Keyword Extraction: from TF-IDF to BERT

How to perform keyword extraction in Python with TF-IDF, TextRank, TopicRank, YAKE!, and KeyBERT.

Shuyi Yang Nov 25, 2020 · 6 min read ★

The **keyword extraction** is one of the most required text mining tasks: given a document, the extraction algorithm should identify a set of terms that best describe its argument. In this tutorial, we are going to perform keyword extraction with five different approaches: TF-IDF, TextRank, TopicRank, YAKE!, and KeyBERT. Let's see who performs better!

Photo by Patrick Tomasso on Unsplash

Install packages for keywords extraction

Let's install the packages required in this tutorial:

```
pip install trafilatura

pip install summa

pip install git+https://github.com/smirnov-am/pytopicrank.git#egg=pytopicrank

pip install git+https://github.com/LIAAD/yake

pip install keyBERT
```

After having installed the packages above, open a Python session, and download these data from nltk:

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

Now, your environment is ready to test all six keyword extraction approaches!

Download sample text

We are going to test the algorithms by extracting the keywords from five online articles. We report here the links to these articles.

Coronavirus disease 2019

Coronavirus disease 2019(COVID-19) Other names The coronavirus 2019-nCoV acute respiratory disease Novel coronavirus...

en.wikipedia.org

Recession

In economics, a recession is a business cycle contraction when there is a general decline in economic activity...

en.wikipedia.org

Vienna

(Vienna (;[9] [10] German: Wien [viːn] listen)) is the national capital, largest city, and one of nine states of...

en.wikipedia.org

Machine learning

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as...

en.wikipedia.org

Graph database

In computing, a graph database (GDB) is a database that uses graph structures for semantic queries with nodes, edges...

en.wikipedia.org

As you can see, they regard different topics.

We extract the articles' texts from each webpage and perform some basic text cleaning. We take only the first 5000 characters.

```
import trafilatura

array_links = [
    "https://en.wikipedia.org/wiki/Coronavirus_disease_2019",
    "https://en.wikipedia.org/wiki/Recession",
    "https://en.wikipedia.org/wiki/Vienna",
    "https://en.wikipedia.org/wiki/Machine_learning",
    "https://en.wikipedia.org/wiki/Graph_database"
]

array_text = []

for l in array_links:
    html = trafilatura.fetch_url(l)
    text = trafilatura.extract(html)
    text_clean = text.replace("\n", " ").replace("\'", "")
    array_text.append(text_clean[0:5000])
```

At the end of the execution of the code above, we should have a list of cleaned texts (documents). Of course, you can substitute these texts with anything you want to test!

Now we are ready to test the keyword extraction algorithms!

Keywords Extraction with TF-IDF

TF-IDF (term frequency-inverse document frequency) is a weighting statistic that indicates if a word is important in a particular document of a corpus. For instance, the corpus could be the set of all Wikipedia articles

while a document is a particular article. Given a particular document, if we consider the word “the”, it is present several times so it has a high TF, but it is present also in almost every article on Wikipedia, hence, IDF is very low. Overall, the word “the” has a low TF-IDF score. We suppose that the keywords of a document should have a high TF-IDF score.

Given a document, it is easy to compute the TF score of every word but for the IDF score, we need a huge corpus of similar documents which is not provided most of the time. In this tutorial, we are going to use the IDF scores computed on Wikipedia. You can download them here:

Wikipedia TF-IDF Dataset Release

Mikhail Galkin; Valentin Malykh Pre-computed IDF stats over all EN Wiki articles. Source: 5989879 articles from EN...

zenodo.org

We use the file wiki_tfidf_terms.csv inside the zip folder but you could try to use the stems file in order to improve the accuracy for the keyword extractor (don’t forget to perform the stemming in this case!).

```
from itertools import islice
from tqdm.notebook import tqdm
from re import sub

num_lines = sum(1 for line in open("wiki_tfidf_terms.csv"))

with open("wiki_tfidf_terms.csv") as file:
    dict_idf = {}
    with tqdm(total=num_lines) as pbar:
        for i, line in tqdm(islice(enumerate(file), 1, None)):
            try:
                cells = line.split(",")
                idf = float(sub("[^0-9.]", "", cells[3]))
                dict_idf[cells[0]] = idf
            except:
                print("Error on: " + line)
            finally:
                pbar.update(1)
```

Then, for each article inside our list, we compute the TF score of its words.

```
from sklearn.feature_extraction.text import CountVectorizer
from numpy import array, log

vectorizer = CountVectorizer()
tf = vectorizer.fit_transform([x.lower() for x in array_text])
tf = tf.toarray()
tf = log(tf + 1)
```

Now, we are ready to multiply TF with IDF.

```
tfidf = tf.copy()
words = array(vectorizer.get_feature_names())

for k in tqdm(dict_idf.keys()):
    if k in words:
        tfidf[:, words == k] = tfidf[:, words == k] * dict_idf[k]
    pbar.update(1)
```

Let’s print the extracted keywords.

```
for j in range(tfidf.shape[0]):
    print("Keywords of article", str(j+1), words[tfidf[j,
:].argsort()[-5:][::-1]])
```

Output:

```
Keywords of article 1
 ['covid' 'coronavirus' 'symptoms' 'respiratory' 'cov']
Keywords of article 2
 ['recessions' 'recession' 'nber' 'gdp' 'shaped']
Keywords of article 3
 ['vienna' 'km2' 'citys' 'sq' 'vedunia']
Keywords of article 4
 ['learning' 'machine' 'algorithms' 'tasks' 'unsupervised']
```

```
Keywords of article 5
 ['graph' 'databases' 'relational' 'database' 'nosql']
```

Keywords Extraction with TextRank

TextRank is an unsupervised method to perform keyword and sentence extraction. It is based on a graph where each node is a word and the edges are constructed by observing the co-occurrence of words inside a moving window of predefined size. Important nodes of the graph, computed with an algorithm similar to PageRank, represent keywords in the text.

We are going to use the keywords extractor implemented in summa.

```
from summa import keywords

for j in range(len(array_text)):
    print("Keywords of article", str(j+1), "\n",
(keywords.keywords(array_text[j], words=5)).split("\n"))
```

Output:

```
Keywords of article 1
 ['symptoms', 'include', 'including', 'covid', 'cough', 'coughs',
'respiratory']
Keywords of article 2
 ['recession', 'recessions', 'economics', 'economic', 'shapes',
'shape', 'governments', 'government', 'policies', 'policy']
Keywords of article 3
 ['vienna', 'viennas', 'city', 'cities', 'citys', 'rank', 'ranked',
'mi', 'world', 'worlds']
Keywords of article 4
 ['learn', 'learns', 'machine learning', 'algorithms', 'algorithm',
'data', 'machines', 'tasks', 'task']
Keywords of article 5
 ['graphs', 'graph database', 'databases', 'model', 'models', 'data',
'relates', 'relational', 'relation']
```

Keywords Extraction with TopicRank

TopicRank is another unsupervised graph-based keyphrase extractor. Different from TextRank, in this case, the nodes of the graph are topics and each topic is a cluster of similar single and multiword expressions.

Let's try the Python implementation of this keywords extractor.

```
from pytopicrank import TopicRank

for j in range(len(array_text)):
    tr = TopicRank(array_text[j])
    print("Keywords of article", str(j+1), "\n", tr.get_top_n(n=5,
extract_strategy='first'))
```

Output:

```
Keywords of article 1
 ['cause covid', 'symptoms', 'day', 'infected person', 'spreads']
Keywords of article 2
 ['onset recession', 'consecutive quarters', 'decline', 'shaped',
'activity economic']
Keywords of article 3
 ['population citys', 'viˈɛnə vienna', 'heritage worlds', 'austria',
'elevation sq mi']
Keywords of article 4
 ['machines', 'computer', 'tasks', 'field', 'data']
Keywords of article 5
 ['relationships database', 'data', 'commercial database graph',
'edge', 'graph']
```

Keywords Extraction with YAKE!

YAKE! is an unsupervised keyword extraction algorithm based on the features extracted from the documents and it is multilingual: English, Italian, German, Dutch, Spanish, Finnish, French, Polish, Turkish, Portuguese, and Arabic.

You can try their demo directly on their website:

YAKE! Demo for Keyword Extraction

Edit description

yake.inesctec.pt

Luckily, also in this case, there is already a Python implementation. Let's try it on our documents.

```
from yake import KeywordExtractor

kw_extractor = KeywordExtractor(lan="en", n=1, top=5)
for j in range(len(array_text)):
    keywords = kw_extractor.extract_keywords(text=array_text[j])
    keywords = [x for x, y in keywords]
    print("Keywords of article", str(j+1), "\n", keywords)
```

Output:

```
Keywords of article 1
 ['symptoms', 'respiratory', 'acute', 'coronavirus', 'disease']
Keywords of article 2
 ['economic', 'recession', 'recessions', 'gdp', 'united']
Keywords of article 3
 ['vienna', 'city', 'capital', 'state', 'german']
Keywords of article 4
 ['learning', 'machine', 'computer', 'tasks', 'computers']
Keywords of article 5
 ['graph', 'databases', 'database', 'data', 'relationships']
```

Keywords Extraction with BERT

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model for natural language processing. Pretrained models can transform sentences or words in language representation consisting of an array of numbers (embedding). Sentences or words having similar latent representations (embedding) should have similar semantic meanings. An implementation that uses this approach to extract the keywords of a text is KeyBERT.

Let's see how this keyword extractor performs.

```
from keybert import KeyBERT

kw_extractor = KeyBERT('distilbert-base-nli-mean-tokens')

for j in range(len(array_text)):
    keywords = kw_extractor.extract_keywords(array_text[j],
keyphrase_length=1, stop_words='english')
    print("Keywords of article", str(j+1), "\n", keywords)
```

Output:

```
Keywords of article 1
 ['coronavirus', 'pneumonia', 'vaccines', 'virus', 'viral']
Keywords of article 2
 ['recessions', 'recession', 'unemployment', 'pandemic', 'worsening']
Keywords of article 3
 ['austrias', 'austria', 'vienna', 'austrian', 'beethoven']
Keywords of article 4
 ['algorithms', 'algorithm', 'computational', 'computers',
'mathematical']
Keywords of article 5
 ['graphs', 'graph', 'databases', 'web', 'database']
```

Conclusion

In this tutorial, we have seen different approaches to extract keywords from a given text. All of them **don't require any training**. This list is far from being complete! Check the most recent review to have a complete overview!

We have tested both simple approaches like TF-IDF and advanced models like BERT. Unfortunately, there isn't a model that performs well on every document. It depends on the type of document, the context, and the corpus used for the model pretraining. A more complete preprocessing with a combination of different keyword extraction approaches should definitely improve the performance!

References

- https://scholar.google.it/scholar?q=keyword+extraction+review
- http://www.tfidf.com/
- https://www.aclweb.org/anthology/W04-3252.pdf
- https://www.aclweb.org/anthology/I13-1062/
- http://yake.inesctec.pt/
- https://towardsdatascience.com/keyword-extraction-with-bert-724efca412ea

Contacts: LinkedIn | Twitter

Thanks to Anne Bonner.

Keyword Extraction Bert NLP Tf Idf Algorithms