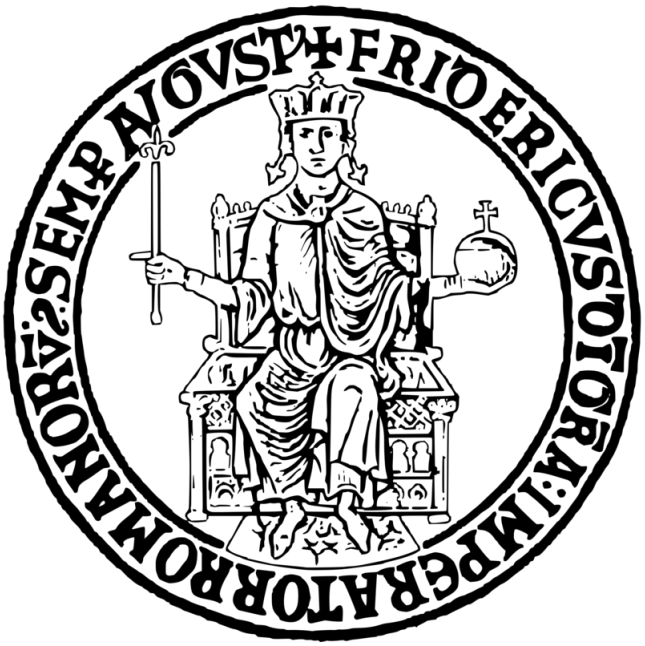
Natural Language Processing for Bibliometrics

February 24, 2021



MASTER'S DEGREE DATA SCIENCE
UNIVERSITY OF NAPLES FEDERICO II

Italo Alberto Ferrante P37000005
Francesco Romano P37000022
Celestino Santagata P37000007

Contents

1 Introduction

Natural language processing (NLP)[1] is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves. It is a discipline that focuses on the interaction between data science and human language, and is scaling to countless industries. Today, NLP is booming thanks to huge improvements in the access to data and increases in computational power, which are allowing practitioners to achieve meaningful results in areas like healthcare, media, finance, and human resources, among others.[2]

For this work we decided to apply text mining for a well-identified discipline, bibliometrics. It is the branch of library science concerned with the application of mathematical and statistical analysis to bibliography; the statistical analysis of books, articles, or other publications.[3] In particular, we used NLP to carry out an analysis of non-medical academic papers dealing with covid-19.

[1] https://en.wikipedia.org/wiki/Naturallanguageprocessing

[2] https://medium.com/@ODSC/an-introduction-to-natural-language-processing-nlp-8e476d9f5f59

[3] https://www.lexico.com/definition/bibliometrics

2 Web Of Science and Bibliometrix

To get a bibliographic database we used Web Of Science.
Web Of Science (WOS) is a website that provides subscription-based access to multiple databases that provide comprehensive citation data for many different academic disciplines[4]. We entered a query to download a dataset of non-medical papers themed covid-19. A bibliographic database is a set of bibliographic metadata which summarizes information about a scientific document.
So. we got a dataset, that is a collection of document metadata, in .txt format with more than three thousand records in a period from January 2020 to half January 2021. To conduct an explorative analysis on this mass of data we used Bibliometrix, id est an open-source tool for quantitative research in scientometrics and bibliometrics that includes all the main bibliometric methods of analysis[5]. In particular, to speed our analysis up without losing datail we used Biblioshiny, the web interface of Bibliometrix.
Let's see charts returned by Bibliometrix.

[4] https://en.wikipedia.org/wiki/WebofScience
[5] https://bibliometrix.org/

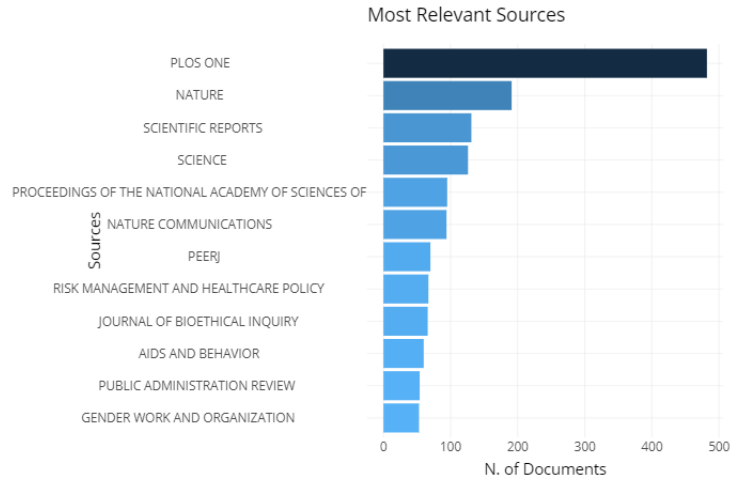


Figure 1: Top 12 sources

As we can see, Plos One is the journal that most of all wrote about coronavirus, followed by Nature, Scientific Reports, Science and so on... But though it spoke so much of it, as showed by next image, it is not the most cited source:

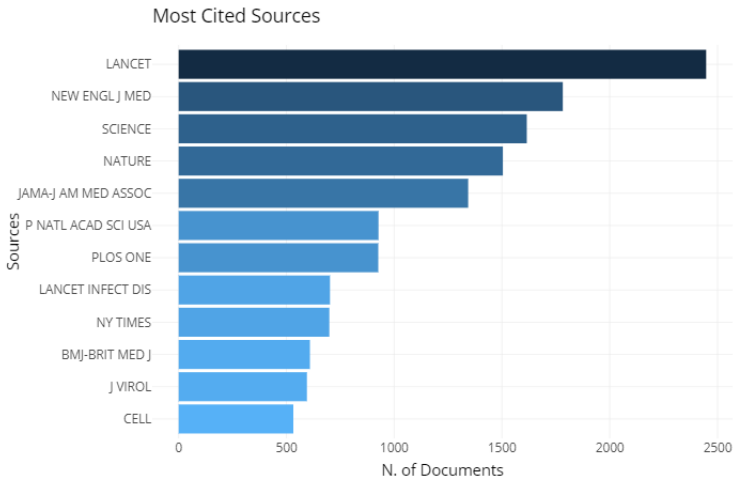


Figure 2: Top 12 cited sources

the reason according to us is the high reputation of Lancet and the higher intellectual content of their articles.
Always for analysing sources, let's see the chart of Bradford's Law.
The law states that, if we consider a list of journals drawn up taking into account the decreasing productivity in relation to the number of items within a given scientific sector, the result is a group of magazines that concentrate most of the articles on the subject, and for this reason it is considered the core of that disciplinary area; and other groups formed by a number of journals which, in order to contain the same number of articles of those that form the nucleus, grows exponentially. The number of magazines that form each group is expressed by the proportion:

$$1 : n : n^2 \tag{1}$$

This means that if journals in a given subject area are classified according to to the number of articles in three groups, one third of the articles are published by a limited number of periodicals specialized in the disciplinary sector in question, another third from other periodicals dedicated to a similar topic but not coincident with the previous remaining third from many generalist periodicals and therefore less relevant. It is therefore clear from the proportion that in order to reach 100% of the bibliographic coverage on the subject it is necessary to add to the nucleus formed by a few specialized journals a much larger number of journals which among other things grows exponentially.

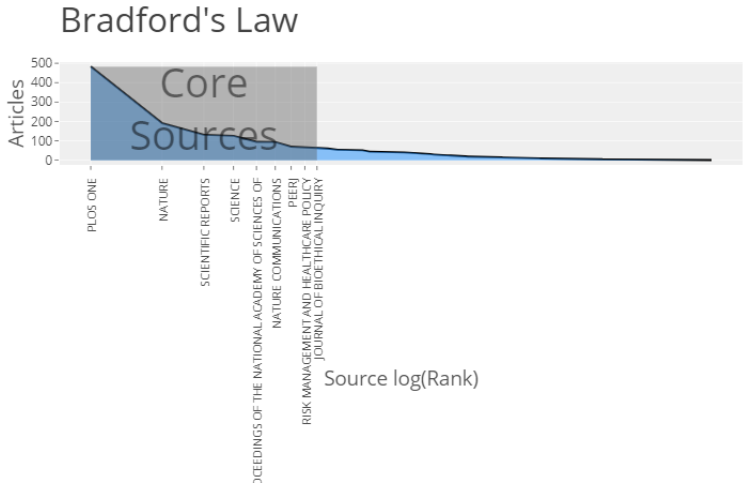


Figure 3: Bradford's Law curve

As we can note, core sources are the first eight relevant sources, so these three sources published one third of the total amount of documents.
These charts gave us a 'quantitative' pov of the sources, but if we want to investigate the 'qualitative' importance of a source we have to conduct an analysis on

the impact. To better understand this topic we have to introduce H-Index and G-Index.
The Hirsch Index (H-Index) is an author's (or journal's) number of published articles (h) each of which has been cited in other papers at least h time.
G-Index was introduced by Hegghe in 2006 and is an improvement of the h-index in order to measure the global citation performance of a set of articles. If this set is ranked in decreasing order of the number of citation that they received, the G-Index is the (unique) largest number such that the top g articles received (together) at least a number of citations equals to the square of g.
Using H-Index we record an equal impact of Nature and Science:

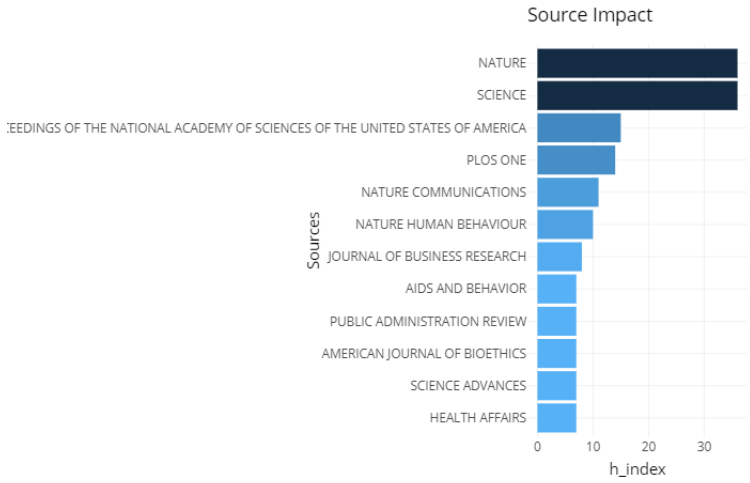


Figure 4: Source Impact with H-Index

instead with G-Index the impact of Nature is higher than Science's one.

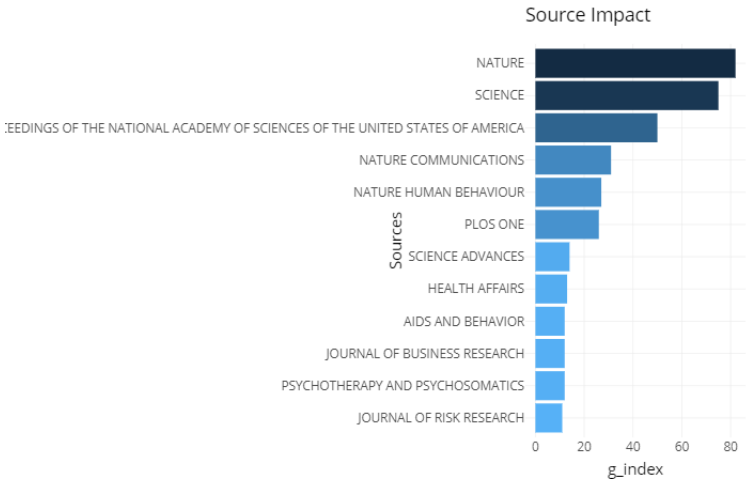


Figure 5: Source Impact with G-Index

Now let's take an overview of the authors.

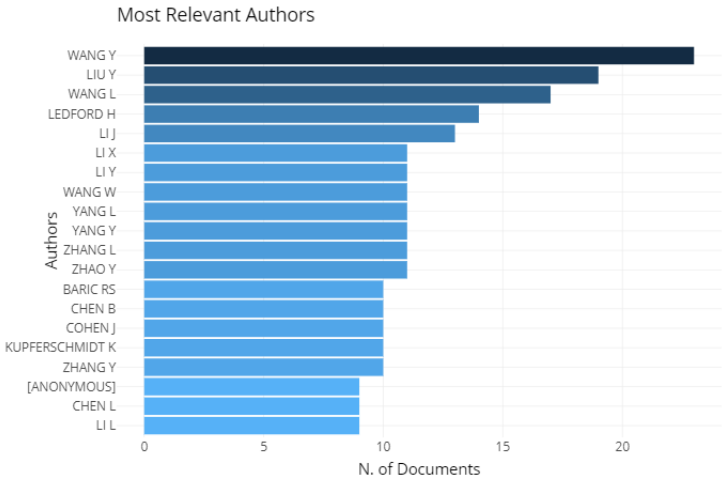


Figure 6: 20 top authors

As emerge by the chart, chinese authors are most relevant ones, maybe because their Country of origin is the Country where coronavirus has spread first. To give a more accurate panoramic of the authors, we analysed Lotka's Law. It describes the frequency of pubblication by authors in any given field. Lotka's Law is an approximaty inverse-square Law, where the number of authors publishing a certain number of articles is a fixed ration to the number of authors publishing a single article. Lotka's Law affirms: *"As the number of articles published increases, authors producing that many publications become less frequent"*.

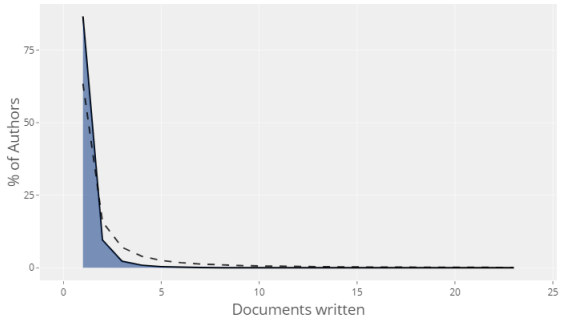


Figure 7: The Frequency Distribution of Scientific Productivity

It is clear that almost all authors wrote a small number of scientif articles and the core area is so tiny that it seems nonexistent.
As done before with the sources, we used G-Index to evaluate author impact:

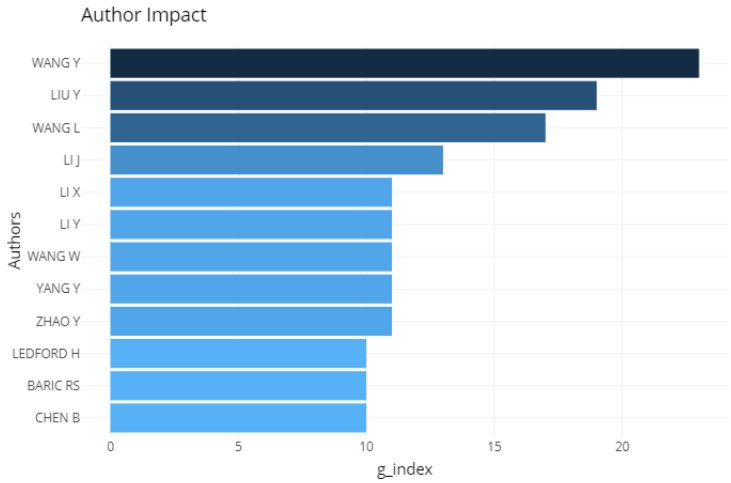


Figure 8: Top 12 Author's Impact

Reading this barchart, we see that almost all authors are Chinese and top three authors are Wang Y., Liu Y. and Wang L.
Now it's time to study affiliations.

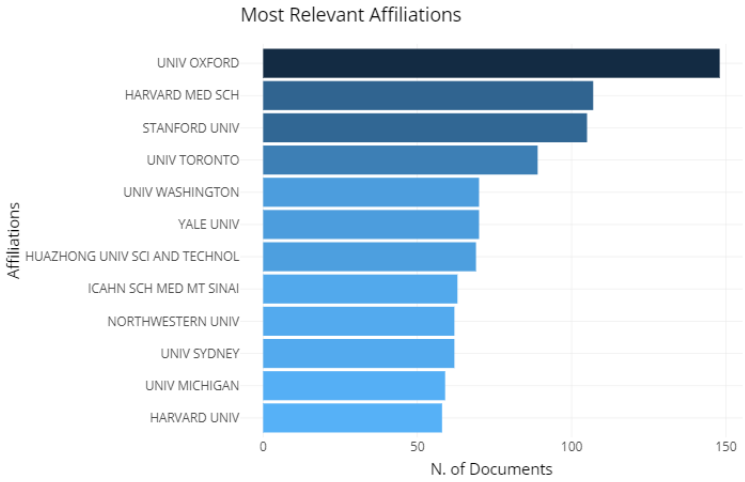


Figure 9: Top 12 Affiliations

Despite the dominance of Chinese authors seen before, to find a Chinese institution we have to go down to the seventh place of the ranking. In fact, first places are dominated by british and american affiliations (Oxford, clearly in first place, followed by Harvard and Stanford).

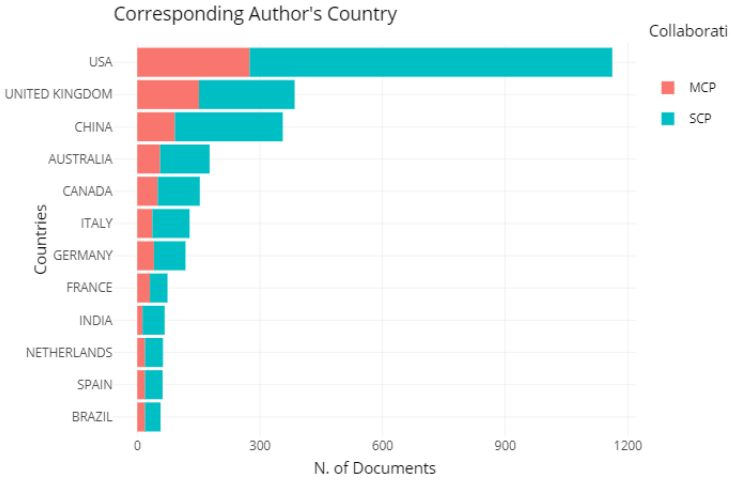


Figure 10: MCP and SCP per top 12 Contry

This is a more interesting and less intuitive chart and to be understood we have to clarify the difference between MCP and SCP. The first is the acronym of Multiple Countries Publications and indicates, for each Country, the number of documents in which there is at least one co-author from a different Country, while SCP (Single Country Publication) indicates the number of documents wrote by authors of that Country.

Every Country has a MCP higher than SCP with with a roughly constant ratio between these measures. Global scientific production is represented in these map:

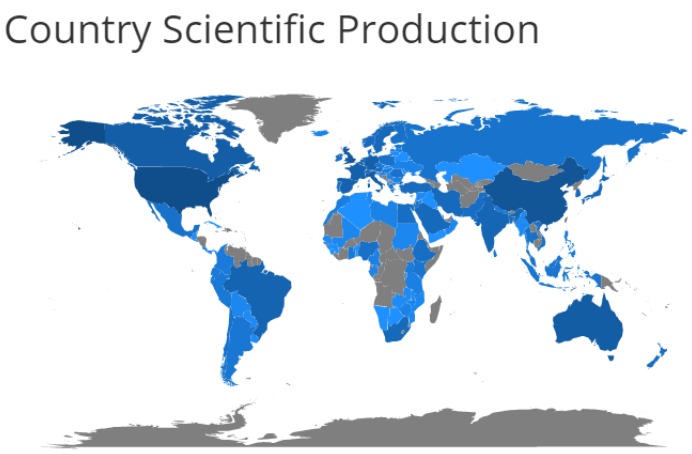


Figure 11: Global Scientific Production

With increasing shades of blue are indicated Contries with a greater scientific production. Europe, America, Australia and China are the areas where color blue is louder while there are areas like Central America, Central Africa and Middle East, plus some scattered countries that are colored by grey (id est they have a poor scientific production on the subject). At this point we looked for the most mentioned countries:

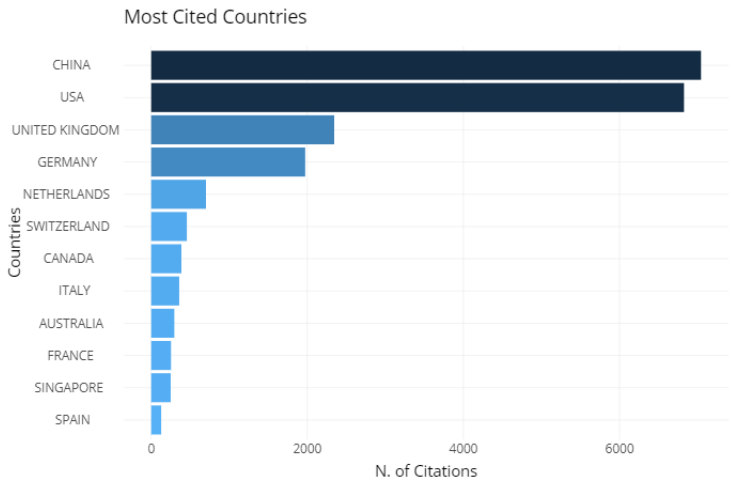


Figure 12: Top 12 Cited Countries

how was probably predictable, most cited Countries are China ed Usa. For those interested in seeing these data better, we also insert the table of the first twenty Countries with the indication of the number of published articles, the frequency, SCP, MCP and MCP Ratio. Trivially SCP Ratio can be calculated as 1 - MCP Ratio.

Country	Articles	Freq	SCP	MCP	MCP_Ratio
USA	1162	0.318182	887	275	0.237
UNITED KINGDOM	385	0.105422	234	151	0.392
CHINA	356	0.097481	264	92	0.258
AUSTRALIA	177	0.048467	121	56	0.316
CANADA	153	0.041895	103	50	0.327
ITALY	128	0.035049	91	37	0.289
GERMANY	118	0.032311	77	41	0.347
FRANCE	74	0.020263	43	31	0.419
INDIA	67	0.018346	54	13	0.194
NETHERLANDS	63	0.017251	44	19	0.302
SPAIN	62	0.016977	43	19	0.306
BRAZIL	57	0.015608	38	19	0.333
SOUTH AFRICA	42	0.011501	32	10	0.238
KOREA	41	0.011227	33	8	0.195
SWITZERLAND	40	0.010953	23	17	0.425
ISRAEL	39	0.010679	29	10	0.256
SWEDEN	35	0.009584	24	11	0.314
JAPAN	34	0.009310	30	4	0.118
DENMARK	31	0.008488	15	16	0.516
BELGIUM	30	0.008215	15	15	0.500

Figure 13: Articles, SCP, MCP and MCP Ratio of Top 20 Countries

At this point, we have to look into documents. For 'documents' we intend to a scientific document (article, review, conference meeting...) included in a bibliographic collection. For 'cited documents' we refer to a scientific document included in a bibliographic collection and, at the same time, it is cited in at least one another document of the collection. Let's see cited documents barplot:

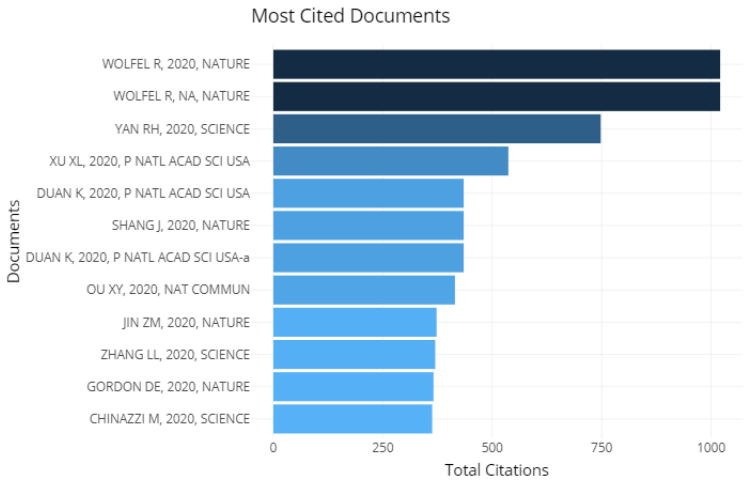


Figure 14: Top 12 Cited Documents

In first places we find probably the same article of Wolfel published for Nature, so we consider in second place an article of Yan appeared on Science. This ranking is dominated by articles published on Nature, Science and Proceedings of the National Academy of Sciences (an academic magazine). Instead, for 'reference' we refer to a scientific document included in at least one of the reference lists (bibliography) of the document set. Then we can say that *"a reference is cited by one or more documents"*.

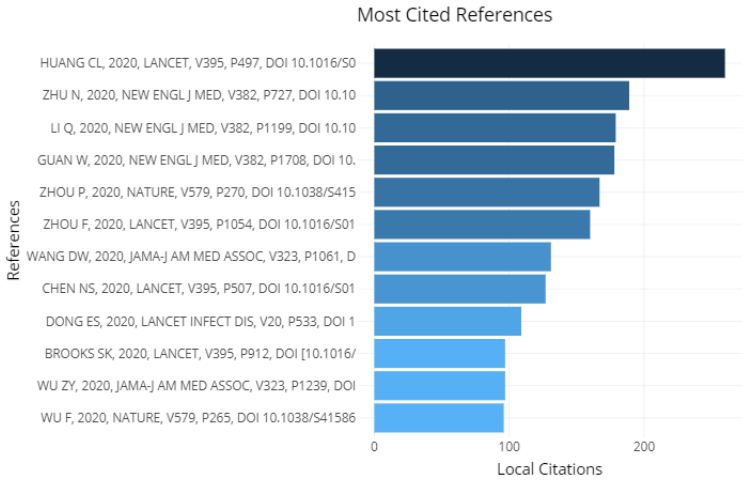


Figure 15: Top 12 Cited References

The first, a reference of an article of Huang published on Lancet, received more than 200 local citations. These are the number of citation a document has received from documents included in the analyzed collection. So local citation measure the impact of a documents in the analyzed collection.
At this point we began the part of science mapping, i.e. the set of activities aimed at displaying the structural and dynamic aspects of scientific research. Science Mapping allows investigating scientific knowledge from a statistical point of view. In particular:

- conceptual structure is what science talks about, the main themes and trends;
- intellectual structure is how the work of an author influences a given scientific community;
- social structure is how authors, institutions and Countries interact each other.

Conceptual structure represents relations among concepts or words in a set of publications:

- Words, which appear together in a document, will be related in a network. It is also known as the co-words network. This structure is be used to understand the topics covered by a research field to define what are the most important and the most recent issues (so called, research front). It could also help in the study of the evolution of subjects over time.
- Similarly to network analysis, factorial analysis (data reduction techniques) is helpful in identifying subfields. Various dimensionality reduction techniques can be applied, such as correspondence analysis (CA), multiple correspondence analysis (MCA), multidimensional scaling (MDS), principal

component analysis (PCA). Clustering algorithms can be used in both cases of network or factorial analysis.

- Mixed approach. Starting from a conceptual network, you identify thematic networks that plot on a bi-dimensional matrix, where axis are function of centrality and density of the thematic network. Dividing the timespan in time slices, it is possible to represent the thematic evolution within a specific research field through an alluvial graph.

The basic idea behind factorial approaches is to reduce dimensionality of data and represent it in a low-dimensional space. The proximity between words correspond to shared-substance.
We choose to represent keywords plus with Multiple Correspondence Analysis (MCA) and arbitrarily set 3 clusters and 60 words. This is the output:

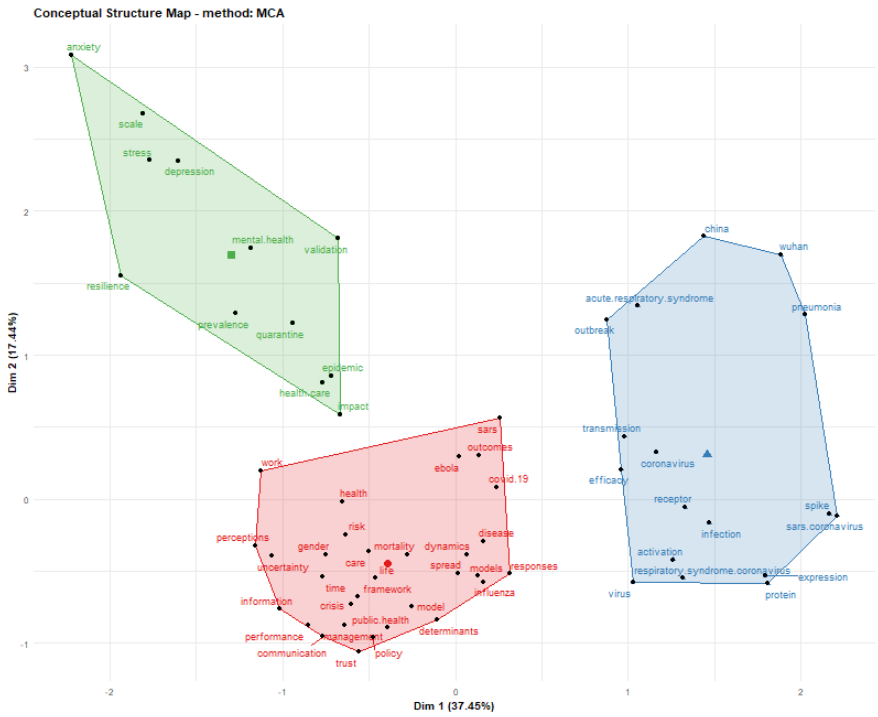


Figure 16: MCA of Keywords

We composed a dial whose first dimension explains 34.75% of whole information and the second 17.44%. We have thus identified three areas: the green area is the 'psicological' area, infact we found 'mental' words like 'stress', 'mental health', 'anxiety' etc.; always according our interpretation in blue polygon there are 'medical' words like 'protein', 'pneumonia' and 'acute respiratory syndrome'; in red area instead there are 'socio-economic' words like 'crisis', 'management',

'information', 'policy', 'trust' and so on... We have expanded the study making a network analysis. Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines). A distinction is made between undirected graphs, where edges link two vertices symmetrically, and directed graphs, where edges, then called arrows, link two vertices asymmetrically. In Science mapping, a network graph is used to represent co-occurrences among bibliographic metadata. Each vertex represents an item (in this case, a word), the vertex size is proportional to the item occurence, the edge size is proportional to item co-occurences. Let's see the output:

Figure 17: Keywords Network

This time, colors have different meanings: red label means 'socio-economic' area while green, blue and violet areas contain 'medical' words. So, due to this

poor ability to create groups of words in the same field, we prefer to this graph the previous one in which we run MCA.
For Intellectual Structure we propose co-occurence newtork:

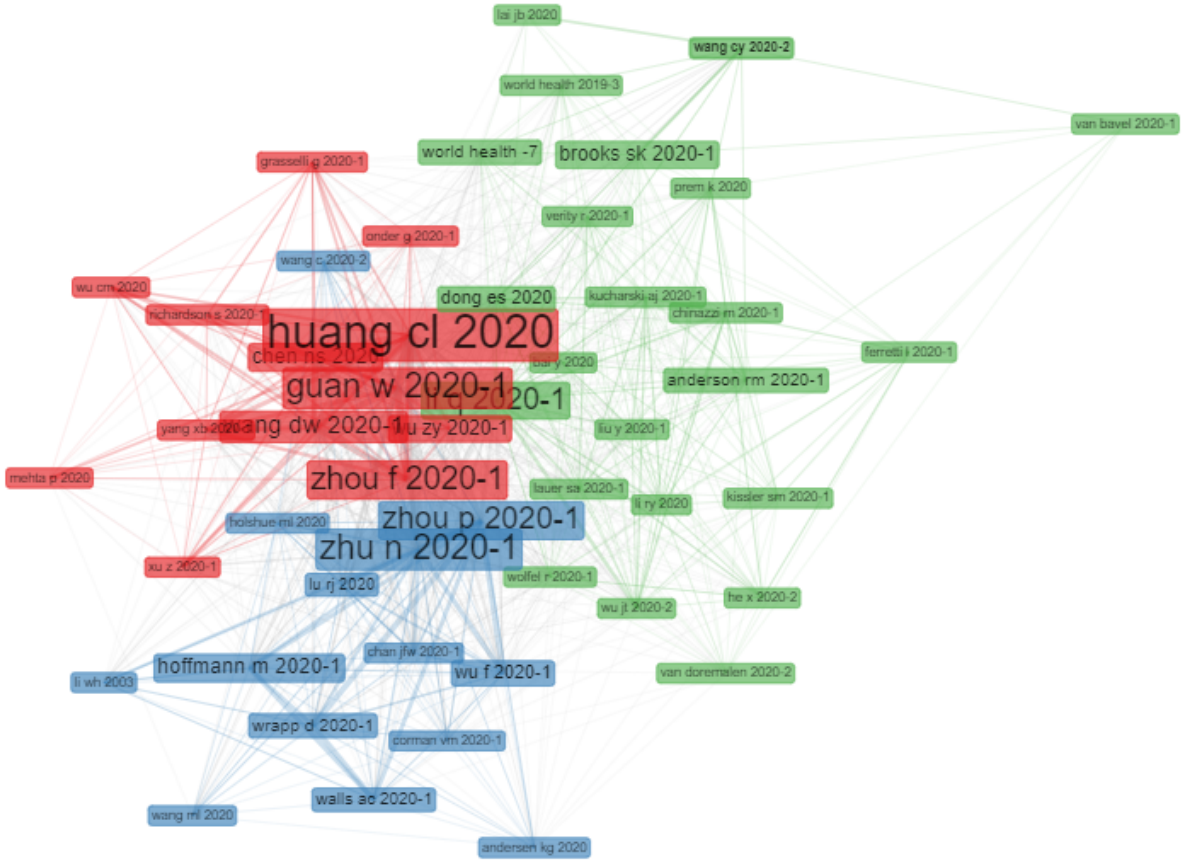


Figure 18: Co-citation Network

Social structure shows how authors or institutions relate to others in the field of scientific research. The most common kind of social structure is co-authorship network (Peters et al. 1991). With co-authorship networks can be discovered, for example, groups of regular authors, influent authors, hidden communities of authors, relevant institutions in a specific research field, etc. Let's see chart of institutions:

Figure 19: Institutions Network

More interesting is the one that shows connection among Countries.

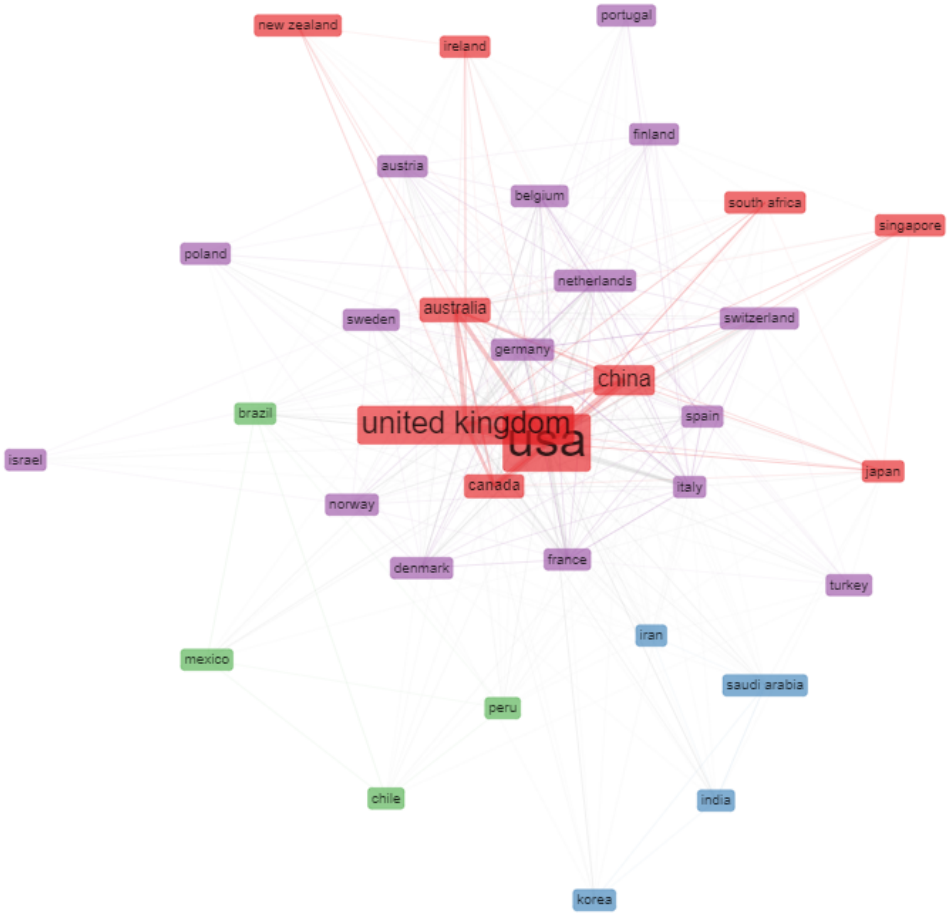


Figure 20: Countries Network

In this way, we clusterized four groups: the violet with most of EU Countries, with a red label Countries connected to China and America, in green Nations of Center and South American, and a residual area with South-Korea, India, Saudi Arabia and Iran. For a spatial representation of connection among Nations, we propose this chart in which we ruled out edges with less than 10 connections:

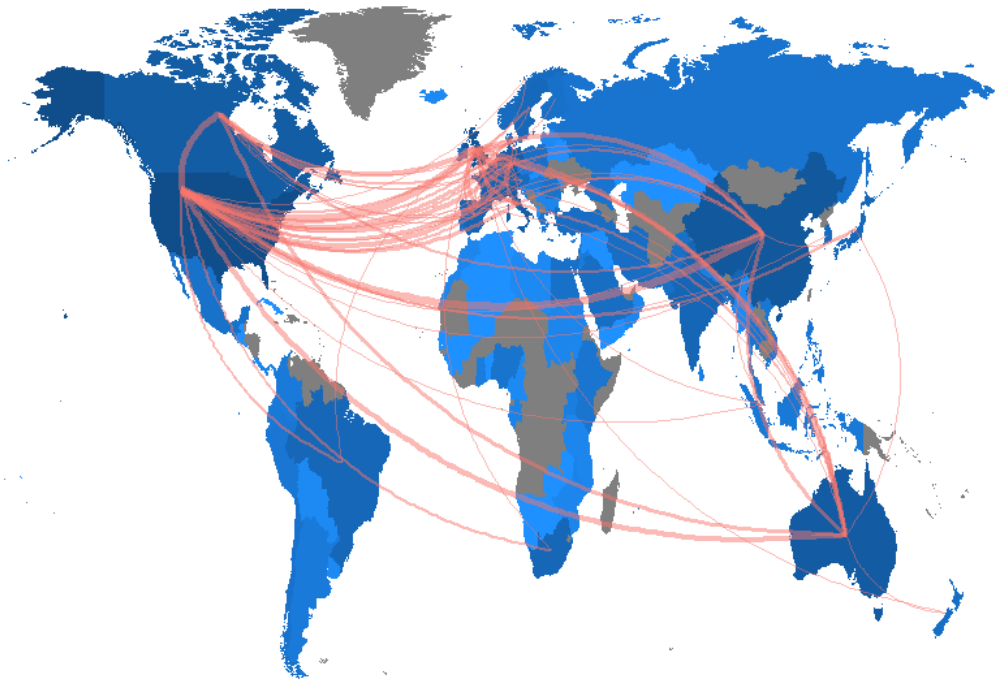


Figure 21: Countries Network

3 Summarization

What is Text Summarization in NLP? Text Summarization involves condensing a piece of text into a shorter version, reducing the size of the original text while preserving key information and the meaning of the content. Since manual text synthesis is a long and generally laborious task, task automation is gaining in popularity and therefore a strong motivation for academic research. The intention to summarize a text is to create an accurate and fluid summary containing only the main points described in the document.

Extractive vs Abstractive Summarization. Text Summarization is classified into two main types: Extraction Approach and Abstraction Approach. Now let's go through both these approaches before we dive into the coding part. The Extractive approach takes sentences directly from the document according to a scoring function to form a cohesive summary. This method works by identifying the important sections of the text cropping and assembling parts of the content to produce a condensed version. The Abstraction approach aims to produce a summary by interpreting the text using advanced natural language techniques to generate a new, shorter text – parts of which may not appear in the original document, which conveys the most information. We used the extractive approach to summarize text using Machine Learning and Python, and in particular we used the TextRank, a graph-based ranking model for text processing which can be used in order to find the most relevant sentences in text and also to find keywords. More specifically, TextRank is an algorithm based on PageRank.
PageRank (PR) is an algorithm used to calculate the weight for web pages. We can take all web pages as a big directed graph. In this graph, a node is a webpage. If webpage A has the link to web page B, it can be represented as a directed edge from A to B. After we construct the whole graph, we can assign weights for web pages by the following formula:

$$S\left(V_i\right) = (1-d) + d * \sum_{j \in \mathrm{In}(v_i)} \frac{1}{\left|Out\left(V_j\right)\right|} S\left(V_j\right)$$

where:

- $S(Vi)$ - the weight of webpage i
- d - damping factor, in case of no outgoing links
- In(Vi) - inbound links of i, which is a set

- Out(Vj) - outgoing links of j, which is a set

- |Out(Vj)| - the number of outbound links

The difference between TextRank and PageTank is that PageRank is for webpage ranking while TextRank is for text ranking.
The webpage in PageRank is the text in TextRank, so the basic idea is the same. In order to find the most relevant sentences in text, a graph is constructed where the vertices of the graph represent each sentence in a document and the edges between sentences are based on content overlap, namely by calculating the number of words that 2 sentences have in common. Based on this network of sentences, the sentences are fed into the Pagerank algorithm which identifies the most important sentences. When we want to extract a summary of the text, we can now take only the most important sentences.

3.1 Libraries

Python Libraries are a set of useful functions that eliminate the need for writing codes from scratch, there are over 137,000 python libraries present today. Python libraries play a vital role in developing machine learning, data science, data visualization, image and data manipulation applications and more.

```
import numpy as np
import pandas as pd
from nltk.corpus import stopwords
from nltk.cluster.util import cosine_distance
import re
import networkx as nx
```

Figure 22: List of libraries imported

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical or shape manipulation. It's quintessential for any kind of scientific code development.

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

NLTK is a platform for building Python programs to work with human language data. It's very famous among NLP (Natural Language Processing) projects as it can provide many useful tools such as Stopwords for most used languages and also Lemmatizers or Stemmatizers for working with the grammar of the texts. Stopwords are important for removing redundant or useless words to our study.

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. In this case we used the library to rank sentences generated in the similarity matrix.

3.2 Loading and cleaning data

We gathered our data from a platform called Web of Science[6]. Web of Science (WoS) is the world's leading scientific citation search and analytical information platform. It is used as both a research tool supporting a broad array of scientific tasks across diverse knowledge domains as well as a dataset for large-scale data-intensive studies.
Data consists of Covid19-related scientific papers that don't concern Medicine or Biology fields. We took papers concerning Economics or Psychology for example. Our dataset contains different variables, such as Authors, Addresses, Publisher etc. all different parts of a scientific paper. We just needed the abstracts for this part of the study, to apply the Text Summarization algorithms.
An abstract summarizes, usually in one paragraph of 300 words or less, the major aspects of the entire paper in a prescribed sequence that includes the overall purpose of the study and the research problem(s) you investigated; the basic design of the study; major findings or trends found as a result of your analysis and a brief summary of your interpretations and conclusions.

```
df = pd.read_csv(r'C:\Users\COMPUTER\Desktop\wos.csv') #creates a dataframe
df['Abstract'] = df['Abstract'].astype(str) # converts abstracts to string
df = df.reset_index(drop=True) # resets indices
df.columns # shows columns
```

Figure 23: Code for importing data.

First thing first, we downloaded the papers from WoS webpage, loaded it with Pandas and changed the type of Abstracts to be a "string". At first the dataset had 3841 rows, but we found out that some of these had "NaN" values, so with the

[6]https://en.wikipedia.org/wiki/Web_of_Science

help of pandas built-in command "*notna()*" we easily deleted those, resulting in 3095 total abstracts of papers.

3.3 Algorithm for text summarization with TextRank

Text summarization is very important for representing the most relevant and important information from a certain document. As technology and globalization keep rising in term of power, so papers and their usage and exchange keep increasing. The purpose of our summarization is to create a tool capable of summarizing Abstracts of papers.

To start, we created a function to open and read each article. The function is pretty simple, once opened, the file gets "read" by the function, so it can get processed immediately after. For the purpose of splitting each sentence, we utilize the *.split()* command of Python, to cut each sentence from the document. Those sentences will be used later. To each sentence then a *.replace()* command gets applied to replace with a space every character that isn't a letter. We do this replacement to obtain clean sentences made of words alone.

```
def sentence_similarity(sent1, sent2, stopwords=None):
    '''It looks at similarity of sentences'''
    if stopwords is None:
        stopwords = []
    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]
    all_words = list(set(sent1 + sent2))
    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)
    for w in sent1: # build the vector for the first sentence
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1
    for w in sent2: # build the vector for the second sentence
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1
    return 1 - cosine_distance(vector1, vector2)
```

Figure 24: Code for the cosine similarity function.

The next step is computing similarity between sentences. We did this using a module from the library **nltk** called "*cosine_distance*". Each sentence gets first

cleaned of any stopwords, then a vector gets build based on the number and type of words in that sentence. For each couple of sentences then, a cosine similarity gets computed, this similarity represents the logical distance between those two sentences. The process is executed for each sentence and couple of sentences, returning only the cosine similarity between those.

With our scores computed it's time to build a matrix with all the similarity scores. Taking as input the length of sentences, and the output from the last function, first checks if the sentence is the same so it can ignore any doubles. After, it builds the matrix to be used by the next and last function for this algorithm.

Our main function uses the before-mentioned functions to read, split, clean and vectorize sentences in a given document. After all this works all that's left to do from the main function is to sort the rankings and pick the ones with the best score. It then returns the picked ones as output.

```
Indexes of top ranked_sentence order are  [(0.1873752688851547, ['E', 'i', 'controllori', 'del', 'volo', 'del', 'rover', '"hann
o', 'confermato', 'che', 'il', 'rover,', 'con', "l'elicottero", 'Ingenuity', 'Mars', 'attaccato', 'alla', 'pancia,', 'Ã¨', 'att
errato', 'in', 'sicurezza"', 'nel', 'cratere', 'Jezero,', 'un', 'bacino', 'largo', '45', 'chilometri', 'appena', 'a', 'Nord',
"dell'equatore", 'marziano,', 'che', 'circa', '3,9', 'miliardi', 'di', 'anni', 'fa', 'conteneva', 'un', 'lago']), (0.1751099793
92209, ['La', 'missione', 'Mars', '2020', 'Ã¨', 'destinata', 'a', 'cercare', 'tracce', 'di', 'vita', 'passata', 'e', 'a', 'racc
ogliere', 'i', 'primi', 'campioni', 'del', 'suolo', 'marziano', 'che', 'nel', '2031', 'saranno', 'portati', 'sulla', 'Terra',
'da', 'una', 'staffetta', 'di', 'missioni', 'nella', 'quale', "l'Italia", 'ha', 'un', 'ruolo', 'importante']), (0.1501161688371
948, ['Il', 'rover', 'Perseverance', 'della', 'Nasa', 'Ã¨', 'arrivato', 'su', 'Marte']), (0.14179846226389559, ['Ora', 'si', 'a
pre', 'una', 'nuova', 'pagina', 'delle', 'esplorazioni', 'spaziali,', 'con', 'il', 'quinto', 'rover', 'che', 'tocca', 'Mart
e']), (0.13810951502529067, ['Il', 'mio', 'primo', 'sguardo', 'sulla', 'casa', 'che', 'abiterÃ²', 'per', 'sempre",', 'si', 'leg
ge', 'sul', 'profilo', 'Twitter', 'del', 'rover']), (0.12699042535569066, ['Tutti', 'in', 'piedi', 'a', 'esultare,', 'grida',
'di', 'gioia', 'e', 'un', 'lungo', 'commosso', 'applauso', 'dalla', "'cabina", 'di', "regia'", 'della', 'Nasa', 'hanno', 'sanci
to', "l'avvenuto", 'touchdown', 'di', 'Perseverance']), (0.059521159261328185, ['E', 'da', 'lÃ¬,', 'dal', 'Pianeta', 'Rosso,',
'ha', 'inviato', 'le', 'sue', 'prime', 'immagini']), (0.020979020979236294, ['"Ciao', 'mondo'])]


Summarize Text:
 E i controllori del volo del rover "hanno confermato che il rover, con l'elicottero Ingenuity Mars attaccato alla pancia, Ã¨ a
tterrato in sicurezza" nel cratere Jezero, un bacino largo 45 chilometri appena a Nord dell'equatore marziano, che circa 3,9 mi
liardi di anni fa conteneva un lago. E i controllori del volo del rover "hanno confermato che il rover, con l'elicottero Ingenu
ity Mars attaccato alla pancia, Ã¨ atterrato in sicurezza" nel cratere Jezero, un bacino largo 45 chilometri appena a Nord del
l'equatore marziano, che circa 3,9 miliardi di anni fa conteneva un lago
```

Figure 25: Output from the text summarization function.

This is the result of a summarized article found online. As we can see the first part consist in sorting the ranked sentences computed using the cosine_similarity function. Last it returns as output the summarized text.

3.4 Interactive Search Engine

With the Text Summarization Algorithm completed, we can apply it on those papers taken from Web of Science. As mentioned before we are only using the abstracts, so first thing first we drop the other section of the dataset to save computational space, saving only Title, Author Name(s), and Abstract columns. This is the result after performing all the cleaning.

```
In [7]: df.head(5)
```

Out[7]:

	Author Full Names	Article Title	Abstract
1	Singer, Merrill	Deadly Companions: COVID-19 and Diabetes in Me...	In this commentary, I assess the adverse synde...
2	Cao, Yukun; Han, Xiaoyu; Gu, Jin; Li, Yumin; L...	Prognostic value of baseline clinical and HRCT...	The aim of this study was to assess the progno...
3	Nishio, Mizuho; Noguchi, Shunjiro; Matsuo, Hid...	Automatic classification between COVID-19 pneu...	This study aimed to develop and validate compu...
4	Ponga, Mauricio	Quantifying the adhesive strength between the ...	The binding affinity and adhesive strength bet...
5	Tian, Ran; Wu, Wei; Wang, Chunyao; Pang, Haiyu...	Clinical characteristics and survival analysis...	Since the outbreak of COVID-19 in China at the...

Figure 26: Head of the cleaned dataset.

The search engine works in a simple yet effective way: first it asks the user if he wishes to search the paper by Article or Author Name; After the choice is made and after providing the input name or article, the function returns the ordered ranked sentences, the summarized abstract and original one, using the before-mentioned functions from the last section. The function makes sure the output is not lost; first it will create a .txt file to write the summarized abstract there, so it doesn't get lost. It writes two different files based on the method of research.

This is the step-by-step process of the search engine:

```
Vuoi procedere ad una ricerca per autore o per articolo? Autore

Che autore stai cercando? Scrivi prima il cognome e poi il nome separati da una virgola
Ponga, Mauricio
```

Figure 27

As mentioned before, the code asks the user to input a research method, in this case we chose Author. Then it asks the user to input the surname, followed by a comma, and the name of the author. Then returns the summarized text following the same pattern as the example shown before: with the ordered ranked sentences, the summarized text and the full text to be compared.

Using this way we built a tool capable of cleaning, processing, analyzing and summarizing abstracts from a raw standard dataset, with the user-friendly search tool for easier access.

4 Keyword Extraction

While Text Summarization is a great tool to synthesize useful information from a given document, it's not the only solution to the problem, another trick under our sleeve can be keyword extraction. Keyword extraction is the automated process of extracting the words and phrases that are most relevant to an input text.
While there are many approaches for the making of a Keyword Extraction algorithm, we decided to build one with the *TF-TDF* or Term Frequency - Inverse Dense Frequency. This is a technique which is used to find meaning of sentences consisting of words and cancels out the incapabilities of Bag of Words technique, useful for text classification and for helping a machine read sentences. The way TF-IDF works is that increase the importance of each word in proportion to the number of times a word appears in the document (Text Frequency) but is offset by the frequency of the word in the corpus (Inverse Document Frequency)

To implement our TF-IDF algorithm we used a library called *sklearn*, or SciKit Learn. This library is a valuable tool containing all beginner and professional tools to work with data and machine learning in Python. Its range of tools and its user-friendly style makes it one of the most used packages.

```
import pandas as pd
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn.impute import SimpleImputer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from matplotlib import pyplot as plt
```

Figure 28: Libraries imported for this part of project.

4.1 Preprocessing Data

The data used to work with this algorithm is the same used in last chapter for the Text Summarization, taken from Web of Science. Again, before being used the dataset required some cleaning; for this reason we coded a function called **pre_processing**. It's a simple function that follows a series of commands: first it converts every letter into a lowercase one, removes all tags, characters and digits. Then converts the string into a list of sentences and removes any word that might prove too redundant to the analysis. Last it uses the nltk built-in command *Word-*

NetLemmatizer(), this is very important as we want each word to be the word we would find in the dictionary, so it can be read better by the algorithm.

```
def pre_process(text):
    '''This function preprocesses a text'''
    text=text.lower() # lowercase
    text=re.sub("&lt;/?.*?&gt;"," &lt;&gt; ",text) #remove tags
    text=re.sub("(\\d|\\W)+"," ",text) # remove special characters and digits
    text = text.split() #Convert to list from string
    text = [word for word in text if word not in stop_words] # remove stopwords
    text = [word for word in text if len(word) >= 3] # remove words less than three letters
    lmtzr = WordNetLemmatizer() # lemmatize
    text = [lmtzr.lemmatize(word) for word in text]
    return ' '.join(text)

docs = abstract_df['Abstract'].apply(lambda x:pre_process(x))
```

Figure 29: Code for the pre-processing function.

4.2 TF-IDF Algorithm

After using the pre_processing function to clean our data we set up the command *CountVectorizer* from the library SciKit Learn. The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. In order to create a proper CountVectorizer instance we first setup the max DF, aka ignore the words that appears in 95 percent of the document, then setup the max features in order to build a vocabulary that selects just the top 10.000 words ordered by term frequency. Last we setup ngram_range: the values of the parameters work in a way that that selects the lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted, in our case we set it up to contain unigrams, bigrams and trigrams.

```
cv=CountVectorizer(max_df=0.95,          # ignore words that appear in 95% of documents
                   max_features=10000,   # the size of the vocabulary
                   ngram_range=(1,3)     # vocabulary contains single words, bigrams, trigrams
                  )
word_count_vector=cv.fit_transform(docs)
```

Figure 30: First code block of the TF-IDF algorithm

We then utilize the SciKit Learn command *tfidf_transformer* to calculate the reverse frequency of documents.

```
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)
```

Figure 31: Section where the tfidf$_{transformer}$ command is given.

After doing that we are ready for the final step: extract the keywords using the TF-IDF vectorization. First we compute a function to get feature names, sorted items and the TF-IDF score of the top 10 items. Then the process it's pretty simple: first, we generate the TF-IDF from a given document, in our case our WoS papers dataset, picking a single paper's abstract, we then sort the TF-IDF by descending order of scores, last, we take the top 10 word-scores combination. Overall the process is very similar to the algorithm used in the Text Summarization part, here we show an output as example:

```
===Keywords===
mexico 0.44
diabetes 0.359
major concern 0.252
country like 0.246
caseload 0.246
mellitus 0.228
diabetes mellitus 0.228
suffering 0.189
growing 0.173
adverse 0.173
```

Figure 32: Output keywords from a random abstract.

4.3 Total Principal Keywords

We wanted to find the principal keywords for the whole dataset of papers; seeing that they both have in common being non-medical papers concerning covid19, it may prove useful in extracting insight useful for future researches.

In order to do it we applied the same methods used for the Keywords Extraction, we computed the TF-IDF using the beforementioned methods and ranked each keyword. However this time, after ranking all the top words, we put them in a dictionary to count all the occurencies of that word. This way, after sorting the dictionary, we could visualize the most recurrent and also important ones in the whole dataset. Here is the output of our visualization.

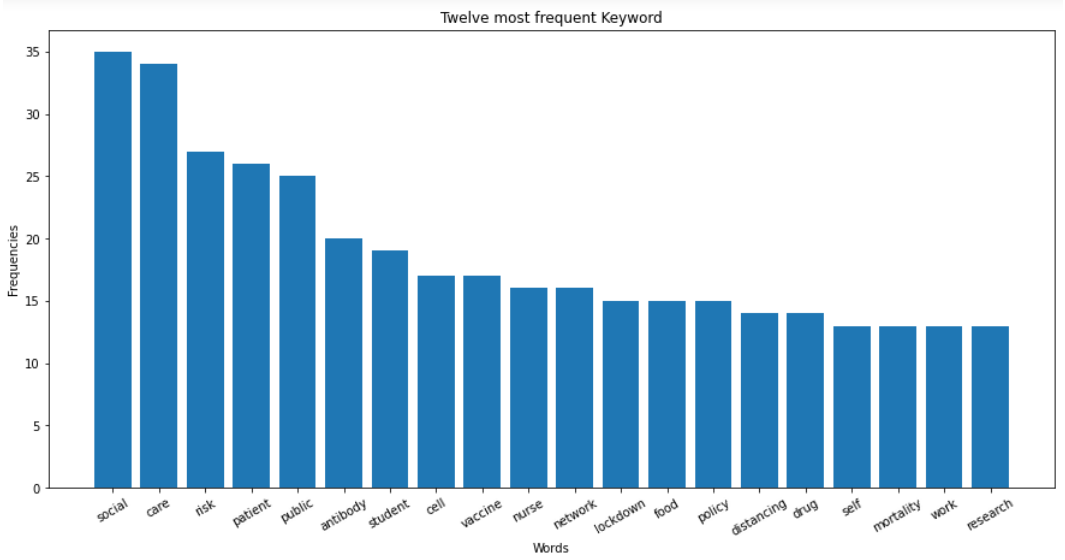


Figure 33

4.4 WordCloud

A WordCloud is a novelty visual representation of text data, typically used to depict keyword metadata (tags) on websites, or to visualize free form text. Tags are usually single words, and the importance of each tag is shown with font size or color. This format is useful for quickly perceiving the most prominent terms to determine its relative prominence. Bigger term means greater weight.

As last paragraph of this part of the project we decided to include a WordCloud. As explained above WordCloud can be neat tools to visualize data and deliver it in a more user-friendly way. In order to compute one we used the Python library *wordcloud*. We also imported a library named *PIL*, used in many ways to handle and process images with Python.

```
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image
```

Figure 34: Libraries used for the WordCloud

Making a WordCloud is pretty easy, all we needed was our list of words taken from all the sentences of all the abstracts of our dataset. We then coded a function to plot and show the WordCloud with the selected size and shape.

```
def plot_cloud(wordcloud):
    # Set figure size
    plt.figure(figsize=(40, 30))
    # Display image
    plt.imshow(wordcloud)
    # No axis details
    plt.axis("off")
    plt.title('Word Cloud with a """Covid 19""" shape')
    plt.savefig('Word Cloud')
    plt.show;
```

Figure 35: Code for the plot fucntion.

In order to stay on-topic with the papers, we picked a particular shape for our WordCloud. Our datasets contains papers concerning covid19, so we picked the shape of a covid19 virus as seen on the microscope. After running the functions this is our result:

Figure 36

5 Sentiment Analysis

Sentiment Analysis can help us to understand the mood and emotions of a community and gather insightful information regarding the context. Sentiment Analysis is a process of analyzing data and classifying it based on the need of the research. These sentiments can be used for a better understanding of various events and impact caused by it. In the Machine Learning World it is possible to see many different names, sentiment analysis, opinion mining, opinion extraction, sentiment mining, subjectivity analysis, affect analysis, emotion analysis, review mining, however all of them have similar purposes and belong to the subject of sentiment analysis or opinion mining. By analysing these sentiments, we may find what people like, what they want and what their major concerns are.
The extraordinary potential of the Web has made itself clear to everyone with the introduction of social networks. Web-based social applications are constantly evolving and creating increasing amount of data, this data is a treasure trove of information about user preferences, their connections, and their influences on others. Therefore, it is natural to leverage this data for analytical insights [1].
In this context, Twitter has an outstanding role. It is an online service which lets subscribers post short messages ("tweets") of up to 140 characters about anything, from good-morning messages to political stands.
Such micro-texts are a precious mine for grasping opinions of groups of people, possibly about a specific topic or product. This is even more so, since tweets are associated to several kinds of meta-data, such as geographical coordinates of where the tweet was sent from, the id of the sender, information that can be combined with text analysis to yield an even more accurate picture of who says what, and where, and when.

5.1 Datasets

To perform analysis we used two datasets:

1. **Covid19 tweets**: these tweets are collected using Twitter API and a Python script. A query for this high-frequency hashtag (#covid19) is run on a daily basis for a certain time period, to collect a larger number of tweets samples.
   Link: Covid19 tweets

2. **Sentiment tweets**: this is the sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter API. The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment.
   Link: Sentiment tweets

5.2 Sentiment analysis with TextBlob

TextBlob is a python library for Natural Language Processing (NLP) and is based on Naive Bayes algorithm, that use *Natural Language ToolKit* (NLTK) to achieve its tasks. NLTK is a library which gives an easy access to a lot of lexical resources and allows users to work with categorization, classification and many other tasks. For lexicon-based approaches, a sentiment is defined by its semantic orientation and the intensity of each word in the sentence. This requires a pre-defined dictionary classifying negative and positive words. Generally, a text message will be represented by bag of words. After assigning individual scores to all the words, final sentiment is calculated by some pooling operation like taking an average of all the sentiments. TextBlob returns polarity and subjectivity of a sentence. *Polarity* lies between [-1,1], -1 defines a negative sentiment and 1 defines a positive sentiment (negation words reverse the polarity). TextBlob has semantic labels that help with fine-grained analysis (emoticons, exclamation mark, emojis, etc). *Subjectivity* lies between [0,1]: it quantifies the amount of personal opinion and factual information contained in the text. The higher subjectivity means that the text contains personal opinion rather than factual information. TextBlob has one more parameter: *intensity*. TextBlob calculates subjectivity by looking at the 'intensity' which determines if a word modifies the next word.
To perform analysis we used **Covid19 tweets**: these tweets are collected using Twitter API and a Python script. A query for this high-frequency hashtag (#covid19) is run on a daily basis for a certain time period, to collect a larger number of tweets samples. Link: Covid19 tweets
First we load the necessary libraries, the dataset and then we extract the text of the tweets. To the tweet text we apply a cleaning function (fig.37) and then pass the cleaned text for two functions: one that provides polarity and one that returns subjectivity (fig.38). At this point it is possible to obtain sentiment through the appropriate function (fig.39)

```
stop_nltk= stopwords.words("english")
stop_updated= stop_nltk+["https",'t.co','...']
lemm= WordNetLemmatizer()
tweet_tok=  TweetTokenizer()

def clean_text(text):
    tokens= tweet_tok.tokenize(text.lower())
    tokens1 = [re.sub('^https://t.co/[\w]+','',tok) for tok in tokens]
    tokens2 = [re.sub('^@[\w]+','',tok) for tok in tokens1]
    tokens3 = [re.sub('^#[\w]+','',tok) for tok in tokens2]
    tokens4 = [re.sub('[0-9]+','',tok) for tok in tokens3]
    lemmed=[lemm.lemmatize(term) for term in tokens4 if term not in stop_updated
    res=' '.join(lemmed)
    return res
```

Figure 37: Function for text cleaning

```
# Function that return subjectivity
def get_subjectivity(text):
    return TextBlob(text).sentiment.subjectivity

# Function that return polarity
def get_polarity(text):
    return  TextBlob(text).sentiment.polarity


# Add Subjectivity and Polarity columns to dataframe
df['Subjectivity'] = df['text'].apply(get_subjectivity)
df['Polarity'] = df['text'].apply(get_polarity)
df
```

Figure 38: Subjectivity and Polarity function

```
def get_sentiment(score):
    if score < -0.05:
        return 'Negative'
    elif score > 0.05:
        return 'Positive'
    else:
        return 'Neutral'
df['Analysis'] = df['Polarity'].apply(get_sentiment)
df
```

Figure 39: Function that returns sentiment

	text	Subjectivity	Polarity	Analysis
0	smelled scent hand sanitizers today someone pa...	0.250000	-0.250000	Negative
1	hey made sense player pay respect	0.000000	0.000000	Neutral
2	trump never claimed hoax claim effort	0.000000	0.000000	Neutral
3	one gift give appreciation simple thing always...	0.357143	0.000000	Neutral
4	july medium bulletin novel	0.000000	0.000000	Neutral
...	...	...	...	...
179103	thanks nominating challenge nominate	0.200000	0.200000	Positive
179104	year insanity lol	0.700000	0.800000	Positive
179105	powerful painting juan lucena tribute grandpar...	1.000000	0.300000	Positive
179106	student test positive major university abc news	0.522727	0.144886	Positive

Figure 40: Dataset with Subjectivity, Polarity and Sentiment

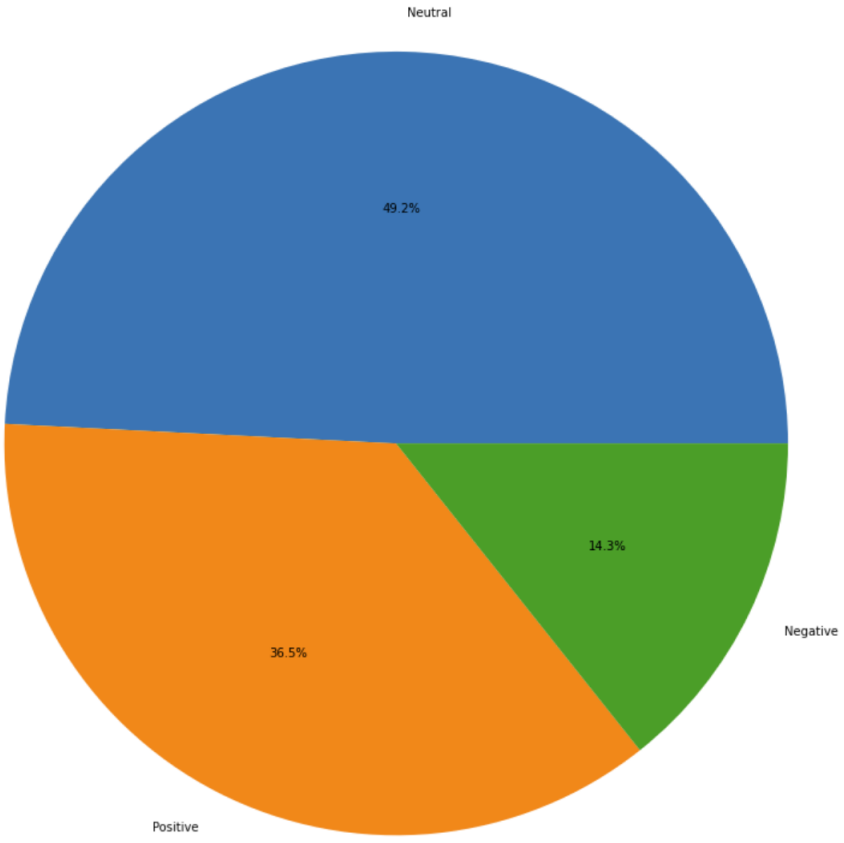


Figure 41: Sentiment pie chart

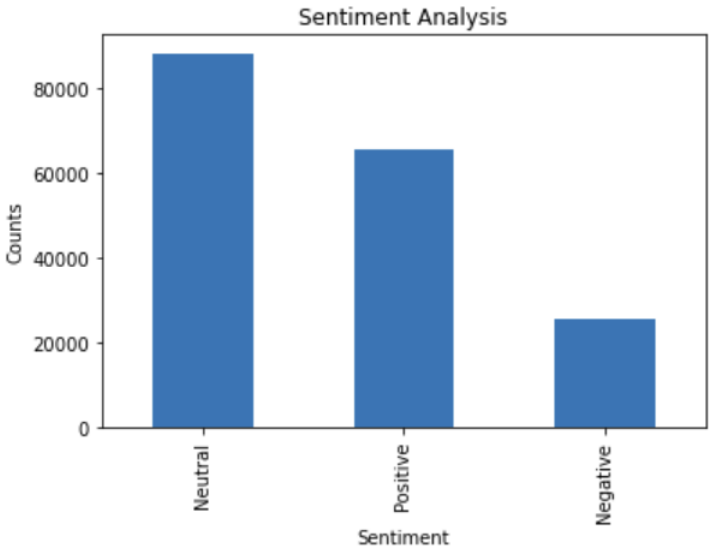


Figure 42: Sentiment bar plot

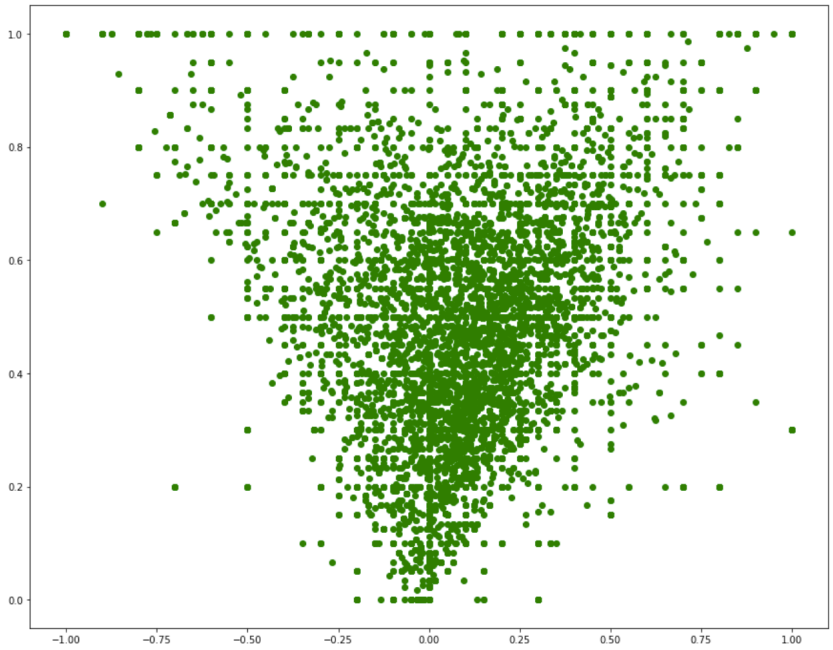


Figure 43: Polarity-Subjectivity scatter plot

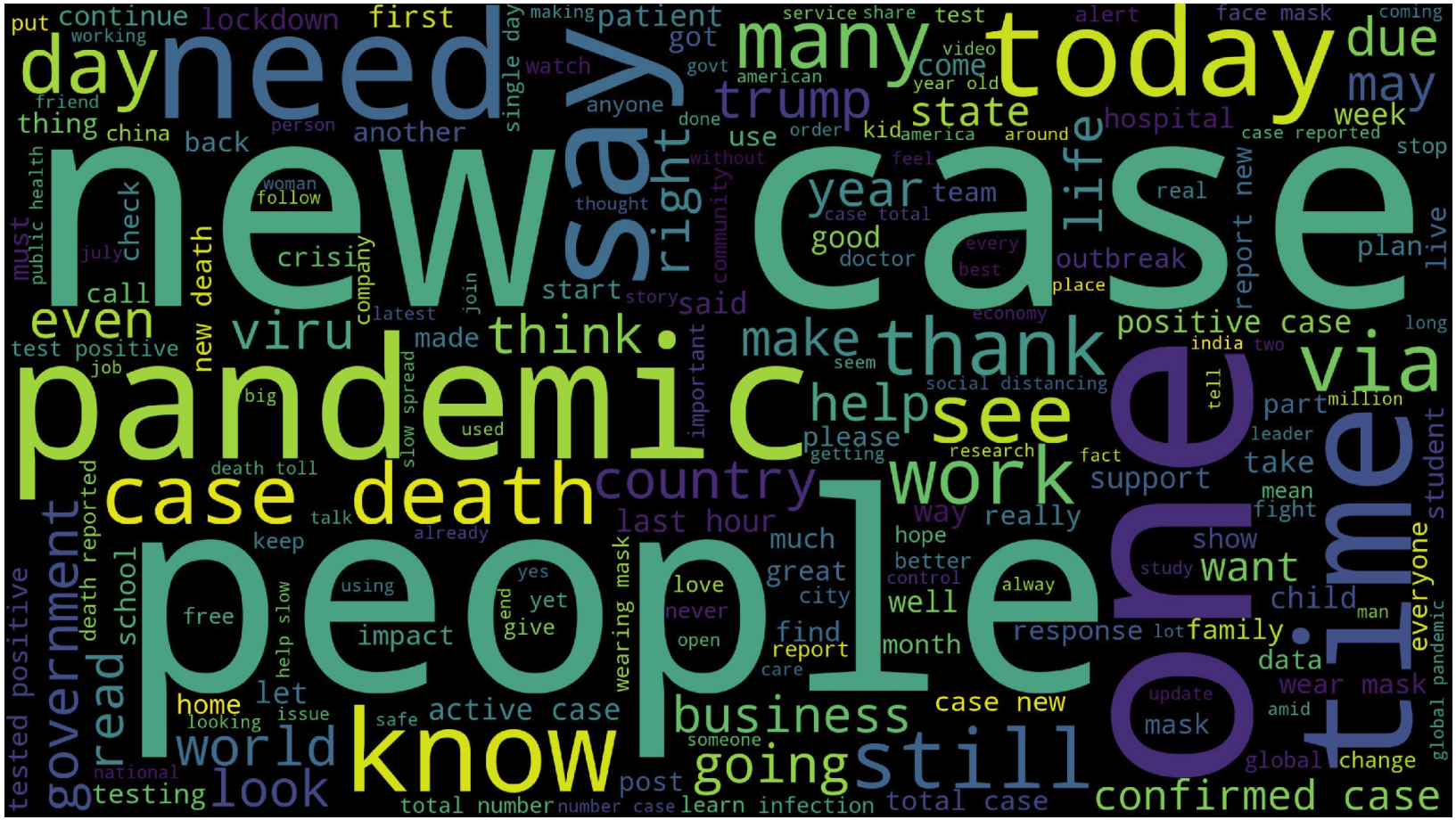


Figure 44: Word Cloud

5.3 Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. Convolutional neural nets, or CNNs, get their name from the concept of sliding (or convolving) a small window over the data sample, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. Regular Neural Nets do not scale well to big datasets/, fully-connected layers would have a large number of parameters and would quickly lead to overfitting. There are three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. We will stack these layers to form a ConvNet architecture.

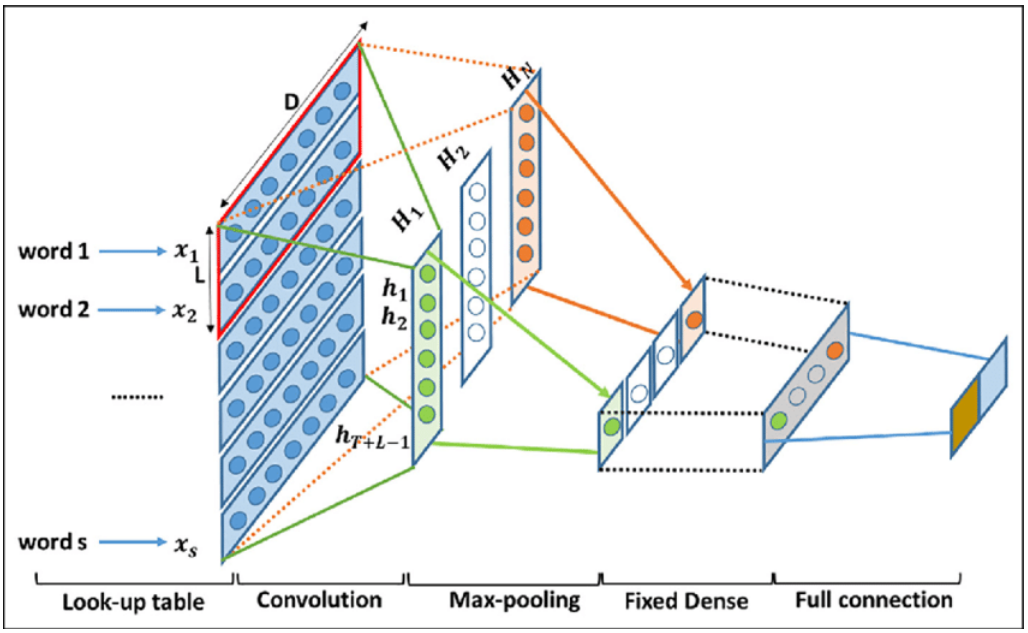


Figure 45: Convolutional Neural Network

Convolutional Layer

The convolutional layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. During the forward pass, we convolve each filter across the width and height of the input volume and compute dot products between the en-

tries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce an activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of feature. At the end, we will have an entire set of filters in each CONV layer, and each of them will produce a separate activation map. We will stack these activation maps along the depth dimension and produce the output volume.
When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the *receptive field* of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume.
We have explained the connectivity of each neuron in the Conv Layer to the input volume, but we have not yet discussed how many neurons there are in the output volume or how they are arranged. Three hyperparameters control the size of the output volume: the *depth*, *stride* and *zero-padding*.

1. The depth corresponds to the number of filters we would like to use, each learning to look for something different in the input.

2. The distance each convolution "travels" is known as the stride: we must specify the stride with which we slide the filter and is typically set to 1.

3. Sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.

Pooling Layer
It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Pooling is the convolutional neural network's path to dimensionality reduction: its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it using the MAX operation (*max pooling*). In addition to max pooling, the pooling units can also perform other functions, such as *average pooling* or even *L2-norm pooling*.

Dropout
Dropout is a special technique developed to prevent overfitting in neural networks. It isn't specific to natural language processing, but it does work well here. The

idea is that on each training pass, if we "turn off" a certain percentage of the input going to the next layer, randomly chosen on each pass, the model will be less likely to learn the specifics of the training set, "overfitting," and instead learn more nuanced representations of the patterns in the data and thereby be able to generalize and make accurate predictions when it sees completely novel data.

5.3.1 Implementation of a CNN

To carry out the analysis through the network we used a dataset of pre-labelized tweets as training for the network. **Sentiment tweets** is the Sentiment140 dataset. It contains 1,600,000 tweets extracted using the Twitter API. The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment.
Link: Sentiment tweets
As usual, we've made all the transformations necessary to get the cleaned tweets. We used Tokenizer to vectorize the text and convert it into sequence of integers after restricting the tokenizer to use only top most common 2000 words (fig.46). Then we used pad-sequences to convert the sequences into 2-D array.

```
max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X, padding='post', maxlen=40)
```

Figure 46: Tokenizer and pad_sequences

Finally we have built our model (fig.47) and compiled it using *binary crossentropy* as loss function and Nadam as optimizer. Then we have fitted the model using EarlyStopping which is a form of regularization used to avoid overfitting when training a learner with an iterative method. We have one hyperparameter: embed_dim. The embedding layer encodes the input sequence into a sequence of dense vectors of dimension embed_dim. Network training stopped after 14 epochs achieving the results shown in the table 1 below.

CNN results		
	Train	**Test**
Loss	*0.40*	*0.43*
Accuracy	*0.82*	*0.81*

Table 1

```
# Setting hyperparameters

embed_size = 128
```

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import *

overfitCallback = EarlyStopping(monitor='val_loss', min_delta=0, patience = 3)
```

```
cnn = Sequential([
    Embedding(max_features, embed_size, input_length = X.shape[1]),
    Conv1D(128, 3, padding='same', activation='relu'),
    Dropout(0.4),
    Flatten(),
    Dense(2, activation='softmax')
])

cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 40, 128)           256000
_________________________________________________________________
conv1d (Conv1D)              (None, 40, 128)           49280
_________________________________________________________________
dropout (Dropout)            (None, 40, 128)           0
_________________________________________________________________
flatten (Flatten)            (None, 5120)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 10242
=================================================================
Total params: 315,522
Trainable params: 315,522
Non-trainable params: 0
_________________________________________________________________
```

Figure 47: Convolutional Neural Network model

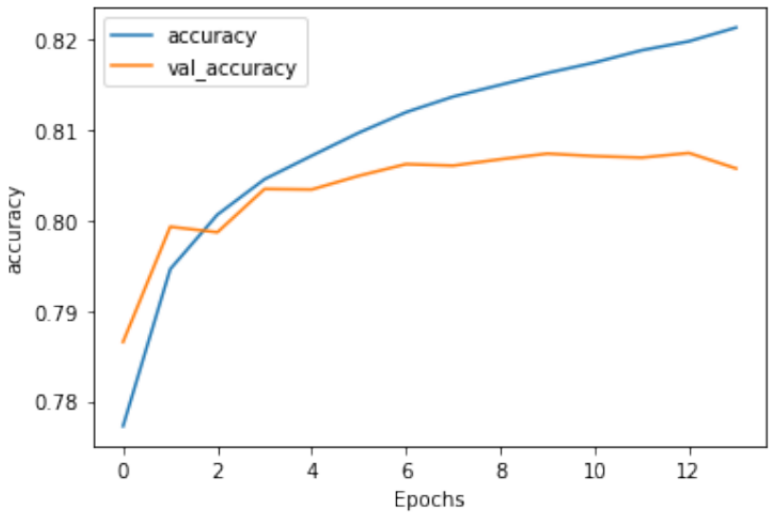


Figure 48: CNN accuracy

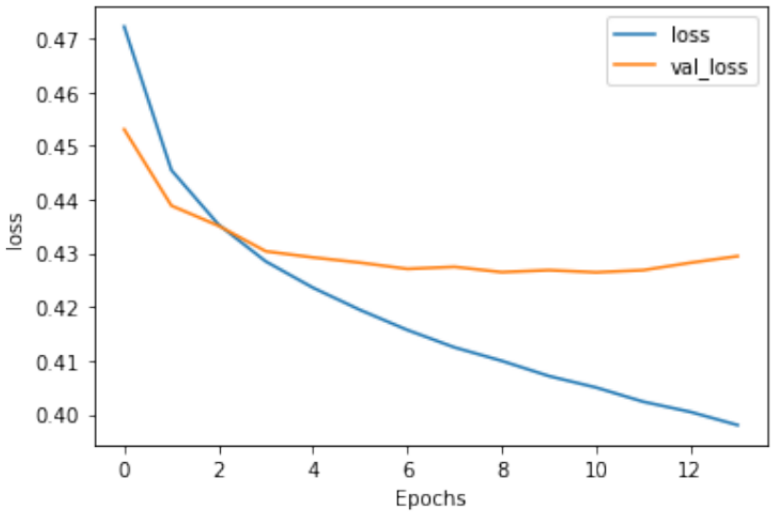


Figure 49: CNN loss

5.4 Long Short-term Memory - LSTM

Certain data types such as time-series, text, and biological data contain sequential dependencies among the attributes. Text is often processed as a bag of words, one can obtain better semantic insights when the ordering of the words is used. In such cases, it is important to construct models that take the sequencing information into account. Text data is the most common use case of recurrent neural networks.
A recurrent neural network (RNN) processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far. In effect, an RNN is a type of neural network that has an internal loop. However, a simple RNN is generally too simplistic to be of real use. Simple RNN has a major issue: although it should theoretically be able to retain at time *t* information about inputs seen many timesteps before, in practice, such long-term dependencies are impossible to learn. This is due to the vanishing gradient problem, an effect that is similar to what is observed with non-recurrent networks (feedforward networks) that are many layers deep: the network eventually becomes untrainable.

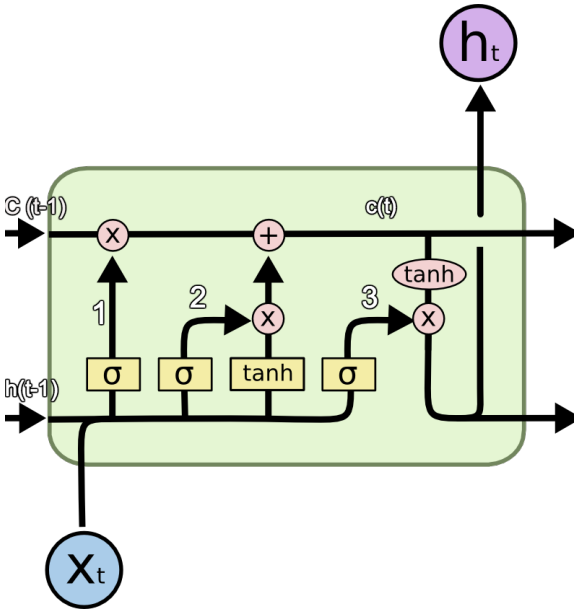


Figure 50: LSTM

The *LSTM (Long Short-Term Memory)* layer are designed to solve this problem, saving information for later, thus preventing older signals from gradually vanishing during processing. LSTM has a special architecture which enables it to forget the unnecessary information. The sigmoid layer takes the input *X(t)* and *h(t-1)* and decides which parts from old output should be removed (by outputting

a 0). This gate is called forget gate *f(t)*. The next step is to decide and store information from the new input *X(t)* in the cell state. A *Sigmoid* layer decides which of the new information should be updated or ignored. A *tanh* layer creates a vector of all the possible values from the new input. These two are multiplied to update the new cell sate. This new memory is then added to old memory to give the new one. Finally, a *Sigmoid* layer decides which parts of the cell state is going to output. Then, cell state pass through a *tanh* generating all the possible values and multiply it by the output of the sigmoid gate.

5.4.1 Implementation of a LSTM

For the LSTM we used same configuration and hyperparameters as CNN, obtaining the results below.

```
lstm = Sequential([
    Embedding(max_features, embed_size, input_length = X.shape[1]),
    Bidirectional(LSTM(50, dropout=0.5, return_sequences=True)),
    GlobalMaxPool1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])

lstm.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 40, 128)           256000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 40, 100)           71600
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                6464
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 130
=================================================================
Total params: 334,194
Trainable params: 334,194
Non-trainable params: 0
_________________________________________________________________
```

Figure 51: LSTM model

LSTM results		
	Train	**Test**
Loss	*0.42*	*0.42*
Accuracy	*0.81*	*0.81*

Table 2

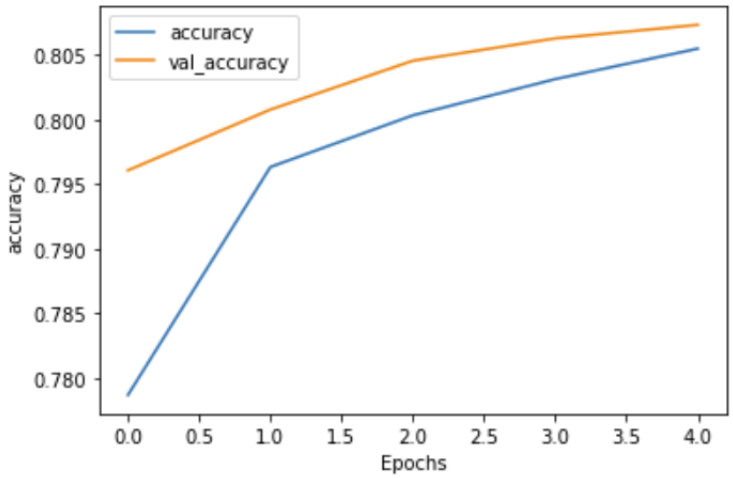


Figure 52: LSTM accuracy

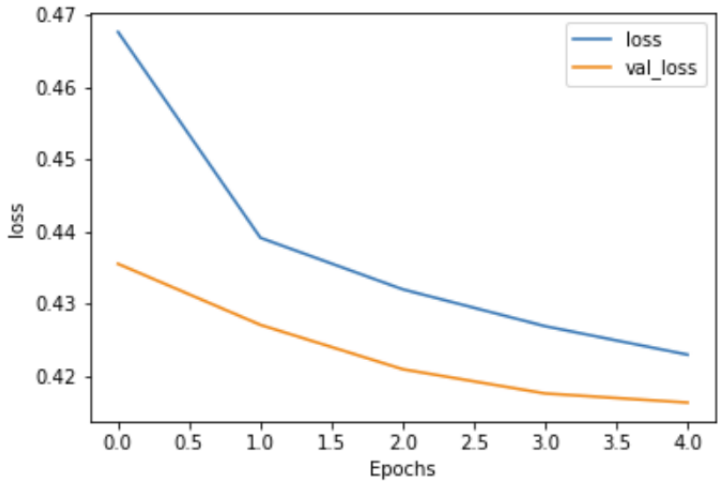


Figure 53: LSTM loss

5.5 Model validation

At this point we loaded the Covid-19 tweets dataset used for sentiment analysis with TextBlob to see how the model performs with this data.

```
score,acc = cnn.evaluate(X_val, Y_val, verbose = 1, batch_size = 32)
print("CNN score: %.2f" % (score))
print("CNN acc: %.2f" % (acc))
```

```
2844/2844 [==============================] - 8s 3ms/step - loss: 0.6099 - accuracy: 0.7126
CNN score: 0.61
CNN acc: 0.71
```

```
score,acc = lstm.evaluate(X_val, Y_val, verbose = 1, batch_size = 32)
print("LSTM score: %.2f" % (score))
print("LSTM acc: %.2f" % (acc))
```

```
2844/2844 [==============================] - 15s 5ms/step - loss: 0.6310 - accuracy: 0.6600
LSTM score: 0.63
LSTM acc: 0.66
```

Figure 54: Validation over Covid-19 tweets dataset