*Summarization can be defined as a task of producing a concise and fluent summary while preserving key information and overall meaning.*

**Impact**

Summarization systems often have additional evidence they can utilize in order to specify the most important topics of document(s). For example, when summarizing blogs, there are discussions or comments coming after the blog post that are good sources of information to determine which parts of the blog are critical and interesting.

In scientific paper summarization, there is a considerable amount of information such as cited papers and conference information which can be leveraged to identify important sentences in the original paper.

**How text summarization works**

In general there are two types of summarization, **abstractive**and **extractive**summarization.

1. **Abstractive Summarization:**Abstractive methods select words based on semantic understanding, even those words did not appear in the source documents. It aims at producing important material in a new way. They interpret and examine the text using advanced natural language techniques in order to generate a new shorter text that conveys the most critical information from the original text.

It can be correlated to the way human reads a text article or blog post and then summarizes in their own word.

***Input document → understand context → semantics → create own summary.***

**2. Extractive Summarization:**Extractive methods attempt to summarize articles by selecting a subset of words that retain the most important points.

This approach weights the important part of sentences and uses the same to form the summary. Different algorithm and techniques are used to define weights for the sentences and further rank them based on importance and similarity among each other.

***Input document → sentences similarity → weight sentences → select sentences with higher rank.***

The limited study is available for abstractive summarization as it requires a deeper understanding of the text as compared to the extractive approach.

Purely extractive summaries often times give better results compared to automatic abstractive summaries. This is because of the fact that abstractive summarization methods cope with problems such as semantic representation,  
inference and natural language generation which is relatively harder than data-driven approaches such as sentence extraction.

There are many techniques available to generate extractive summarization. To keep it simple, I will be using an [**unsupervised learning**](https://en.wikipedia.org/wiki/Unsupervised_learning)approach to find the sentences similarity and rank them. One benefit of this will be, you don’t need to train and build a model prior start using it for your project.

It’s good to understand **Cosine similarity**to make the best use of code you are going to see. **Cosine similarity** is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Since we will be representing our sentences as the bunch of vectors, we can use it to find the similarity among sentences. Its measures cosine of the angle between vectors. Angle will be **0**if sentences are similar.

*All good till now..?* *Hope so :)*

**Next, Below is our code flow to generate summarize text:-**

*Input article → split into sentences → remove stop words → build a similarity matrix → generate rank based on matrix → pick top N sentences for summary.*

Let’s create these methods.

**1. Import all necessary libraries**

from nltk.corpus import stopwords  
from nltk.cluster.util import cosine\_distance  
import numpy as np  
import networkx as nx

**2. Generate clean sentences**

**def read\_article(file\_name):**  
 file = open(file\_name, "r")  
 filedata = file.readlines()  
 article = filedata[0].split(". ")  
 sentences = []

for sentence in article:  
 print(sentence)  
 sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))  
 sentences.pop()   
   
 return sentences

**3. Similarity matrix**

This is where we will be using cosine similarity to find similarity between sentences.

**def build\_similarity\_matrix(sentences, stop\_words):**  
 # Create an empty similarity matrix  
 similarity\_matrix = np.zeros((len(sentences), len(sentences)))  
   
 for idx1 in range(len(sentences)):  
 for idx2 in range(len(sentences)):  
 if idx1 == idx2: #ignore if both are same sentences  
 continue   
 similarity\_matrix[idx1][idx2] = sentence\_similarity(sentences[idx1], sentences[idx2], stop\_words)

return similarity\_matrix

**4. Generate Summary Method**

Method will keep calling all other helper function to keep our summarization pipeline going. Make sure to take a look at all # Steps in below code.

**def generate\_summary(file\_name, top\_n=5):**  
 stop\_words = stopwords.words('english')  
 summarize\_text = []

**# Step 1 - Read text and tokenize**  
 sentences = read\_article(file\_name)

**# Step 2 - Generate Similary Martix across sentences**  
 sentence\_similarity\_martix = build\_similarity\_matrix(sentences, stop\_words)

**# Step 3 - Rank sentences in similarity martix**  
 sentence\_similarity\_graph = nx.from\_numpy\_array(sentence\_similarity\_martix)  
 scores = nx.pagerank(sentence\_similarity\_graph)

**# 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 texr**  
 print("Summarize Text: \n", ". ".join(summarize\_text))

All put together, here is the complete code.

**Let’s look at it in action.**

The complete text from an article titled ***Microsoft Launches Intelligent Cloud Hub To Upskill Students In AI & Cloud Technologies***

In an attempt to build an AI-ready workforce, Microsoft announced Intelligent Cloud Hub which has been launched to empower the next generation of students with AI-ready skills. Envisioned as a three-year collaborative program, Intelligent Cloud Hub will support around 100 institutions with AI infrastructure, course content and curriculum, developer support, development tools and give students access to cloud and AI services. As part of the program, the Redmond giant which wants to expand its reach and is planning to build a strong developer ecosystem in India with the program will set up the core AI infrastructure and IoT Hub for the selected campuses. The company will provide AI development tools and Azure AI services such as Microsoft Cognitive Services, Bot Services and Azure Machine Learning.According to Manish Prakash, Country General Manager-PS, Health and Education, Microsoft India, said, "With AI being the defining technology of our time, it is transforming lives and industry and the jobs of tomorrow will require a different skillset. This will require more collaborations and training and working with AI. That’s why it has become more critical than ever for educational institutions to integrate new cloud and AI technologies. The program is an attempt to ramp up the institutional set-up and build capabilities among the educators to educate the workforce of tomorrow." The program aims to build up the cognitive skills and in-depth understanding of developing intelligent cloud connected solutions for applications across industry. Earlier in April this year, the company announced Microsoft Professional Program In AI as a learning track open to the public. The program was developed to provide job ready skills to programmers who wanted to hone their skills in AI and data science with a series of online courses which featured hands-on labs and expert instructors as well. This program also included developer-focused AI school that provided a bunch of assets to help build AI skills.

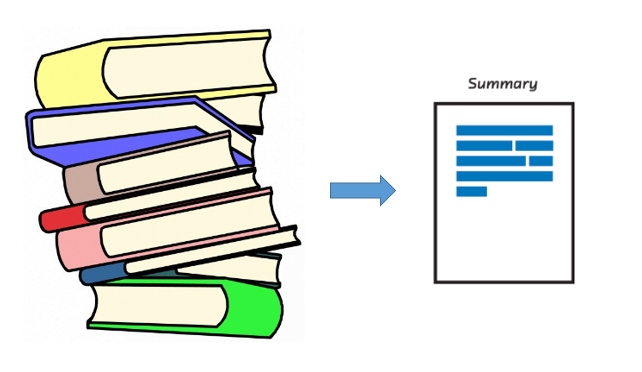
and the summarized text with *2**lines* as an input is

Envisioned as a three-year collaborative program, Intelligent Cloud Hub will support around 100 institutions with AI infrastructure, course content and curriculum, developer support, development tools and give students access to cloud and AI services. The company will provide AI development tools and Azure AI services such as Microsoft Cognitive Services, Bot Services and Azure Machine Learning. According to Manish Prakash, Country General Manager-PS, Health and Education, Microsoft India, said, "With AI being the defining technology of our time, it is transforming lives and industry and the jobs of tomorrow will require a different skillset.

## Table of Contents

1. Text Summarization Approaches
2. Understanding the TextRank Algorithm
3. Understanding the Problem Statement
4. Implementation of the TextRank Algorithm
5. What’s next?

## Text Summarization Approaches



Automatic Text Summarization gained attention as early as the 1950’s. A [research paper](http://courses.ischool.berkeley.edu/i256/f06/papers/luhn58.pdf), published by Hans Peter Luhn 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](http://courses.ischool.berkeley.edu/i256/f06/papers/edmonson69.pdf), done by Harold P Edmundson 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.

Text summarization can broadly be divided into two categories — **Extractive Summarization** and **Abstractive Summarization**.

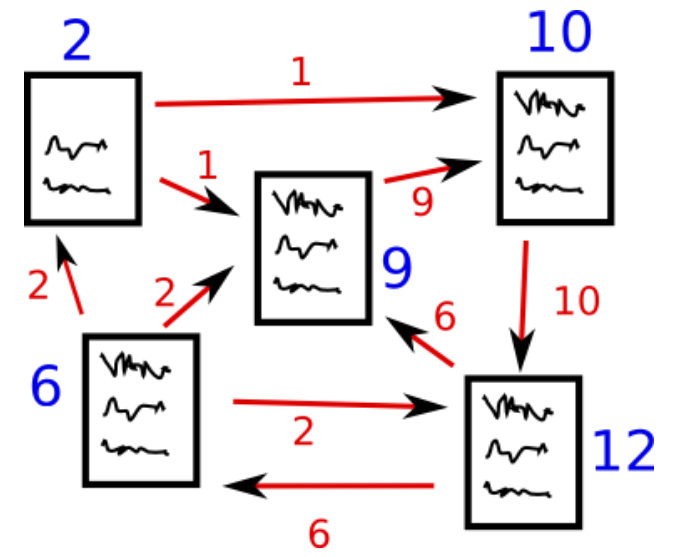
1. **Extractive Summarization:** These methods rely on extracting several parts, such as phrases and sentences, from a piece of text and stack them together to create a summary. Therefore, identifying the right sentences for summarization is of utmost importance in an extractive method.
2. **Abstractive Summarization:** These methods use advanced NLP techniques to generate an entirely new summary. Some parts of this summary may not even appear in the original text.

In this article, we will be focusing on the **extractive summarization** technique.

## Understanding the TextRank Algorithm

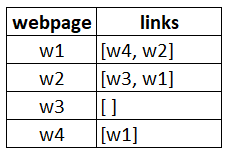
Before getting started with the TextRank algorithm, there’s another algorithm which we should become familiar with – the PageRank algorithm. In fact, this actually inspired TextRank! **PageRank is used primarily for ranking web pages in online search results.** Let’s quickly understand the basics of this algorithm with the help of an example.

### PageRank Algorithm



Source: http://www.scottbot.net/HIAL/

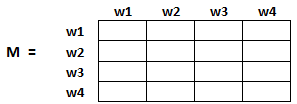
Suppose we have 4 web pages — w1, w2, w3, and w4. These pages contain links pointing to one another. Some pages might have no link – these are called dangling pages.



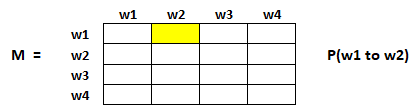
* Web page w1 has links directing to w2 and w4
* w2 has links for w3 and w1
* w4 has links only for the web page w1
* w3 has no links and hence it will be called a dangling page

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.



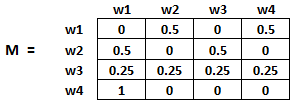
Each element of this matrix denotes the probability of a user transitioning from one web page to another. For example, the highlighted cell below contains the probability of transition from w1 to w2.



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)**

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



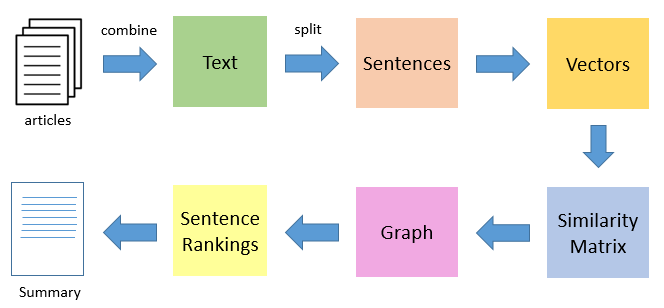
Finally, the values in this matrix will be updated in an iterative fashion to arrive at the web page rankings.

### 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

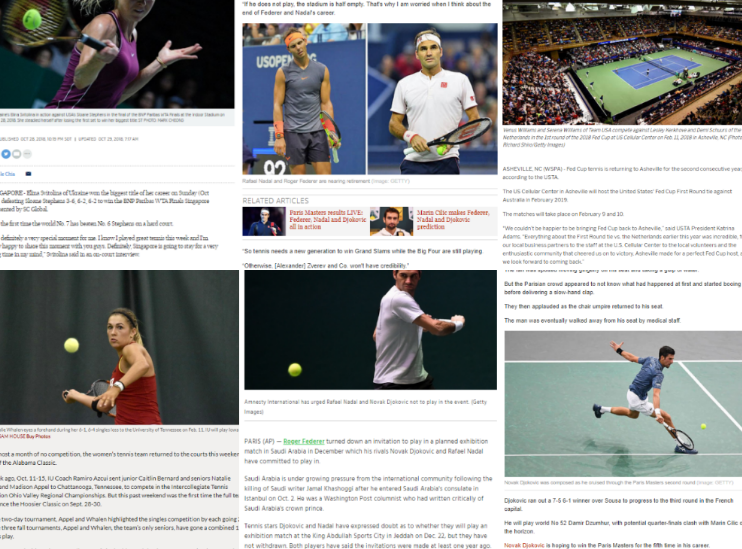
So, without further ado, let’s fire up our Jupyter Notebooks and start coding!

*Note: If you want to learn more about Graph Theory, then I’d recommend checking out this*[*article*](https://www.analyticsvidhya.com/blog/2018/09/introduction-graph-theory-applications-python/)*.*

## Understanding the Problem Statement

Being a major tennis buff, I always try to keep myself updated with what’s happening in the sport by religiously going through as many online tennis updates as possible. However, this has proven to be a rather difficult job! There are way too many resources and time is a constraint.

Therefore, I decided to design a system that could prepare a bullet-point summary for me by scanning through multiple articles. How to go about doing this? That’s what I’ll show you in this tutorial. We will apply the TextRank algorithm on a dataset of scraped articles with the aim of creating a nice and concise summary.



Please note that this is essentially a single-domain-multiple-documents summarization task, i.e., we will take multiple articles as input and generate a single bullet-point summary. Multi-domain text summarization is not covered in this article, but feel free to try that out at your end.

#### You can download the dataset we’ll be using from [here](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2018/10/tennis_articles_v4.csv).

## Implementation of the TextRank Algorithm

So, without any further ado, fire up your Jupyter Notebooks and let’s implement what we’ve learned so far.

#### Import Required Libraries

First, import the libraries we’ll be leveraging for this challenge.

import numpy as np

import pandas as pd

import nltk

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

import re

#### Read the Data

Now let’s read our dataset. I have provided the link to download the data in the previous section (in case you missed it).

df = pd.read\_csv("tennis\_articles\_v4.csv")

#### Inspect the Data

Let’s take a quick glance at the data.

df.head()

  
We have 3 columns in our dataset — ‘article\_id’, ‘article\_text’, and ‘source’. We are most interested in the ‘article\_text’ column as it contains the text of the articles. Let’s print some of the values of the variable just to see what they look like.

df['article\_text'][0]

**Output:**

"Maria Sharapova has basically no friends as tennis players on the WTA Tour. The Russian player

has no problems in openly speaking about it and in a recent interview she said: 'I don't really

hide any feelings too much. I think everyone knows this is my job here. When I'm on the courts

or when I'm on the court playing, I'm a competitor and I want to beat every single person whether

they're in the locker room or across the net...

df['article\_text'][1]

BASEL, Switzerland (AP), Roger Federer advanced to the 14th Swiss Indoors final of his career by beating

seventh-seeded Daniil Medvedev 6-1, 6-4 on Saturday. Seeking a ninth title at his hometown event, and a 99th

overall, Federer will play 93th-ranked Marius Copil on Sunday. Federer dominated the 20th-ranked Medvedev and had

his first match-point chance to break serve again at 5-1...

df['article\_text'][2]

Roger Federer has revealed that organisers of the re-launched and condensed Davis Cup gave him three days to

decide if he would commit to the controversial competition. Speaking at the Swiss Indoors tournament where he will

play in Sundays final against Romanian qualifier Marius Copil, the world number three said that given the

impossibly short time frame to make a decision, he opted out of any commitment...

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.

#### Split Text into Sentences

Now the next step is to break the text into individual sentences. We will use the sent\_tokenize( ) function of the nltk library to do this.

from nltk.tokenize import sent\_tokenize

sentences = []

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

sentences.append(sent\_tokenize(s))

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

Let’s print a few elements of the list sentences.

sentences[:5]

**Output:**

['Maria Sharapova has basically no friends as tennis players on the WTA Tour.',

"The Russian player has no problems in openly speaking about it and in a recent

interview she said: 'I don't really hide any feelings too much.",

'I think everyone knows this is my job here.',

"When I'm on the courts or when I'm on the court playing,

I'm a competitor and I want to beat every single person whether they're in the

locker room or across the net.So I'm not the one to strike up a conversation about

the weather and know that in the next few minutes I have to go and try to win a tennis match.",

"I'm a pretty competitive girl."]

#### Download GloVe Word Embeddings

[GloVe](https://nlp.stanford.edu/projects/glove/) word embeddings are vector representation of words. These word embeddings will be used to create vectors for our sentences. We could have also used the Bag-of-Words or TF-IDF approaches to create features for our sentences, but these methods ignore the order of the words (and the number of features is usually pretty large).

We will be using the pre-trained **Wikipedia 2014 + Gigaword 5** GloVe vectors available [here](https://nlp.stanford.edu/data/glove.6B.zip). Heads up – the size of these word embeddings is 822 MB.

!wget http://nlp.stanford.edu/data/glove.6B.zip

!unzip glove\*.zip

Let’s extract the words embeddings or word vectors.

# Extract word vectors

word\_embeddings = {}

f = open('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()

len(word\_embeddings)

400000

We now have word vectors for 400,000 different terms stored in the dictionary – ‘word\_embeddings’.

#### Text Preprocessing

It is always a good practice to make your textual data noise-free as much as possible. So, let’s do some basic text cleaning.

# 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]

Get rid of the stopwords (commonly used words of a language – is, am, the, of, in, etc.) present in the sentences. If you have not downloaded nltk-stopwords, then execute the following line of code:

nltk.download('stopwords')

Now we can import the stopwords.

from nltk.corpus import stopwords

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

Let’s define a function to remove these stopwords from our dataset.

# 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]

We will use clean\_sentences to create vectors for sentences in our data with the help of the GloVe word vectors.

#### Vector Representation of Sentences

# Extract word vectors

word\_embeddings = {}

f = open('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()

Now, let’s create vectors for our sentences. We will first fetch vectors (each of size 100 elements) for the constituent words in a sentence and then take mean/average of those vectors to arrive at a consolidated vector for the sentence.

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)

Note: For more text preprocessing best practices, you may check our video course, [*Natural Language Processing (NLP) using Python*](https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018_T1/about?utm_source=blog&utm_medium=introduction-text-summarization-textrank-python).

#### Similarity Matrix Preparation

The next step is to find similarities between the sentences, and we will use the cosine similarity approach for this challenge. Let’s create an empty similarity matrix for this task and populate it with cosine similarities of the sentences.

Let’s first define a zero matrix of dimensions (n \* n).  We will initialize this matrix with cosine similarity scores of the sentences. Here, **n**is the number of sentences.

# similarity matrix

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

We will use Cosine Similarity to compute the similarity between a pair of sentences.

from sklearn.metrics.pairwise import cosine\_similarity

And initialize the matrix with cosine similarity scores.

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]

#### 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

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)

# Extract top 10 sentences as the summary

for i in range(10):

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

When I'm on the courts or when I'm on the court playing, I'm a competitor and I want to beat every single person

whether they're in the locker room or across the net.So I'm not the one to strike up a conversation about the

weather and know that in the next few minutes I have to go and try to win a tennis match.

Major players feel that a big event in late November combined with one in January before the Australian Open will

mean too much tennis and too little rest.

Speaking at the Swiss Indoors tournament where he will play in Sundays final against Romanian qualifier Marius

Copil, the world number three said that given the impossibly short time frame to make a decision, he opted out of

any commitment.

"I felt like the best weeks that I had to get to know players when I was playing were the Fed Cup weeks or the

Olympic weeks, not necessarily during the tournaments.

Currently in ninth place, Nishikori with a win could move to within 125 points of the cut for the eight-man event

in London next month.

He used his first break point to close out the first set before going up 3-0 in the second and wrapping up the

win on his first match point.

The Spaniard broke Anderson twice in the second but didn't get another chance on the South African's serve in the

final set.

"We also had the impression that at this stage it might be better to play matches than to train.

The competition is set to feature 18 countries in the November 18-24 finals in Madrid next year, and will replace

the classic home-and-away ties played four times per year for decades.

Federer said earlier this month in Shanghai in that his chances of playing the Davis Cup were all but non-existent.

And there we go! An awesome, neat, concise, and useful summary for our articles.

### What’s Next?

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)

### End Notes

I hope this post helped you in understanding the concept of automatic text summarization. It has a variety of use cases and has spawned extremely successful applications. Whether it’s for leveraging in your business, or just for your own knowledge, text summarization is an approach all NLP enthusiasts should be familiar with.

I will try to cover the abstractive text summarization technique using advanced techniques in a future article. Meanwhile, feel free to use the comments section below to let me know your thoughts or ask any questions you might have on this article.