

Open in app ↗



Search



This member-only story is on us. [Upgrade](#) to access all of Medium.

★ Member-only story

Text Clustering with TF-IDF in Python

Explanation of a simple pipeline for text clustering. Full example and code



Andrea D'Agostino · [Follow](#)

Published in MLearning.ai

8 min read · Nov 24, 2021



Listen



Share

... More



Photo by [Andrew Wulf](#) on [Unsplash](#)

TF-IDF is a well known and documented **vectorization technique** in data science. Vectorization is the act of converting data into a numerical format in such a way that

a statistical model can interpret it and make predictions.

In this article we will see how to convert a corpus of text into numerical format and apply machine learning algorithms to bring out interesting patterns and anomalies.

Methodology

We will use a dataset provided by Sklearn to have a replicable corpus. After that, we will use the KMeans algorithm to group the vectors generated by the TF-IDF. We will then use Principal Component Analysis to visualize our groups and bring out common or unusual characteristics of the texts present in our corpus.

Here is what we'll do

- import the dataset
- apply **preprocessing** to our corpus to remove words and symbols which, when converted into numerical format, do not add value to our model
- use **TF-IDF** as a vectorization algorithm
- apply **KMeans** to group our data
- apply **PCA** to reduce the dimensionality of our vectors to 2 for visualization purposes
- **interpret** the data

The Analysis

Our Dataset

For this example we will use Scikit-Learn's API, *sklearn.datasets*, which allows us to access a famous dataset for linguistic analysis, the **20newsgroups dataset**. A newsgroup is an online user discussion group, such as a forum. Sklearn allows us to access different categories of content. We will use texts that have to do with technology, religion and sport.

```
1  # import the dataset from sklearn
2  from sklearn.datasets import fetch_20newsgroups
3  from sklearn.feature_extraction.text import TfidfVectorizer
4  from sklearn.cluster import KMeans
5  from sklearn.decomposition import PCA
6
7  # import other required libs
8  import pandas as pd
9  import numpy as np
10
11 # string manipulation libs
12 import re
13 import string
14 import nltk
15 from nltk.corpus import stopwords
16
17 # viz libs
18 import matplotlib.pyplot as plt
19 import seaborn as sns
20
21
22 categories = [
23     'comp.graphics',
24     'comp.os.ms-windows.misc',
25     'rec.sport.baseball',
26     'rec.sport.hockey',
27     'alt.atheism',
28     'soc.religion.christian',
29 ]
30 dataset = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, remove=('head
```

clustering-ana-1.py hosted with ❤️ by GitHub

view raw

If we access the first element with `dataset['data'][0]` we see

They tried their best not to show it, believe me. I'm surprised they couldn't find a sprint car race (mini cars through pigpens, indeed!) On short notice.

George

It is possible that your data is different due to `shuffle=True`, which randomizes the order of the elements of the dataset. The number of items in our dataset is 3451.

Let's create a Pandas dataframe from our dataset

```
1 df = pd.DataFrame(dataset.data, columns=["corpus"])
```

clustering_3.py hosted with ❤ by GitHub

[view raw](#)

	corpus
0	\nThey tried their best not to show it, believ...
1	\nStankiewicz? I doubt it.\n\nKoufax was one ...
2	\n[deletia- and so on]\n\nI seem to have been ...
3	Excuse the sheer newbiness of this post, but ...
4	=====...
...	...
3446	\n Or, with no dictionary available, they cou...
3447	\n\nSorry to disappoint you but the Red Wings ...
3448	\n: Can anyone tell me where to find a MPEG vi...
3449	\n
3450	\nHey Valentine, I don't see Boston with any w...
3451 rows x 1 columns	

Our corpus before the cleaning phase

Note that there are `\n`, `===` and other symbols that should be removed to properly train our model.

Preprocessing

Along with the symbols mentioned, we also want remove stopwords . These are a series of words that add no information to our model. An example of a stopword in English are articles, conjunctions, and so on.

We will use the NLTK library to import the stopwords and display them

```

1  import nltk
2  from nltk.corpus import stopwords
3  # nltk.download('stopwords')
4
5  stopwords.words("english")[:10] # <-- import the english stopwords

```

clustering_eng_4.py hosted with ❤ by GitHub

[view raw](#)

>>> ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"] and so on.

Now let's create a *preprocess_text* function that takes some text as input and returns a clean version of it.

```

1  def preprocess_text(text: str, remove_stopwords: bool) -> str:
2      """This utility function sanitizes a string by:
3          - removing links
4          - removing special characters
5          - removing numbers
6          - removing stopwords
7          - transforming in lowercase
8          - removing excessive whitespaces
9      Args:
10         text (str): the input text you want to clean
11         remove_stopwords (bool): whether or not to remove stopwords
12      Returns:
13         str: the cleaned text
14      """
15
16      # remove links
17      text = re.sub(r"http\S+", "", text)
18      # remove special chars and numbers
19      text = re.sub("[^A-Za-z]+", " ", text)
20      # remove stopwords
21      if remove_stopwords:
22          # 1. tokenize
23          tokens = nltk.word_tokenize(text)
24          # 2. check if stopword
25          tokens = [w for w in tokens if not w.lower() in stopwords.words("english")]
26          # 3. join back together
27          text = " ".join(tokens)
28      # return text in lower case and stripped of whitespaces
29      text = text.lower().strip()
30      return text

```

clustering_eng_preprocessing.py hosted with ❤ by GitHub

[view raw](#)

Here is the same previous document, preprocessed

tried best show believe im surprised couldnt find sprint car race mini cars pigpens indeed short notice george

Let's apply the function to the entire dataframe

```
1 df['cleaned'] = df['corpus'].apply(lambda x: preprocess_text(x, remove_stopwords=True))
```

clustering_5.py hosted with ❤ by GitHub

[view raw](#)

	corpus	cleaned
0	\nThey tried their best not to show it, believ...	they tried their best not to show it believe m...
1	\nStankiewicz? I doubt it.\n\nKoufax was one ...	stankiewicz i doubt it koufax was one of two j...
2	\n[deletia- and so on]\n\nI seem to have been ...	deletia and so on i seem to have been rather u...
3	Excuse the sheer newbiness of this post, but ...	excuse the sheer newbiness of this post but i...
4	=====...	
...
3446	\n Or, with no dictionary available, they cou...	or with no dictionary available they could gai...
3447	\n\nSorry to disappoint you but the Red Wings ...	sorry to disappoint you but the red wings earn...
3448	\n: Can anyone tell me where to find a MPEG vi...	can anyone tell me where to find a mpeg viewer...
3449	\n	
3450	\nHey Valentine, I don't see Boston with any w...	hey valentine i dont see boston with any world...

Cleaned corpus series added to our dataframe

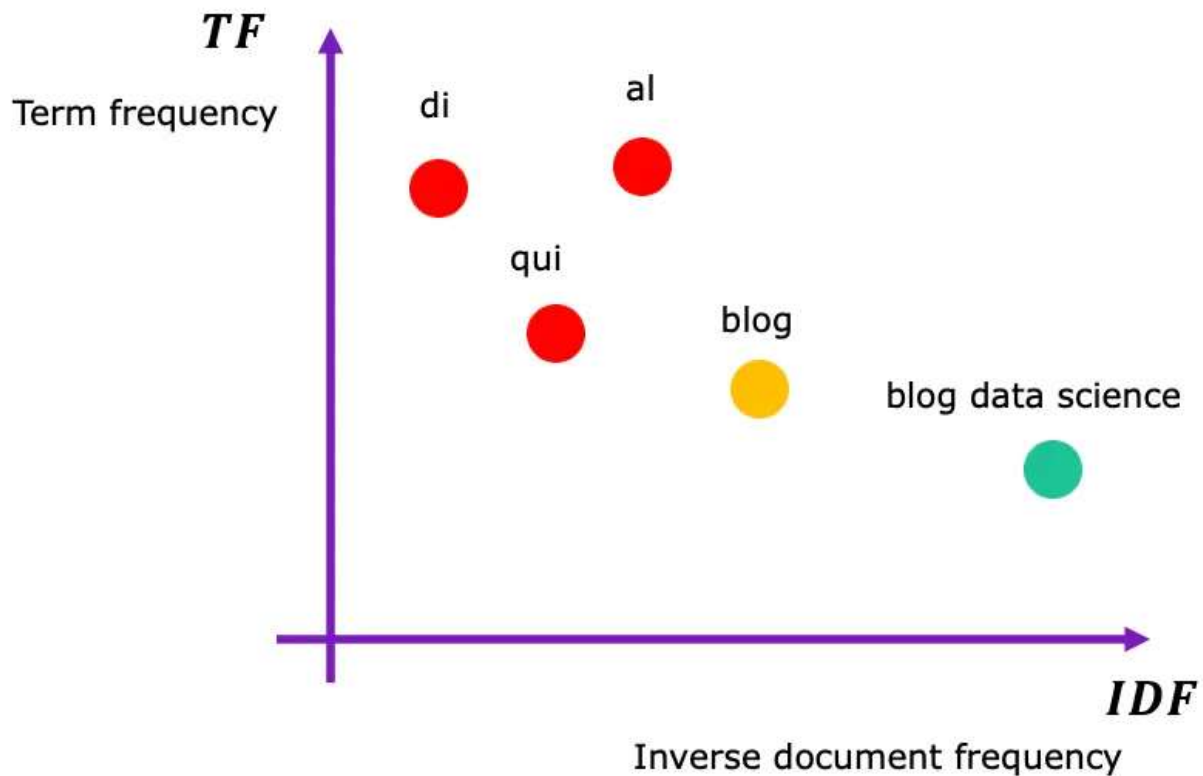
Now we are ready to perform vectorization.

TF-IDF Vectorization

The TF-IDF converts our corpus into a numerical format by bringing out specific terms, weighing very rare or very common terms differently in order to assign them a low score.

TF stands for term frequency, while IDF stands for inverse document frequency.

The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain that word.



TF-IDF: the term in green gets a high score because it is more specific

With this vectorization technique we are able to group our documents considering the most important terms that constitute them. To read more about how TF-IDF works, read [here](#).

With Sklearn, applying TF-IDF is trivial

```
1 # initialize the vectorizer
2 vectorizer = TfidfVectorizer(sublinear_tf=True, min_df=5, max_df=0.95)
3 # fit_transform applies TF-IDF to clean texts - we save the array of vectors in X
4 X = vectorizer.fit_transform(df['cleaned'])
```

clustering_eng_vec.py hosted with ❤ by GitHub

[view raw](#)

X is the array of vectors that will be used to train the KMeans model. The default behavior of Sklearn is to create a **sparse matrix**. Vectorization generates vectors similar to this

$vector_a = [1.204, 0, 0, 0, 0, 0, 0, \dots, 0]$

The vector is made up of a single value not equal to 0. In Sklearn, a sparse matrix is nothing more than a matrix that indexes the position of values and 0s instead of

storing it like any other matrix. This is a mechanism for saving RAM and computing power. The convenience is that the sparse matrix is accepted by most machine learning algorithms, as well as KMeans. In fact, the latter will use the data present in the sparse matrix to find groups and patterns.

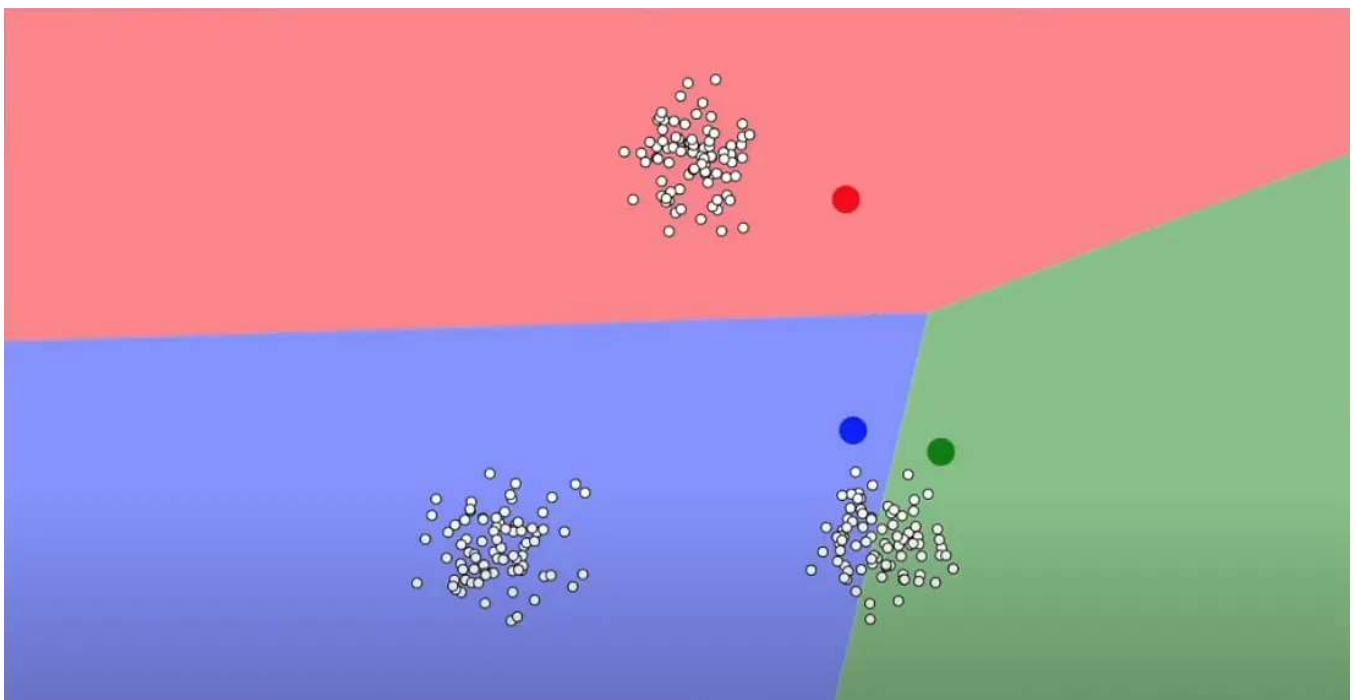
If we use `X.toarray()` we actually see the complete matrix, not sparse.

```
array([[0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       [0.10747419, 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       ...,
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ],
       [0., 0., 0., ..., 0., 0., ]])
```

Vector representation of the sparse matrix

Implementation of KMeans

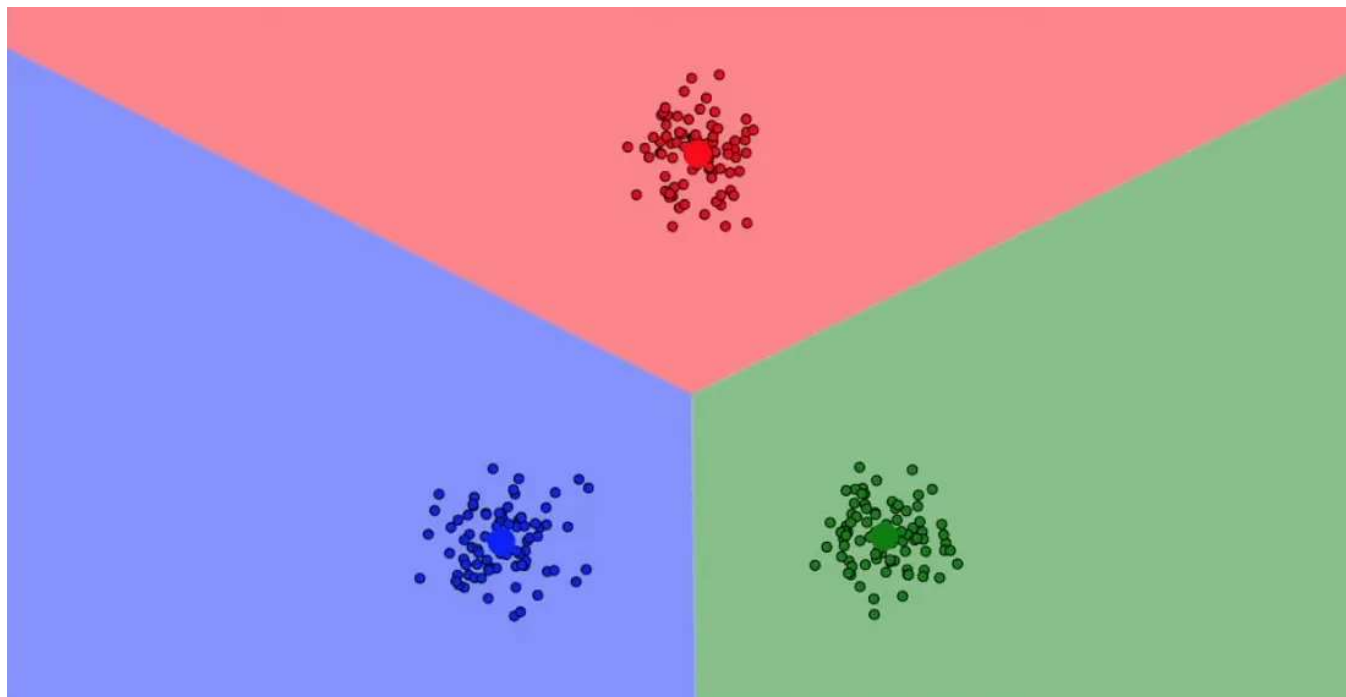
KMeans is one of the most known and used unsupervised algorithms in data science and is used to group a set of data into a defined number of groups. The idea behind it is very simple: the algorithm initializes random positions (called centroids, the red, blue and green points in the screenshot below) in the vector plane and assigns the point to the nearest centroid.



First step of KMeans — initialization of the centroids

Screenshot taken from https://www.youtube.com/watch?v=R2e3Ls9H_fc

The algorithm calculates the average position (or, if it helps the interpretation, the “center of gravity”) of the points and moves the respective centroid to that position and updates the group to which each point belongs. The algorithm converges when all points are at the minimum distance from their respective centroid.



Convergence of the KMeans — cluster discovery

Screenshot taken from https://www.youtube.com/watch?v=R2e3Ls9H_fc

Let's continue with our project by going to use Sklearn again

With this code snippet we trained the KMeans with the vectors returned by the TF-IDF and we assigned the groups to the *clusters* variable.

```
1 [c for c in clusters][:10]  
[0, 0, 2, 1, 0, 0, 0, 1, 1, 1]
```

Now we can proceed to the visualization of our groups and evaluate their segmentation and / or presence of anomalies.

Dimensional Reduction and Visualization

We have our X from the TF-IDF and we have a KMeans model and related clusters. Now we want to put these two pieces together to visualize the relation between text and group.

As we know, a graph is usually presented in 2 dimensions and rarely in 3. Surely we cannot visualize more. If we look at the dimensionality of X with `X.shape` we see that it is (3451, 7390). There are 3451 vectors (one for each text), each with 7390 dimensions. It is definitely impossible to visualize them.

Fortunately for us, there is a technique called **PCA (Principal Component Analysis)** **which reduces the dimensionality of a data set to an arbitrary number while preserving most of the information contained in it.** As with the TF-IDF, I won't go into detail about how it works, and you can read more about how it works [here](#). For the purpose of this article, it is enough for us to know that PCA tends to preserve the dimensions that best summarize the total variability of our data, by removing dimensions that contribute little to the latter.

Our X will go from 7390 in size to 2. `Sklearn.decomposition.PCA` is what we need.

```
1 x0
array([-0.00152024, -0.03644996, -0.06548033, ...,  0.18883194,
        0.03557314, -0.05786917])

1 x1
array([-0.00430002, -0.03914495,  0.08785332, ...,  0.05718469,
        -0.03828756, -0.10444227])
```

Two two reduced dimensions generated by the PCA algorithm

If we now check the dimensionality of x_0 and x_1 we see that they are respectively (3451,), so one point (x_0, x_1) for each text. This enables us to create a scatter chart.

Visualize the Clustering

Before creating our chart let's better organize our dataframe by creating columns *cluster*, x_0 , x_1

	corpus	cleaned	cluster	x0	x1
0	\nThey tried their best not to show it, believ...	tried best show believe surprised find sprint ...	0	-0.001520	-0.004300
1	\nStankiewicz? I doubt it.\n\nKoufax was one ...	stankiewicz doubt koufax one two jewish hofs h...	0	-0.036450	-0.039145
2	\n[deletia- and so on]\n\nI seem to have been ...	deletia seem rather unclear asking please show...	2	-0.065480	0.087853
3	Excuse the sheer newbiness of this post, but ...	excuse sheer newbiness post looking decent pa...	1	0.164120	0.062958
4	=====		0	0.035573	-0.038288
...
3446	\n Or, with no dictionary available, they cou...	dictionary available could gain first hand kno...	0	-0.010061	-0.010073
3447	\n\nSorry to disappoint you but the Red Wings ...	sorry disappoint red wings earned victory easi...	0	-0.029921	-0.158443
3448	\n: Can anyone tell me where to find a MPEG vi...	anyone tell find mpeg viewer either dos window...	1	0.188832	0.057185
3449	\n		0	0.035573	-0.038288
3450	\nHey Valentine, I don't see Boston with any w...	hey valentine see boston world series rings fi...	0	-0.057869	-0.104442

3451 rows x 5 columns

Dataset ready for visualization after KMeans and PCA application

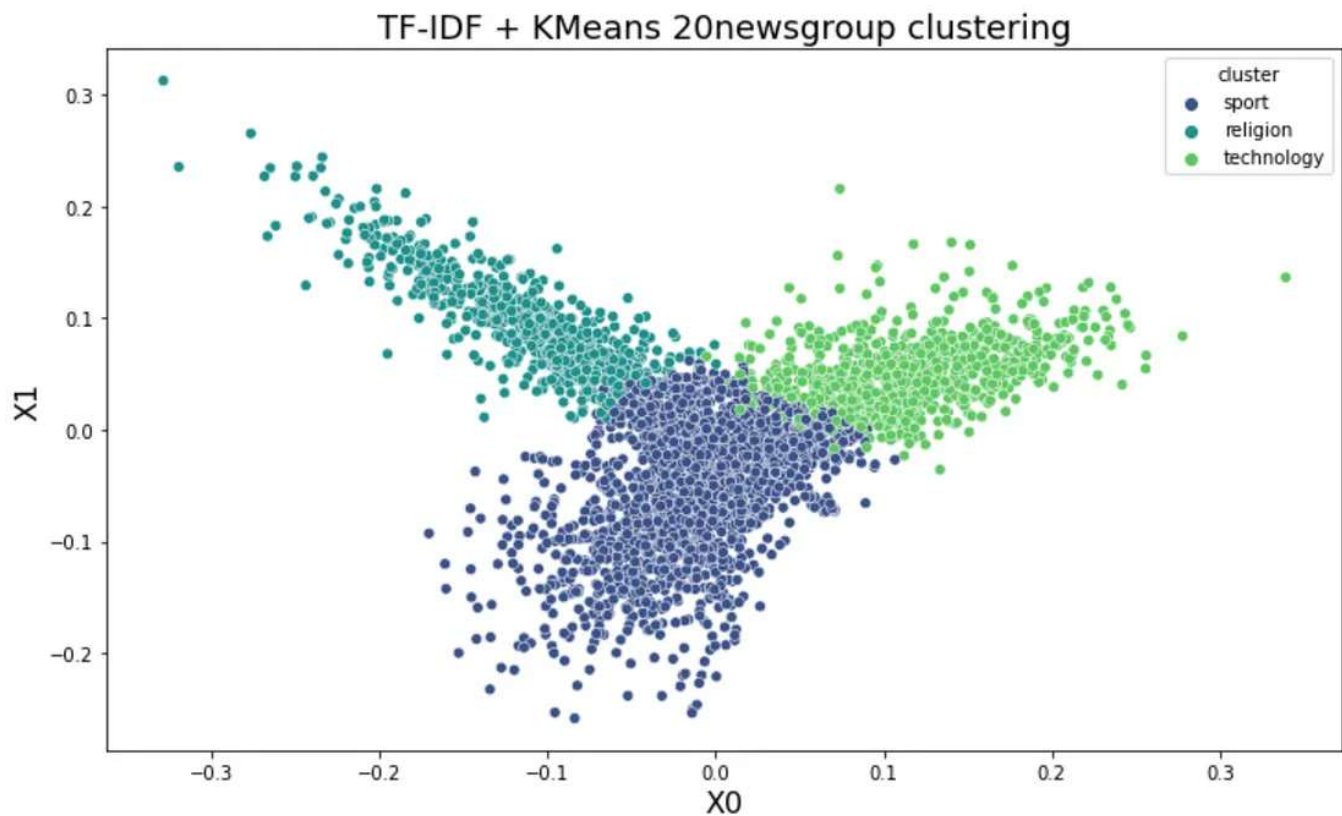
Let's see which are the most relevant keywords for each centroid so that we can rename each cluster with a better label

```
Cluster 0  
good, last, games, like, would, year, think, one, team, game  
  
Cluster 1  
please, dos, use, know, program, anyone, files, file, thanks, windows  
  
Cluster 2  
christians, say, think, bible, believe, jesus, one, would, people, god
```

Keywords per centroid

Good! The KMeans has correctly created 3 distinct groups, one for each category present in the dataset. Cluster 0 refers to sport, cluster 2 to software / tech, cluster 3 to religion. Let's apply the mapping

Let's proceed with the Seaborn library to visualize our grouped texts in a very simple way.



Text data clustering using TF-IDF and KMeans. Each point is a vectorized text belonging to a defined category

As we can see, the clustering activity worked well: the algorithm found three distinct groups; as they can be found in our dataset. Imagine the power of this approach in finding clusters in unlabeled data :)

Interpretation

The interpretation is quite simple: there are no particular anomalies, except for the fact that there are texts belonging to the technology category that overlap slightly with those belonging to sport, between the dark blue and bright green border. This is due to the presence of common terms among some of these texts which, when vectorized, obtain equal values for some dimensions.

As a follow-up it would be interesting to investigate how these texts are written and understand if the hypothesized motivation is well founded or not.

Conclusion

Glad you made it here. Hopefully you'll find this article useful and implement snippets of it in your codebase.

Recommended Reads

For the interested, here are a list of books that I recommended for each ML-related topic. There are ESSENTIAL books in my opinion and have greatly impacted my

professional career.

Disclaimer: these are Amazon affiliate links. I will receive a small commission from Amazon for referring you these items. Your experience won't change and you won't be charged more, but it will help me scale my business and produce even more content around AI.

- **Intro to ML:** [Confident Data Skills: Master the Fundamentals of Working with Data and Supercharge Your Career](#) by Kirill Eremenko
- **Sklearn / TensorFlow:** [Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow](#) by Aurelien Géron
- **NLP:** [Text as Data: A New Framework for Machine Learning and the Social Sciences](#) by Justin Grimmer
- **Sklearn / PyTorch:** [Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python](#) by Sebastian Raschka
- **Data Viz:** [Storytelling with Data: A Data Visualization Guide for Business Professionals](#) by Cole Knaflie

Useful Links (written by me)

- **Learn how to perform a top-tier Exploratory Data Analysis in Python:** [Exploratory Data Analysis in Python — A Step-by-Step Process](#)
- **Learn the basics of TensorFlow:** [Get started with TensorFlow 2.0 — Introduction to deep learning](#)
- **Perform text clustering with TF-IDF in Python:** [Text Clustering with TF-IDF in Python](#)

If you want to support my content creation activity, feel free to follow my referral link below and join Medium's membership program. I will receive a portion of your investment and you'll be able to access Medium's plethora of articles on data science and more in a seamless way.

Join Medium with my referral link - Andrea D'Agostino

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

medium.com

Have a great day. Stay well 🙌

The Code

Here is all the complete code, in copy-paste format

MLearning.ai Submission Suggestions

How to become a writer on Mlearning.ai

medium.com

Data Science

Text Clustering

Unsupervised Text

Machine Learning

NLP



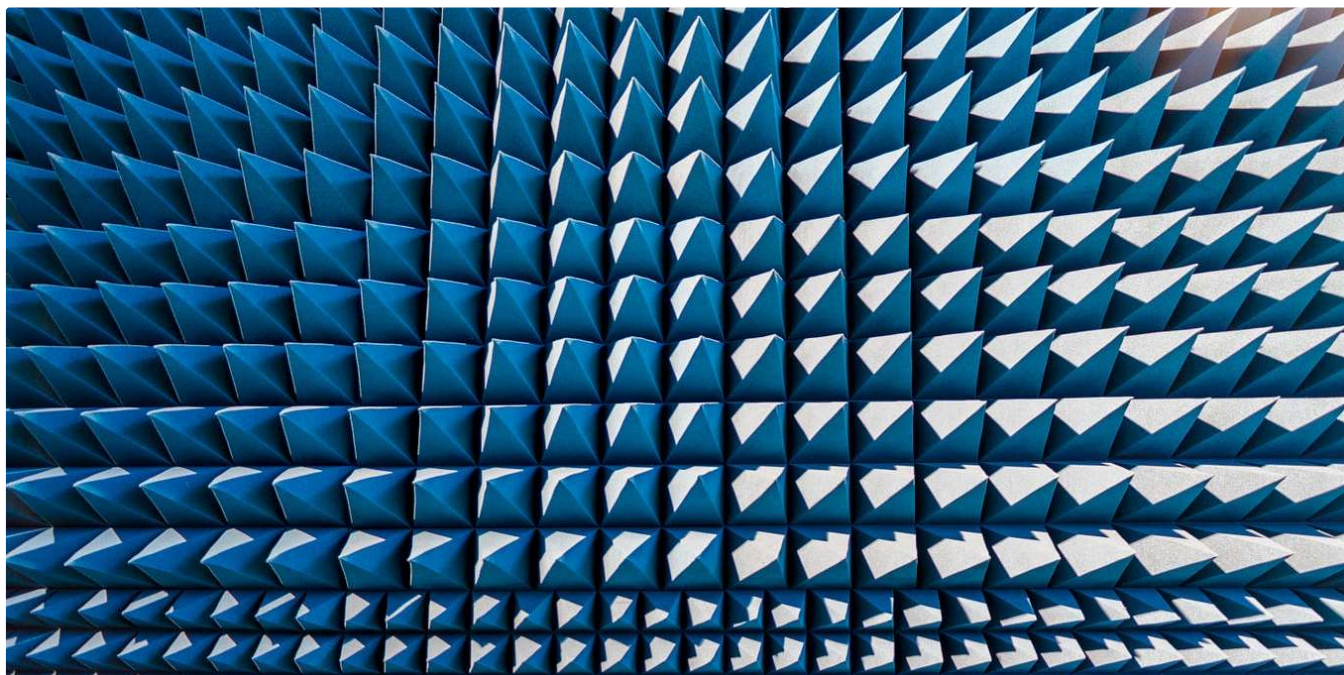
Follow

Written by Andrea D'Agostino

1.1K Followers · Writer for MLearning.ai

Data scientist. I write about data science, machine learning and analytics. I also write about career and productivity tips to help you thrive in the field.

More from Andrea D'Agostino and MLearning.ai



Andrea D'Agostino in Towards Data Science

Introduction to PCA in Python with Sklearn, Pandas, and Matplotlib

Learn the intuition behind PCA in Python and Sklearn by transforming a multidimensional dataset into an arbitrary number of dimensions and...

★ • 13 min read • Sep 7



104



FS Ndzomga in MLearning.ai

LangChain Is The Past, Here Is The Future Of LLM-based Apps

I recently came across EmbedChain, a framework for building chatbots using LLMs that can interact with various types of data.

★ • 4 min read • Aug 29



912



27





Maximilian Vogel in MLearning.ai

The ChatGPT list of lists: A collection of 3000+ prompts, examples, use-cases, tools, APIs...

Updated Oct-18, 2023. New developer resources, marketing & SEO prompts, new prompt engineering courses, masterclasses and tutorials.

11 min read • Feb 8



9.1K



119



Andrea D'Agostino in Towards Data Science

Exploratory Data Analysis in Python—A Step-by-Step Process

What is exploratory analysis, how it is structured and how to apply it in Python with the help of Pandas and other data analysis and...

🌟 · 14 min read · Jul 7, 2022

 554

 2





See all from Andrea D'Agostino

See all from MLearning.ai

Recommended from Medium

3 And this is the third one.

4 Is this the first document?

DF VALUES

and	document	first	is	one	second	the	third	this
1	3	2	4	1	1	4	1	4

IDF VALUES

and	document	first	is	one	second	the	third	this
1.91629073	1.22314355	1.51082562	1	1.91629073	1.91629073	1	1.91629073	1

TF VALUES

	and	document	first	is	one	second	the	third	this
1	0	1	1	1	0	0	1	0	1
2	0	2	0	1	0	1	1	0	1
3	1	0	0	1	1	0	1	1	1
4	0	1	1	1	0	0	1	0	1

TFIDF VALUES

	and	document	first	is	one	second	the	third	this
1	0	0.46979139	0.58028582	0.38408524	0	0	0.38408524	0	0.38408524

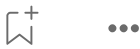



Mohamad Mahmood in Dev Genius

TFIDF Calculation Using SKLearn's TfidfVectorizer

accompanied by the step-by-step manual TFIDF calculation

8 min read · Jul 20



 Heka.ai

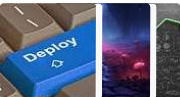
Labeling text clusters with keywords

We propose to explore several keyword extraction techniques to label text clusters obtained after a Text Clustering or a Topic Modeling...

9 min read · Jun 8



Lists



Predictive Modeling w/ Python

20 stories · 502 saves



Practical Guides to Machine Learning

10 stories · 570 saves



Natural Language Processing

721 stories · 319 saves



New_Reading_List

174 stories · 151 saves

```
or([[ -1.8824e-01, -7.6512e-04, 1.0336e-01, ..., -2.0809e-01, -3.9280e-01, 7.9072e-01],  
    [ -4.1441e-01, -1.7607e-01, 5.4728e-02, ..., -1.0659e-01, -3.8406e-01, 7.9451e-01],  
    [ -5.5160e-01, 1.7655e-01, 2.4592e-01, ..., 1.5593e-01, -5.1121e-01, 1.3524e+00],  
    [ -2.6705e-01, 2.0308e-01, -3.6436e-02, ..., -9.6218e-01, -5.8836e-01, 6.6819e-01],  
    [ -1.7557e-01, 1.8462e-01, 3.5970e-02, ..., -1.4965e-01, -3.1363e-01, 6.9866e-01],  
    [ -1.6752e-01, -3.2122e-01, 7.4659e-02, ..., -2.1811e-01, -3.7288e-01, 7.0560e-01]])
```

 Abhijat Sarari in Python in Plain English

How to Generate Word Embedding Using BERT?

Introduction

9 min read · Aug 28

 2 



 Farhan Sarguroh

03: Basic Text Preprocessing (NLP)

Text preprocessing is an essential step in natural language processing (NLP) tasks that involves cleaning and transforming raw text data...

6 min read · Jul 11



Shahar Gino in MLearning.ai

Clustering—Beyonds KMeans+PCA...

Perhaps the most popular way of clustering is K-Means. It's natively supports only numerical data, so typically an encoding is applied...

9 min read · Jul 14

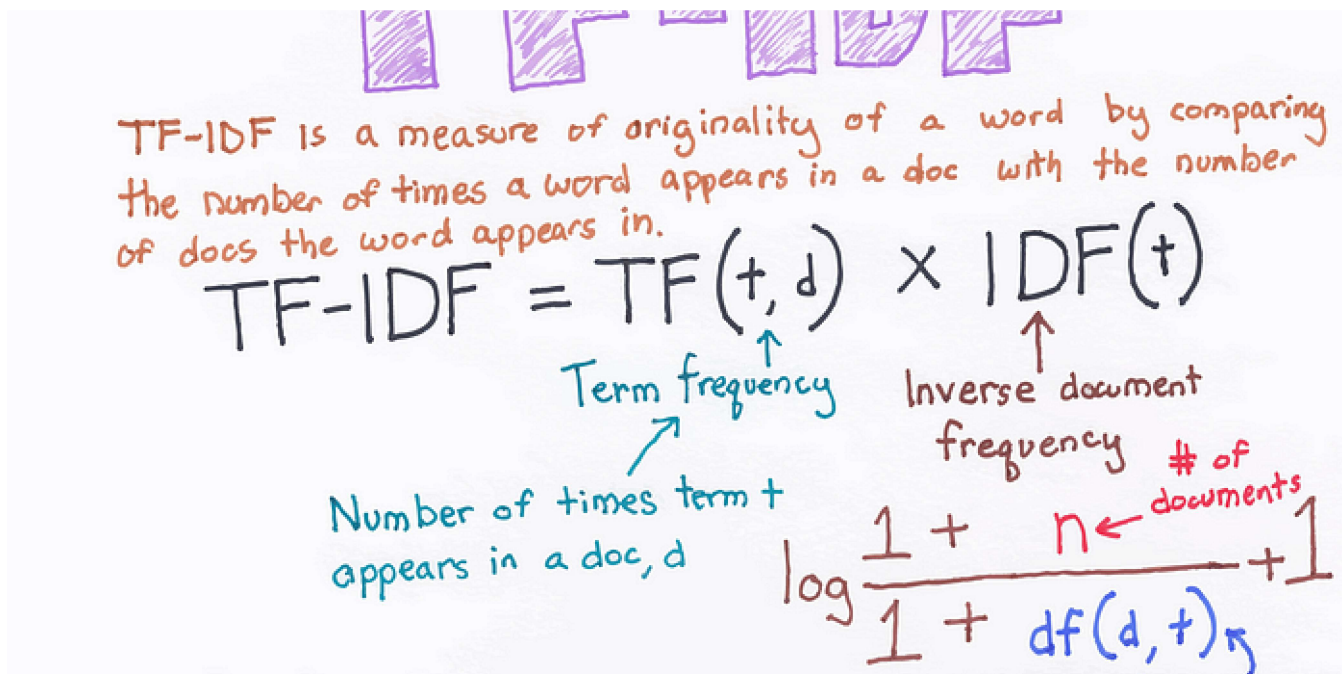


48



1





Rahul S

NLP: TF-IDF (Term Frequency-Inverse Document Frequency)

Convert words into numbers

★ • 3 min read • Jun 8



5



See more recommendations