2020

Extractive Text Summarization for News Articles

FAISAL RIAZ, WASEEM ABBAS
F2019313020, F2019313020

Contents

Abstract

Text summarization is the process of reducing a text Document in order to create a summary that retains the most important points of the original document. As The problem of information overload has grown, and the quantity of data has increased, so has interest in automatic summarization. It is very difficult for human beings to manually summarize large documents of text. Text Summarization methods can be classified into extractive and abstractive summarization. We have used an extractive summarization method consists of selecting important sentences, paragraphs etc. from the original News Articles and concatenating them into shorter form. The importance of sentences is decided based on cosine distance of sentence vectors, similarity matrix and generating the scores using text rank algorithm. Extractive methods work by selecting a subset of existing words, phrases, or sentences in the original text to form the summary. The extractive summarization systems are typically based on techniques for sentence extraction and aim to cover the set of sentences that are most important for the overall understanding of a given News Article.

Problem statement

Being a genuine political buff, I continuously attempt to keep myself overhauled with what's happening within the political issues by religiously going through as numerous online political overhauls as conceivable. Be that as it may, this has demonstrated to be a or maybe troublesome work! There are way as well many resources and time could be a constraint. Therefore, we chosen to plan a program that may get ready a bullet-point outline for me by filtering through multiple articles. How to go almost doing this? That's what we are going appear you in this article. We are going apply the Text Rank calculation on a dataset of scratched articles with the point of making a decent and brief text summary.

Text summarization

Text summarization is the technique for generating a concise and precise summary of voluminous texts while focusing on the sections that convey useful information, and without losing the overall meaning.

Automatic text summarization aims to transform lengthy documents into shortened versions, something which could be difficult and costly to undertake if done manually. Machine learning algorithms can be trained to comprehend documents and identify the sections that convey important facts and information before retrieving the required summarized texts. (Dhaji)

Automatic text summarization methods are greatly needed to address the ever-growing amount of text data available online to both better help retrieve relevant information and to consume relevant information faster. There are two type of ***text summarization***.

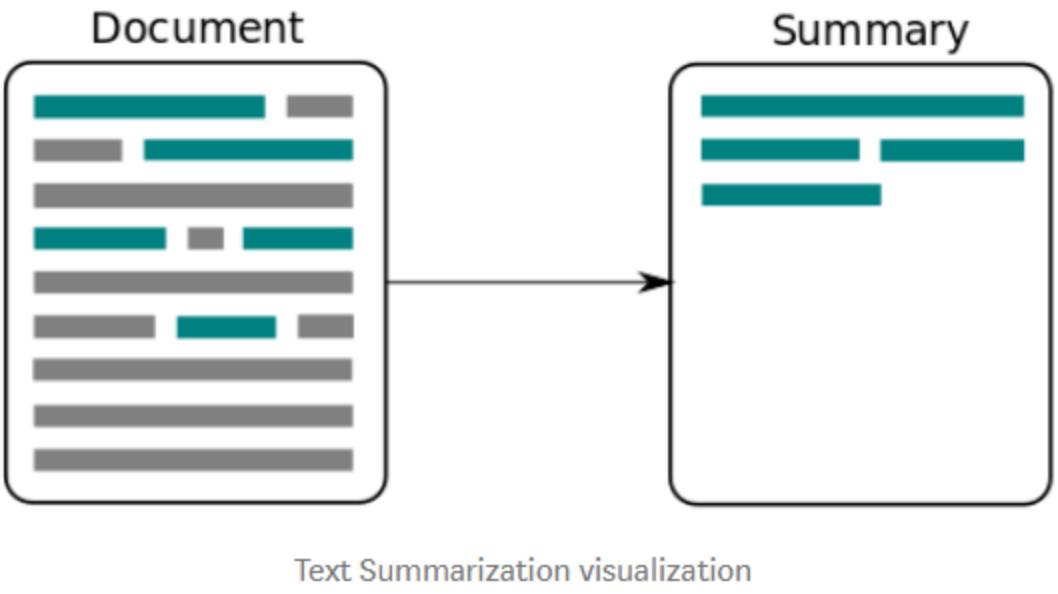


Fig1.Text Summarizer

Text Summarization Approaches:

Automatic Text Summarization gained attention as early as the 1950's. A research paper, published by Hans Peter Luhn (Luhun) in the late 1950s, titled "The automatic creation of literature abstracts", used features such as word frequency and phrase frequency to extract important sentences from the text for summarization purposes.

Another important research, done by Harold P Edmundson (Edmonson) in the late 1960's, used methods like the presence of cue words, words used in the title appearing in the text, and the location of sentences, to extract significant sentences for text summarization. Since then, many important and exciting studies have been published to address the challenge of automatic text summarization.

There are two type of ***text summarization***.

Extraction-based summarization

The extractive text summarization technique involves pulling key phrases from the source document and combining them to make a summary. The extraction is made according to the defined metric without making any changes to the texts.

Here is an example:

Source text: ***Joseph*** *and* ***Mary*** *rode on a donkey to* ***attend*** *the annual* ***event*** *in* ***Jerusalem****. In the city, Mary gave* ***birth*** *to a child named* ***Jesus***.

Extractive summary*: Joseph and Mary attend event Jerusalem. Mary birth Jesus*.

As you can see above, the words in bold have been extracted and joined to create a summary — although sometimes the summary can be grammatically strange.

Abstraction-based summarization

The abstraction technique entails paraphrasing and shortening parts of the source document. When abstraction is applied for text summarization in deep learning problems, it can overcome the grammar inconsistencies of the extractive method.

The abstractive text summarization algorithms create new phrases and sentences that relay the most useful information from the original text — just like humans do.

Therefore, abstraction performs better than extraction. However, the text summarization algorithms required to do abstraction are more difficult to develop; that's why the use of extraction is still popular.

Here is an example:

Abstractive summary: *Joseph and Mary came to Jerusalem where Jesus was born*. (Garbade)

In our project we are going to develop a code based on Extractive summarization technique using Text Rank Algorithm.

Text Rank Algorithm

In order to understand Text Rank Algorithm, we have to understand Page Rank Algorithm. Page Rank Algorithm was developed by Larry Page and give birth to the Google which we know now. Page Rank is used for ranking web pages in online search results. At first, we will briefly discuss Page Rank Algorithm and then we will make an analogy of Text Rank Algorithm to Page Rank.

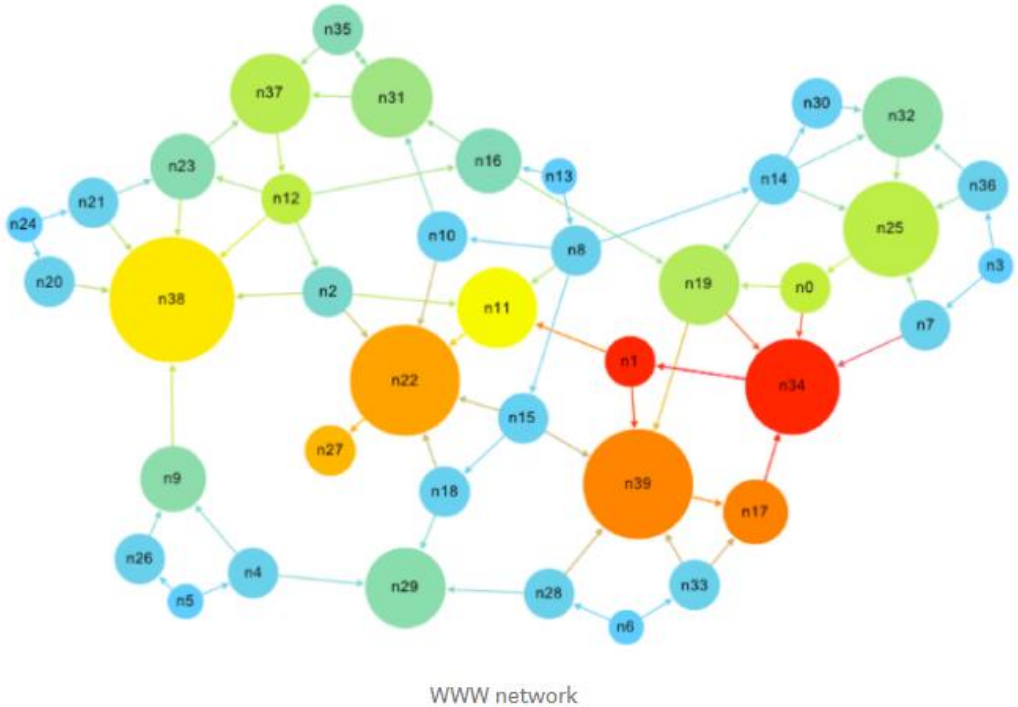


Fig2.Page Rank Graph

Page Rank

Suppose we have 4 web pages — w1, w2, w3, and w4. These pages contain links pointing to one another. Some pages might have no link – these are called dangling pages.

webpage	links
w1	[w4, w2]
w2	[w3, w1]
w3	[ ]
w4	[w1]

- Web page w1 has links directing to w2 and w4
- w2 has links for w3 and w1
- w4 has links only for the web page w1
- w3 has no links and hence it will be called a dangling page

In order to rank these pages, we would have to compute a score called the PageRank score. This score is the probability of a user visiting that page.

To capture the probabilities of users navigating from one page to another, we will create a square matrix M, having n rows and n columns, where n is the number of web pages.

M =

	w1	w2	w3	w4
w1				
w2				
w3				
w4				

- Each element of this matrix denotes the probability of a user transitioning from one web page to another. For example, the highlighted cell below contains the probability of transition from w1 to w2.

M =

	w1	w2	w3	w4
w1				
w2				
w3				
w4				

P(w1 to w2)

- The initialization of the probabilities is explained in the steps below:

- Probability of going from page i to j, i.e., M[i][j], is initialized with 1/(number of unique links in web page wi)
- If there is no link between the page i and j, then the probability will be initialized with 0
- If a user has landed on a dangling page, then it is assumed that he is equally likely to transition to any page. Hence, M[i][j] will be initialized with 1/(number of web pages)
- Hence, in our case, the matrix M will be initialized as follows:

M =

	w1	w2	w3	w4
w1	0	0.5	0	0.5
w2	0.5	0	0.5	0
w3	0.25	0.25	0.25	0.25
w4	1	0	0	0

- Finally, the values in this matrix will be updated in an iterative fashion to arrive at the web page rankings. (Prateek)

Text Rank Algorithm

Let's understand the Text Rank algorithm, now that we have a grasp on PageRank. I have listed the similarities between these two algorithms below:

- In place of web pages, we use sentences.
- Similarity between any two sentences is used as an equivalent to the web page transition. probability.
- The similarity scores are stored in a square matrix, similar to the matrix M used for PageRank.

Text Rank is an ***extractive and unsupervised text summarization*** technique. Let's take a look at the flow of the Text Rank algorithm that we will be following:

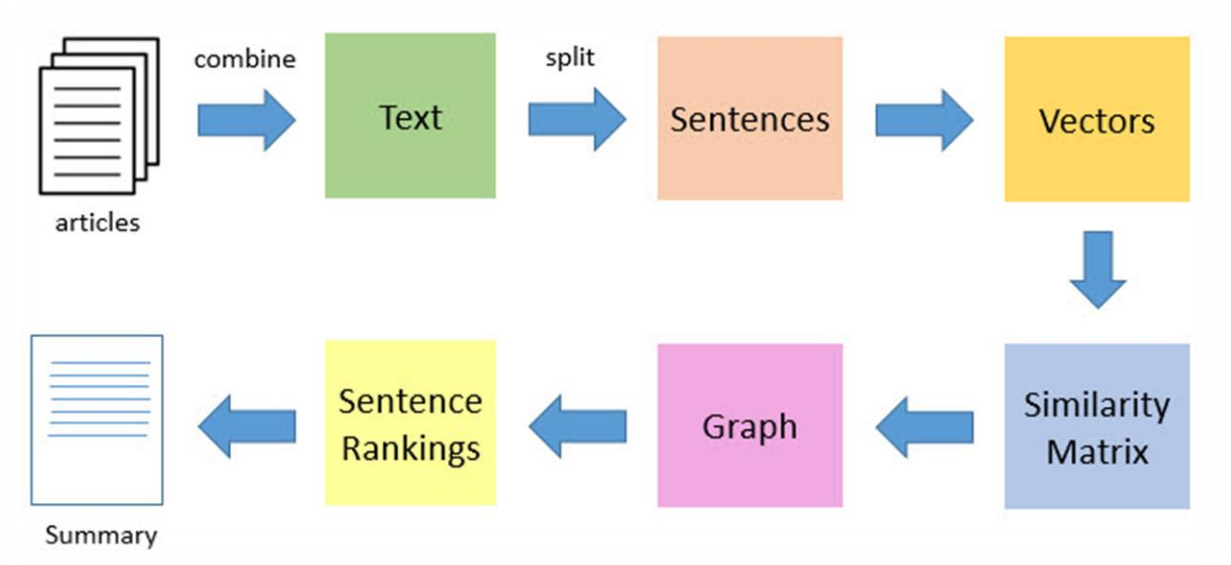


Fig3.Text Rank Flow

- Selecting and article from a dataset and extracting all the text.
- Splitting the Text into sentences.
- Converting all the words in a sentence to vectors.
- Generating similarity matrix across sentences from their vectors.
- Similarity matrix is then converted into graph, where sentences are the Nodes/Vertices and the scores as edges to calculate sentence rank.
- Graph for all the sentences is drawn.
- Sentences ranks are sorted in descending order.
- N numbers of top ranked sentences are selected to generate summarized text.

Key concepts

We have outlined some key concepts here which are required to be understood before we explain the implementation of code.

- Vector representation of words.
- Cosine Similarity and Cosine distance.

Vector Representation of words

Representation of words to vectors is also known as word embedding. According to Wikipedia. Word embedding is the collective name for a set of language modelling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. — Wikipedia.

Many Techniques are available to convert words to vectors, we will only explain the most basic one:

- Count Vectorizer
- TF-IDF Vectorizer

We have used the Count Vectorizer Technique which is the most basic technique, in our code so we are only going to explain Count Vectorizer only.

Count Vectorizer

We will explain it by considering a Dataset based on four sentences. Sample text Dataset is

```
"He is playing in the field.
He is running towards the football.
The football game ended.
It started raining while everyone was playing in the field."
```

Step 1: Identify unique words in the complete text data and converting them to lower case. In our case, the list is as follows (17 words):

Unique List = `['ended', 'everyone', 'field', 'football', 'game', 'he', 'in', 'is', 'it', 'playing', 'raining', 'running', 'started', 'the', 'towards', 'was', 'while']`

Step 2: For each sentence, we'll create an array of **zeros** with the same length as above (17)

Vector1 = `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].`

Step 3: Taking each sentence one at a time, we'll read the first word, find its total occurrence in the sentence. Once we have the number of times it appears in that sentence, we'll identify the position of the word in the list above and replace the same zero with this count at that position. This is repeated for all words and for all sentences

Example

Let's take the first sentence, ***He is playing in the field.***

The first word is ***He***. Its position is ***6th*** from the starting (Unique List). I'll just update its vector and it will now be:

Vector1 `=[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].`

Similarly, we will find each word from the **Unique List** is searched and its respective index is updated (**0+1**) in the **Vector1.** Complete representation of ***He is playing in the field*** would become as:

Vector1 `=[0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0].` (Bhanot, n.d.)

In similar way we will generate vectors for each sentence.

TF-IDF Vectorizer

Count Vectorizer converts each sentence into its own vector, it does not consider the importance of a word across the complete list of sentences. For example, He is in two sentences and it provides no useful information in differentiating between the two. Thus, it should have a lower weight in the overall vector of the sentence. This is where the TF-IDF Vectorizer comes into the picture.

TF-IDF is a product of two parts:

TF (Term Frequency) — It is defined as the number of times a word appears in the given sentence.

IDF (Inverse Document Frequency) — It is defined as the **log** to the **base 2** of number of the total documents divided by the documents in which the word appears. Considering the previous example and sentence. So, for ***He***

```
Total documents (N): 4

Documents in which the word appears (n): 2

Number of times the word appears in the first sentence: 1

Number of words in the first sentence: 6

Term Frequency(TF) = 1

Inverse Document Frequency(IDF) = log(N/n)
                                = log(4/2)
                                = log(2)

TF-IDF value = 1 * log(2)
             = 0.69314718
```

Updated **Vector1 =** `[0, 0, 0, 0, 0, 0.69314718, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

The same will get repeated for all other words

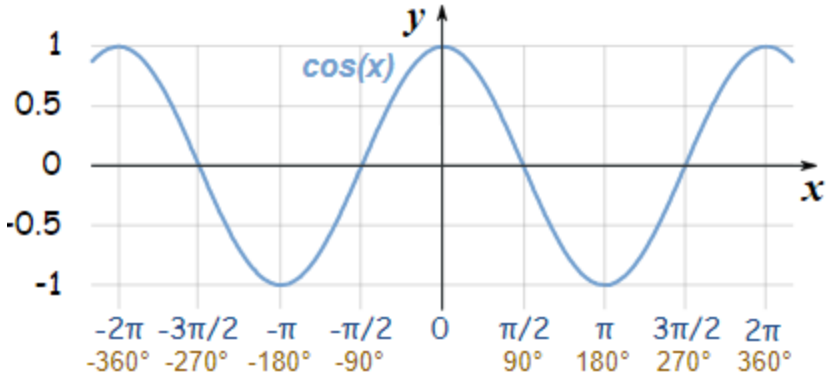
Cosine Similarity and Cosine Distance

Cosine similarity is used to determine the similarity between documents, sentences or vectors. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.

The relation between cosine similarity and cosine distance can be define as below.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Cosine similarity says that to find the similarity between two points or vectors we need to find Angle between them. If **A** and B represents the vectors then cosine similarity would be the $\cos(\theta)$. We know that Cosine values varies between 1 and -1.



Since we want to find the similarity between two sentences so less distance mean more similarity and vice versa. So, formula for cosine distance is

$$\text{Cosine distance} = 1 - \cos\theta$$

Implementation of Text Summarizer in Python

We have implemented Text Summarization of News Articles building a simple Text summarizer from scratch. We will explain the implementation step by step.

Importing Libraries

We have imported the following libraries as a starting point.

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

import matplotlib.pyplot as plt
from PIL import Image
from wordcloud import WordCloud
```

Dataset

Dataset we have used here is a bunch articles based on the statements of Imran Khan and Donald Trump. Attributes of the Dataset are also shown below:

```
df = pd.read_csv("IK_Trump.csv", engine ='python')
df.head(5)
```

	ArticleID	ArticleTitle	ArticleText	ArticleSource
0	0	On the Coronavirus, Pakistan's Government Is M...	During a televised broadcast on March 22, Paki...	https://foreignpolicy.com/2020/05/11/on-corona...
1	1	Pakistan Plans Another COVID-19 Lockdown. Will...	The government in Pakistan is considering the ...	https://thediplomat.com/2020/05/pakistan-plans...
2	2	COVID-19: Pakistan has a 'dependency syndrome'...	As Pakistan grapples to combat coronavirus wit...	https://gulfnews.com/world/asia/pakistan/covid...
3	3	COVID19 and Pakistan: The Economic Fallout	In Pakistan, the first confirmed cases of COVI...	https://www.orfonline.org/research/covid19-and...
4	4	Donald Trump says he's taking hydroxychloroqui...	President Donald Trump said he's taking antima...	https://www.theverge.com/2020/5/18/21262873/tr...

ArticleTitle: Title of the Article

ArticleText: column containing the text of the article

ArticleSource: is the URL of the article

Since we are only interested in the Article Text so let's display an article text.

```
df['ArticleText'][2]
```

```
'As Pakistan grapples to combat coronavirus with increasing number of cases, Prime Minister Imran Khan has urged the need to bu
ild its own reliable medical infrastructure.\n"The coronavirus has brought to the fore that Pakistan needs to build its medical
infrastructure and reduce dependency on foreign aid," added Imran.\nSpeaking during this visit to COMSTECH exhibition in Islama
bad on Wednesday, Prime Minister Imran said:"We have a dependency syndrome; we don't have that self-belief, we didn't progress
towards knowledge economy because we didn't spend on education and research. Pakistanis excel once they immigrate; why can't yo
u create that system here?""""We have to focus on building our medical infrastructure so that we are prepared for any such emerg
ency situation in the future,"  he said and added that the COVID-19 crisis provided an opportunity to produce locally manufactu
red ventilators and protective equipment as everything cannot be imported. OMSTECH stands for the Organisation of Islamic Coope
ration's Standing Committee on Scientific and Technological Cooperation for the promotion and cooperation of science and techno
logy activities among the OIC member states.\n"It is only now – with a global shortage – that we have found out that building v
entilators isn't that hard. The country that had the capacity to make nuclear bombs, how hard can it be for it to make ventilat
ors?" he said. "The biggest quality that takes a nation upwards is self-belief. This belief increases as the nation progresses
and reaches a stage where the nation thinks it can overcome any challenge. The small island of Britain dominated the world. Wha
t did it have that was so special? It was self belief," he noted Prime Minister Imran also congratulated Minister for Science a
nd Technology Fawad Chaudhry for seeing the potential in a dormant industry, reported Dawn news. Addressing the ceremony, Fawad
Chaudhry said that one of Pakistan's biggest mistakes was not linking military and civil research. "We have informally linked t
hem now, defence and production industry," Chaudhry said while adding that this has helped Pakistan produce equipment for battl
ing coronavirus much faster. He said that NTRC and Nust have made disinfecting robots, drones can be used to disinfect areas un
der quarantine. "A thermal monitoring camera has been developed, will detect without touching," Chaudhry said. He also praised
the role of private sector in producing protective equipment for frontline workers. "We were facing a shortage of protective eq
uipment in Pakistan a few weeks ago, now all of Faisalabad city is making protective suits for doctors and frontline workers. N
ow there is a need for us to consider how many of these we need, the rest we should export," Chaudhry said. He further announce
d that Pakistan is now producing its own N-95 mask prototype. "While the masks being imported by Pakistan are costing Rs1,100,
the ones we have created are going to cost Rs90 – a huge difference." He also said that Pakistan took mere weeks to produce its
own coronavirus testing kits. Addressing the ceremony, Fawad Chaudhry said that one of Pakistan's biggest mistakes was not link
ing military and civil research. "We have informally linked them now, defence and production industry," Chaudhry said while add
ing that this has helped Pakistan produce equipment for battling coronavirus much faster. He said that NTRC and Nust have made
disinfecting robots, drones can be used to disinfect areas under quarantine. "A thermal monitoring camera has been developed, w
ill detect without touching," Chaudhry said. He also praised the role of private sector in producing protective equipment for f
rontline workers. "We were facing a shortage of protective equipment in Pakistan a few weeks ago, now all of Faisalabad city is
making protective suits for doctors and frontline workers. Now there is a need for us to consider how many of these we need, th
e rest we should export," Chaudhry said. He further announced that Pakistan is now producing its own N-95 mask prototype. "Whil
e the masks being imported by Pakistan are costing Rs1,100, the ones we have created are going to cost Rs90 – a huge differenc
e." He also said that Pakistan took mere weeks to produce its own coronavirus testing kits. He said Pakistan was facing a short
age of hand sanitisers and disinfectants a month and a half ago but "today, we are producing our own sanitisers and disinfectan
ts and are in the position to export them. We are looking for the commerce ministry to lift the ban on its export for us to be
able to do that," Chaudhry said. Meanwhile, the federal government is preparing to loosen coronavirus lockdown restrictions as
the number of infections and deaths are "well below previous projections," said Asad Umar, Minister of Planning. Pakistan despi
te registering more than 15,7500 cases of Covid-19 including 346 deaths has already granted exemptions to dozens of sectors to
open up over the last few days. Umar said that infections and deaths in Pakistan were 30-35% lower than projections and, if thi
ngs remained this way, the country could open up further in coming days. Experts say Pakistan's low numbers are due to limited
testing. Currently Pakistan, a country of more than 207 million people, conducts about 8,000 tests a day. \n\n'
```

Splitting Text to Sentences

- Article Id is passed to function.
- Text is read from the Articles DataFrame and split into sentences with period(.).
- All non-alphabets characters are replaced with space and Sentence list is populated.
- Sentences is a list of each sentence tokenized. [[sent1],[sent2],.....[sentn]]

```
def read_article_text(article_id):
    article = df['ArticleText'][article_id].split(". ")
    sentences = []

    for sentence in article:
        #print(sentence)
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" ")) #replacing all non characters with "space"
    sentences.pop()

    return sentences
```

Similarity Matrix

- NxN matrix where N is length/no of sentence
- S= Sentence and N=3
- Creating a matrix with zeros NxN where N = no of sentences.

- Check if sentences are not similar than pass on the both sentences to sentence similarity function.
- Sentence similarity function will calculate the cosine similarity and the matrix is updated according to sentence index position.
- Thus, populating the whole NxN similarity matrix.

$$\begin{bmatrix} S_1S_1 & S_1S_2 & S_1S_3 \\ S_2S_1 & S_2S_2 & S_2S_3 \\ S_3S_1 & S_3S_2 & S_3S_3 \end{bmatrix}$$

```
def create_similarity_matrix(sentences, stop_words):

    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences), len(sentences)))
    print("length of sentences is |",len(sentences))

    for index1 in range(len(sentences)):

        for index2 in range(len(sentences)):
            if index1 == index2: #ignore if both are same sentences

                continue

            similarity_matrix[index1][index2] = sentence_similarity(sentences[index1], sentences[index2], stop_words)

    print("returning_similarity mat shape", similarity_matrix.shape)
    print("returning_similarity mat", similarity_matrix)
    return similarity_matrix
```

Sentence Similarity

- Two Sentences which are not equal indices are passed to this function.
- Two Vectors are created filled with 0 equal to the number of unique tokens in all words.
- Search the for each Token in all words list and update the index of token in Vector1 and then for Vector2.
- Returning the Cosine distance and returning to previous function to update similarity values for two sentences.

```
[[0.         0.08526091 0.         ... 0.05756737 0.         0.        ]
 [0.08526091 0.         0.06350006 ... 0.04356068 0.         0.        ]
 [0.         0.06350006 0.         ... 0.         0.         0.        ]
 ...
 [0.05756737 0.04356068 0.         ... 0.         0.         0.0766965 ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.0766965  0.         0.        ]]
```

Sentence Similarity Matrix

```
def sentence_similarity(sent1, sent2, stopwords=None):
    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)

    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue

        vector1[all_words.index(w)] += 1


    # build the vector for the second sentence
    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1


    print("vector 1 is = ", vector1)
    print("vector 2 is = ", vector2)
    print ("Cosine_Distance = ",1-cosine_distance(vector1,vector2))
    return (1 - cosine_distance(vector1, vector2))
```

```
unique list of words in two sentences
['', 'economy', 'as', 'we', 'imported', 'why', 'its', 'you', 'crisis', 'here?”“”we', 'provided', 'infrastructure.\n“the',
'are', 'with', 'produce', 'minister', 'own', 'visit', 'combat', 'aid,”', 'imran.\nspeaking', 'foreign', 'immigrate;',
'on', 'protective', 'medical', 'during', 'wednesday,', 'system', 'progress', 'have', 'building', 'khan', 'that', 'the',
'situation', 'imran', 'pakistan', 'to', 'fore', 'towards', 'knowledge', 'reduce', 'he', 'and', 'opportunity', 'need',
'focus','self-belief,', 'said', 'they', 'has', 'number', 'cases,', 'don’t', 'coronavirus', 'our', 'syndrome;', 'reliable',
'because', 'of', 'exhibition', 'in', 'prepared', 'create', 'spend', 'said:“we', 'for', 'didn’t', 'can’t', 'such', 'excel',
'a', 'grapples', 'pakistanis', 'brought', 'future,”', 'added', 'everything', 'education', 'this', 'comstech', 'manufactured',
'locally', 'covid-19', 'once', 'emergency', 'any', 'increasing', 'needs', 'an', 'equipment', 'prime', 'so', 'cannot', 'build',
'urged', 'islamabad', 'infrastructure', 'be', 'research', 'dependency', 'ventilators']

vector 1 is =  [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 1, 1, 1, 1, 1, 0, 0, 0, 2, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 2,
                2, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 2, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 2, 0, 0, 0, 0, 1,
                0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 2, 0, 0, 2, 1, 1, 1, 0, 1, 2, 0]
vector 2 is =  [1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]

Cosine_Distance =  0.08526091126389745
```

Sentence Ranking

We have applied Page Rank algorithm imported from ***nx*** library. Here we have calculated the sentence similarity graph and the scores.

```
    # Step 3 - Rank sentences in similarity martix
    sentence_similarity_graph = nx.from_numpy_array(sentence_similarity_martix)
    print("sentence similarity graph = ", sentence_similarity_graph)
    scores = nx.pagerank(sentence_similarity_graph)
    print("score of the similariy graph = ",scores )
   # creating the graph for similarity matrix.
    nx.draw(sentence_similarity_graph,with_labels=True)
```

We have drawn sentence similarity graph with each sentence in the article as node and shown below. Each sentence here is represented by node/edge. Whereas, vertices represent the cosine distance (Rada Mihalcea, 2004)between the nodes.

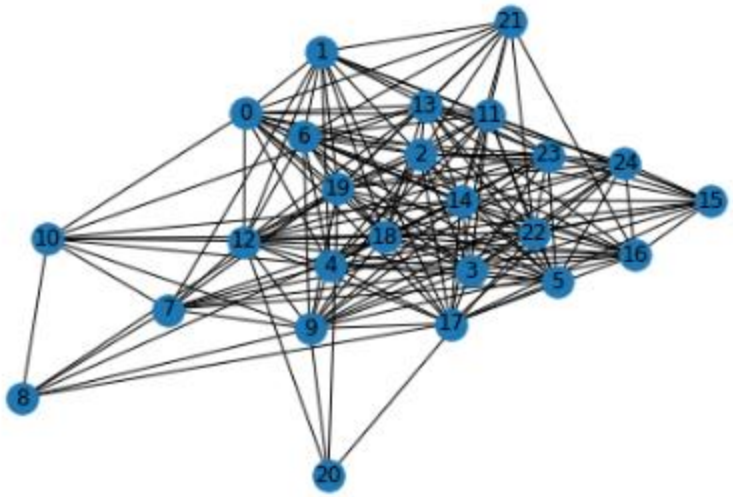


Fig4. Sentence Similarity Graph

Text Summarizer

Sentences are ranked according to the scores in descending order. Summary of the article is generated for ***n*** number of top ranked sentences. Output is shown as below:

```
# Step 4 - Sort the rank and pick top sentences
ranked_sentence = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)

print("Indexes of top ranked_sentence order are ", ranked_sentence)

for i in range(top_n):
  summarize_text.append(" ".join(ranked_sentence[i][1]))

# Step 5 - Offcourse, output the summarize text
print("Summarize Text: \n", ". ".join(summarize_text))
```

```
Summarize Text:
 “We have informally linked them now, defence and production industry,” Chaudhry said while adding that this has helped Pakista
n produce equipment for battling coronavirus much faster. “We have informally linked them now, defence and production industr
y,” Chaudhry said while adding that this has helped Pakistan produce equipment for battling coronavirus much faster. “While the
masks being imported by Pakistan are costing Rs1,100, the ones we have created are going to cost Rs90 — a huge difference.” He
also said that Pakistan took mere weeks to produce its own coronavirus testing kits
```

Fig5.Text Summary

Summary Word Cloud

Word clouds can identify trends and patterns that would otherwise be unclear or difficult to see in a tabular format. Frequently used keywords stand out better in a word cloud. Common words that might be overlooked in tabular form are highlighted in larger text making them pop out when displayed in a word cloud. We have drawn the word cloud for Summarized Text below to emphasize the most frequent words appearing in the summarize text.

```
def word_cloud_generate(file):
    summary = ""
    with open(file, encoding='cp1252') as f:
        summary = ''.join(f.readlines())

    wordcloud = WordCloud(width=1600, height=800).generate(summary)
    plt.figure( figsize=(16,8), facecolor='k')
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.savefig('wordcloud.png', facecolor='k', bbox_inches='tight')
```

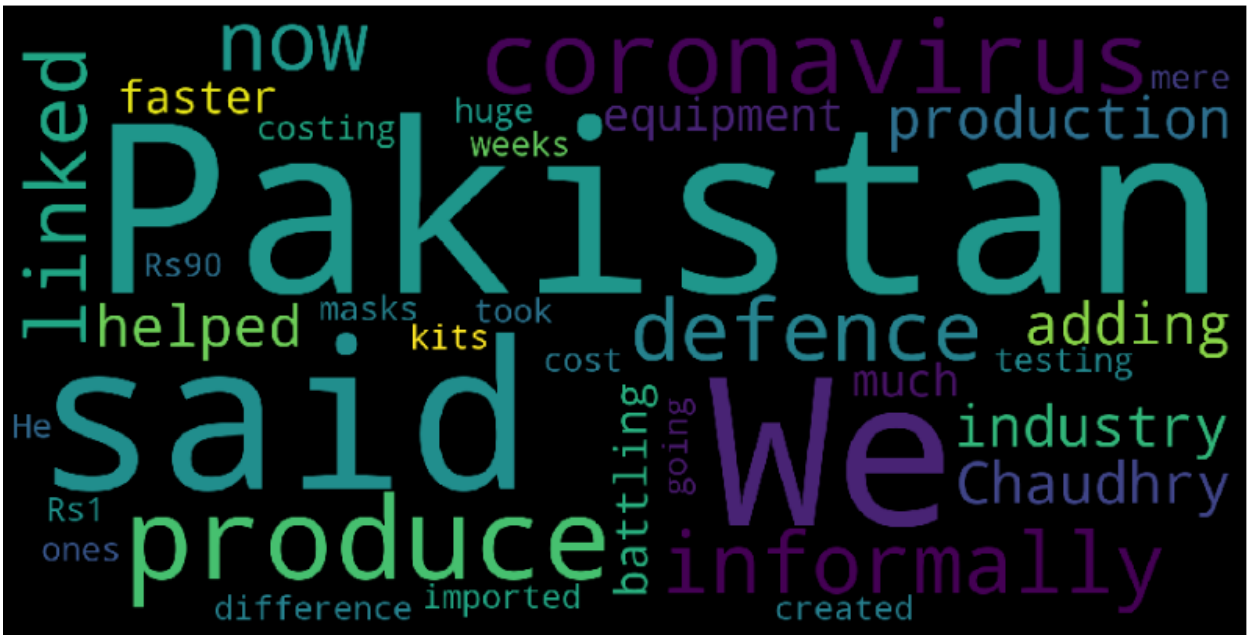


Fig6.Summary Word Cloud

Conclusions

We have reasonably performed well, since Automatic Text Summarization is a hot topic of research, and in this article, we have covered just the tip of the iceberg. Going forward, we will explore the abstractive text summarization technique where deep learning plays a big role. In addition, we can also look into the following summarization tasks:

- Problem-specific
- Multiple domain text summarization
- Single document summarization
- Cross-language text summarization (source in some language and summary in another language)

- Algorithm-specific
- Text summarization using RNNs and LSTM
- Text summarization using Reinforcement Learning
- Text summarization using Generative Adversarial Networks (GANs)

References

Bhanot, K. (n.d.). *Different techniques to represent words as vectors (Word Embeddings)*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/different-techniques-to-represent-words-as-vectors-word-embeddings-3e4b9ab7ceb4

Dhaji, H. (n.d.). *Text Summarization-Key Concepts.* Retrieved from Medium: https://medium.com/@harshdarji_15896/text-summarization-key-concepts-23df617bfb3e

Edmonson. (n.d.). *http://courses.ischool.berkeley.edu/i256/f06/papers/edmonson69.pdf.*

Garbade, D. M. (n.d.). *a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f.* Retrieved from https://towardsdatascience.com: https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f

Luhun. (n.d.). *http://courses.ischool.berkeley.edu/i256/f06/papers/luhn58.pdf.*

Prateek, J. (n.d.). *An Introduction to Text Summarization using the TextRank Algorithm.* Retrieved from https://www.analyticsvidhya.com/: https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/

Rada Mihalcea, P. T. (2004). *TextRank: Bringing Order into Text*. Retrieved from https://www.semanticscholar.org: https://www.semanticscholar.org/paper/TextRank%3A-Bringing-Order-into-Text-Mihalcea-Tarau/7b95d389bc6affe6a127d53b04bcfd68138f1a1a