**Text Summarization Methods**

## **Introduction**

Text Summarization is one of those applications of Natural Language Processing (NLP) which is bound to have a huge impact on our lives. With growing digital media and ever-growing publishing – who has the time to go through entire articles / documents / books to decide whether they are useful or not? Text Summarization helps to resolve this problem.

Have you come across the mobile app **inshorts**? It’s an innovative news app that converts news articles into a 60-word summary. And that is exactly what we are going to do in this project.

The methods of automatic text summarization fall into two primary categories:

1. **Extractive summarization: -**

* Mark the sentences in the text as per relevance.
* Select the sentences with highest relevance so that they represent the information conveyed by the text.
* Extractive text summarization does not use words aside from the ones already in the text, and selects some combination of the existing words most relevant to the meaning of the source.
* This is a Unsupervised Learning Method

1. **Abstractive text summarization: -**

* Follows an Encode-Decode Structure.
* The text is encoded into a latent factor and then decoded as a summary.
* Abstractive text summarization involves generating entirely new phrases and sentences to capture the meaning of the text.
* This is a Supervised Learning Method

In this document we will be focusing on Extractive Text Summarization.

**Extractive Text Summarization Method (UNSUPERVISED LEARNING): -**

For extractive summarization, we will use the TextRank algorithm, which is based on Google’s PageRank algorithm.

Before going into details, here is how it works: -

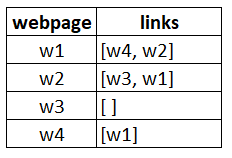
* TextRanks works by transforming the text into a graph.
* It regards words as vertices and the relation between words in phrases or sentences as edges. Each edge also has different weight.
* When one vertex links to another one, it is basically casting a vote of importance for that vertex.
* The importance of the vertex also dictates how heavily weighted its votes are.
* TextRank uses the structure of the text and the known parts of speech for words to assign a score to words that are keywords for the text.

## **Understanding the TextRank Algorithm**

Before getting started with the TextRank algorithm, there’s another algorithm which we should become familiar with – the PageRank algorithm. In fact, this actually inspired TextRank!

**PageRank is used primarily for ranking web pages in online search results.** Let’s quickly understand the basics of this algorithm with the help of an example.

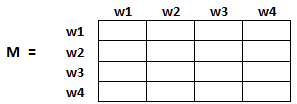
Suppose we have 4 web pages — w1, w2, w3, and w4. These pages contain links pointing to one another. Some pages might have no link – these are called dangling pages.



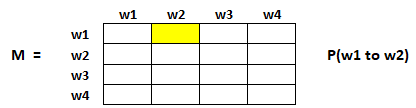
* Web page w1 has links directing to w2 and w4
* w2 has links for w3 and w1
* w4 has links only for the web page w1
* w3 has no links and hence it will be called a dangling page

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.



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.



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

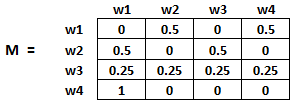
1. Probability of going from page i to j, i.e., M[ i ][ j ], is initialized with **1/(number of unique links in web page wi)**
2. If there is no link between the page i and j, then the probability will be initialized with **0**
3. If a user has landed on a dangling page, then it is assumed that he is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

So, M[w1][w2]=1/2 (one link out of the two links in w1 redirects to w2).

Also, M[w1][w3]=0 (no link in w1 redirects to w3)

M[w3][wi]=1/4 (w3 is a dangling page)

Hence, in our case, the matrix M will be initialized as follows:



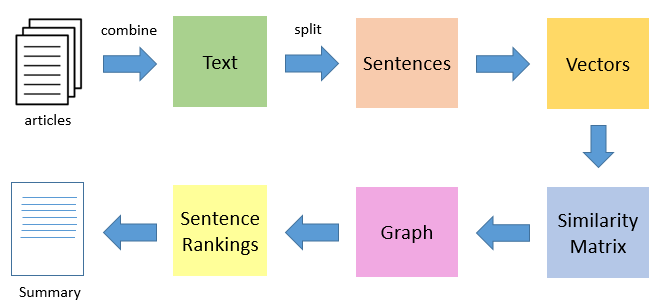
Finally, the values in this matrix will be updated in an iterative fashion to arrive at the web page rankings.

### **TextRank Algorithm**

Let’s understand the TextRank 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

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



* The first step would be to concatenate all the text contained in the articles
* Then split the text into individual sentences
* In the next step, we will find vector representation (word embeddings) for each and every sentence
* Similarities between sentence vectors are then calculated and stored in a matrix
* The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation
* Finally, a certain number of top-ranked sentences form the final summary

Now let’s get into an in-detail explanation on how we will be executing the TextRank Algorithm.

In this task, we will summarize a number of articles on latest tennis updates and provide the user with a concise summary.

**# Import Required Libraries**

import numpy as np

import pandas as pd

import nltk

**# Punkt Sentence Tokenizer. This tokenizer divides a text into a list of sentences**

nltk.download('punkt') **# one time execution**

import re

**# Read the Data**

df = pd.read\_csv("tennis\_articles.csv", encoding='cp1252')

df.head()

df['article\_text'][0]

**# Now we have 2 options – we can either summarize each article individually, or we can generate a single summary for all the articles.**

**# For our purpose, we will go ahead with the latter**.

import numpy as np

**# Split Text into Sentences**

from nltk.tokenize import sent\_tokenize # Return a sentence-tokenized copy of text

sentences = []

for s in df['article\_text']:

sentences.append(sent\_tokenize(s))

sentences = [y for x in sentences for y in x] # flatten list

sentences

**# Text Preprocessing**

**# remove punctuations, numbers and special characters**

clean\_sentences = pd.Series(sentences).str.replace("[^a-zA-Z]", " ")

**# make alphabets lowercase**

clean\_sentences = [s.lower() for s in clean\_sentences]

clean\_sentences

**# Remove stop words**

nltk.download('stopwords')

from nltk.corpus import stopwords

stop\_words = stopwords.words('english')

**# function to remove stopwords**

def remove\_stopwords(sen):

sen\_new = " ".join([i for i in sen if i not in stop\_words])

return sen\_new

**# remove stopwords from the sentences**

clean\_sentences = [remove\_stopwords(r.split()) for r in clean\_sentences]

clean\_sentences

**# Vector Representation of Sentences**

**# Extract features using TFIDF Vectorizer**

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max\_features=100)

sentence\_vectors = vectorizer.fit\_transform(clean\_sentences)

sentence\_vectors.shape

df\_idf = pd.DataFrame(vectorizer.idf\_, index=vectorizer.get\_feature\_names(),columns=["tfidf\_weights"])

df\_idf.sort\_values(by=['tfidf\_weights'],ascending = False).head()

**# Similarity Matrix Preparation**

**# similarity matrix**

sim\_mat = np.zeros([len(sentences), len(sentences)])

sim\_mat

**# We will use Cosine Similarity to compute the similarity between a pair of sentences And initialize the matrix with cosine similarity scores.**

from sklearn.metrics.pairwise import cosine\_similarity

for i in range(len(sentences)):

for j in range(len(sentences)):

if i != j:

sim\_mat[i][j] = cosine\_similarity(sentence\_vectors[i].reshape(1,100), sentence\_vectors[j].reshape(1,100))[0,0]

sim\_mat

**# Applying PageRank Algorithm**

**# Before proceeding further, let’s convert the similarity matrix sim\_mat into a graph.**

**# The nodes of this graph will represent the sentences and the edges will represent the similarity scores between the sentences.**

**# On this graph, we will apply the PageRank algorithm to arrive at the sentence rankings.**

import networkx as nx

import matplotlib.pyplot as plt

nx\_graph = nx.from\_numpy\_array(sim\_mat)

scores = nx.pagerank(nx\_graph)

**# Summary Extraction**

**# Finally, it’s time to extract the top N sentences based on their rankings for summary generation.**

ranked\_sentences = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)

ranked\_sentences

**# Extract top 10 sentences as the summary**

for i in range(10):

print(ranked\_sentences[i][1])