How to compute text similarity on a website with TF-IDF in Python

A simple and effective approach to text similarity with TF-IDF and Pandas

Calculating the similarity between two pieces of text is a very useful activity in the field of data mining and natural language processing (NLP). This allows both to **isolate anomalies and diagnose for specific problems**, for example very similar or very different texts on a blog, or to group similar entities into useful categories.

In this article we are going to use a script published here to scrape a blog and create a small corpus on which to apply a similarity calculation algorithm based on TF-IDF in Python.

In particular, we will use a library called Trafilatura to retrieve all the articles from the target website via its sitemap and place them in a Pandas dataframe for processing.

I invite the reader to read the article I linked above to understand how the extraction algorithm works in more detail.

For simplicity, in the example we are going to analyze diariodiunanalista.it, my own blog in Italian on data science, in order to understand if there are articles that are too similar to each other. In the process I will diagnose my own work, and perhaps come up with some cool insights!

This has significant SEO repercussions — in fact, similar articles give rise to the phenomenon of *content cannibalization*: when two pieces belonging to the same website compete for the same position on Google. **We want to avoid that and identifying these kinds of situations is the first step in doing so.**

The requirements

The libraries we will need will be Pandas, Numpy, NLTK, Sklearn, TQDM, Matplotlib and Seaborn.

Let’s import them into our Python script.

```python
import pandas as pd
import numpy as np

import nltk
from nltk.corpus import stopwords
import string

from tqdm import tqdm

import matplotlib.pyplot as plt
import seaborn as sns
```

text_sim_tfidf_ita1.py hosted with ❤ by GitHub view raw

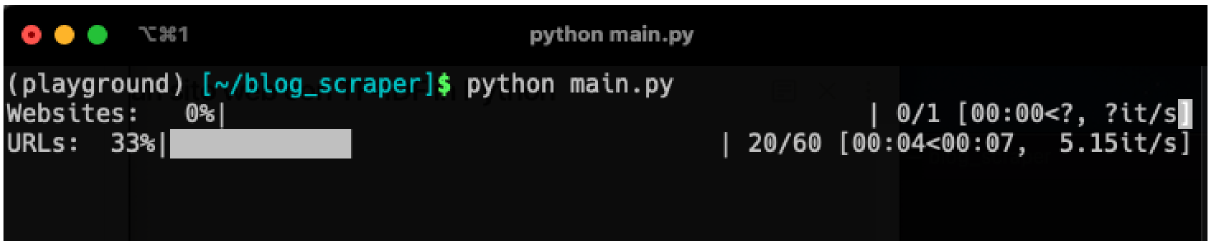
Additionally, we will need to run the nltk.download('stopwords') command to install the NLTK stopwords. A stopword is a word that does not contribute in an important way to the meaning of a sentence and we will need them to preprocess our texts.

Creating the dataset

Let’s run the software from in the article mentioned above.

```python
if __name__ == "__main__":

  list_of_websites = [
    "https://www.diariodiunanalista.it/",
  ]

  df = create_dataset(list_of_websites)

  df.to_csv("dataset.csv", index=False)
```

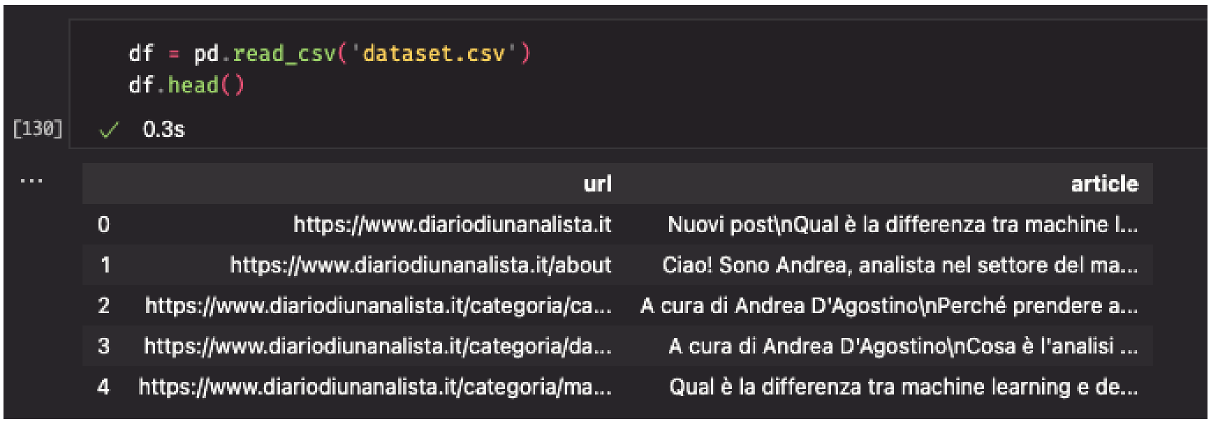
text_sim_tfidf_ita2.py hosted with ❤ by GitHub view raw



```
(playground) [~/blog_scraper]$ python main.py
Websites:   0%|                                          | 0/1 [00:00<?, ?it/s]
URLs:  33%|█████████                    | 20/60 [00:04<00:07,  5.15it/s]
```

Data collection process. Image by author.

Let’s take a look at our dataset.



```
df = pd.read_csv('dataset.csv')
df.head()
```

[130] ✓ 0.3s

	url	article
0	https://www.diariodiunanalista.it	Nuovi post\nQual è la differenza tra machine l...
1	https://www.diariodiunanalista.it/about	Ciao! Sono Andrea, analista nel settore del ma...
2	https://www.diariodiunanalista.it/categoria/ca...	A cura di Andrea D'Agostino\nPerché prendere a...
3	https://www.diariodiunanalista.it/categoria/da...	A cura di Andrea D'Agostino\nCosa è l'analisi ...
4	https://www.diariodiunanalista.it/categoria/ma...	Qual è la differenza tra machine learning e de...

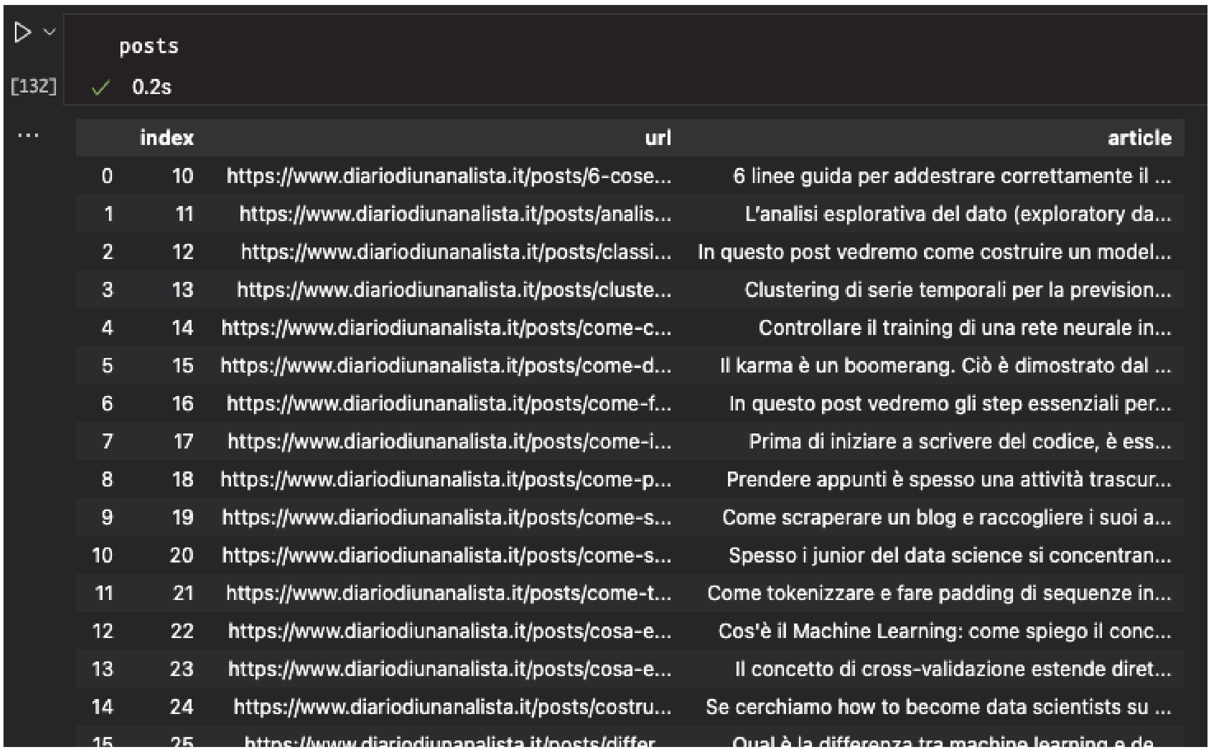
Preview of our dataset. Image by author.

From the taxonomy of the URLs we notice how all the posts are collected under */posts/* — this allows us to isolate only the actual articles, leaving out pages, categories, tags and more.

We use the following code to apply this selection

```python
posts = df[df.url.str.contains('post')]
posts.reset_index(inplace=True)
```

text_sim_tfidf_ita3.py hosted with ❤ by GitHub view raw



```
posts
```

[132] ✓ 0.2s

	index	url	article
0	10	https://www.diariodiunanalista.it/posts/6-cose...	6 linee guida per addestrare correttamente il ...
1	11	https://www.diariodiunanalista.it/posts/analis...	L'analisi esplorativa del dato (exploratory da...
2	12	https://www.diariodiunanalista.it/posts/classi...	In questo post vedremo come costruire un model...
3	13	https://www.diariodiunanalista.it/posts/cluste...	Clustering di serie temporali per la prevision...
4	14	https://www.diariodiunanalista.it/posts/come-c...	Controllare il training di una rete neurale in...
5	15	https://www.diariodiunanalista.it/posts/come-d...	Il karma è un boomerang. Ciò è dimostrato dal ...
6	16	https://www.diariodiunanalista.it/posts/come-f...	In questo post vedremo gli step essenziali per...
7	17	https://www.diariodiunanalista.it/posts/come-i...	Prima di iniziare a scrivere del codice, è ess...
8	18	https://www.diariodiunanalista.it/posts/come-p...	Prendere appunti è spesso una attività trascur...
9	19	https://www.diariodiunanalista.it/posts/come-s...	Come scraperare un blog e raccogliere i suoi a...
10	20	https://www.diariodiunanalista.it/posts/come-s...	Spesso i junior del data science si concentran...
11	21	https://www.diariodiunanalista.it/posts/come-t...	Come tokenizzare e fare padding di sequenze in...
12	22	https://www.diariodiunanalista.it/posts/cosa-e...	Cos'è il Machine Learning: come spiego il conc...
13	23	https://www.diariodiunanalista.it/posts/cosa-e...	Il concetto di cross-validazione estende diret...
14	24	https://www.diariodiunanalista.it/posts/costru...	Se cerchiamo how to become data scientists su ...
15	25	https://www.diariodiunanalista.it/posts/differ...	Qual è la differenza tra machine learning e de...

Preview of our dataset. Image by author.

And here is our corpus. At the time of writing this piece there are around 30 articles — therefore it is a very small corpus. It will still be fine for our example.

Preprocessing of texts

We will apply some bare minimum text preprocessing to replicate a real application pipeline. This can be expanded to complement the reader's requirements.

Preprocessing steps

We are going to apply these preprocessing steps:

- punctuation removal
- lowercase application

This will all be done in a very simple function, which uses the standard *string* and NLTK libraries.

```python
remove_punctuation_map = dict((ord(char), None) for char in string.punctuation)


def preprocess(text):
  return nltk.word_tokenize(text.lower().translate(remove_punctuation_map))
```

text_sim_tfidf_ita4.py hosted with ❤ by GitHub view raw

This function will be used by the TF-IDF vectorizer (which we will define shortly) to normalize the text.

The similarity calculation algorithm

First, let’s define our stopwords by saving them in a variable

ita_stopwords = stopwords.words('italian')

Now we import TfIdfVectorizer from Sklearn, passing it the preprocessing function and the stopwords.

```python
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(tokenizer=preprocess, stop_words=ita_stopwords)
```

text_sim_tfidf_ita5.py hosted with ❤ by GitHub view raw

The TF-IDF vectorizer will convert each text into its vector representation. This will allow us to treat each text as a series of points in a multidimensional space.

The way in which we are going to calculate the similarity will be through the **computation of the cosine** between the vectors that make up the texts we are comparing. The similarity value is between -1 and +1. **A value of +1 indicates two essentially equal texts, while -1 indicates complete dissociation.**

I invite the interested reader to read more on the subject on the dedicated Wikipedia page.

Now we'll define a function called `compute_similarity` which will use the vectorizer to convert the texts to number and apply the function to calculate the similarity cosine with TF-IDF vectors.

```python
def compute_similarity(a, b):
  tfidf = vectorizer.fit_transform([a, b])

  return ((tfidf * tfidf.T).toarray())[0,1]
```

text_sim_tfidf_ita6.py hosted with ❤ by GitHub view raw

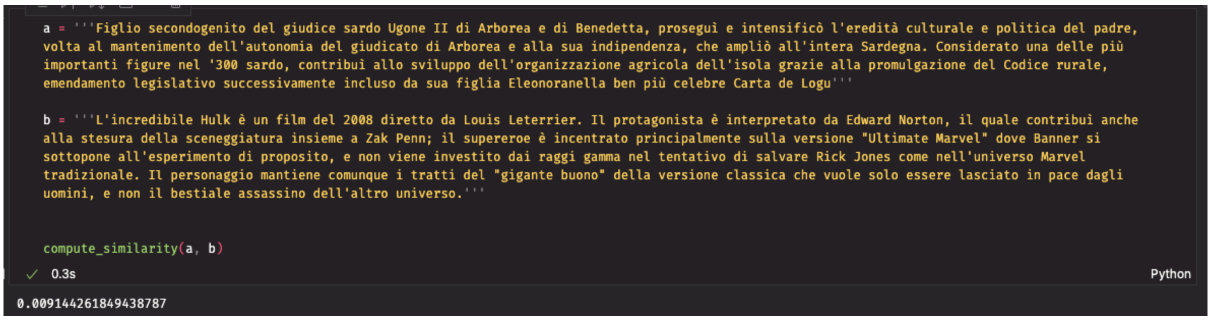
Let’s test the code

Let’s take two texts from Wikipedia in Italian as an example.

Figlio secondogenito del giudice sardo Ugone II di Arborea e di Benedetta, proseguì e intensificò l’eredità culturale e politica del padre, volta al mantenimento dell’autonomia del giudicato di Arborea e alla sua indipendenza, che ampliò all’intera Sardegna. Considerato una delle più importanti figure nel ‘300 sardo, contribuì allo sviluppo dell’organizzazione agricola dell’isola grazie alla promulgazione del Codice rurale, emendamento legislativo successivamente incluso da sua figlia Eleonora nella ben più celebre Carta de Logu

L’incredibile Hulk è un film del 2008 diretto da Louis Leterrier. Il protagonista è interpretato da Edward Norton, il quale contribuì anche alla stesura della sceneggiatura insieme a Zak Penn; il supereroe è incentrato principalmente sulla versione Ultimate si sottopone all’esperimento di proposito, e non viene investito dai raggi gamma nel tentativo di salvare Rick Jones come nell’universo Marvel tradizionale. Il personaggio mantiene comunque i tratti del “gigante buono” della versione classica che vuole solo essere lasciato in pace dagli uomini, e non il bestiale assassino dell’altro universo.

Let's apply the `cosine_similarity` function to test how similar these two texts are. I expect a fairly low value as they deal with different topics and don’t use the same terminology.

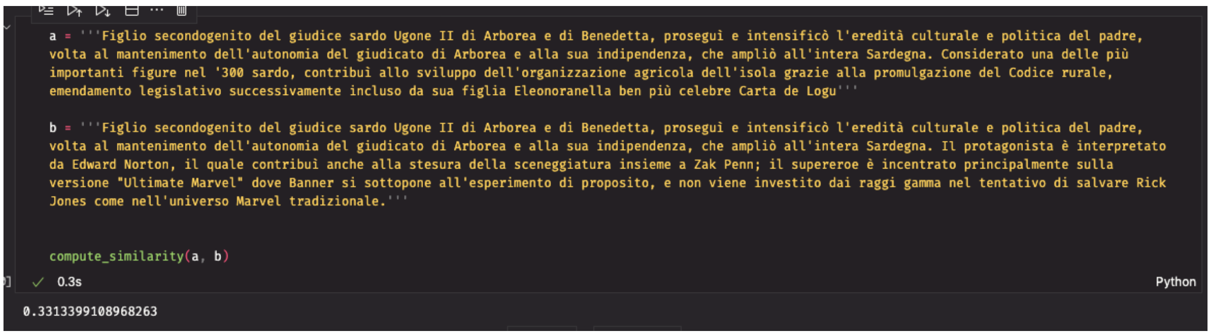


```
a = '''Figlio secondogenito del giudice sardo Ugone II di Arborea e di Benedetta, proseguì e intensificò l'eredità culturale e politica del padre,
volta al mantenimento dell'autonomia del giudicato di Arborea e alla sua indipendenza, che ampliò all'intera Sardegna. Considerato una delle più
importanti figure nel '300 sardo, contribuì allo sviluppo dell'organizzazione agricola dell'isola grazie alla promulgazione del Codice rurale,
emendamento legislativo successivamente incluso da sua figlia Eleonoranella ben più celebre Carta de Logu'''

b = '''L'incredibile Hulk è un film del 2008 diretto da Louis Leterrier. Il protagonista è interpretato da Edward Norton, il quale contribuì anche
alla stesura della sceneggiatura insieme a Zak Penn; il supereroe è incentrato principalmente sulla versione "Ultimate Marvel" dove Banner si
sottopone all'esperimento di proposito, e non viene investito dai raggi gamma nel tentativo di salvare Rick Jones come nell'universo Marvel
tradizionale. Il personaggio mantiene comunque i tratti del "gigante buono" della versione classica che vuole solo essere lasciato in pace dagli
uomini, e non il bestiale assassino dell'altro universo.'''


compute_similarity(a, b)
```

```
0.009144261849438787
```

Application of the algorithm on two pieces of text. Image by author.

The two texts show a very low similarity, close to 0. Let's test it now with two similar texts. L'll copy part of the second text into the first, keeping a similar length.



```
a = '''Figlio secondogenito del giudice sardo Ugone II di Arborea e di Benedetta, proseguì e intensificò l'eredità culturale e politica del padre,
volta al mantenimento dell'autonomia del giudicato di Arborea e alla sua indipendenza, che ampliò all'intera Sardegna. Considerato una delle più
importanti figure nel '300 sardo, contribuì allo sviluppo dell'organizzazione agricola dell'isola grazie alla promulgazione del Codice rurale,
emendamento legislativo successivamente incluso da sua figlia Eleonoranella ben più celebre Carta de Logu'''

b = '''Figlio secondogenito del giudice sardo Ugone II di Arborea e di Benedetta, proseguì e intensificò l'eredità culturale e politica del padre,
volta al mantenimento dell'autonomia del giudicato di Arborea e alla sua indipendenza, che ampliò all'intera Sardegna. Il protagonista è interpretato
da Edward Norton, il quale contribuì anche alla stesura della sceneggiatura insieme a Zak Penn; il supereroe è incentrato principalmente sulla
versione "Ultimate Marvel" dove Banner si sottopone all'esperimento di proposito, e non viene investito dai raggi gamma nel tentativo di salvare Rick
Jones come nell'universo Marvel tradizionale.'''


compute_similarity(a, b)
```

```
0.3313399108968263
```

Application of the algorithm on two pieces of text. Image by author.

The similarity is now 0.33. It seems to work fine.
Now we apply this method to all of corpus, in a pairwise fashion.

```
M = np.zeros((posts.shape[0], posts.shape[0])) # we create a 30x30 matrix to contain the results


for i, row in tqdm(posts.iterrows(), total=posts.shape[0], desc='1st level'): # we define i
        for j, next_row in posts.iterrows(): # we define j
                M[i, j] = compute_similarity(row.article, next_row.article) # we populate the mat
```

text_sim_tfidf_eng1.py hosted with ❤ by GitHub view raw

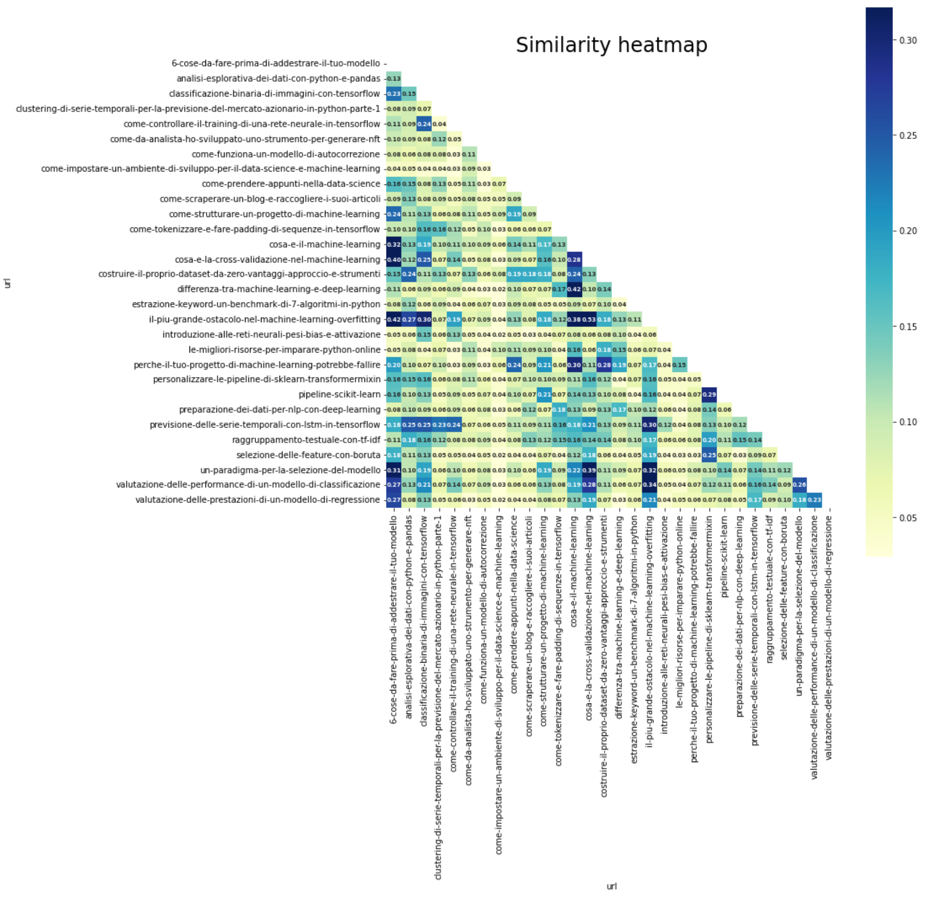
Let's go into detail on what this piece of code does.

- We create a 30x30 matrix called M
- We iterate line by line on the dataframe to access article_i
- We iterate line by line on the same dataframe again, to access article_j
- We run compute_similarity on article_i and on article_j to obtain the similarity
- We save this value in M at position i, j

M can easily be converted into a Pandas dataframe and this allows us to **build a heatmap** with Seaborn.

```python
labels = posts.url.str.split('/').str[3:].str[1] # we extract the titles of the articles from th
similarity_df = pd.DataFrame(M, columns=labels, index=labels) # let's create the dataframe
mask = np.triu(np.ones_like(similarity_df)) # we apply a mask to remove the top of the heatmap

# let's create the viz
plt.figure(figsize=(12, 12))
sns.heatmap(
                        similarity_df,
                        square=True,
                        annot=True,
                        robust=True,
                        fmt='.2f',
                        annot_kws={'size': 7, 'fontweight': 'bold'},
                        yticklabels=similarity_df.columns
                        xticklabels=similarity_df.columns,
                        cmap="YlGnBu",
                        mask=mask
)

plt.title('Similarity heatmap', fontdict={'fontsize': 24})
plt.show()
```

text_sim_tfidf_eng2.py hosted with ❤ by GitHub view raw



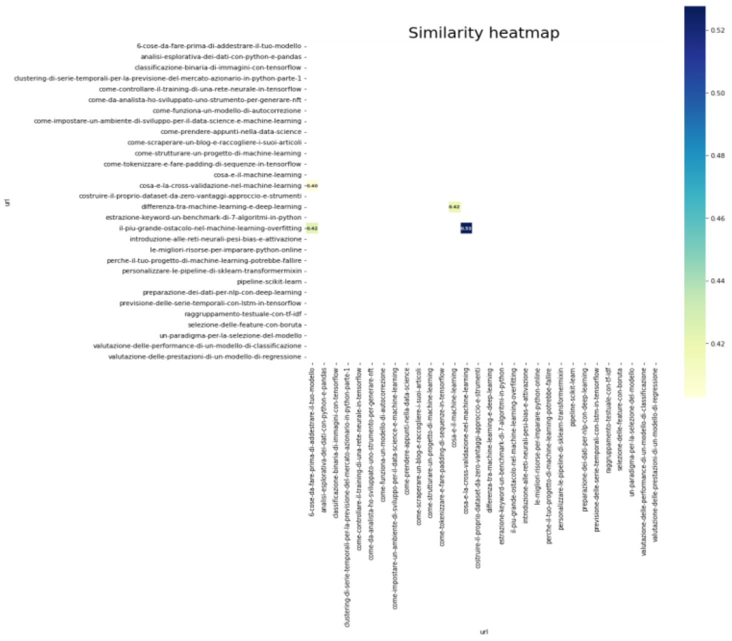
Heatmap of similarities. Image by author.

The heatmap highlights anomalies using brighter or duller colors based on the similarity value obtained.

Let’s make a small change to the code to select only elements that have **similarity greater than 0.40.**

```python
top = similarity_df[similarity_df > 0.4] # change this
mask = np.triu(np.ones_like(top))

# let's create the viz
plt.figure(figsize=(12, 12))
sns.heatmap(
                        top,
                        square=True,
                        annot=True,
                        robust=True,
                        fmt='.2f',
                        annot_kws={'size': 7, 'fontweight': 'bold'},
                        yticklabels=top.columns
                        xticklabels=top.columns,
                        cmap="YlGnBu",
                        mask=mask
)

plt.title('Similarity heatmap', fontdict={'fontsize': 24})
plt.show()
```

text_sim_tfidf_eng4.py hosted with ❤ by GitHub view raw



Heatmap of similarities filtered by values above 0.4. Image by author.

We see 4 pages with a similarity index greater than 0.4.

In particular we see these combinations:

- 6 things to do before training your model -> the biggest obstacle in machine learning — overfitting
- 6 things to do before training your model -> what is cross-validation in machine learning
- what is cross-validation in machine learning -> the biggest obstacle in machine learning — overfitting
- what is machine learning -> what is the difference between machine learning and deep learning

The similarity between some of these pairs is also present among other pairs that show high similarity.

These articles share the topic, namely that of machine learning and some best practices.