronavirus', 'sars', 'cov', 'virus', 'first',  
      'confirmed', 'spread', 'italy', 'january', 'two', 'chinese',  
      'tourists', 'rome', 'tested', 'positive', 'virus'], dtype='<U11')
```

To obtain the array of words we call .take(n) method after pre_process(...) function.

Word counter algorithm

Word counter algorithm: introduction

Our task is to design a distributed algorithm to count the occurrences of all words inside our documents. This is a very common task in NLP (Natural Language Processing).

The algorithm is defined as follow:

- **map phase:** putting together all document texts in one big text D and produce a set of intermediate pairs $(word, 1)$ for each word $\in D$.
- **reduce phase:** for each word, gather all previous pairs $(word, 1)$ and return the final pair $(word, N(word))$ where $N(word)$ = number of occurrences of word in all documents.

Word counter algorithm: code

```
# For each word that appears this function returns a dictionary made of the word and its number of appearances
```

```
def binop(t,x):  
    return {"name":t["name"], "count":sum(da["count"] for da in (t,x))}
```

```
# Pre processing all articles and putting them together in one text
```

```
futures = client.submit(pre_process,b)  
pre_processed_b = client.gather(futures)
```

```
res = ( pre_processed_b  
        # Hadoop map  
        .map(lambda x: {"name":x,"count":1})  
        # Combined groupby w.r.t. "name"  
        # and reduction w.r.t. binop function  
        .foldby("name", binop)  
        # At this point we have something like: ( ( word, {'name': word , 'count' : number of counts} ) , ... )  
        # Taking the [1] element we obtain the Hadoop reduce result  
        .map(lambda x:x[1]) )
```

```
# Taking 12 biggest elements in collection
```

```
res_top = res.topk(12,key=lambda x: x["count"]).compute()
```

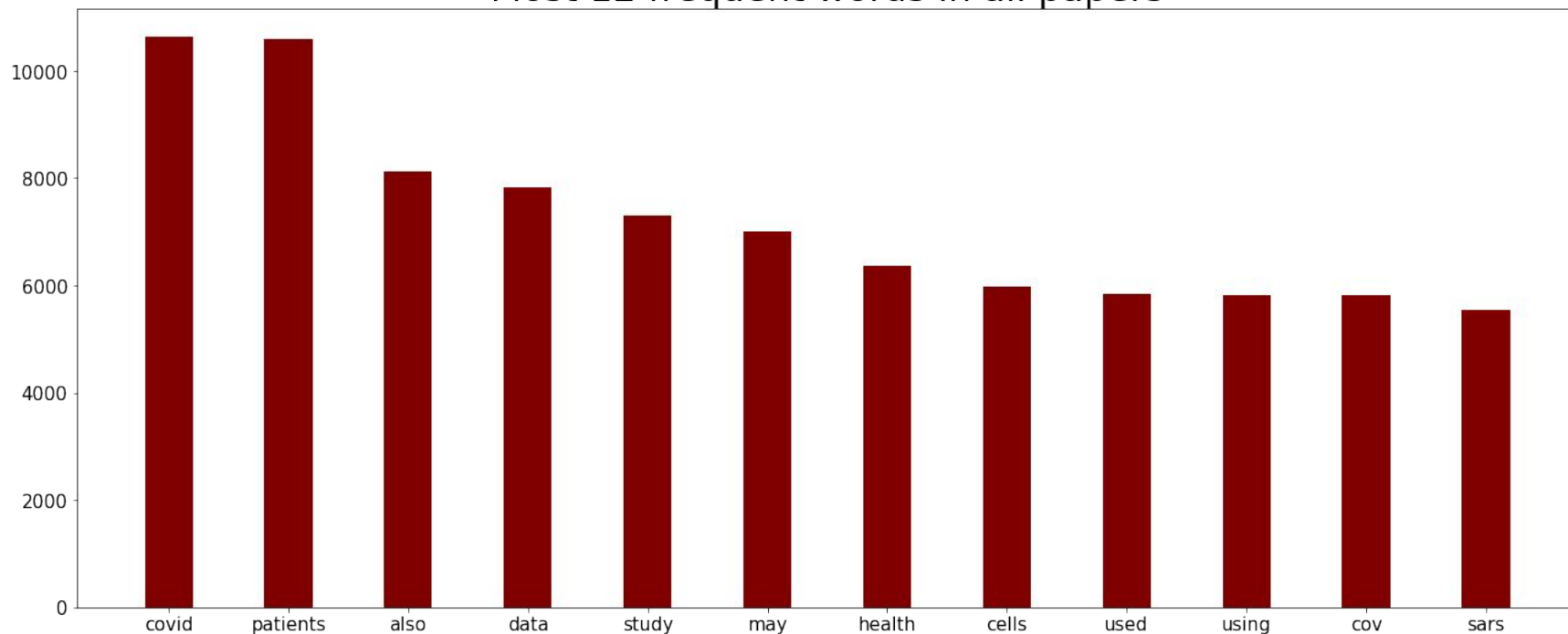
```
# To pandas dataframe
```

```
res_top_df = pd.DataFrame(res_top)
```

```
res_top_df
```

Word counter algorithm: results

Most 12 frequent words in all papers





Country and Institution contribution

Country contribution: dataframe conversion

Now our goal is to quantify the contribution of each country and institution in the research, in particular checking the country and institution of every paper author.

This task becomes extremely easy converting our `dask bag` of files in a `dask dataframe`.

```
# Function that replaces empty dictionary {} with empty string ""
# This is fundamental because for some authors 'institution' or 'country' is not known and they are = {}
def check(var, args):
    cc = var
    for i in range(len(args)):
        if args[i] in cc:
            cc = cc[args[i]]
            if cc=={}:
                return("")
        else:
            return("")
    # removing uppercase
    return(cc.lower())
```

Country contribution: dataframe conversion

```
# This function returns a list for every nested item, every author of each paper.  
# In this way we are creating a common structure for every author of each paper, in this way  
# we can build our Dataframe using the same structure.
```

```
def denormalize(record):  
    res = []  
    for author in record["metadata"]["authors"]:  
        res.append({'title': record['metadata']['title'],  
                    'first': author['first'],  
                    'middle': author['middle'],  
                    'last': author['last'],  
                    'suffix': author['suffix'],  
                    'laboratory': check(author, ['affiliation', 'laboratory']),  
                    'institution': check(author, ['affiliation', 'institution']),  
                    'settlement': check(author, ['affiliation', 'location', 'settlement']),  
                    'country': check(author, ['affiliation', 'location', 'country']),  
                    'email': author['email']  
                })  
    return res
```

```
# considering 99 files we have 99 partitions  
f = b.map(denormalize).flatten()  
f
```

```
dask.bag<flatten, npartitions=99>
```


Country contribution: dataframe conversion

```
# Dataframe structure is the same imposed by the denormalize() function.  
# f is divided in 1 partition per file. The dataframe has the same number of partitions!  
df = f.to_dataframe()  
df.head()
```

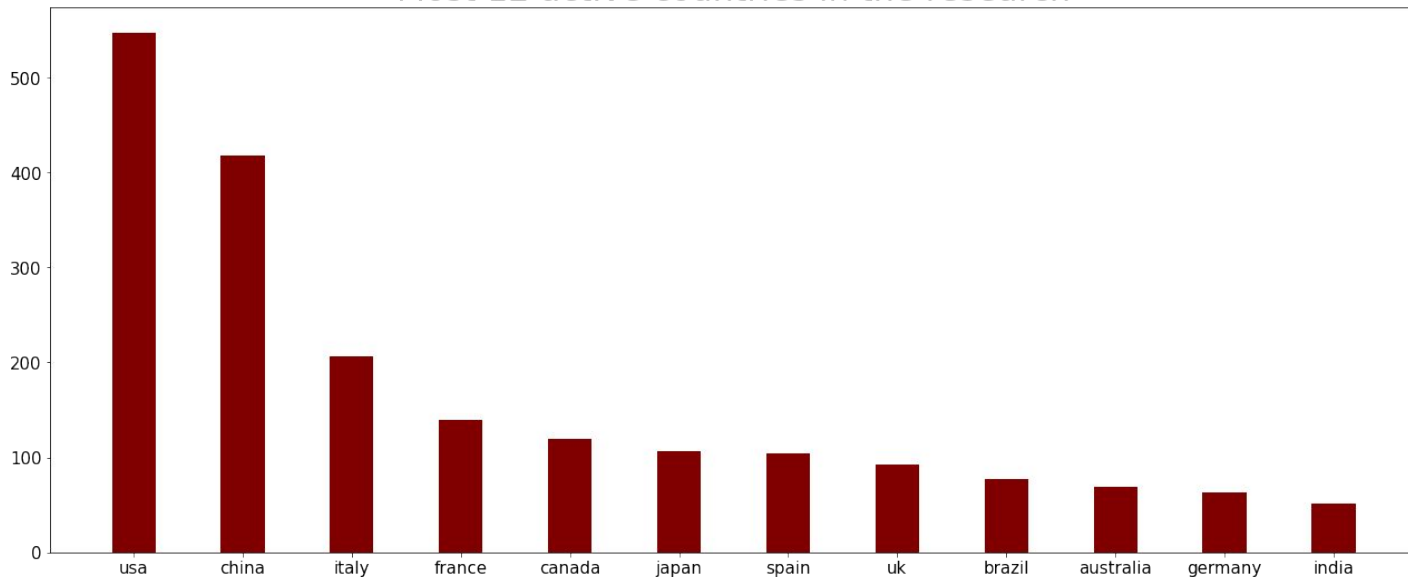
	title	first	middle	last	suffix	laboratory	institution	settlement	country	email
0	Dromedary camels in northern Mali have high se...	Darryl		Falzarano		laboratory of virology	rocky mountain laboratories	hamilton	usa	
1	Dromedary camels in northern Mali have high se...	Badian		Kamissoko		laboratoire central vétérinaire		bamako	mali	
2	Dromedary camels in northern Mali have high se...	Emmie		De Wit		laboratory of virology	rocky mountain laboratories	hamilton	usa	
3	Dromedary camels in northern Mali have high se...	Ousmane		Maïga			techniques and technologies of bamako	bamako		
4	Dromedary camels in northern Mali have high se...	Jacqueline		Cronin		laboratory of virology	rocky mountain laboratories	hamilton	usa	

Country contribution: results

```
# Considering only valid names
df = df[df.country.map(isalpha) == True]

# Counting country appearances, than picking the largest 20
m_series = ( df[df.country != '']
             .country
             .value_counts()
             .nlargest(20)
             .compute() )
```

Most 12 active countries in the research



Institution contribution: results

Most active institutions in the research:	
kemri-wellcome trust research programme	30
university of thessaly	27
huazhong university of science and technology	24
chinese academy of sciences	22
johns hopkins university	19
owkin, inc. new york	18
the university of hong kong	18
university of california	17
university of pittsburgh	16
kyoto university	16
china medical university	15
keio university school of medicine	15
pak emirates military hospital (pemh)	15
neurosurgery	15
emory university	15
ascension st. john hospital	14
institut pasteur	14
university of utah	13
hong kong special administrative region	13
national institute for infectious diseases 'l. spallanzani' irccs	13



Kaggle version

Kaggle version

In this version of the notebook thanks to the kaggle resources, we can load the embedding file entirely with the given **load_vectors()** function, without any change.

```
import io

def load_vectors(fname):
    fin = io.open(fname, 'r', encoding='utf-8', newline='\n', errors='ignore')
    n, d = map(int, fin.readline().split())
    data = {}
    for line in fin:
        tokens = line.rstrip().split(' ')
        data[tokens[0]] = list(map(float, tokens[1:]))
    return data

model = load_vectors('../input/wikinews300d1msubwordvec/wiki-news-300d-1M-subword.vec') # 2GB dataset
```

Kaggle version

In this version of the notebook thanks to the kaggle resources, we can load the embedding file entirely with the given **load_vectors()** function, without any change.



After extracting the list of titles from the bag previously created, the computation of the title-embedding of each paper is done using the map function on them.

```
# embedding of a list of words and also this do a preprocess on the sentence
```

```
def embedding(sentence, model=model):  
    emb = []  
    for s in sentence.split():  
        s = s.lower()  
        if (s.isalpha()) and (not s in stop_words) and (s in model):  
            emb.append(np.array(model[s]))  
  
    # if words are not found embedding is Empty  
    if(len(emb)==0):  
        return("Empty")  
  
    return sum(emb)
```

```
# Compute the title-embedding
```

```
titles = b.pluck('metadata').pluck('title')  
  
emb = list( map(embedding,titles) )
```



Title embedding

Word embedding

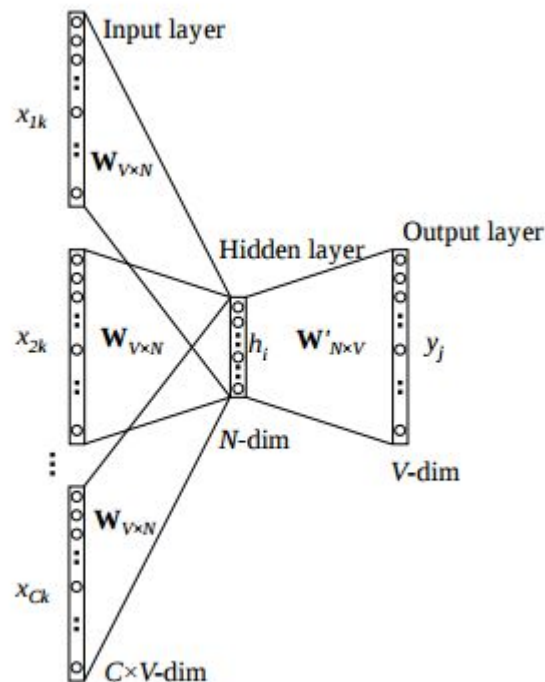
Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words and other features. In NLP the Features are the representation of a sequence of words or sentence in the numeric vector.

Example:

people



[0.0104, 0.1101, -0.0982, -0.0106, -0.0696, ..., -0.1039, -0.1243, -0.0287, -0.0954, 0.1598, -0.0329, 0.0798]



Word embedding in Dask

Import of the FastText file

```
1 %%time
2
3 def nsplit(x):
4     tit=x[0]
5     res=[float(ri) for ri in x[1:]]
6     return {"name":tit.lower(), "embed":res}
7
8 fname = "file:///../../mnt/nfs/covid_distributed/wiki-news-300d-1M.vec"
9 datas = db.read_text(fname, blocksize="10MiB")
10 model = (datas.map(lambda x: x.split())
11          .map(nsplit))
12 av_names = (model.pluck("name")
13             .compute())
14 av_names[100:110]
```

Word embedding in Dask

Import of the FastText file

```
1 %%time
2
3 def nsplit(x):
4     tit=x[0]
5     res=[float(ri) for ri in x[1:]]
6     return {"name":tit.lower(), "embed":res}
7
8 fname = "file:///.../mnt/nfs/covid_distributed/wiki-news-300d-1M.vec"
9 datas = db.read_text(fname, blocksizes="10MiB")
10 model = (datas.map(lambda x: x.split())
11         .map(nsplit))
12 av_names = (model.pluck("name")
13             .compute())
14 av_names[100:110]
```

1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (2GiB)

Indicates the memory size of each partition in which the bag is splitted

Each line of the text is divided in the word and the numeric embedding of it

Only the words within the file are computed in order to execute later some comparison faster

Word embedding in Dask

```
10 def embedding(sentence, model=model):
11     emb = []
12     for s in sentence.split():
13         s = s.lower()
14         if (s.isalpha()) and (not s in stop_words) and (s in av_names):
15             res=model.filter(lambda x: x["name"]==s).take(1)
16             if len(res)!=0:
17                 emb.append(np.array(res[0]["embed"]))
18     if(len(emb)==0):
19         return("Empty")
20     return sum(emb)
21
22 def denormalize2(record):
23     return {
24         'paper_id': record['paper_id'],
25         'title': record['metadata']['title'],
26         'title-embedding': embedding(record['metadata']['title'])
27     }
```

Word embedding in Dask

```
10 def embedding(sentence, model=model):
11     emb = []
12     for s in sentence.split():
13         s = s.lower()
14         if (s.isalpha()) and (not s in stop words) and (s in av names):
15             res=model.filter(lambda x: x["name"]==s).take(1)
16             if len(res)!=0:
17                 emb.append(np.array(res[0]["embed"]))
18     if(len(emb)==0):
19         return("Empty")
20     return sum(emb)
21
22 def denormalize2(record):
23     return {
24         'paper_id': record['paper_id'],
25         'title': record['metadata']['title'],
26         'title-embedding': embedding(record['metadata']['title'])
27     }
```

Check if the words allowed and if they are available in the FastText file

Only compute the embedding vector of the considered word

Some titles can contain no allowed words. In this case the function returns "Empty"

Summing the embedding vectors of all the words in the title in order to lighten the final result and to prepare it for the cosine similarity task

Word embedding in Dask

Then we converted the bag into a Dask dataframe with 3 columns: **paper_id**, **title**, **title-embedding**. For each column we provided the meta information which specifies the data type and allows to work with the data much faster because Dask doesn't have to infer the data type every time.

```
1 %%time
2
3 # I create a df with paper_id and title
4
5 f2 = (b.map(denormalize2)
6       .filter(lambda x: x["title-embedding"]!="Empty"))
7
8 rre=f2.to_dataframe(meta={'paper_id' : str,
9                           'title' : str,
10                          'title-embedding':object})
```

```
1 %%time
2
3 rre.head(5, npartitions=6)
4
```

The computation of the whole dataframe is too heavy, so we only take some elements to show how the table looks like:

paper_id	title	title-embedding
0004774b55eb0dad880aba9b572efe362660c5e0	Disaster Perceptions	[-0.0206, 0.0155, -0.0024, -0.0014, 0.0177, 0....
00bf93c7ce5dccbfc4f28c8075bffa1ad40d07	Nucleotide Sequence of Bovine Rotavirus Gene 1...	[-0.148900000000000003, -0.2459, -0.05080000000...
0098190ff26a17c7046fccba55196195fac5281	Winter School on sEMG Signal Processing: An In...	[-0.483399999999999994, 0.5336, 0.139099999999...
0043d044273b8eb1585d3a66061e9b4e03edc062	Evaluation of the tuberculosis programme in Ni...	[-0.319700000000000004, -0.22829999999999998, -...
00ef21603a690ae62a62c48bac9c847059c0455f	Research in Veterinary Science	[-0.15589999999999998, 0.2055, -0.0885, -0.112...

Cosine similarity

Cosine similarity

The **Cosine similarity** is a metric used to measure how similar two documents are irrespective of their size. Mathematically, it measures the **cosine of the angle** between two vectors projected in a multi-dimensional space.

$$\textit{cosine similarity}(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||}$$

Note: to compute the cosine similarity we have to know all the values of the numeric vectors but we are limited by our available resources. For this reason, we load only 100 papers in this section.

Cosine similarity with Dask

The **dask_distance** package provides a function **cosine()** which allow us to compute 1 - cosine similarity

```
1 %%time
2
3 # compute the cosine similarity using the dask_distance.cosine function
4 amress = dask_distance.cosine(marr,marr).compute()
```


Cosine similarity with Dask

The **dask_distance** package provides a function **cosine()** which allow us to compute $1 - \text{cosine similarity}$



We have modified the source code to obtain our desired quantity: but this function can be used only on **dask vectors**

```
1 %%time
2
3 marr = (clean_rre
4         .to_dataframe(meta=dict(zip(np.arange(300), [float]*300)))
5         .to_dask_array(lengths=curr_len)
6         .persist())
7
```

Cosine similarity with Dask

```
1 def title_check(sentence, model=model):
2     emb = []
3     for s in sentence.split():
4         s = s.lower()
5         if (s.isalpha()) and (not s in stop_words) and (s in av_names):
6             return 1
7     if len(emb)==0:
8         return(0)
```

We have modified the source code to obtain our desired quantity: but this function can be used only on **dask vectors**

```
1 %%time
2
3 marr = (clean_rre
4         .to_dataframe(meta=dict(zip(np.arange(300), [float]*300)))
5         .to_dask_array(lengths=curr_len)
6         .persist())
7
```

Highest cosine similarity

Couple 1 :

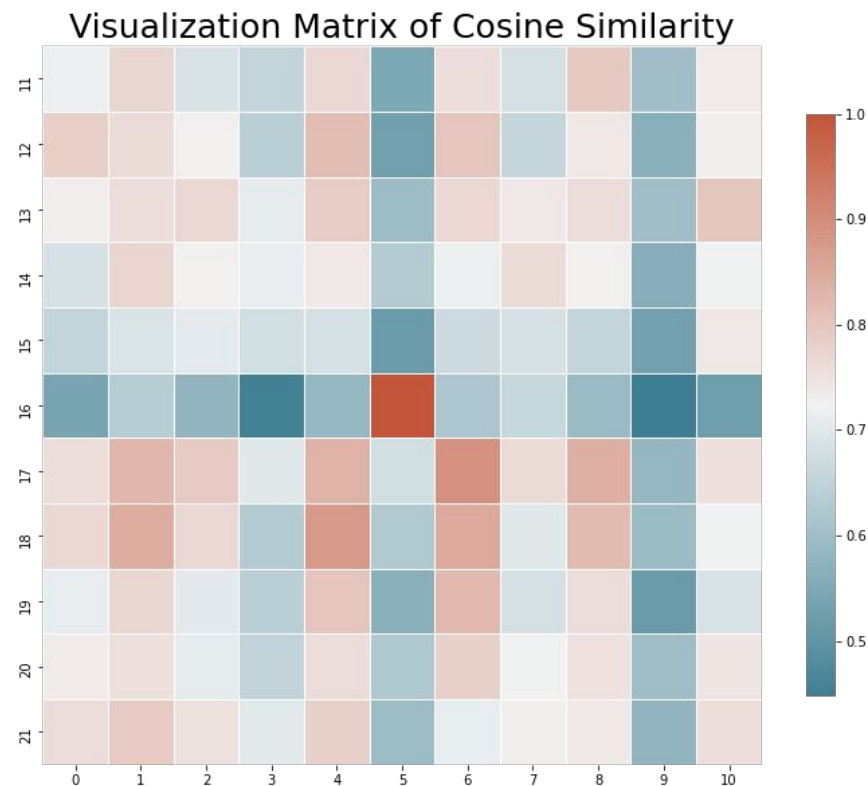
- Open Access
- Open Access

Couple 2 :

- An Official Learning Resource of AASLD review impact of COviD-19 response on Hepatitis Prevention Care and Treatment: results From Global survey of Providers and Program Managers
- 'Necessity is the mother of invention': Specialist palliative care service innovation and practice change in response to COVID-19. Results from a multinational survey (CovPall)

Highest cosine similarity

	Paper title legend
0	Symptoms of Anxiety and Depression in Relation...
1	Tracing and analysis of 288 early SARS-CoV-2 i...
2	The COVID-19 Pandemic in 2021: Avoiding Overdi...
3	SARS-CoV-2, bacterial co-infections, and AMR: ...
4	Validation of reported risk factors for diseas...
5	Open Access
6	The future of medical scribes documenting in t...
7	Leptin Deficiency, Caused by Malnutrition, Mak...
8	Supplementary Materials Insulator Based Dielec...
9	EGFR-specific single-chain variable fragment a...
10	Extracorporeal life support for immune reconst...
11	Physics of Fluids ARTICLE scitation.org/journa...
12	Cross-sectional survey on physician burnout du...
13	Veterinary Microbiology Identification of ente...
14	Coronaviruses Detected in Brazilian Wild Birds...
15	Human infections associated with wild birds
16	Open Access
17	Immunity credentials using self-sovereign iden...
18	A comparative analysis on risk communication b...
19	Overall Quality of Sporting Events and Emotion...
20	Arbidol combined with the Chinese medicine Lia...
21	Diffusion as a First Model of Spread of Viral ...



Some Dask considerations

N Workers impact

- It is important to have **more than one thread** per worker
- More **balanced-workers** lead to better performances
- It is better to have good amount of memory related to the task to avoid worker killing

	Word Counter [s]	Country and Institutions [s]	Title Embedding [s]	Bonus Point [m]	Total time [s]	Nr Papers
3 workers ([4 thre. & 4 GiB RAM] 2, [2 thre. & 8 GiB RAM] 1)	32.3	70.6	18.8	5.25	437	50
4 workers ([8 thre. & 8 GiB RAM] 1, [4 thre. & 8 GiB RAM] 2, [2 thre. & 8 GiB RAM] 1)	32.2	40.22	26.8	7.72	562	50
4 workers ([8 thre. & 8 GiB RAM] 2, [8 thre. & 16 GiB RAM] 1, [4 thre. & 6 GiB RAM] 1)	32.6	30.5	24.03	10.53	700	100
7 workers ([4 thre. & 4 GiB RAM] 7)	34.3	18.3	19.7	4.3	330	100
8 workers ([4 thre. & 4 GiB RAM] 2, [2 thre. & 4 GiB RAM] 4, [1 thre. & 4 GiB RAM] 2)	24.7	24.05	8.423	8.05	540	100
9 workers ([2 thre. & 4 GiB RAM] 9)	40.3	18.76	18.6	16.42	1063	100
Kaggle:	-	-	-	-	-	-
4 workers ([1 thre. & 4 GiB RAM] 4)	25.1	34.5	4	0.21	77	1000

N Partitions impact

Comparing these results we can see some trends:

- Word Counter time is more or less constant;
- Country and Institution time decreases until the number of partitions is equal to the number of files, then increases again;
- Title Embedding time increases if the number of partitions decreases.

	Word Counter [s]	Country and Institutions [s]	Title Embedding [s]	Total time [s]	Nr Partitions
7 workers ([4 thre. & 4 GiB RAM] 7)	26.2	51.3	>300	>377	7
7 workers ([4 thre. & 4 GiB RAM] 7)	28.0	56.9	162	247	100
7 workers ([4 thre. & 4 GiB RAM] 7)	30.0	62.9	51.0	144	250
7 workers ([4 thre. & 4 GiB RAM] 7)	35.8	26.16	22.8	85	500
7 workers ([4 thre. & 4 GiB RAM] 7)	42.9	27.4	18.2	89	750
7 workers ([4 thre. & 4 GiB RAM] 7)	35.2	18.4	15.8	70	1000
7 workers ([4 thre. & 4 GiB RAM] 7)	39.1	38.61	16.59	94	2000
7 workers ([4 thre. & 4 GiB RAM] 7)	86.0	132.5	22.3	241	4000

N Partitions impact

Let's try to understand these trends.

N Partitions impact

Let's try to understand these trends.

In **Word Counter section**, we manage a bag and it seems it is not dependent on the number of partitions. Instead we can see that in the other two sections, where we use a dataframe, the partitions affects much more the performances.

N Partitions impact

Let's try to understand these trends.

In **Word Counter section**, we manage a bag and it seems it is not dependent on the number of partitions. Instead we can see that in the other two sections, where we use a dataframe, the partitions affects much more the performances.

In **Country and Institution section**, we notice a parabolic trend and this is because we manage a single information for each file and if the number of partitions is equal to the number of files all the computation is optimized.

N Partitions impact

Let's try to understand these trends.

In **Word Counter section**, we manage a bag and it seems it is not dependent on the number of partitions. Instead we can see that in the other two sections, where we use a dataframe, the partitions affects much more the performances.

In **Country and Institution section**, we notice a parabolic trend and this is because we manage a single information for each file and if the number of partitions is equal to the number of files all the computation is optimized.

In **Title Embedding section**, we compute the head of the dataframe and if we have too large partitions dask has to do more operations than necessary to do it.

Conclusions

In this work, we have tested the different functionalities of dask, reaching these conclusions:

- dask allows us to manage **large amount of data**;
- by creating a **LAN cluster** we are able to perform some tasks that we cannot execute on a single machine, e.g. the FastText file loading;
- we can take advantage of the **lazy computation** method of dask that allows us not to compute intermediate results;
- some limitations are linked to **overhead** and **network connection**;
- we can check the real-time processes using the dask **diagnostic dashboard**.

In conclusion we can say that Dask is a powerful tool but strongly dependent on the **available resources** (we should have had 40 computers to perform the given task).