Open in app Get started

Published in MLearning.ai

Andrea D'Agostino Follow
Nov 24, 2021 · 7 min read · Listen

# Text Clustering with TF-IDF in Python

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

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

## 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.

```
# import the dataset from sklearn
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
```

```
import numpy as np

# string manipulation libs
import re
import string
import nltk
from nltk.corpus import stopwords

# viz libs
import matplotlib.pyplot as plt
import seaborn as sns


categories = [
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'alt.atheism',
 'soc.religion.christian',
]
dataset = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, remove=(
```

clustering_eng_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

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

| | 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 newbieness 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 × 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

```
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.

```
def preprocess_text(text: str, remove_stopwords: bool) -> str:
    """This utility function sanitizes a string by:
    - removing links
    - removing special characters
    - removing numbers
    - removing stopwords
    - transforming in lowercase
    - removing excessive whitespaces
    Args:
        text (str): the input text you want to clean
        remove_stopwords (bool): whether or not to remove stopwords
    Returns:
        str: the cleaned text
    """

    # remove links
    text = re.sub(r"http\S+", "", text)
    # remove special chars and numbers
    text = re.sub("[^A-Za-z]+", " ", text)
    # remove stopwords
    if remove_stopwords:
        # 1. tokenize
        tokens = nltk.word_tokenize(text)
        # 2. check if stopword
        tokens = [w for w in tokens if not w.lower() in stopwords.words("english")]
        # 3. join back together
        text = " ".join(tokens)
    # return text in lower case and stripped of whitespaces
    text = text.lower().strip()
```

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

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

clustering_5.py hosted with ❤ by GitHub view raw

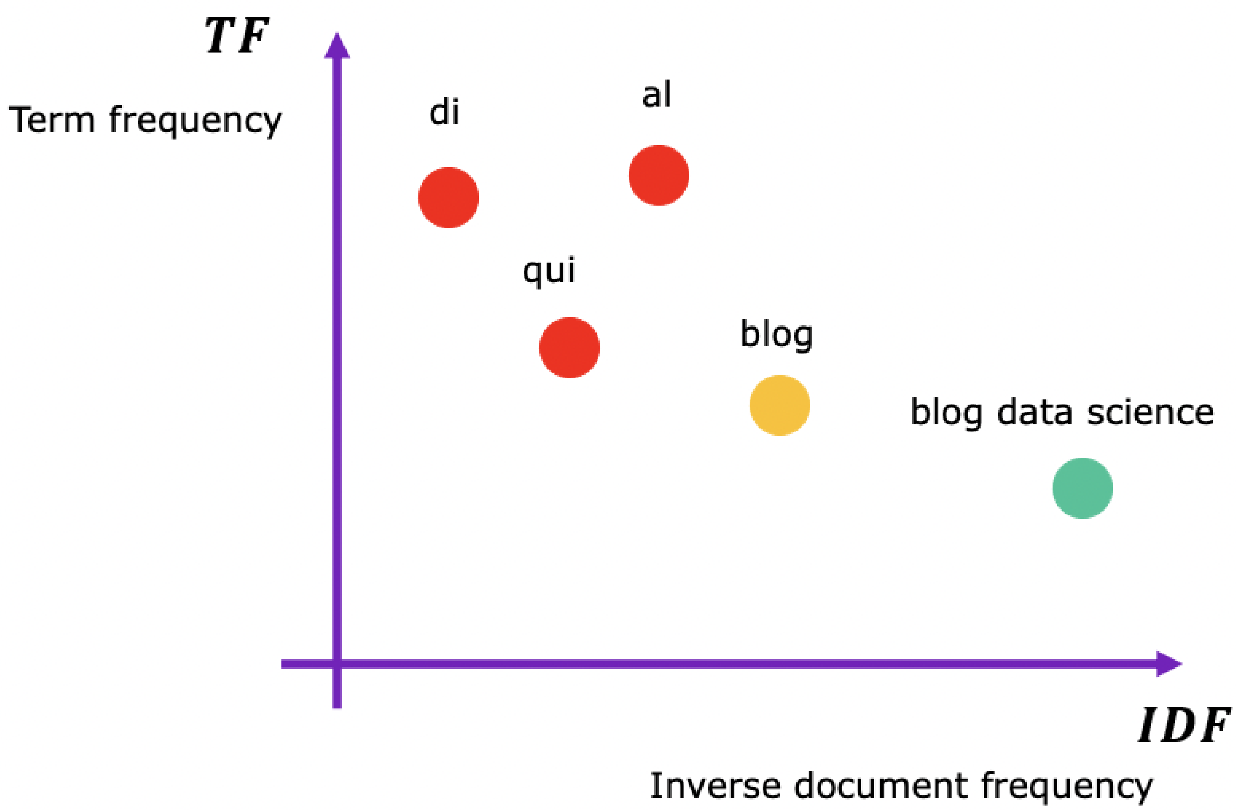
| | corpus | cleaned |
|---|---|---|
| 0 | \nThey tried their best not to show it, believ... | they tried their best not to show it believe m... |
| 1 | \nStankiewicz? I doubt it.\n\nKoufax was one ... | stankiewicz i doubt it koufax was one of two j... |
| 2 | \n[deletia- and so on]\n\nI seem to have been ... | deletia and so on i seem to have been rather u... |
| 3 | Excuse the sheer newbieness of this post, but ... | excuse the sheer newbieness 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-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

```
# initialize the vectorizer
vectorizer = TfidfVectorizer(sublinear_tf=True, min_df=5, max_df=0.95)
# fit_transform applies TF-IDF to clean texts - we save the array of vectors in X
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 *vector_a = [1.204, 0, 0, 0, 0, 0, 0, …, 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.

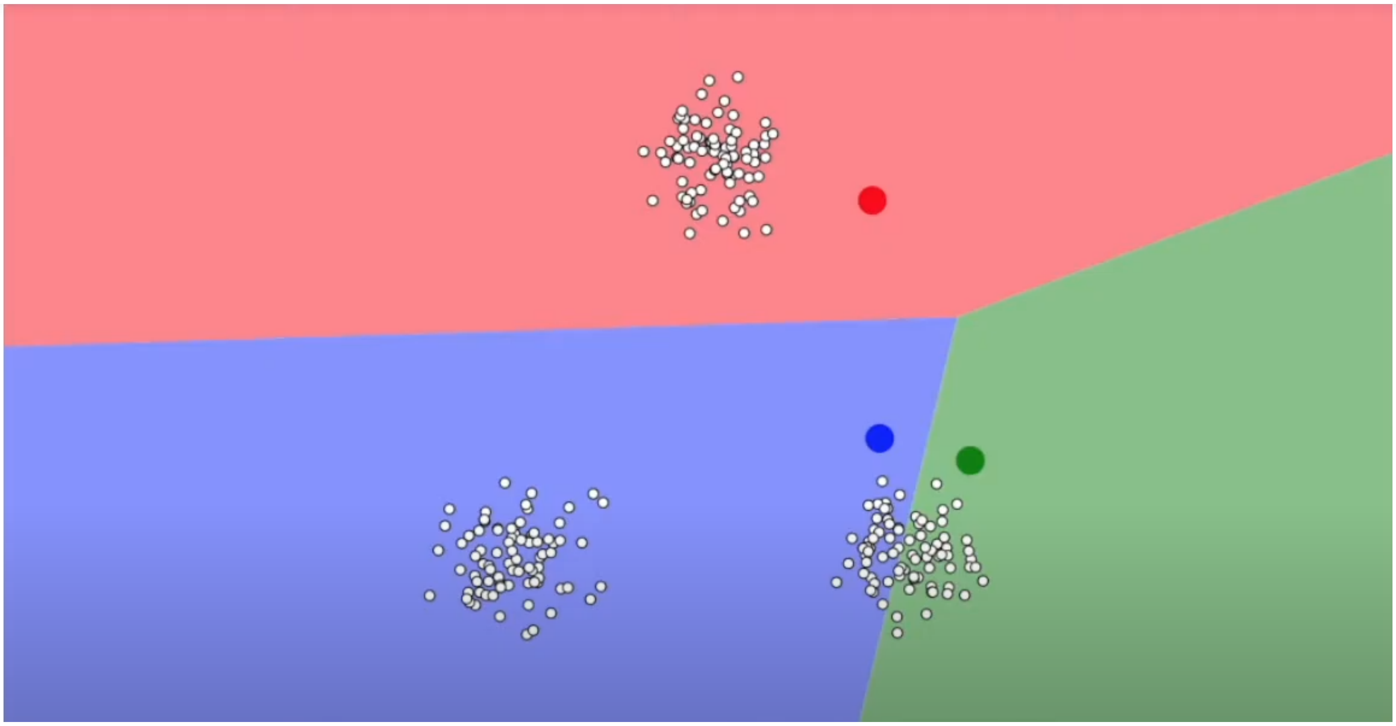
If we use *X.toarray()* we actually see the complete matrix, not sparse.

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

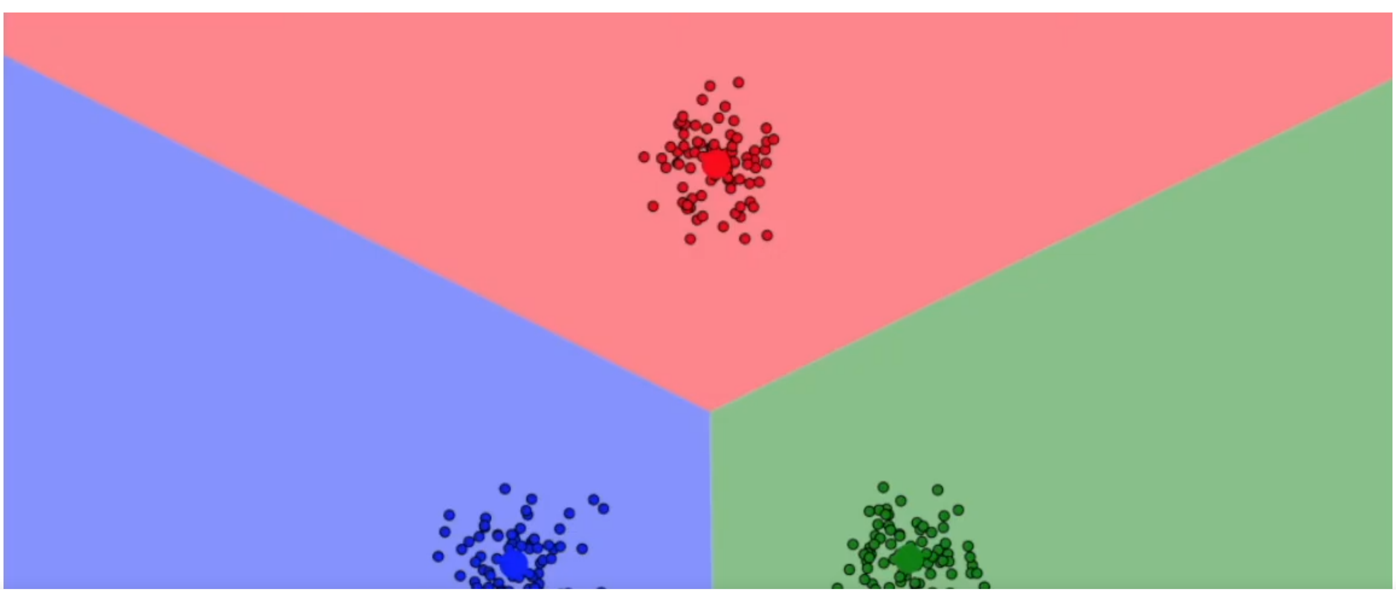
**Implementation of KMeans**

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

```
from sklearn.cluster import KMeans

# initialize kmeans with 3 centroids
kmeans = KMeans(n_clusters=3, random_state=42)
# fit the model
kmeans.fit(X)
# store cluster labels in a variable
clusters = kmeans.labels_
```

clustering_eng_kmeans.py hosted with ❤ by GitHub view raw

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.

```
[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

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.

```
from sklearn.decomposition import PCA

# initialize PCA with 2 components
pca = PCA(n_components=2, random_state=42)
# pass our X to the pca and store the reduced vectors into pca_vecs
pca_vecs = pca.fit_transform(X.toarray())
# save our two dimensions into x0 and x1
x0 = pca_vecs[:, 0]
x1 = pca_vecs[:, 1]
```

clustering_eng_pca.py hosted with ❤ by GitHub view raw

```
x0
```

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

```
x1
```

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

Two two reduced dimensions generated by the PCA algorithm

Before creating our chart let's better organize our dataframe by creating columns *cluster, x0, x1*

```
# assign clusters and pca vectors to our dataframe
df['cluster'] = clusters
df['x0'] = x0
df['x1'] = x1
```

clustering_eng_pca.py hosted with ❤ by GitHub view raw

| | 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 newbieness of this post, but ... | excuse sheer newbieness 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 × 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

```
def get_top_keywords(n_terms):
    """This function returns the keywords for each centroid of the KMeans"""
    df = pd.DataFrame(X.todense()).groupby(clusters).mean() # groups the TF-IDF vector by c
    terms = vectorizer.get_feature_names_out() # access tf-idf terms
    for i,r in df.iterrows():
        print('\nCluster {}'.format(i))
        print(','.join([terms[t] for t in np.argsort(r)[-n_terms:]])) # for each row of the

get_top_keywords(10)
```

Open in app Get started

```
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

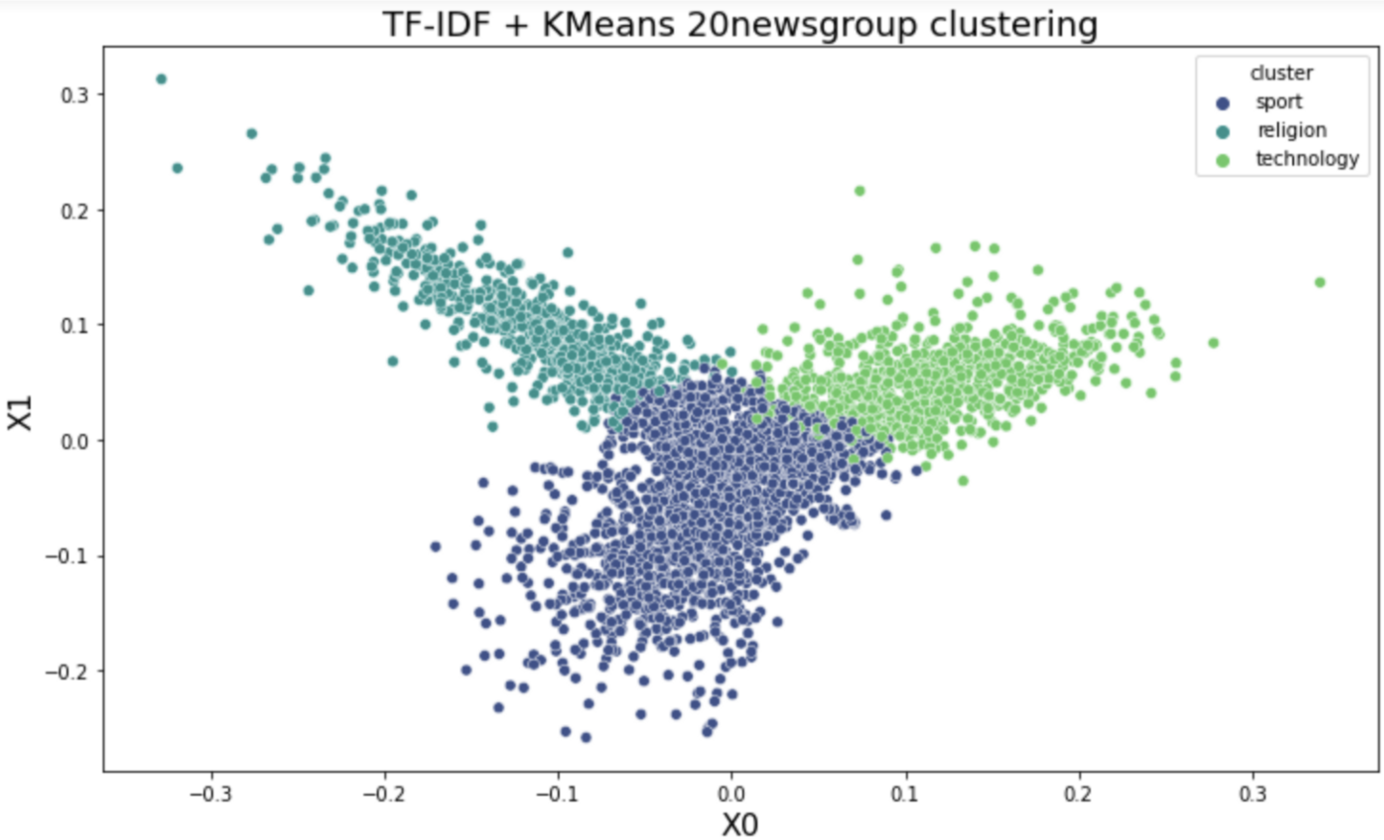
```
# map clusters to appropriate labels
cluster_map = {0: "sport", 1: "tech", 2: "religion"}
# apply mapping
df['cluster'] = df['cluster'].map(cluster_map)
```

clustering_eng_mapping.py hosted with ❤ by GitHub view raw

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

```
# set image size
plt.figure(figsize=(12, 7))
# set a title
plt.title("TF-IDF + KMeans 20newsgroup clustering", fontdict={"fontsize": 18})
# set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})
# create scatter plot with seaborn, where hue is the class used to group the data
sns.scatterplot(data=df, x='x0', y='x1', hue='cluster', palette="viridis")
plt.show()
```

clustering_eng_viz.py hosted with ❤ by GitHub view raw




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.

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# import other required libs
import pandas as pd
import numpy as np

# string manipulation libs
import re
import string
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords

# viz libs
import matplotlib.pyplot as plt
import seaborn as sns

categories = [
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'alt.atheism',
 'soc.religion.christian',
]
dataset = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, remove=(

df = pd.DataFrame(dataset.data, columns=["corpus"])
df['cleaned'] = df['corpus'].apply(lambda x: preprocess_text(x, remove_stopwords=True))

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

# initialize KMeans with 3 clusters
```

```python
pca = PCA(n_components=2, random_state=42)
# pass X to the pca
pca_vecs = pca.fit_transform(X.toarray())
# save the two dimensions in x0 and x1
x0 = pca_vecs[:, 0]
x1 = pca_vecs[:, 1]

# assign clusters and PCA vectors to columns in the original dataframe
df['cluster'] = clusters
df['x0'] = x0
df['x1'] = x1

cluster_map = {0: "sport", 1: "technology", 2: "religion"} # mapping found through get_top
df['cluster'] = df['cluster'].map(cluster_map)

# set image size
plt.figure(figsize=(12, 7))
# set title
plt.title("Raggruppamento TF-IDF + KMeans 20newsgroup", fontdict={"fontsize": 18})
# set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})
#  create scatter plot with seaborn, where hue is the class used to group the data
sns.scatterplot(data=df, x='x0', y='x1', hue='cluster', palette="viridis")
plt.show()
```

**clustering_eng_full.py** hosted with ❤️ by **GitHub** view raw

```python
def preprocess_text(text: str, remove_stopwords: bool) -> str:
    """This function cleans the input text by
    - removing links
    - removing special chars
    - removing numbers
    - removing stopwords
    - transforming in lower case
    - removing excessive whitespaces
    Arguments:
        text (str): text to clean
        remove_stopwords (bool): remove stopwords or not
    Returns:
```

```
    text = re.sub("[^A-Za-z]+", " ", text)
    # remove stopwords
    if remove_stopwords:
        # 1. creates tokens
        tokens = nltk.word_tokenize(text)
        # 2. checks if token is a stopword and removes it
        tokens = [w for w in tokens if not w.lower() in stopwords.words("english")]
        # 3. joins all tokens again
        text = " ".join(tokens)
    # returns cleaned text
    text = text.lower().strip()
    return text

def get_top_keywords(n_terms):
    """This function returns the keywords for each centroid of the KMeans"""
    df = pd.DataFrame(X.todense()).groupby(clusters).mean() # groups tf idf vector per clu
    terms = vectorizer.get_feature_names_out() # access to tf idf terms
    for i,r in df.iterrows():
        print('\nCluster {}'.format(i))
        print(','.join([terms[t] for t in np.argsort(r)[-n_terms:]])) # for each row of th

```

clustering_eng_functions.py hosted with ❤ by GitHub view raw

**Mlearning.ai Submission Suggestions**

How to become a writer on Mlearning.ai

medium.com