Published in Towards Data Science

TU Follow

Feb 1, 2021 · 11 min read · Listen

How to cluster similar sentences using TF-IDF and Graph partitioning in Python

What data science articles gain more attraction from the readers (Part 2)

Photo by Hanson Lu on Unsplash

In this series of articles we are analysing historical archives of data science publications to understand what topics are more popular with the readers. Previously we covered how to get the data that will be used for further analysis.

We will cover how to clean text data we collected earlier , group similar topics using network graphs and establish patterns within these clusters in this article.

Data summary

Let's remind ourselves how the data looks like. It is combination of articles obtained from three data sources [field: 'Source'] — Analytics Vidhya ['avd'], TDS ['tds'] and Towards AI ['tai'].

We collected titles, subtitles, claps and responses from individual articles in archives of the publications.

```
import pandas as pd

# Reading the data obtained using code here.
avd = pd.read_csv('analytics_vidhya_data.csv')
tds = pd.read_csv('medium_articles.csv')
tai = pd.read_csv('towards_ai_data.csv')

avd['source'] = 'avd'
tds['source'] = 'tds'
tai['source'] = 'tai'

# Create single data set, join title and subtitle
single_matrix = pd.concat([avd, tds, tai])
single_matrix['title_subtitle'] = [' '.join([str(i),str(j)]) for i, j
in zip(single_matrix['Title'].fillna(''),
single_matrix['Subtitle'].fillna(''))]
```

	Title	Subtitle	Claps	Responses	source	title_subtitle
0	Reporting In Qlikview | Ad Hoc Reporting	NaN	211	0	avd	Reporting In Qlikview | Ad Hoc Reporting
1	10 Ultimate Tips and Tricks on Data Visualizat...	NaN	75	0	avd	10 Ultimate Tips and Tricks on Data Visualizat...
2	Use of variables in QlikView to create powerfu...	NaN	90	0	avd	Use of variables in QlikView to create powerfu...
3	A Comprehensive Guide to Data Exploration	NaN	204	0	avd	A Comprehensive Guide to Data Exploration
4	An Introduction to Implementing Neural Network...	NaN	47	0	avd	An Introduction to Implementing Neural Network...
5	Bayesian Statistics explained to Beginners in ...	NaN	4	0	avd	Bayesian Statistics explained to Beginners in ...
6	CNN Architectures: LeNet, AlexNet, VGG, GoogLe...	NaN	4000	18	avd	CNN Architectures: LeNet, AlexNet, VGG, GoogLe...

Articles data set

We added an additional column in the data set called ‘title_subtitle’ which is the join of columns *‘Title’* and *‘Subtitle’*, we will mainly use this column in order to have a better view of the topic the article belongs to. Quite interestingly *39%* of articles don’t have subtitles and a very small proportion *(0.13%)* don’t have titles.

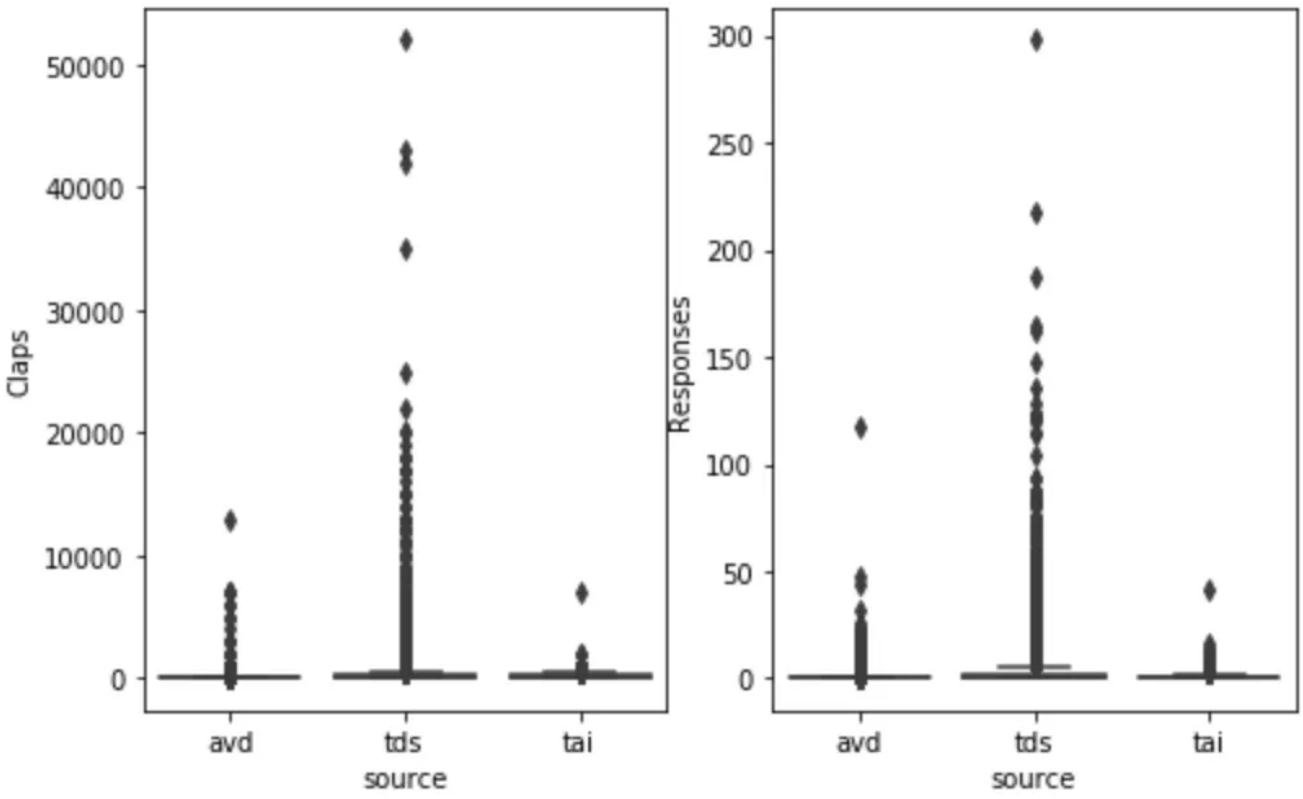
Let’s quickly look at the claps and responses distributions for every data source. We start with box plots, we use *seaborn* library in Python to create our plots.

```
# We will use seaborn to create all plots
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(8, 5))
# Claps
```

```
sns.boxplot(ax=axes[0], x="source", y="Claps", data=single_matrix)

# Responses
sns.boxplot(ax=axes[1], x="source", y="Responses", data=single_matrix)
```

We can see that Towards Data Science has not only more activity, but also quite a few outliers with individual articles gaining a lot of attraction from readers. Of course, the activity for each source depends on the size of publication, for larger publications we observe more writers and readers.

When it comes to responses, we observe far less activity in comparison to claps across all sources, although such behaviour is not very unexpected.



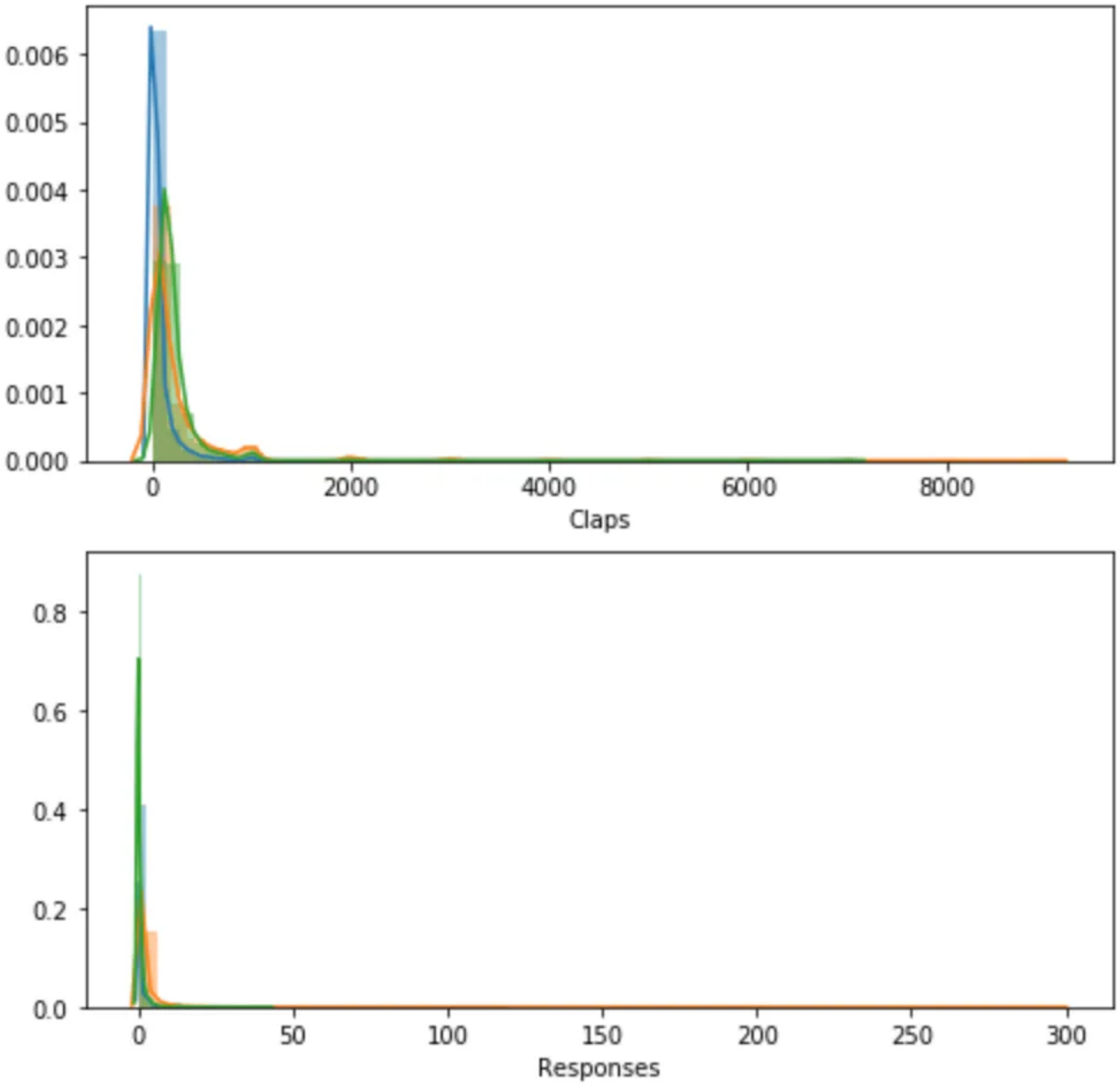
Box plots for claps and responses split by source

Next, we remove outliers and visualise distributions of the fields to have a clearer picture.

```
# Code to create distribution subplots
fig, axes = plt.subplots(2, 1, figsize=(8, 8))
```

```
# Claps
sns.distplot(avd['Claps'][avd['Claps']<10000], hist=True, rug=False,
ax=axes[0])
sns.distplot(tds['Claps'][tds['Claps']<10000], hist=True, rug=False,
ax=axes[0])
sns.distplot(tai['Claps'][tai['Claps']<10000], hist=True, rug=False,
ax=axes[0])

# Responses
sns.distplot(avd['Responses'], hist=True, rug=False, ax=axes[1])
sns.distplot(tds['Responses'], hist=True, rug=False, ax=axes[1])
sns.distplot(tai['Responses'], hist=True, rug=False, ax=axes[1])
```

We can see that both distributions are skewed to the left, meaning that the most articles get very little claps and even less responses. Yet again this is not surprising, since the success of articles depends on many factors, such as good quality writing, relevant topics and many more. Striking a good balance is not a simple task!



Claps and responses distributions split by sources

How to clean text data?

Cleaning data is an important step (if not the most important part) when working with text. There are standard practices in place that one follows when dealing with such tasks. We will undertake following steps to process titles and subtitles:

- remove punctuation marks and other symbols
- remove stop words and digits
- lemmatise words

We will use a mixture of regular expressions and *nltk* library to ***remove punctuation marks, symbols, stop words and digits.***

```
import re
single_matrix['title_subtitle'] = [re.findall(r'\w+', i.lower()) for i
in single_matrix['title_subtitle'].fillna('NONE')]
```

The code above matches one or more word characters, in fact *r'\w+'* is the same as *r'[a-zA-Z0–9_]+'.* Also, when applying *re.findall()* and *i.lower()* commands, they conveniently split sentences into words and transform them into lower case. This will come very useful in next steps. So, the sentence *'Reporting In Qlikview | Ad Hoc Reporting'* becomes *[reporting, in, qlikview, ad, hoc, reporting].*

Next, we will use *nltk* library to upload a dictionary of stop words so we can remove them from the sentences. Additionally we append words 'use' and 'part ' to the list, since they are overused in the data set. To remove the stop words we use for loop to iterate over each sentence, when doing so we also ensure to remove the digits from the sentences.

```
# The code to upload list of stop words and remove them from sentences

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

stopwords_eng = stopwords.words('english')
stopwords_eng += ['use', 'using', 'used', 'part']

new_titles_sub = []
for title_sub in single_matrix['new_title_subtitle']:
    new_title_sub = []
    for w_title in title_sub:
        if w_title not in stopwords_eng and not w_title.isdigit():
            new_title_sub.append(w_title)

    new_titles_sub.append(new_title_sub)

single_matrix['new_title_subtitle'] = new_titles_sub
```

Finally, we are going to lemmatise words in the sentences. Lemmatisation transforms the word to its meaningful root form taking into consideration the context. Frequently stemming is used as a computationally faster alternative, however less accurate one. Once again we use *nltk* to lemmatise words

```
nltk.download('wordnet')
nltk.download('words')
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

new_titles_sub = []
for title_sub in single_matrix['title_subtitle']:
    new_title_sub = []
    for w_title in title_sub:
        new_title_sub.append(wordnet_lemmatizer.lemmatize(w_title,
pos="v"))
    new_titles_sub.append(new_title_sub)

single_matrix['new_title_subtitle'] = new_titles_sub
single_matrix['new_title_subtitle'] = [' '.join(i) for i in
single_matrix['new_title_subtitle']]
```

Let's look how the sentences look like after all the transformations

title_subtitle	new_title_subtitle
Reporting In Qlikview | Ad Hoc Reporting	report qlikview ad hoc report
10 Ultimate Tips and Tricks on Data Visualizat...	ultimate tip trick data visualization qlikview
Use of variables in QlikView to create powerfu...	variables qlikview create powerful data stories
A Comprehensive Guide to Data Exploration	comprehensive guide data exploration

How to vectorise text data using TF-IDF?

TF-IDF stands for term frequency-inverse document frequency and it is *a numerical measure of how relevant a keyword is to a document in some specific set of documents.* It is commonly used in text analysis, some of the examples include content ranking and information retrieval. Here is quite a useful paper that talks about the approach in more detail.

As a name suggests the measure consists of two parts, one that finds frequency of a word appearing in a document (TF) and another the extent of word uniqueness in a corpus (IDF). Let's look at the simplified version the formula and its components:

$$tf = (number\ of\ times\ the\ keyword\ appears\ in\ a\ document)/(total\ number\ of\ words\ in\ a\ document)$$
$$idf = log((total\ number\ of\ documents\ in\ a\ set)/(number\ of\ documents\ containing\ the\ keyword))$$
$$tf - idf = tf * idf$$

We can see that words appearing more frequently will result in a lower TF-IDF score and for rare words the score will be higher. This weight adjustment is quite important, since overused words will have no additional meaning.

The easiest way to understand the calculations is by example, in our data set a single title is a document and all the titles form a corpus (set of documents). Consider the word *'create'* in the title *'use variables qlikview create powerful data stories',* the document has 7 words and *'create'* appears only once, so *TF(create) = 1/7*. The total number of articles in one of the data sources is *12963* and word *'create'* appears in *268* titles so *IDF(create)=log(12963/268) =3.88*. Thus, *TF-IDF =0.14*3.88 = 0.55* is the score for the word *'create'*.

Now that we know how the scores are calculated for each word in a document, we can vectorise the data set with articles titles and subtitles. For this we will use *sklearn* library in Python, in particular *TfidfVectorize*r function.

Note: *TfidfVectorize*r uses a slightly different formula than the one specified above, it adds 1 to IDF. This is done to ensure that the words that appear in every document are not neglected.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df=0)

tfidf_matrices = []
data_sets = []
for source in ['avd', 'tai', 'tds']:
    source_data = single_matrix[single_matrix['source'] ==
source].drop_duplicates()
    data_sets.append(source_data['new_title_subtitle'])
    tfidf_matrices.append(tf.fit_transform(
source_data['new_title_subtitle']))
```

We introduced the for loop that iterates through the data sources, this is done to improve computational time. This will also be needed when looking at the distances between all pairs of sentences and partitioning them into groups. The output is the sparse matrix, where rows are documents and columns all unique words in a corpus.

Now that we have vectorised titles and subtitles, we can **calculate pairwise distances between all the sentences.** We will use **cosine similarity** between pairs of vectors that represent sentences. The measure considers the angle between two vectors and commonly used in text analysis. Some good explanations of the chosen similarity measure can be found here, the paper not only provides clear definition it also discusses context based uses.

We used *sklearn* library to calculate pairwise cosine similarities, yet again splitting by the source.

```
from sklearn.metrics.pairwise import linear_kernel
matrix_with_cos_sim = []
for m in tfidf_matrices:
    matrix_with_cos_sim.append(linear_kernel(m, m))
```

The output for each data source is a numpy array (NxN) with pairwise similarities between all sentences where N is number of titles/subtitles for a single data source.

How to group similar sentences using network graphs?

Our aim is to find clusters that have articles covering similar data science topics, to achieve this we will start by building a weighted graph where nodes are articles and edges are their cosine similarity. We are then able to create clusters by applying graph partitioning with the goal to find meaningful subgraphs (also called communities) without imposing a number of communities.

We will use Python libraries *networkx* and *community* to build and partition the graph. Before we proceed with building the graph, we will only select top 15 similar titles for every document in a corpus, the number was chosen based on the measure called modularity which gives indications how good partition is. This approach not only sharpens the graph but also helps with computational speed.

```
import numpy as np
from tqdm import tnrange
top_n_sentences = []
for cs, t in zip(matrix_with_cos_sim, data_sets):
    no_dups = np.array(t)
    i = 0
    top_frame = []
    for c, z in zip(cs, tnrange(len(cs))):
        # Create vector of titles
        start_name = pd.Series([no_dups[i]]*15)
        # Index of top 15 similar titles
        ix_top_n = np.argsort(-c)[0:15]
        cos_sim = pd.Series(c[ix_top_n])
        names = pd.Series(no_dups[ix_top_n])
        i +=1
        top_frame.append(pd.DataFrame([start_name, names,
cos_sim]).transpose())

    top_frame = pd.concat(top_frame)
    top_frame.columns = ['title1', 'title2', 'cos_sim']
    # Remove the similarities for the same sentences
    top_frame['is_same'] = [bool(i==j) for i, j in
zip(top_frame['title1'], top_frame['title2'])]
    top_frame = top_frame[top_frame['is_same'] != True]

    top_n_sentences.append(top_frame)
```

The script produces the data frame with top 15 similar titles for every title in the data set (split by source), it will be used as an input to building the graph. Let's look at the example of the data frame for one of the sources

	title1	title2	cos_sim	is_same
1	report qlikview ad hoc report	learn stop worry trust script automatically cr...	0.165474	False
2	report qlikview ad hoc report	sales report pandas	0.141703	False
3	report qlikview ad hoc report	churn report pandas	0.139912	False
4	report qlikview ad hoc report	classification report brief discussion classif...	0.128978	False
5	report qlikview ad hoc report	automate report python schedule create send pe...	0.123157	False
6	report qlikview ad hoc report	report dashboard python dash	0.117644	False
7	report qlikview ad hoc report	build enterprise report app desktop let discus...	0.112799	False
8	report qlikview ad hoc report	birt generate report csv	0.105572	False
9	report qlikview ad hoc report	medical report generation deep learn	0.104479	False
10	report qlikview ad hoc report	build schedule report whatsapp python	0.0959972	False
11	report qlikview ad hoc report	get google analytics report data pandas datafr...	0.0923236	False
12	report qlikview ad hoc report	cool hack azure ad	0.0860952	False
13	report qlikview ad hoc report	orchestrate dynamic report python r rmd file	0.0821002	False
14	report qlikview ad hoc report	automate stock portfolio daily analysis report...	0.078943	False

We will continue by building and partitioning the graph, we will do it for the source that has the second largest number of articles which is Analytics Vidhya. The snippets of code can be applied to all the sources covered in this article.

```
# We start by defining the structure of the graph

top_frame = top_n_sentences[2] #TDS articles

edges = list(zip(top_frame['title1'], top_frame['title2']))
weighted_edges = list(zip(top_frame['title1'], top_frame['title2'],
top_frame['cos_sim']))
nodes = list(set(top_frame['title1']).union(set(top_frame['title2'])))
```

We now can use *networkx* to build the graph using structure defined above

```
import networkx as nx
G = nx.Graph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)
G.add_weighted_edges_from(weighted_edges)
```

Next we partition the graph using *community* library, before module imports ensure to install *python-louvain* library to avoid errors.

```
# !pip install python-louvain

import community
partition = community.best_partition(G)
modularity = community.modularity(partition, G)
```

Earlier we mentioned modularity, the measure how good the partition is, the value in this case is *0.7*. Usually, values above *0.6* considered to be decent enough partitioning.

```
# Takes some time for larger graphs
import matplotlib.pyplot as plt
pos = nx.spring_layout(G, dim=2)
community_id = [partition[node] for node in G.nodes()]
fig = plt.figure(figsize=(10,10))
nx.draw(G, pos, edge_color = ['silver']*len(G.edges()),
cmap=plt.cm.tab20,
        node_color=community_id, node_size=150)
```

The code above produces the graph and communities we just found, although the plot looks quite busy we are still able to see quite a few clusters found by the approach.

Titles and subtitles partitioned into communities

Before we look into clusters in more detail, we are going to transform *partition* variables we created earlier to a more readable format.

```
title, cluster = [], []
for i in partition.items():
    title.append(i[0])
    cluster.append(i[1])

frame_clust = pd.DataFrame([pd.Series(title),
pd.Series(cluster)]).transpose()
frame_clust.columns = ['Title', 'Cluster']
```

The output of the code above is the data frame with all the titles and subtitles and the community they belong to with 45 clusters identified by partitioning the graph.

	Title	Cluster
0	accuracy precision recall much accuracy prefer...	0
1	portfolio analysis indian stock minutes	1
2	cost function logistic regression logistic reg...	2
3	segment mix type data case study k medoids sub...	3
4	different optimization algorithm deep neural n...	4

Titles and subtitles partitioned into communities

Now that we have obtained clusters, we can create summary statistics for each of them to understand if any of them have more activity. We will merge the data set that has partitions with the data set that has claps and responses then we will calculate min, max, mean, median and number of articles for each group. Although mainly will focus on median, since we saw earlier that the data is skewed towards smaller values and has outliers present.

```
avd = single_matrix[single_matrix['source'] ==
'avd'].drop_duplicates()

frame_clust = frame_clust.merge(tds[['Title', 'new_title_subtitle',
'Claps', 'Responses']], how='left', left_on='Title',
right_on='new_title_subtitle')

grouped_mat = frame_clust.groupby('Cluster').agg(
{'Claps': ['max', 'mean', 'sum', 'median'],
 'Responses': ['max', 'mean', 'sum', 'median'],
 'Title_x': 'count'}).reset_index()

grouped_mat.columns = ['cluster', 'claps_max', 'claps_mean',
'claps_sum', 'claps_median','responses_max', 'responses_mean',
'responses_sum', 'responses_median', 'title_count']
```

```
grouped_mat = grouped_mat.sort_values(by = ['claps_median',
'title_count'])
```

For representation purposes will only look at three communities with lowest reader activity and three with highest. We first consider the data set and then we are going to visualise word cloud to determine the common topic in each group.

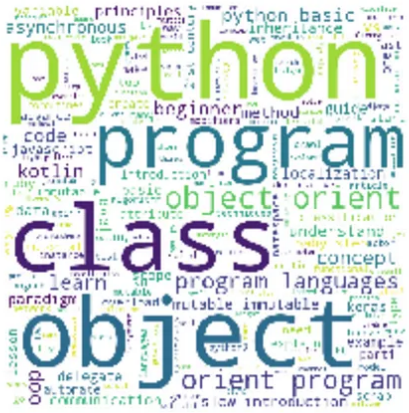
	cluster	claps_max	claps_mean	claps_sum	claps_median	responses_max	responses_mean	responses_sum	responses_median	title_count
19	19	72	9.880000	247	4.0	11	0.440000	11	0	25
39	39	609	53.603053	7022	6.0	2	0.114504	15	0	131
38	38	327	42.095238	3536	7.0	2	0.345238	29	0	84
43	43	1000	79.081481	10676	24.0	8	0.488889	66	0	135
28	28	721	87.681159	6050	32.0	6	0.710145	49	0	69
7	7	548	82.294118	4197	43.0	2	0.254902	13	0	51

Communities wit lowest and highest reader activity

The table above shows that there groups are not very large, let's see what are the common themes in each cluster we will use wordcloud library for this.

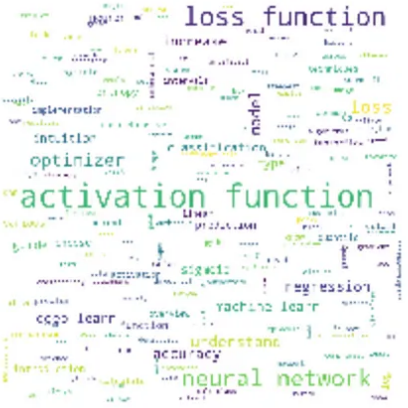
```
from wordcloud import WordCloud
fig, ax = plt.subplots(1, 3, figsize=(12.5,6.5))

clusters = [19, 39, 38] #lowest activity groups
# clusters = [43, 28, 7] #highest activity groups

for cluster, col in zip(clusters, [0, 1, 2]):
    corpus = ' '.join(frame_clust['new_title_subtitle'].
[frame_clust['Cluster'] == cluster])
    ax[col].imshow(WordCloud(width = 800,
                             height = 800,
                             background_color ='white',
                             min_font_size = 10).generate(corpus))
    ax[col].axis("off")

plt.show()
```

We first look at the communities with lowest activity, it seems that cluster 19 has mostly articles that belong to one author which would explain lower activity. The other two clusters consist of more articles that were written by multiple authors. Quite interestingly we can observe that topics such as 'object oriented programming in python' and 'fraud detection' attracted least interest from the readers.



Clusters with lower readers activity

Moving on to the clusters with highest activity the highlighted topics that cause more interest from the readers are natural language processing, neural networks, activation functions and support vector machines.



Clusters with higher reader activity

Wrap up

Although we were able to establish common themes in low and high readers activity groups, we still observed articles that didn't have many claps and responses as well as ones that had high activity in each group. The analysis can come handy when trying to establish general patterns of what readers are interested in, as well as the topics that have higher saturation of articles. However choosing a relevant topic doesn't guarantee success of the article, as many other important factors contribute towards gaining attraction from the reader.

NLP Python Network Machine Learning Clustering