|  |
| --- |
| 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 | 2/2/21 | Machine Learning | |

**DISCLAIMER**

This document contains the original code snippets and their explanations. The program logic and the code may change over time and may not be fully updated here. While every care has been given to record accurately the process flow and logics, 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. Only when there are sufficiently large blocks of code borrowed from other sources are they acknowledged. It 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 or performance issues. This software and documentation are provided as they are without any warranty about its operation and accuracy.

**DETAILS**

Document Name: cnlp-tc software documentation

Software Name: cnlp-tc

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.

# 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 software. 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.

# 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

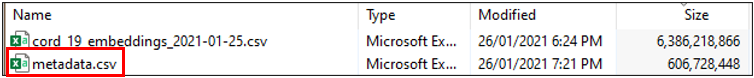
The accuracy of the method is the percentage of total lines that are correctly predicted, with WF or CS, to belong in a category. Each line will have a recorded category name and a predicted category. If these are different the result is negative. A ‘mis-match’ is one where the WF score stands out, but 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

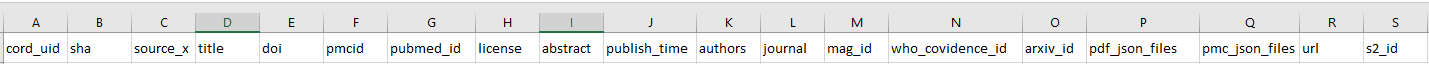
The software is modular and easy to understand. Docstrings and inline comments explain the operation of each function. However, beware of some outdated comments and docstrings. Given below are high level descriptions of the functions that will complement the comments and docstrings within the program.

## 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.

### 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.

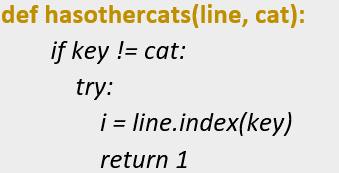
*keywords = ['cancer', 'covid', 'heart']*

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']}*

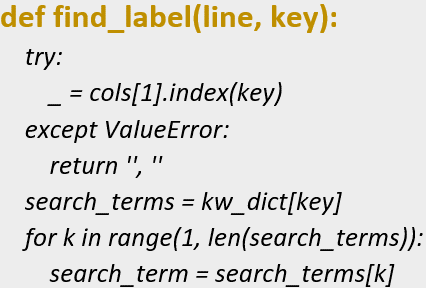
The lines are parsed to pick those which have the current category keyword, key (e.g. ‘covid’) and at least one search term in ‘kw\_dict[key]’. The line should not have the category keyword for the other two categories.

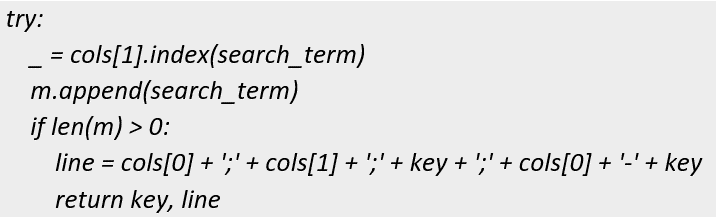
*jj = hasothercats(line, key)  
if jj:  
 continue*



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



### 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.

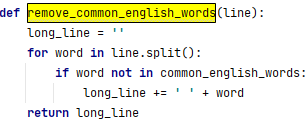
*line\_nn;content;category;line\_nn-category  
 e.g. line\_48;epicatechins purified...;cancer;line\_48-cancer*

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)*

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)