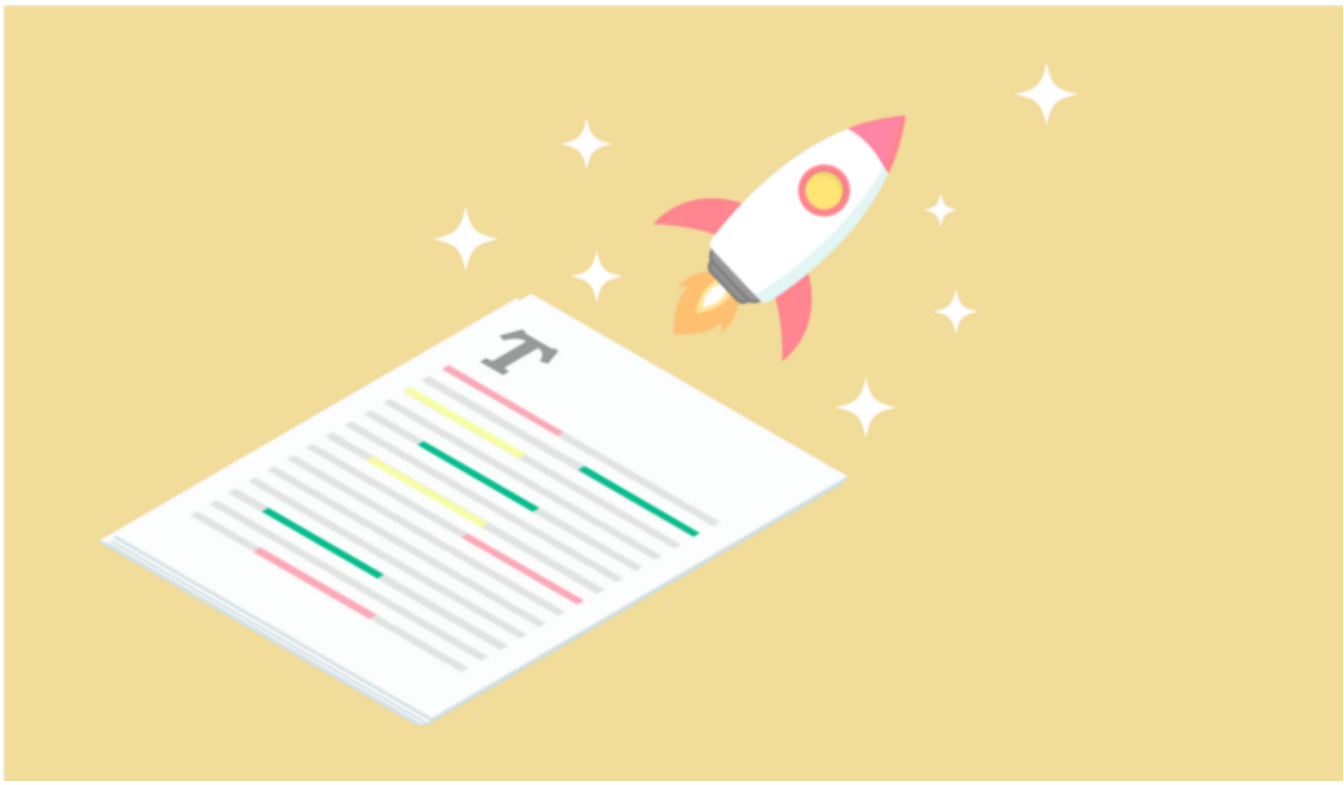
Automatic Text Summarization Made Simple with Python

Luís Gonçalves
Apr 19 · 4 min read ★



Summarization condenses a longer document into a short version while retaining core information. When this is done through a computer, we call it Automatic Text Summarization. This process can be seen as a form of compression, and it inevitably suffers from information loss, but it is essential to tackle the information overload due to abundance of textual material available on the internet, which needs to be effectively summarized to be useful.

Why Summarize Texts

There are a number of valid reasons in favor of the automatic summarization of documents:

1. Summaries reduce reading time
2. When researching documents, summaries make the selection process easier.
3. Automatic summarization algorithms are less biased than human summarizers.
4. Personalized summaries are useful in question-answering systems as they provide personalized information.

Besides, a summary enables readers to identify the basic content of a document quickly and accurately, to determine its relevance to their interests, and thus to decide whether they need to read the document in its entirety.

Approaches for automatic summarization

In general, summarization algorithms are either extractive or abstractive based on the summary generated. Extractive algorithms form summaries by identifying and pasting together relevant sections of the input text. Thus, they depend only on extraction of sentences from the original text. For such a reason, extractive methods yield naturally grammatical summaries and require relatively little linguistic analysis.

In contrast, abstractive algorithms are most human-like and mimic the process of paraphrasing a text, which may generate new text that is not present in the initial document. Texts summarized using this technique looks more human-like and produces condensed summaries. However, abstractive techniques are much harder to implement than extractive summarization techniques. It's worth mentioning existing abstractive summarizers often rely on an extractive preprocessing component to produce the abstract of the text.

Extractive summarization with Python

In this post, we will follow the extractive approach. Specifically, we will develop a method for single-document summarization using Python. To run the code, you need Scikit-learn and Spacy installed:

```
$ pip install -U spacy
$ pip install -U scikit-learn
```

Moreover, it's necessary to run the following commands on the terminal to download the spacy models:

```
$ python -m spacy download pt_core_news_sm
```

In the next step, we load the general-purpose spacy model in Portuguese and then open the file that contains the text to be summarized. Then, we apply the pipeline to the loaded text.

```
import spacy
from spacy.lang.pt.stop_words import STOP_WORDS
from sklearn.feature_extraction.text import CountVectorizer
import pt_core_news_sm
nlp = pt_core_news_sm.load()

with open("original_text.txt", "r", encoding="utf-8") as f:
        text = " ".join(f.readlines())

doc = nlp(text)
```

The text is further divided into sentences and then we convert them to a matrix of token counts. Next, we create a dictionary that contains the extracted words and their respective frequencies. Besides, we discard unnecessary words like *a, an, the, is, of*, and others, which are known as stop words. These are high-frequency words that do not carry any information and don't serve any purpose towards our goal of summarization.

```
corpus = [sent.text.lower() for sent in doc.sents ]
cv = CountVectorizer(stop_words=list(STOP_WORDS))
cv_fit=cv.fit_transform(corpus)
word_list = cv.get_feature_names();
count_list = cv_fit.toarray().sum(axis=0)

word_frequency = dict(zip(word_list,count_list))
```

Almost all extractive summarization methods have three main obstacles. The first one is how you rank sentences or words. The second obstacle is how to select the ranked units. The third obstacle is how to ensure that the selected units form an understandable summary rather than being a set of disconnected words or sentences. To solve the ranking problem, we will use a simple algorithm that determines the relevance of sentences based on the cumulative frequency of their words. Then, firstly we compute the relative frequency of each word:

```
val=sorted(word_frequency.values())
higher_word_frequencies = [word for word,freq in
word_frequency.items() if freq in val[-3:]]

print("\nWords with higher frequencies: ", higher_word_frequencies)

# gets relative frequency of words
higher_frequency = val[-1]
for word in word_frequency.keys():
    word_frequency[word] = (word_frequency[word]/higher_frequency)
```

Next, the sentences are ranked and we put each ranked sentence in an ordered list, where most important sentences come first and are chosen as part of the subset of sentences which will form the summary. Check the next code for a more detailed view.

```
sentence_rank={}
for sent in doc.sents:
    for word in sent :
        if word.text.lower() in word_frequency.keys():
            if sent in sentence_rank.keys():

sentence_rank[sent]+=word_frequency[word.text.lower()]
            else:

sentence_rank[sent]=word_frequency[word.text.lower()]

top_sentences=(sorted(sentence_rank.values())[::-1])
top_sent=top_sentences[:3]
```

Finally, we compound the summary based on the most important sentences. In this example, we used the top three best-ranked sentences:

```
summary=[]
for sent,strength in sentence_rank.items():
    if strength in top_sent:
        summary.append(sent)
    else:
        continue

for i in summary:
    print(i,end=" ")
```

Sometimes you just want to dive into code. So, the complete source code for this post can be found here. Of course, there are more complex approaches to automatic text

summarization using machine learning techniques.

I hope you have found this post useful. Leave a comment if you have any questions or ideas for new posts.

Endnote: English is not my native language. So, let me know if you have found any errors in the text. I will be grateful if you can leave your feedback at comments section.

NLP Machine Learning Text Summarization Python

About Help Legal

Get the Medium app