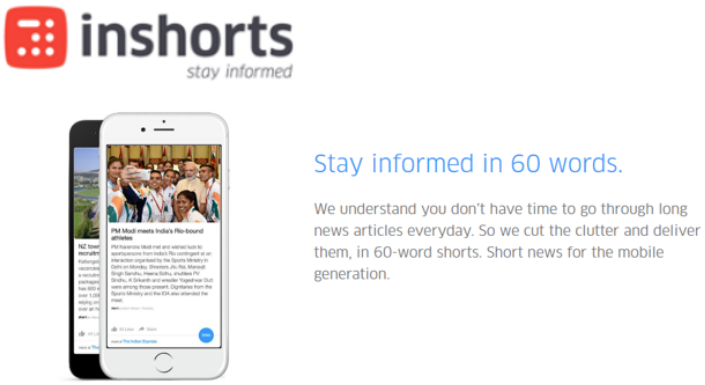
An Introduction to Text Summarization using the TextRank Algorithm (with Python implementation)

ALGORITHM INTERMEDIATE NLP PYTHON RANKING TECHNIQUE TEXT UNSTRUCTURED DATA UNSUPERVISED

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? Thankfully – this technology is already here.

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 learn in this article — **Automatic Text Summarization**.



Automatic Text Summarization is one of the most challenging and interesting problems in the field of Natural Language Processing (NLP). It is a process of generating a concise and meaningful summary of text from multiple text resources such as books, news articles, blog posts, research papers, emails, and tweets.

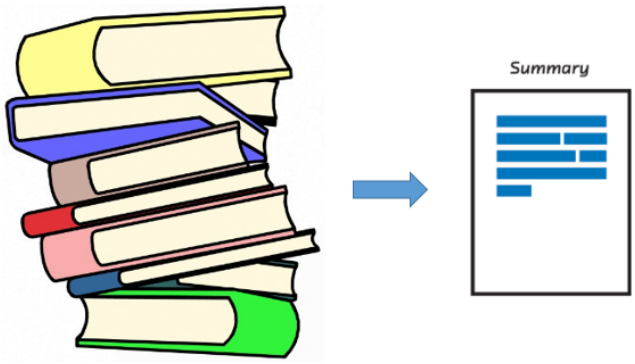
The demand for automatic text summarization systems is spiking these days thanks to the availability of large amounts of textual data.

Through this article, we will explore the realms of text summarization. We will understand how the TextRank algorithm works, and will also implement it in Python. Strap in, this is going to be a fun ride!

Table of Contents

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, 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, 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**.

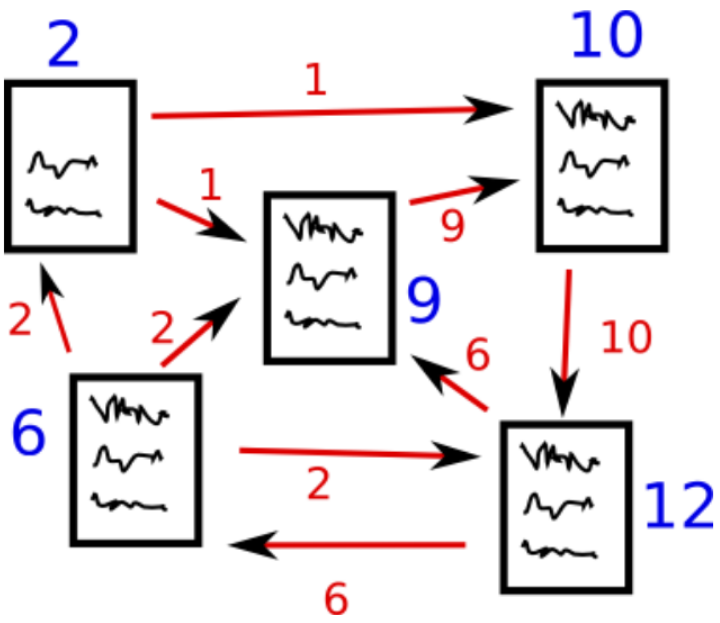
1. **Extractive Summarization:** These methods rely on extracting several parts, such as phrases and sentences, from a piece of text and stack them together to create a summary. Therefore, identifying the right sentences for summarization is of utmost importance in an extractive method.
2. **Abstractive 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.

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:

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:

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

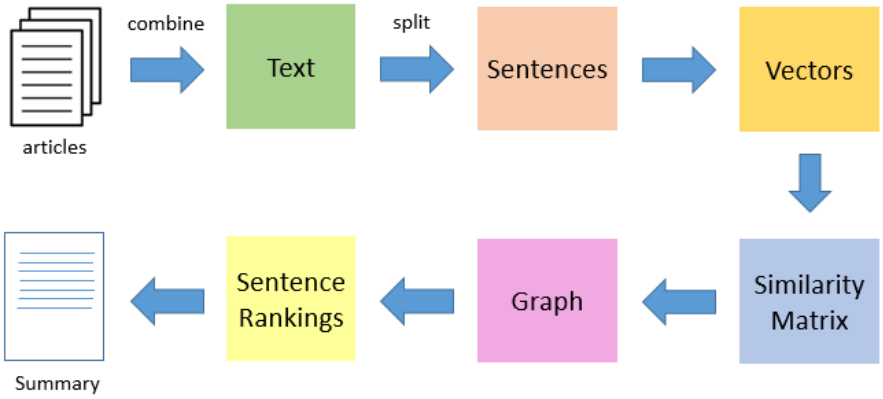
Finally, the values in this matrix will be updated in an iterative fashion to arrive at the web page rankings.

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

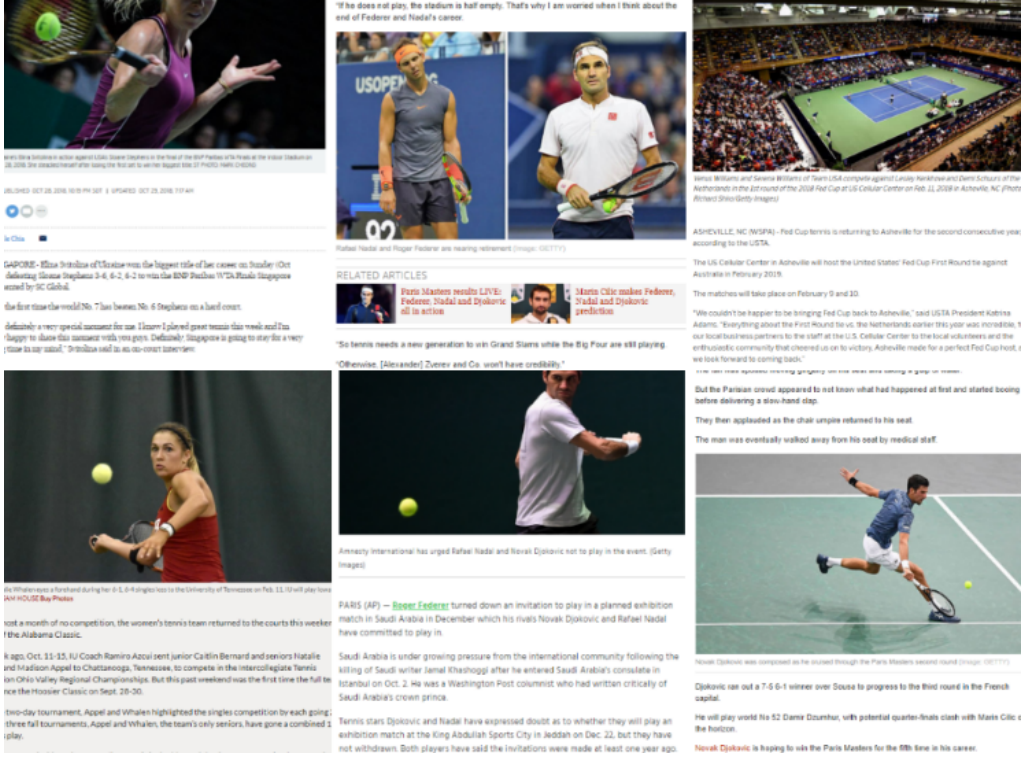
So, without further ado, let's fire up our Jupyter Notebooks and start coding!

Note: If you want to learn more about Graph Theory, then I'd recommend checking out this article.

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.

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()
```

	article_id	article_text	source
0	1	Maria Sharapova has basically no friends as te...	https://www.tennisworldusa.org/tennis/news/Mar...
1	2	BASEL, Switzerland (AP), Roger Federer advance...	http://www.tennis.com/pro-game/2018/10/copil-s...
2	3	Roger Federer has revealed that organisers of ...	https://scroll.in/field/899938/tennis-roger-fe...
3	4	Kei Nishikori will try to end his long losing ...	http://www.tennis.com/pro-game/2018/10/nishiko...
4	5	Federer, 37, first broke through on tour over ...	https://www.express.co.uk/sport/tennis/1036101...

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 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. *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.*

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.

Please find the code in this GitHub Repo.

Article Url - https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/

Prateek Joshi

Data Scientist at Analytics Vidhya with multidisciplinary academic background. Experienced in machine learning, NLP, graphs & networks. Passionate about learning and applying data science to solve real world problems.