|  |  |  |
| --- | --- | --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |  |  |
| **CNLP Text Classification**  ML Python software for classifying medical article abstracts |  |  |
| |  |  |  | | --- | --- | --- | | Arapaut Sivaprasad | 02/02/21 | Machine Learning | |  |  |

**DOCUMENT DETAILS**

Document Name: cnlp-tc software documentation.docx

Software Filename: medical\_tc.py

Date Created: 2 February, 2021

Last Modified: 3 February 2021

Author: Arapaut V. Sivaprasad ([arapaut.sivaprasad@abs.gov.au](mailto:arapaut.sivaprasad@abs.gov.au); [avs@webgenie.com](mailto:avs@webgenie.com))

Copyright © 2021 by Australian Bureaue of Statistics and Arapaut V. Sivaprasad.

# ABSTRACT

Text classification is a challenging area and requires sophisticated algorithms and models to make correct predictions. This document describes a Python software that creates, trains and uses models based on word frequency. In instances where the model fails to predict a category, the Cosine Similarity is used to resolve the classification. The dataset consits of abstracts from medical journal articles on three topics, ‘cancer’, ‘covid’ and ‘heart’. Unlike other ML model training, this software is extremely fast and prepares the models in minutes instead of hours. Testing the model with data finishes in seconds. The average accuracy is 97.25% (range: 91% to 100%) over 3,700 lines of text. Unless it can be bettered, in terms of accuracy or speed, this software may be useful for classifying the census feedback. Since AWS platform is preferred, we will investigate and compare it with this software and will also look into incorporating this software with AWS Sage Maker.

**DISCLAIMER**

This is an evolving document and contains the code snippets, their explanations, and logics. The program logic and the code may change over time and may not be fully updated here. While every care has been taken to record accurately the process flow, omissions and errors may exist. These will be corrected as the document is updated. The code is mostly original but may contain snippets from third party sources. Acknowledgments to third parties are given only when there are sufficiently large blocks of code or data borrowed from other sources. The software uses only data and code available in public domain. While the software has been tested multiple times, cannot guarantee that it is free of bugs and performance issues. This software is provided as is without any warranty about its operation and accuracy.

Contents

[ABSTRACT 1](#_Toc63280120)

[INTRODUCTION 3](#_Toc63280121)

[BRIEF DESCRIPTION OF ALGORITHM 3](#_Toc63280122)

[Labelling 3](#_Toc63280123)

[WF creation 3](#_Toc63280124)

[NLP reference lines 4](#_Toc63280125)

[First level screening 4](#_Toc63280126)

[Second level screening 4](#_Toc63280127)

[Accuracy determination 4](#_Toc63280128)

[PROGRAM MODULES 4](#_Toc63280129)

[Raw Data 4](#_Toc63280130)

[metadata.csv: 4](#_Toc63280131)

[Create and train the model 4](#_Toc63280132)

[def prepare\_covid19\_data 5](#_Toc63280133)

[def create\_data\_file 5](#_Toc63280134)

[def create\_word\_frequecy\_dict 6](#_Toc63280135)

[def create\_docsnlp 7](#_Toc63280136)

[Testing the model 8](#_Toc63280137)

[def key\_value\_percent\_similarity 8](#_Toc63280138)

[RESULTS AN INTERPRETATION 10](#_Toc63280139)

[Interpretation 11](#_Toc63280140)

[Way forward 11](#_Toc63280141)

[GLOSSARY 12](#_Toc63280142)

# INTRODUCTION

Classifying text into known categories and creating new categories when enough texts do not fall into an existing category are the primary focuses of this software R&D. The ultimate objective is to classify the enormous amounts of Census feedback from users into a finite set of categories. Profanity detection is another objective, but it does not form part of this. Another software derived from this may be used for it.

This software uses a supervised learning to build and train the models. The data used in development comes from a collection of medical articles (abstracts) stored at [Kaggle](https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge). This data was collated in response to the COVID-19 pandemic by the White House and a coalition of leading research groups and is known as the COVID-19 Open Research Dataset (CORD-19). CORD-19 is a resource of over 400,000 scholarly articles, including over 150,000 with full text, about COVID-19, SARS-CoV-2, and related coronaviruses. It is freely available to the research community. Though it has been prepared in response to COVID pandemic, the collection has articles on many subjects dating back several years. The vast majority are COVID-related articles.

For developing the text classification software (“cnlp-tc”) the above data has been parsed to take the relevant abstracts for three medical areas *viz*., *cancer, covid* and *heart*. For further testing we may be able to source more data related to other medical terms. Each row in the parsed dataset (in CSV format) will be called a ‘document’ or ‘line’ in the following sections.

# BRIEF DESCRIPTION OF ALGORITHM

The data as one document per line is collected and labelled as ‘cancer’, ‘covid’ or ‘heart’. We use “Word Frequency (WF)” and “Cosine Similarity (CS)” methods to compare the docs and to classify. WF is calculated by the software itself and is not using Word2Vec. CS is calculated using vectorized model ‘en’ but we may try other models such as ‘en\_core\_web\_sm’, ‘en\_core\_web\_md’ or ‘en\_core\_web\_lg’. The ‘en’ model was used primarily because of my previous observation that the ‘sm’ model does not have vectors, ‘md’ lacks vectors for technical words and ‘lg’ is too large for our needs. This choice can be re-visited later.

## Labelling

Correct labelling of the data is of utmost importance in supervised learning. Otherwise, it can lead to incorrect model and in turn wrong classifications.

See later for details of the various algorithms that are briefly described here.

The first step in **cnlp-tc** was to collect the articles belonging to the three categories. For this, a simple algorithm to look for keywords was used. Though there will inevitably overlaps between the categories, heuristically a record that contains at least three specific keywords for a category, while not having such for the other two, should safely give a set of documents for creating the models.

## WF creation

To create the WF a concatenated line, of maximum 1M characters, from each category was selected. After removing the stop words and common English words from these lines, which generally reduced the length to about half, the line was split into words and the relative frequencies of words were recorded.

## NLP reference lines

To calculate Cosine Similarity, a list of 10 lines from each category was selected at random, tokenised and stored in a dictionary object. The test line, when necessary to compare the CS score, was compared with these reference lines.

## First level screening

Since CS scoring takes longer than WF calculation, the first screening was by taking a line and comparing it against the WF for the category. The scores are normalised by taking relative frequencies into an array. The category that scored the highest is taken.

## Second level screening

If the category selected in the first level screening is different to the category name recorded[[1]](#footnote-1) in the line, then the CS scoring is used to do a second level of screening. The CS scoring is used when the WF scores are close together for all categories as well.

## Accuracy determination

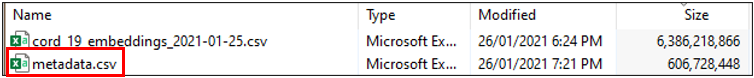
The accuracy of the method is the percentage of total lines that are correctly predicted, with WF or CS, to belong in a category. Each line will have a recorded category name and a predicted category. If these are different the result is negative. A ‘mis-match’ is one where the WF score stands out, but the category is different. A ‘new category’[[2]](#footnote-2) is where the WF scores are close together and CS scoring did not pick the correct category.

# PROGRAM MODULES

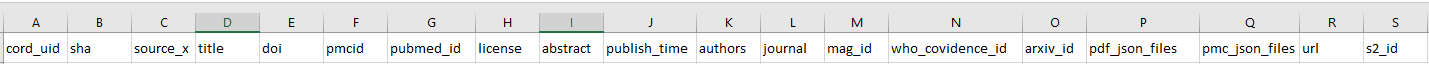
The software is modular and easy to understand. Docstrings and inline comments explain the operation of each function. However, beware of some outdated comments and docstrings. Given below are high level descriptions of the functions that will complement the comments and docstrings within the program.

## Raw Data

The data is [downloaded](https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge) from Kaggle, a subsidiary of Google LLC. The ZIP file, ‘archive.zip’, contains several files of which only the highlighted one is required for our purpose.



### metadata.csv:



There are 433,162 lines in this file of which 432,941 have non-blank titles and 312,618 non-blank abstracts. The columns ‘title’ and ‘abstract’ are copied to a TSV file. This file is now parsed as given below.

## Create and train the model

Functions described below prepare the data and then create the models and data for comparing the documents against. They are required only once, and the outputs are stored as pickle files.

### def prepare\_covid19\_data

Both cells in each row are concatenated to form a single line and all non-alphanumeric characters removed and double spaces truncated to singles. See code snippet below.

*content = cols[0] + ' ' + cols[1]  
content = re.sub('[^a-zA-Z0-9 -]', ' ', content).strip()  
content = ' '.join(content.split())*

Every line is marked as category ‘medical’ and rewritten into a CSV file as below[[3]](#footnote-3).



*category = "medical"  
comple\_filename = file\_name + "-" + category   
line = f"{file\_name};{content};{category};{comple\_filename}\n"  
fo.writelines(line)*

Output file: covid\_19.csv

### def create\_data\_file

This function parses the ‘covid\_19.csv’ to pick out lines corresponding to three categories having keywords as below.

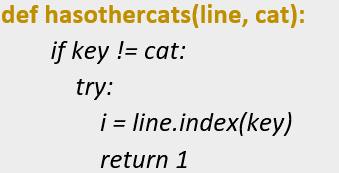
*keywords = ['cancer', 'covid', 'heart']*

Each keyword is used as a key to a dictionary object that holds specific search terms[[4]](#footnote-4) for each category.

*kw\_dict = {****'covid'****: ['covid', 'sars', 'sarscov', 'wuhan', 'astrazeneca', 'coronavirus', 'pandemic', 'lockdown'],****'cancer'****: ['cancer', 'malignancy', 'malignant', 'metastasis', 'metastatic', 'prostate', 'anticancer', 'catechins', 'apoptosis'],****'heart'****: ['heart', 'angiotensin', 'angiography', 'cardiomyopathy', 'cardiac', 'electrocardiogram' 'coronary', 'hypertension']}*

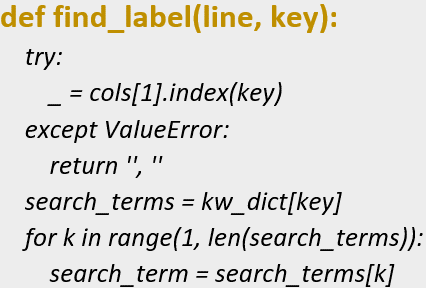
The lines are parsed to pick those which have the current category keyword, key (e.g. ‘covid’) and at least one search term in ‘kw\_dict[key]’. The line should not have the category keyword for the other two categories.

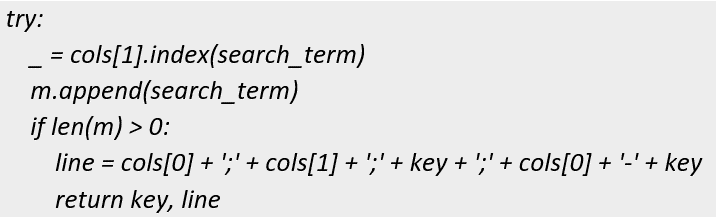
*jj = hasothercats(line, key)  
if jj:  
 continue*



If the line has only the current category key, then see if it has at least one search term for that category.

*cat, line = find\_label(line, key)  
if line:  
 fo.writelines(f"{line}\n")*





Output file: multi\_category.csv



### def create\_word\_frequecy\_dict

The word frequency (WF) is the main component for comparing text. It is created and saved as a dictionary object and written out as {key}\_kvp.pickle. The dictionary data object will be created upon reading the saved file. The pickle file is binary and is like the output from Perl’s storage module. The purpose is to save time creating the data object by creating and saving it once for multiple reads.

#### Process details

The input file, 'multi\_category.csv' created by 'def create\_data\_file' has the lines for three categories as below.

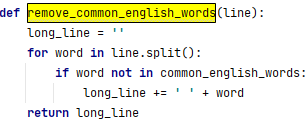
*line\_nn;content;category;line\_nn-category  
 e.g. line\_48;epicatechins purified...;cancer;line\_48-cancer*

The ‘content’ part is taken to calculate the WF. It has been screened for stop words so that common words like ‘the’, ‘an’, ‘of’, etc. will not be present. However, there are the common English words which, if not removed, could add to the noise, and diminish the distinction between categories. These are removed before calculating the WF.

The first step is to concatenate the content to form a long line. An arbitrary maximum length of 1M characters is set for this line.

*for category in categories:  
 long\_line = ""  
 for i in range(0, j):  
 cols = lines[i].split(';')  
 if cols[2] != category:  
 continue  
 line = cols[1]  
 long\_line += " " + line  
 if len(long\_line) > 1000000:  
 break  
 long\_line = remove\_common\_english\_words(long\_line)*

The common English words are downloaded from [here](https://www.ef-australia.com.au/english-resources/english-vocabulary/top-3000-words/).



The WF are calculated by splitting the long line and taking the words from which the hyphen characters (‘-’) are removed. This is to convert words like ‘Covid-19’ and ‘SARS-COVID’ into a uniform format.

*for word in long\_line.split():  
 word = word.replace('-', '')*

To make the word list for categories as unique as possible, the most common words in other categories are removed. This is an arbitrary step and may have to be re-visited.

*if category == 'cancer':  
 if word in* [*kw\_dict*](#kw_dict)*['covid'] or word in kw\_dict['heart']:  
 continue*

The above step was enough to work with categories, ‘covid’ and ‘heart’, possibly because they had relatively low vocabularies. For ‘cancer’ it was necessary to remove some seemingly unrelated words that appeared often to get a high accuracy. This is arbitrary and it may be required to remove others as well.

*cancer\_stopwords = ['purified', 'differentially']  
 if word in cancer\_stopwords:  
 continue*

Now, the word counts are added into a dictionary object.

*wordcount = {}  
if word not in wordcount:  
 wordcount[word] = 1  
else:  
 wordcount[word] += 1*

*#cancer = {'epicatechins': 1,* *'malignancy': 110,…}*

The above dictionary object is sorted on the word count. This is done initially to take only the top-ranking words for further analyses, but later it was decided to take all words. Hence, the code below is currently not critical. It can be replaced with the ‘wordcount’ dict object.

*word\_counter = collections.Counter(wordcount)  
 Counter({'patients': 1564, 'cells': 909,…}*

Convert the above into relative word frequencies. *for key, value in word\_counter.items():  
 tot\_value += value  
 key\_value\_perc = {}  
 for key, value in word\_counter.items():  
 value\_perc = value / tot\_value  
 key\_value\_perc[key] = value\_perc*

*key\_value\_perc:  
 cancer = {'epicatechins': 1.401227475268335e-05, 'patients': 0.02191519771319676,…}*

Save *key\_value\_perc* in a pickle file.

*filename = pickles\_dir + category + "\_kvp.pickle" # cancer\_kvp.pickle  
 with open(filename, 'wb') as output:  
 pickle.dump(key\_value\_perc, output)*

### def create\_docsnlp

This function creates and stores 10 lines each from the categories for CS scoring against if the WF does not give a conclusive result. The objective is to select lines that are representative of each category with little or no overlap with other categories. When a test has picked up two or more categories as possibilities, the line is CS scored against these lines and the one that gives the highest score is selected as its category.

*random.shuffle(lines) # Shuffle the lines in multi\_category.csv  
 docsnlp = {'cancer': [], 'covid': [], 'heart': []} # Dict object. Each key will hold an array of 10 lines.*

*def read\_dictionary():  
 model = 'en'   
 nlp = spacy.load(model)  
 return nlp*

*cols = lines[i].split(';')  
 line = cols[1]  
 if cols[2] == cat:  
 k =* [*hasothercats*](#hasothercats)*(line, cat)  
 if k == 0:  
 n += 1  
 docnlp =* [*nlp*](#nlp)*(line)  
 docsnlp[cat].append(docnlp)  
 if n > 9:  
 break*

The dict object is saved as a pickle file.

*filename = pickles\_dir + "docsnlp.pickle"  
 with open(filename, 'wb') as output:  
 pickle.dump(docsnlp, output)*

## Testing the model

Testing the model aims to find out its efficacy in correctly classifying the documents. This is an iterative process to arrive at the best algorithm. Labelled data, which has been used in creating the model or new, will be used in this. Accuracy of the predictions is determined as the percentage of documents correctly predicted. An accuracy of >90% is aimed for by using the training data, but it may not be achievable with a different set of data. This remains to be seen.

Once a model that gives the highest accuracy is reached, the process ends and the model can be deployed for actual classification of new data.

Two separate methods to classify are used in succession. The first uses the WF-based model and, if the result is inconclusive, the CS method is applied. The CS method may probably be standalone, but the high CPU time for it may become a limiting factor in large data sets. If CS method alone is used, and there are more than three categories to test, the execution time will increase exponentially. The current software has, therefore, implemented CS as the second step alone and doing only a single pair of comparison at any time. The WF method is virtually instantaneous.

### def key\_value\_percent\_similarity

This function is called as below.

*match, mismatch, doubles, new\_cat, ii = key\_value\_percent\_similarity(b, e, lines, nlp, w)*

where,

**b** = Start line number

**e** = End line number

**lines** = The entire content of the CSV file (multi\_category.csv)

**nlp** = The standard English words model.

**w** = Worker number when run in parallel mode. Parallelisation has been implemented but not used, as the process is already fast.

**match** = Number of lines where the recorded category equals the predicted category. This value is returned either by the WF method or the CS method called after it.

**mismatch** = Those which could not be matched.

**doubles** = The cases where the difference between the recorded and predicted categories is very small and hence the line could be classified as belonging to both categories.

**new\_cat** = The scores for all three categories are close together. Hence, the line does not belong to any of the categories. By further analysis of these lines we can pick up word(s) that may form a new category. This part has not yet been implemented.

**ii** = Total number of lines tested. We calculate the accuracy as ‘match/ii \*100’.

#### Load the models

The WF-based models for each category are loaded as below. Only ‘cancer’ model is shown.

*kvp1 = pickles\_dir + "cancer\_kvp.pickle"  
 with open(kvp1, 'rb') as data:* ***cancer\_kvp*** *= pickle.load(data)*

The reference lines for CS scoring are loaded as below.

*infile = pickles\_dir + "docsnlp.pickle"  
 with open(infile, 'rb') as data:* ***docsnlp*** *= pickle.load(data)*

The lines in the CSV file (multi\_category.csv) are now enumerated and compared firstly with WF.

*s, ss,* ***cat, this\_cat*** *= get\_this\_cat(lines, i, s\_cat, cancer\_kvp, heart\_kvp, covid\_kvp)*

#### def get\_this\_cat

This function checks the line against the word frequency model (e.g. cancer\_kvp) to total the frequencies of words in the line for each category.

*e.g. cancer\_kvp = {'epicatechins': 1.6790637540507413e-05,...}*

Thus, for example, for the word ‘epicatechins’, the WF in cancer\_kvp is 1.6790637540507413e-05. This word may or may not be in the other categories and, if present, will usually have a different value. Adding up all values in all category models will result in an array as below. The array indices are [0=cancer, 1=covid, 2=heart].

*s = [0.13738099635643172, 0.04223915521689565, 0.07645095294497688]*

Each value in the above array is then converted into a percentage of total. and added to the list, 'ss'. Further analysis depends on the difference between the scores.

*[****53.65****, 16.5,* ***29.86****] – Predicted category (this\_cat) =* ***cancer****, recorded (cat):* ***cancer****. Diff between cat and this\_cat = 23.79  
[****36.05****, 28.73,* ***35.22****] – this\_cat =* ***cancer****, cat:* ***cancer****. Diff between cat and this\_cat = 0.83  
[****33.22****, 30.73,* ***36.05****] – this\_cat =* ***heart****, cat:* ***cancer****. Diff between cat and this\_cat = 2.83  
[****33.22****,* ***35.05****,* ***31.73****] – this\_cat =* ***covid****, cat:* ***heart****. Diff between largest and smallest = 3.32*

#### Heuristic rules:

The values used below are arbitrary and are not critical.

* Predicted and recorded categories are the same:
  + Accept without further checking if the difference is > 1.00%
  + Check CS score if the diff is < 1.00%
* Predicted and recorded categories are different:
  + Always check the CS score, but there are two scenarios.
    - The maximum and minimum vales are less than 5% apart.
      * Check for CS and, if no match, call it a new category.
    - Maximum and minimum are more than 5% apart.
      * Top two are less than or equal to 5%
        + Check for CS and, if no match, call it a double category.
      * Top two are more than 5%
        + Check for CS and call it a mismatch.

In the first row above, the difference between the value for cancer and the second nearest is ~24%. This value is large enough to call the prediction correct. No further analysis is required.

In the second line, the difference is less than 1%. Though it has called the category correctly, this must be further checked. If no match in the CS checking it is called a mismatch.

In the third line, predicted and recorded categories are different, but the difference is less than 5%. This will be CS checked as above and called a mismatch if no match.

In the fourth line, the diff between maximum and minimum is less than 5%. This will be CS checked and called a new category if no match.

#### def compare\_CS

This function calculates the Cosine Similarity between two texts. The reference text is the line to be tested against different category lines stored in ‘[docsnlp](#docsnlp)’. The CS values are stored in two lists, ‘cs1’ and ‘cs2’ and the category for the list that has the highest value among the 20 values between the lists is taken as the matching category.

*cs1 = []  
 cs2 = []  
 doc1nlp = nlp(this\_line)  
 for doc2nlp in docsnlp[cat]:  
 cs1.append(doc1nlp.similarity(doc2nlp))  
 for doc2nlp in docsnlp[this\_cat]:  
 cs2.append(doc1nlp.similarity(doc2nlp))*

*if max(cs1) > max(cs2):  
 this\_cat = cat  
cs1:  
cancer [0.9237732313597005, 0.9484085805237733, 0.9425249510737472, 0.9102426063470225, 0.9565340838404702, 0.936145870765263, 0.952712852989643, 0.964917437950997, 0.9522471280379634, 0.8776091968511329]*

*cs2:  
heart [0.9308122556511905, 0.959679104848698, 0.9612861189098855, 0.9626877962068543, 0.9394829657319225, 0.9331936038480758, 0.9482550338278134, 0.959163450080754, 0.9562485971584624, 0.9597292836043037]*

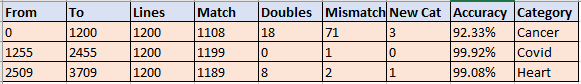
*match: cancer*

Alternative to taking the maximum CS value between the two categories is to take the highest average between the two. So far the first method gives higher accuracy. The validity of this logic must be revisited.

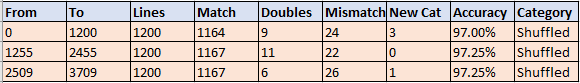
# RESULTS AN INTERPRETATION

Tests were conducted in batches of 1200 lines using the same CSV file used to create the models. This file contains 1254 lines for three categories in the order of cancer, covid and heart.

When tested with lines exclusively from one category at a time, the results were different between the tests. Thus, the category ‘cancer’ gave lower accuracy values than ‘covid’ and ‘heart’.



When the data is shuffled so that the category lines are not grouped together, the accuracy values were uniform across the whole file.



When the whole data was tested, there was no difference in the accuracy percentage between shuffled and unshuffled data.



## Interpretation

The relatively lower accuracy for cancer lines (92.33%) which is still a high rate that can be expected from text classifications in general, could be due to several factors. Most probable cause is that the topic, ‘cancer’, covers a wide range of symptoms and research. Hence, the vocabulary for articles in it would be larger than the areas of research that are narrower. Covid research is very new and may have the lowest vocabulary. Heart diseases are as common as cancer, but the medical terms for it are not as widely variable as for cancer.

Another, more probable, reason is that the labelling is not accurate for cancer lines. There maybe incorrect labelling of lines belonging to the other categories or completely unrelated lines. This must be investigated further.

## Way forward

To prove that the method used here is valid, we need to repeat it by using…

1. Use the same models with new set of data that has not gone into making the models.
2. Create new models from other medical terms (e.g. diabetics, influenza, smoking).

In parallel with the above, we can investigate incorporating this into AWS Sage Maker. The data should give similar results if AWS modelling is as good as or better than this one.

If it does not work with AWS Sage Maker, we can investigate incorporating this software into Sage Maker. I saw somewhere that one’s own software can be incorporated.

This software was inspired by a public domain program to classify the news data but used a different logic due to the excessive time it took to create and train the models. Repeating it with smaller data set should be investigated.

\_\_\_\_\_END OF SECTION\_\_\_\_\_

# GLOSSARY

Terms are listed in the order they appear in the document.

* COVID-19 – Corona Virus Disease 2019
  + (or Chinese Origin Virus Disease?)
* CORD-19 - COVID-19 Open Research Dataset Challenge.
* SARS – Severe Acute Respiratory Syndrome. The family of virus linked to COVID-19
* SARS-COV-2: Specific name of the COVID virus.
* Kaggle - a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.
* NLP: Natural Language Processing
* nlp: English text vectorised data model.
* CNLP-TC: Census Natural Language Processing – Text Classification.
* CS: Cosine Simliarity. Statistical value representing how similar two texts are.
* WF: Word Frequency. Relative frequencies of words in a text.
* Word2vec: a technique for natural language processing, using vectorised words.
* en\_core\_web\_sm: English pipeline optimized for CPU. Components: tok2vec, tagger, parser, senter, ner, attribute\_ruler, lemmatizer. Size: 13MB. No vector data.
* en\_core\_web\_md: Same as above. 44MB. 685k keys, 20k unique vectors (300 dimensions).
* en\_core\_web\_lg: Same as above. 742MB. 685k keys, 685k unique vectors (300 dimensions)
* Labelling: Identifying a text line with a category name.
* Accuracy: Percentage of lines correctly predicted.
* Parallel computing: Execution of code on more than one CPU simultaneously. Requires specific logics in the code.
* AWS Sage Maker: Machine Learning and text classification service from AWS.

\_\_\_\_\_END OF SECTION\_\_\_\_\_

1. In the training and test data sets each line has a label corresponding to the category. Later, when using new data, this label will not be present. Hence, it is necessary to make sure that the highest accuracy is achievable with the test data. [↑](#footnote-ref-1)
2. It is necessary to further analyse these before calling a new category. It will be added later. [↑](#footnote-ref-2)
3. The column headings, ‘File\_name, Content, Category and Complete\_Filename’, are vestiges from a previous program and can be changed.. [↑](#footnote-ref-3)
4. These search terms are arbitrarily selected. [↑](#footnote-ref-4)