Text Rank Algorithm

December 30, 2020

Text Rank algorithm is - **extractive summarization technique**. The assigned problem can be classified as **a single-domain-multiple-document summarization task** which means multiple articles generate a single bullet-point summary on Covid19 dataset

Code reference

[1]:
```
import math
import networkx as nx
import nltk
nltk.download('punkt') # one time execution
import numpy as np
import os
import pandas as pd
import re

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tag import pos_tag # for proper nouns
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Divyank\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

1 Custom functions

1. Cue phrases

[2]:
```
def cue_phrases():
    QPhrases=["incidentally", "example", "anyway", "furthermore","according"
            "first", "second", "then", "now", "thus", "moreover", "therefore",␣
 ↪"hence", "lastly", "finally", "summary"]
    cue_phrases={}
    for sentence in sent_tokens:
        cue_phrases[sentence] = 0
        word_tokens = nltk.word_tokenize(sentence)
        for word in word_tokens:
            if word.lower() in QPhrases:
                cue_phrases[sentence] += 1

    maximum_frequency = max(cue_phrases.values())

    for k in cue_phrases.keys():
        try:
            cue_phrases[k] = cue_phrases[k] / maximum_frequency
            cue_phrases[k] = round(cue_phrases[k], 3)
        except ZeroDivisionError:
            x = 0

    print(cue_phrases.values())

    return cue_phrases
```

2. Numerical data

[3]:
```
def numeric_data():
    numeric_data = {}

    for sentence in sent_tokens:
        numeric_data[sentence] = 0
        word_tokens = nltk.word_tokenize(sentence)

        for k in word_tokens:
            if k.isdigit():
                numeric_data[sentence] += 1

    maximum_frequency = max(numeric_data.values())

    for k in numeric_data.keys():
        try:
            numeric_data[k] = (numeric_data[k]/maximum_frequency)
            numeric_data[k] = round(numeric_data[k], 3)
        except ZeroDivisionError:
            x = 0

    print(numeric_data.values())

    return numeric_data
```

3. Sentence length

[4]:
```
def sent_len_score():
    sent_len_score={}

    for sentence in sent_tokens:
        sent_len_score[sentence] = 0
        word_tokens = nltk.word_tokenize(sentence)

        if len(word_tokens) in range(0,10):
            sent_len_score[sentence] = 1 - 0.05 * (10 - len(word_tokens))

        elif len(word_tokens) in range(7,20):
            sent_len_score[sentence] = 1

        else:
            sent_len_score[sentence] = 1 - (0.05) * (len(word_tokens) - 20)

    for k in sent_len_score.keys():
        sent_len_score[k] = round(sent_len_score[k], 4)

    print(sent_len_score.values())

    return sent_len_score
```

4. Sentence position

[5]:
```
def sentence_position():
    sentence_position={}

    d = 1
    no_of_sent = len(sent_tokens)

    for i in range(no_of_sent):
        a = 1/d
        b = 1/(no_of_sent-d+1)
        sentence_position[sent_tokens[d-1]] = max(a,b)
        d += 1

    for k in sentence_position.keys():
        sentence_position[k] = round(sentence_position[k], 3)

    print(sentence_position.values())

    return sentence_position
```

5. Frequency table

[6]:
```
def word_frequency():
    freqTable = {}

    for word in word_tokens_refined:
        if word in freqTable:
            freqTable[word] += 1
        else:
            freqTable[word] = 1

    for k in freqTable.keys():
        freqTable[k] = math.log10(1 + freqTable[k])

    #Compute word frequnecy score of each sentence
    word_frequency = {}

    for sentence in sent_tokens:
        word_frequency[sentence] = 0
        e = nltk.word_tokenize(sentence)
        f = []

        for word in e:
            f.append(PorterStemmer().stem(word))

        for word,freq in freqTable.items():
            if word in f:
                word_frequency[sentence] += freq

    maximum = max(word_frequency.values())

    for key in word_frequency.keys():
        try:
            word_frequency[key] = word_frequency[key]/maximum
            word_frequency[key] = round(word_frequency[key],3)
        except ZeroDivisionError:
            x = 0

    print(word_frequency.values())

    return word_frequency
```

6. Upper case

[7]:
```
def upper_case():
    upper_case={}

    for sentence in sent_tokens:
        upper_case[sentence] = 0
        word_tokens = nltk.word_tokenize(sentence)

        for k in word_tokens:
            if k.isupper():
                upper_case[sentence] += 1

    maximum_frequency = max(upper_case.values())

    for k in upper_case.keys():
        try:
            upper_case[k] = (upper_case[k]/maximum_frequency)
            upper_case[k] = round(upper_case[k], 3)
        except ZeroDivisionError:
            x = 0

    print(upper_case.values())

    return upper_case
```

7. Proper nouns

[8]:
```
def proper_noun():
    proper_noun={}

    for sentence in sent_tokens:
        tagged_sent = pos_tag(sentence.split())
        propernouns = [word for word, pos in tagged_sent if pos == 'NNP']
        proper_noun[sentence]=len(propernouns)

    maximum_frequency = max(proper_noun.values())

    for k in proper_noun.keys():
        try:
            proper_noun[k] = (proper_noun[k]/maximum_frequency)
            proper_noun[k] = round(proper_noun[k], 3)
        except ZeroDivisionError:
            x = 0
    print(proper_noun.values())

    return proper_noun
```

8. Word matches with heading

[9]:
```
def head_match():
    head_match={}
    heading=sent_tokens[0]
    for sentence in sent_tokens:
        head_match[sentence]=0
        word_tokens = nltk.word_tokenize(sentence)

        for k in word_tokens:
            if k not in my_stopwords:
                k = PorterStemmer().stem(k)

                if k in PorterStemmer().stem(heading):
                    head_match[sentence] += 1

    maximum_frequency = max(head_match.values())

    for k in head_match.keys():
        try:
            head_match[k] = (head_match[k]/maximum_frequency)
            head_match[k] = round(head_match[k], 3)
        except ZeroDivisionError:
            x = 0

    print(head_match.values())

    return head_match
```

2 Creating data frame

[10]:
```
df = pd.DataFrame(columns=['a','b','c', 'd', 'upper', 'f','g','h','key','label'])
print(df)
```

```
Empty DataFrame
Columns: [a, b, c, d, upper, f, g, h, key, label]
Index: []
```

[11]:
```
path = 'D:/COVID_19_dataset/documents/'

filelist = os.listdir(path)

for file in filelist:
    f = open(path + file, "r")
    text = f.read()

    sent_tokens = nltk.sent_tokenize(text)
    word_tokens = nltk.word_tokenize(text)
    word_tokens_lower = [word.lower() for word in word_tokens]

    my_stopwords = list(set(stopwords.words('english')))
    word_tokens_refined = [x for x in word_tokens_lower if x not in my_stopwords]

    Cue_phrases = list(cue_phrases().values())
    Key = list(cue_phrases().keys())
    Numeric_data = list(numeric_data().values())
    Sent_length_score = list(sent_len_score().values())
    Sentence_position = list(sentence_position().values())
    Upper_case = list(upper_case().values())
    Header_match = list(head_match().values())
    Word_frequency = list(word_frequency().values())
    Proper_noun = list(proper_noun().values())

    label = {}
    for sentence in sent_tokens:
        label[sentence] = 0

    o = list(label.values())
    df = df.append(pd.DataFrame({'a': Cue_phrases,'b': Numeric_data,'c':␣
↪Sent_length_score,'d': Sentence_position,
                              'upper': Upper_case, 'f': Header_match, 'g':␣
↪Word_frequency,'h': Proper_noun,
                              'key': Key,'label': o}), ignore_index = True)

    f.close()
```

```
dict_values([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0])
dict_values([0.8, -2.25, 1, -0.1, 1.0, 1, 0.6, 0.35, 1, 0.85])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0])
dict_values([0.87, 1.0, 0.304, 0.478, 0.13, 0.174, 0.13, 0.13, 0.261, 0.217])
dict_values([0.484, 1.0, 0.189, 0.755, 0.238, 0.158, 0.305, 0.191, 0.212,
0.263])
dict_values([0.364, 1.0, 0.0, 0.909, 0.091, 0.0, 0.0, 0.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0,
0.0, 1.0, 0.0])
dict_values([1, 0.6, 1, 0.8, 0.8, -0.05, 1, 1, 1.0, 0.45, -0.35, -0.8, 1, 0.4,
0.15, 0.5])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.125, 0.143,
0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([0.333, 0.667, 0.333, 0.667, 0.0, 0.0, 0.667, 0.0, 0.333, 0.333,
0.0, 0.333, 0.0, 0.0, 1.0, 0.333])
dict_values([1.0, 0.455, 0.182, 0.182, 0.182, 0.273, 0.182, 0.182, 0.273, 0.091,
0.091, 0.364, 0.091, 0.182, 0.364, 0.364])
dict_values([0.497, 0.904, 0.479, 0.545, 0.471, 0.545, 0.504, 0.516, 0.549,
0.569, 0.762, 0.894, 0.427, 0.667, 1.0, 0.53])
dict_values([0.5, 0.625, 0.0, 0.125, 0.0, 0.0, 0.625, 0.125, 0.125, 0.125,
0.875, 1.0, 0.0, 0.25, 0.875, 0.125])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.25, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0])
dict_values([1, 0.6, -0.65, 0.5, 0.35, 0.9, 0.15, 0.5, 0.1, 0.9, 0.35, 0.95,
-0.25, 0.95, 0.9, 1.0])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.125, 0.143,
0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([0.25, 0.0, 0.75, 0.75, 0.5, 0.25, 1.0, 0.75, 0.0, 0.0, 0.5, 0.5,
1.0, 0.25, 0.5, 0.25])
dict_values([1.0, 0.286, 0.429, 0.714, 0.714, 0.571, 0.857, 0.143, 0.286, 0.143,
0.143, 0.286, 0.857, 0.286, 0.429, 0.429])
dict_values([0.427, 0.688, 1.0, 0.674, 0.815, 0.532, 0.833, 0.555, 0.588, 0.342,
0.761, 0.383, 0.982, 0.356, 0.474, 0.642])
dict_values([0.2, 0.3, 0.4, 0.9, 0.7, 0.1, 0.8, 0.6, 0.0, 0.0, 0.6, 0.3, 1.0,
0.0, 0.1, 0.2])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0,
0.5, 0.0, 0.0, 0.5])
dict_values([1, 0.15, 1, 0.7, -0.5, 0.55, 1, 0.75, -1.4, 1, 0.85, 1, 1.0, 0.5,
-0.45, 1, 0.55])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1, 0.111,
0.125, 0.143, 0.167, 0.5, 1.0, 0.333])
dict_values([0.333, 0.333, 0.0, 0.0, 0.0, 0.333, 0.0, 0.0, 0.333, 0.0, 0.0, 0.0,
0.333, 0.0, 1.0, 0.0, 0.0])
dict_values([1.0, 0.375, 0.125, 0.125, 0.5, 0.5, 0.125, 0.125, 0.625, 0.125,
0.375, 0.25, 0.25, 0.375, 0.625, 0.125, 0.375])
dict_values([0.432, 0.648, 0.213, 0.581, 0.786, 0.633, 0.276, 0.466, 1.0, 0.213,
0.662, 0.405, 0.384, 0.689, 0.823, 0.346, 0.689])
dict_values([0.214, 0.214, 0.143, 0.143, 0.286, 0.357, 0.143, 0.0, 0.857, 0.0,
0.214, 0.0, 0.143, 0.214, 1.0, 0.0, 0.214])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([1, 0.7, 0.45, 1.0, 1, 0.55, 0.85, -0.85, 0.45, 0.3, 1.0, 0.9, 0.9,
0.3])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.143, 0.167, 0.2, 0.25,
0.333, 0.5, 1.0])
dict_values([0.0, 1.0, 0.0, 0.5, 0.0, 0.5, 0.0, 0.5, 0.0, 0.5, 1.0, 0.0, 0.0,
0.5])
```

```
dict_values([1.0, 0.273, 0.273, 0.273, 0.091, 0.273, 0.273, 0.545, 0.182, 0.182,
0.364, 0.091, 0.091, 0.273])
dict_values([0.448, 0.77, 0.748, 0.421, 0.476, 0.515, 0.596, 1.0, 0.453, 0.558,
0.457, 0.209, 0.411, 0.644])
dict_values([0.6, 1.0, 0.4, 0.2, 0.2, 0.6, 0.0, 0.4, 0.0, 0.2, 0.8, 0.0, 0.0,
0.2])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([1, 0.6, 0.0, 1, 0.2, 0.85, -0.2, 0.45, 0.95, 0.9, 1, 1, 0.75, 0.9,
1.0, 1, 0.15, 0.45, 1])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1, 0.111,
0.125, 0.143, 0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([0.333, 1.0, 0.0, 0.0, 0.333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.333, 0.667, 0.333, 0.0, 0.0])
dict_values([1.0, 0.182, 0.364, 0.182, 0.182, 0.273, 0.455, 0.273, 0.273, 0.182,
0.091, 0.091, 0.091, 0.091, 0.273, 0.091, 0.182, 0.273, 0.091])
dict_values([0.608, 0.414, 0.928, 0.567, 0.876, 0.621, 1.0, 0.747, 0.457, 0.882,
0.519, 0.327, 0.213, 0.36, 0.482, 0.371, 0.826, 0.462, 0.389])
dict_values([0.5, 0.167, 0.0, 0.0, 0.833, 0.0, 0.0, 0.0, 0.0, 0.333, 0.167, 0.0,
0.333, 0.0, 0.0, 0.167, 1.0, 0.0, 0.167])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 1.0, 0.0, 0.0])
dict_values([1, 0.55, 0.85, 1, 1, 0.9, 1, 1, 1, 1, 1, 0.25])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.167, 0.2, 0.25, 0.333, 0.5,
1.0])
dict_values([0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0])
dict_values([1.0, 0.5, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.6])
dict_values([0.603, 0.691, 0.555, 0.314, 0.381, 0.802, 0.381, 0.286, 0.531,
0.368, 0.273, 1.0])
dict_values([0.5, 0.167, 1.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.333, 0.0, 0.167,
0.833])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0])
dict_values([1, 0.8, 0.85, 1, 1, 1, 0.65, 0.95, 0.85, 1, 0.6, 0.2, 0.95, 1,
0.65, 0.4, 0.35, 1.0, 1, 0.7, 1, 1])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1, 0.091,
0.091, 0.1, 0.111, 0.125, 0.143, 0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0])
dict_values([1.0, 0.625, 0.375, 0.25, 0.125, 0.375, 0.25, 0.25, 0.25, 0.375,
0.125, 0.25, 0.125, 0.25, 0.125, 0.25, 0.5, 0.125, 0.125, 0.25, 0.125, 0.125])
dict_values([0.544, 0.917, 0.774, 0.483, 0.285, 0.604, 0.786, 0.534, 0.55,
0.576, 0.713, 0.872, 0.521, 0.526, 0.424, 0.918, 1.0, 0.671, 0.56, 0.757, 0.743,
0.523])
```

```
dict_values([0.333, 0.167, 0.333, 0.333, 0.0, 0.167, 0.333, 0.333, 0.5, 0.333,
1.0, 0.167, 0.0, 0.333, 0.667, 0.167, 0.333, 0.167, 0.333, 0.167, 1.0, 0.167])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
dict_values([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0])
dict_values([1, 0.45, 0.45, 0.3, 1, 1, 0.45, 0.35, 0.75, 1, 0.65, 0.85, 0.5,
0.35, 0.85, 0.3, 0.9])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.125,
0.143, 0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
1.0, 0.0, 0.0, 1.0])
dict_values([1.0, 0.286, 0.429, 0.429, 0.286, 0.286, 0.571, 0.429, 0.143, 0.143,
0.571, 0.286, 0.857, 0.714, 0.143, 0.286, 0.143])
dict_values([0.497, 0.691, 0.479, 1.0, 0.557, 0.757, 0.949, 0.722, 0.222, 0.186,
0.984, 0.587, 0.772, 0.799, 0.422, 0.715, 0.68])
dict_values([0.182, 0.909, 0.091, 0.364, 0.182, 0.364, 0.273, 0.0, 0.182, 0.0,
0.364, 0.091, 1.0, 0.091, 0.273, 0.0, 0.182])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0])
dict_values([0.0, 0.0, 0.333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.333, 0.0,
0.333, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.333, 0.333, 0.667, 0.0, 0.0, 0.0,
0.333])
dict_values([1, 1, 0.9, 1, 1, 0.65, 0.25, 1.0, 0.7, 0.35, -0.4, 0.95, 0.65, 0.7,
0.95, 0.5, 0.9, 0.65, 0.9, 0.55, -0.15, -1.4, 0.7, 1, -0.15, 0.2, -1.1, 0.25,
0.45, 0.1, 0.9, 0.8, 0.7, 0.75, 0.3, 1, 1, 0.75, 0.6, 1, -0.8, 1.0, -0.05])
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1, 0.091,
0.083, 0.077, 0.071, 0.067, 0.062, 0.059, 0.056, 0.053, 0.05, 0.048, 0.045,
0.048, 0.05, 0.053, 0.056, 0.059, 0.062, 0.067, 0.071, 0.077, 0.083, 0.091, 0.1,
0.111, 0.125, 0.143, 0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.667, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.667, 0.0, 0.0, 0.0, 0.0, 0.333, 0.667, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.333, 0.333, 0.0, 0.0, 0.0, 0.0, 0.333,
0.667])
dict_values([1.0, 0.125, 0.125, 0.125, 0.25, 0.0, 0.125, 0.25, 0.125, 0.125,
0.0, 0.25, 0.125, 0.25, 0.125, 0.25, 0.0, 0.125, 0.25, 0.125, 0.0, 0.375, 0.125,
0.0, 0.375, 0.25, 0.625, 0.25, 0.125, 0.375, 0.0, 0.0, 0.0, 0.125, 0.25, 0.0,
0.125, 0.125, 0.0, 0.0, 0.25, 0.375, 0.375])
dict_values([0.139, 0.268, 0.299, 0.132, 0.319, 0.342, 0.37, 0.304, 0.371,
0.515, 0.58, 0.107, 0.492, 0.528, 0.192, 0.28, 0.399, 0.337, 0.329, 0.358, 0.7,
0.817, 0.402, 0.284, 0.645, 0.625, 1.0, 0.482, 0.65, 0.675, 0.369, 0.185, 0.713,
0.48, 0.454, 0.325, 0.257, 0.372, 0.675, 0.419, 0.663, 0.28, 0.592])
dict_values([0.154, 0.077, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.385, 0.0, 0.538, 0.0,
0.0, 0.154, 0.0, 0.154, 0.231, 0.462, 0.0, 0.385, 0.385, 0.231, 0.0, 0.0, 0.692,
0.077, 1.0, 0.0, 0.308, 0.0, 0.077, 0.0, 0.154, 0.154, 0.077, 0.0, 0.077, 0.0,
0.0, 0.154, 0.077, 0.077, 0.538])
```

```
        ---------------------------------------------------------------------------

        UnicodeDecodeError                        Traceback (most recent call last)

        <ipython-input-11-5586f972b0aa> in <module>
          5 for file in filelist:
          6     f = open(path + file, "r")
    ----> 7     text = f.read()
          8
          9     sent_tokens = nltk.sent_tokenize(text)


        D:\Anaconda\lib\encodings\cp1252.py in decode(self, input, final)
         21 class IncrementalDecoder(codecs.IncrementalDecoder):
         22     def decode(self, input, final=False):
    ---> 23         return codecs.charmap_decode(input,self.
 ↪errors,decoding_table)[0]
         24
         25 class StreamWriter(Codec,codecs.StreamWriter):


        UnicodeDecodeError: 'charmap' codec can't decode byte 0x9d in position␣
 ↪5740: character maps to <undefined>
```

[12]:
```
df = df.to_csv('D:/COVID_19_dataset/documents/output.csv', index = False)
```

[13]:
```
data = pd.read_csv('D:/COVID_19_dataset/documents/output.csv')
data.head()
```

[13]:
```
     a    b     c      d  upper      f      g      h  \
0  0.0  0.0  0.80  1.000    1.0  0.870  0.484  0.364
1  0.0  0.0 -2.25  0.500    1.0  1.000  1.000  1.000
2  0.0  0.0  1.00  0.333    0.0  0.304  0.189  0.000
3  1.0  0.0 -0.10  0.250    0.0  0.478  0.755  0.909
4  0.0  0.0  1.00  0.200    0.0  0.130  0.238  0.091

                                                 key  label
0  Success from two leading coronavirus vaccine p...      0
1  The fact that two coronavirus vaccines recentl...      0
2  The studies showed both vaccines provided stro...      0
3  "With the very good news from Pfizer and Moder...      0
4  While Gates didn't delve into the scientific r...      0
```

[14]:
```
data.tail()
```

[14]:
```
       a      b     c      d  upper      f      g      h  \
181  0.0  0.667  0.60  0.200  0.000  0.000  0.675  0.000
182  0.0  0.000  1.00  0.250  0.000  0.000  0.419  0.154
183  1.0  0.000 -0.80  0.333  0.000  0.250  0.663  0.077
184  0.0  0.000  1.00  0.500  0.333  0.375  0.280  0.077
185  0.0  0.333 -0.05  1.000  0.667  0.375  0.592  0.538

                                                   key  label
181  "We talk about the 90/10 divide in global heal...      0
182        This is part of that story," Ms Wenham said.      0
183  "But there's a difference between the fact tha...      0
184  A landmark global vaccine plan known as Covax ...      0
185  The joint initiative - between the Gavi vaccin...      0
```

2.1 Analysis

Here there are 10 columns the key column is useful because it has text entries of all the files. Printing some values to see what they look like

[15]:
```
data['key'][0]
```

[15]:
```
'Success from two leading coronavirus vaccine programs likely means other
frontrunners will also show strong protection against COVID-19, Bill Gates said
Tuesday.'
```

[16]:
```
data['key'][100]
```

[16]:
```
'The development comes nearly 10 months after news of the coronavirus began to
emerge from Wuhan, China.'
```

[17]:
```
data['label'].value_counts()
```

[17]:
```
0    186
Name: label, dtype: int64
```

3 Methodology

Objective: To generate a single summary for all articles

3.1 Step 1: Split text into sentences

[18]:
```
sentences = [] # empty list

for s in data['key']:
    sentences.append(sent_tokenize(s))

sentences = [y for x in sentences for y in x] # flatten list
sentences[:5]
```

[18]:
```
['Success from two leading coronavirus vaccine programs likely means other
frontrunners will also show strong protection against COVID-19, Bill Gates said
Tuesday.',
 'The fact that two coronavirus vaccines recently showed strong protection
against COVID-19 bodes well for other leading programs led by AstraZeneca,
Novavax, and Johnson & Johnson, Bill Gates said Tuesday.The billionaire
Microsoft founder and philanthropist said it will be easier to boost
manufacturing and distribute these other shots to the entire world, particularly
developing nations.The vaccine space has seen a flurry of good news in recent
days, marked by overwhelming success in late-stage trials by both Pfizer and
Moderna.',
 'The studies showed both vaccines provided strong protection against the virus
compared to a placebo.',
 '"With the very good news from Pfizer and Moderna, we think it\'s now likely
that AstraZeneca, Novavax, and Johnson & Johnson will also likely show very
strong efficacy," Gates told journalist Andrew Ross Sorkin.',
 "While Gates didn't delve into the scientific rationale behind that prediction,
many scientists hold the same hope."]
```

3.2 Step 2: Extract word embeddings

[19]:
```
word_embeddings = {}

f = open('E:/Jupyterfiles/ML_practice/Stanford/glove.6B/glove.6B.100d.txt',␣
 ↪encoding = 'utf-8')

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_embeddings[word] = coefs

f.close()
```

[20]:
```
len(word_embeddings)
```

[20]:
```
400000
```

Word vectors for 400K different terms are stored in the dictionary

3.3 Step 3: Text pre-processing

[21]:
```
filter = data["key"] != ""
data_clean = data[filter]
data_clean = data_clean.dropna()
```

[22]:
```
def preprocess_text(sen):

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sen)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence
```

[23]:
```
X = []
sentences = list(data["key"])
for sen in sentences:
    X.append(preprocess_text(sen))

X[:5]
```

[23]:
```
['Success from two leading coronavirus vaccine programs likely means other
frontrunners will also show strong protection against COVID Bill Gates said
Tuesday ',
 'The fact that two coronavirus vaccines recently showed strong protection
against COVID bodes well for other leading programs led by AstraZeneca Novavax
and Johnson Johnson Bill Gates said Tuesday The billionaire Microsoft founder
and philanthropist said it will be easier to boost manufacturing and distribute
these other shots to the entire world particularly developing nations The
vaccine space has seen flurry of good news in recent days marked by overwhelming
success in late stage trials by both Pfizer and Moderna ',
 'The studies showed both vaccines provided strong protection against the virus
compared to placebo ',
 ' With the very good news from Pfizer and Moderna we think it now likely that
AstraZeneca Novavax and Johnson Johnson will also likely show very strong
efficacy Gates told journalist Andrew Ross Sorkin ',
 'While Gates didn delve into the scientific rationale behind that prediction
many scientists hold the same hope ']
```

[24]:
```
def remove_stopwords(sen):
    sen_new = " ".join([i for i in sen if i not in my_stopwords])

    return sen_new
```

[25]:
```
clean_sentences = [remove_stopwords(r.split()) for r in X]
clean_sentences = [s.lower() for s in clean_sentences]

clean_sentences[:5]
```

[25]:
```
['success two leading coronavirus vaccine programs likely means frontrunners
also show strong protection covid bill gates said tuesday',
 'the fact two coronavirus vaccines recently showed strong protection covid
bodes well leading programs led astrazeneca novavax johnson johnson bill gates
said tuesday the billionaire microsoft founder philanthropist said easier boost
manufacturing distribute shots entire world particularly developing nations the
vaccine space seen flurry good news recent days marked overwhelming success late
stage trials pfizer moderna',
 'the studies showed vaccines provided strong protection virus compared
placebo',
 'with good news pfizer moderna think likely astrazeneca novavax johnson johnson
also likely show strong efficacy gates told journalist andrew ross sorkin',
 'while gates delve scientific rationale behind prediction many scientists hold
hope']
```

Use 'clean_sentences' to create vectors for sentences in data with the help of GloVe word vectors

3.4 Step 4: Vector representation of sentences

[26]:
```
# Extract word vectors
word_embeddings = {}

f = open('E:/Jupyterfiles/ML_practice/Stanford/glove.6B/glove.6B.100d.txt',␣
 ↪encoding = 'utf-8')

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1: ], dtype = 'float32')
    word_embeddings[word] = coefs

f.close()
```

[27]:
```
sentence_vectors = []

for i in clean_sentences:
    if len(i) != 0:
        v = sum([word_embeddings.get(w, np.zeros((100,))) for w in i.split()])/
 ↪(len(i.split()) + 0.001)
    else:
        v = np.zeros((100,))
    sentence_vectors.append(v)
```

3.5 Step 5: Similarity between sentences

Algorithm used: Cosine similarity

Approach: Create an empty similarity matrix and populate it with cosine similarities of sentences

[28]:
```
sim_matrix = np.zeros([len(sentences), len(sentences)]) # zero matrix of size nXn

# initialise matrix with cosine similarity scores
for i in range(len(sentences)):
    for j in range(len(sentences)):
        if i!=j:
            sim_matrix[i][j] = cosine_similarity(sentence_vectors[i].reshape(1,␣
 ↪100), sentence_vectors[j].reshape(1, 100))[0, 0]
```

3.6 Step 6: Apply PageRank algorithm

[29]:
```
nx_graph = nx.from_numpy_array(sim_matrix)
scores = nx.pagerank(nx_graph)
```

3.7 Step 7: Summary extraction

Extract top-N sentences based on rankings for summary generation

[30]:
```
ranked_sentences = sorted(((scores[i],s) for i,s in enumerate(sentences)),␣
 ↪reverse = True)

for i in range(10):
    print(ranked_sentences[i][1])
```

```
Meanwhile, efforts to develop an effective vaccine are continuing - although the
World Health Organization (WHO) has warned that the death toll could hit two
million before one is widely available.
As ministers struggle to get test-and-trace on track, BBC News spoke to key
government figures, scientists and health officials who were involved from the
very start to establish what went wrong - and, crucially, whether the system can
be fixed to hold the virus in check until vaccines come to the rescue.
From Monday, under new government restrictions designed to tackle the fresh
outbreak, residents will only be allowed to see one other person from outside
their household and should work from home if possible.
Delivering a limited supply to the world
Andrea Taylor, who has been leading the Duke analysis, said the combination of
advance purchase agreements and limits on the number of doses that can be
manufactured in the next couple of years meant "we're heading into a scenario
where the rich countries will have vaccines and the poorer countries are
```

unlikely to have access".
The paper says ministers have approved a plan to cut the isolation time to just five days, after which travellers would face tests for the virus that return results within an hour.
Moderna Covid vaccine shows nearly 95% protection
UK orders 5m doses of Moderna vaccine
How a 'warm vaccine' could help India tackle Covid
Rachel Silverman, a policy analyst at the Center for Global Development think-tank in the US, said the most promising vaccines "are largely covered by advanced purchase agreements, mostly from wealthy countries".
“So New Zealand took a precautionary approach and on 26 March, apart from essential workers, the entire country was required to self-quarantine at home.”
WHO Regional Director for the Western Pacific, Dr Takeshi Kasai, explains that New Zealand combined strict physical distancing with strong testing, contact tracing, clinical management of those infected, and clear and regular public communication.
The fact that two coronavirus vaccines recently showed strong protection against COVID-19 bodes well for other leading programs led by AstraZeneca, Novavax, and Johnson & Johnson, Bill Gates said Tuesday.The billionaire Microsoft founder and philanthropist said it will be easier to boost manufacturing and distribute these other shots to the entire world, particularly developing nations.The vaccine space has seen a flurry of good news in recent days, marked by overwhelming success in late-stage trials by both Pfizer and Moderna.
The investigation found a system performing worst in the areas where it is needed the most and still struggling with the legacy of decisions that were made at the outset.
But she added: "There is very little likelihood that it will make it to low- and middle-income countries by the end of next year, at least in any significant numbers for mass vaccination."