cord19-analysis

April 4, 2020

[1]:
```
#!python3 -m spacy download en
import pysolr
import requests
import json
import pandas as pd
import os

base_directory = os.getcwd()
print(base_directory)

pd.set_option('display.max_columns', 50)
df = pd.read_csv('https://ai2-semanticscholar-cord-19.s3-us-west-2.amazonaws.
 ↪com/2020-03-27/metadata.
 ↪csv',usecols=['title','doi','abstract','publish_time'])
df.describe()
```

```
/Users/cbadenes/Downloads
```

[1]:

	title	doi	abstract	publish_time
count	45617	42440	37913	45765
unique	44994	42439	37531	6368
top	Infectious disease surveillance update	10.1097/jcma.0000000000000270	Unknown	2020
freq	24	2	337	637

[2]:
```
df = df.dropna()
df.describe()
```

[2]:

	title	doi	abstract	publish_time
count	35177	35177	35177	35177
unique	34987	35176	34957	5596
top	Abkürzungen	10.1097/jcma.0000000000000270	Unknown	2020
freq	5	2	188	246

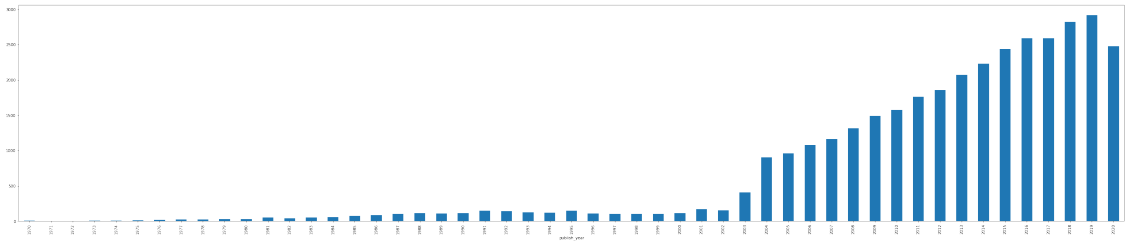
[3]:
```
import numpy as np

def get_creation_year(datetime):
  datetime_values = datetime.split("-")
  year = datetime_values[0]
  return year

df['publish_year'] = np.vectorize(get_creation_year)(df['publish_time'])

df.describe()
```

[3]:

	title	doi	abstract	publish_time	\
count	35177	35177	35177	35177	
unique	34987	35176	34957	5596	
top	Abkürzungen	10.1097/jcma.0000000000000270	Unknown	2020	
freq	5	2	188	246	

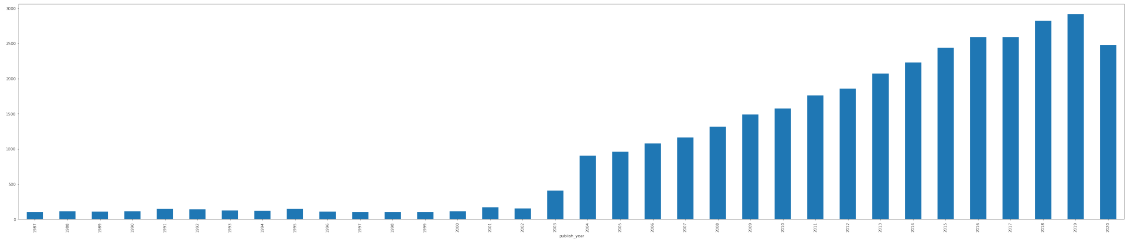
	publish_year
count	35177
unique	51
top	2019
freq	2919

[4]:
```
def show_articles_per_year(dataframe):
  per_year_df = dataframe.groupby('publish_year')
  size = per_year_df.size()
  size.plot(kind='bar', figsize=(50,10))

show_articles_per_year(df)
```



[5]:
```
per_year_df = df.groupby(['publish_year']).size().reset_index(name='counts')

# years with less than 100 articles are removed
valid_year_df = per_year_df[per_year_df['counts'] > 100]

df = df[df['publish_year'].isin(valid_year_df['publish_year'])]
```

```
show_articles_per_year(df)
```



[6]:
```
import numpy as np
from langdetect import detect

def get_language(text):
  try:
    lang = detect(text)
    return lang
  except:
    return "unknown"

df['lang'] = np.vectorize(get_language)(df['abstract'])

df.describe()
```

[6]:
```
              title                              doi abstract publish_time  \
count         34652                            34652    34652        34652
unique        34463                            34651    34442         5427
top     Abkürzungen  10.1097/jcma.0000000000000270  Unknown         2020
freq              5                                2      178          246

       publish_year   lang
count         34652  34652
unique           34     13
top            2019     en
freq           2919  34284
```

[7]:
```
df = df[df['lang'] == 'en']
df.describe()
```

[7]:
```
              title                              doi abstract publish_time  \
count         34284                            34284    34284        34284
unique        34105                            34283    34080         5426
top     Abkürzungen  10.1097/jcma.0000000000000270  Unknown         2020
freq              5                                2      178          242
```

```
       publish_year   lang
count         34284  34284
unique           34      1
top            2019     en
freq           2904  34284
```

[8]:
```
import spacy
from spacy.tokenizer import Tokenizer
from spacy.lang.en import English

nlp = spacy.load('en_core_web_sm')

def tokenize(text):
  tokens = []
  doc = nlp(text)
  for token in doc:
    if not token.is_stop and token.is_alpha and len(token.lemma_) > 1:
        tokens.append(token.lemma_)
  for chunk in doc.noun_chunks:
    tokens.append(chunk.text)
  return tokens

df['abstract'] = df['abstract'].apply(tokenize)

df.head()
```

[8]:
```
                                               title             doi  \
0  SIANN: Strain Identification by Alignment to N…  10.1101/001727
1  Spatial epidemiology of networked metapopulati…  10.1101/003889
2  Sequencing of the human IG light chain loci fr…  10.1101/006866
3  Bayesian mixture analysis for metagenomic comm…  10.1101/007476
4  Mapping a viral phylogeny onto outbreak trees …  10.1101/010389

                                            abstract publish_time  \
0  [generation, sequencing, increasingly, study, …   2014-01-10
1  [emerge, disease, infectious, epidemic, cause,…   2014-06-04
2  [germline, variation, immunoglobulin, gene, IG…   2014-07-03
3  [deep, sequencing, clinical, sample, establish…   2014-07-25
4  [develop, method, reconstruct, transmission, h…   2014-11-11

  publish_year lang
0         2014   en
1         2014   en
2         2014   en
3         2014   en
4         2014   en
```

[9]:
```
def get_size(tokens):
  return len(tokens)

df['size'] = df['abstract'].apply(get_size)

df.head()
```

[9]:
```
                                               title             doi  \
0  SIANN: Strain Identification by Alignment to N…  10.1101/001727
1  Spatial epidemiology of networked metapopulati…  10.1101/003889
2  Sequencing of the human IG light chain loci fr…  10.1101/006866
3  Bayesian mixture analysis for metagenomic comm…  10.1101/007476
4  Mapping a viral phylogeny onto outbreak trees …  10.1101/010389

                                            abstract publish_time  \
0  [generation, sequencing, increasingly, study, …   2014-01-10
1  [emerge, disease, infectious, epidemic, cause,…   2014-06-04
2  [germline, variation, immunoglobulin, gene, IG…   2014-07-03
3  [deep, sequencing, clinical, sample, establish…   2014-07-25
4  [develop, method, reconstruct, transmission, h…   2014-11-11

  publish_year lang  size
0         2014   en   154
1         2014   en   115
2         2014   en   168
3         2014   en   191
4         2014   en   191
```

[10]:
```
df = df[df['size'] > 50]
df = df[df['size'] < 500]
df.describe()
```

[10]:
```
               size
count  32994.000000
mean     175.742286
std       63.651460
min       51.000000
25%      130.000000
50%      172.000000
75%      214.000000
max      499.000000
```

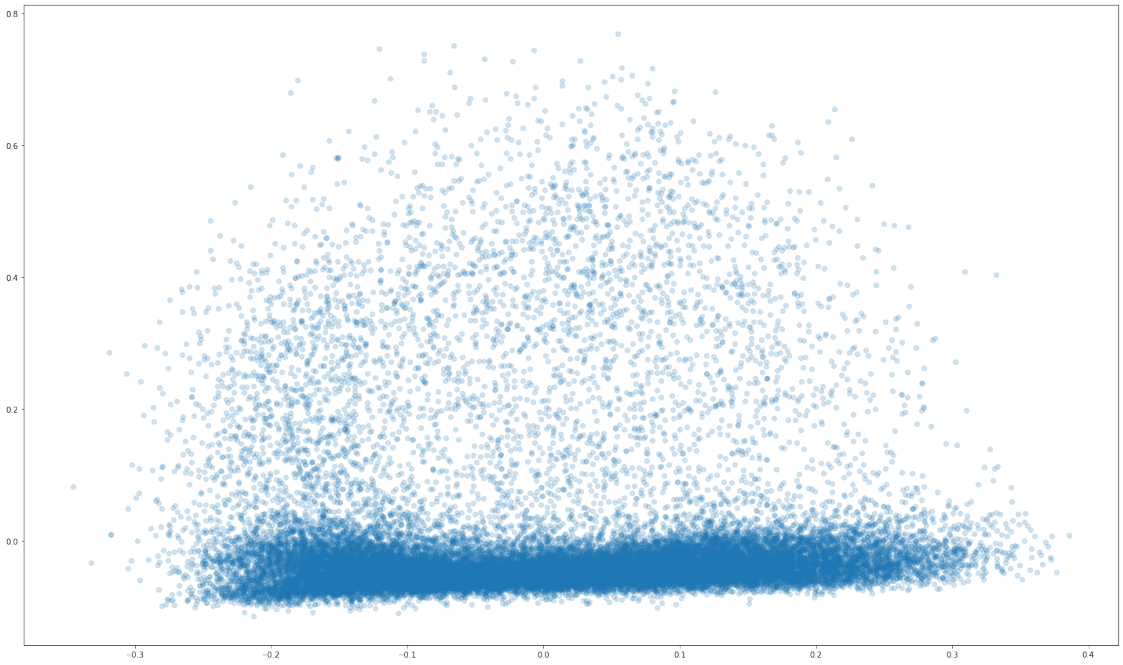
[14]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
#word_corpus = [' '.join(text) for text in df['abstract']]
#word_corpus = df['abstract'].to_list()

#define vectorizer parameters
tfidf_vectorizer = TfidfVectorizer(max_df=0.7, max_features=200000,
                                   min_df=0.01, stop_words=None,
                                   preprocessor=None, analyzer='word',
                                   use_idf=True, tokenizer=None)

%time tfidf_matrix = tfidf_vectorizer.fit_transform([' '.join(text) for text in␣
 ↪df['abstract']]) #fit the vectorizer to synopses

print(tfidf_matrix.shape)
```

```
CPU times: user 8.37 s, sys: 224 ms, total: 8.6 s
Wall time: 10.3 s
(32994, 1935)
```

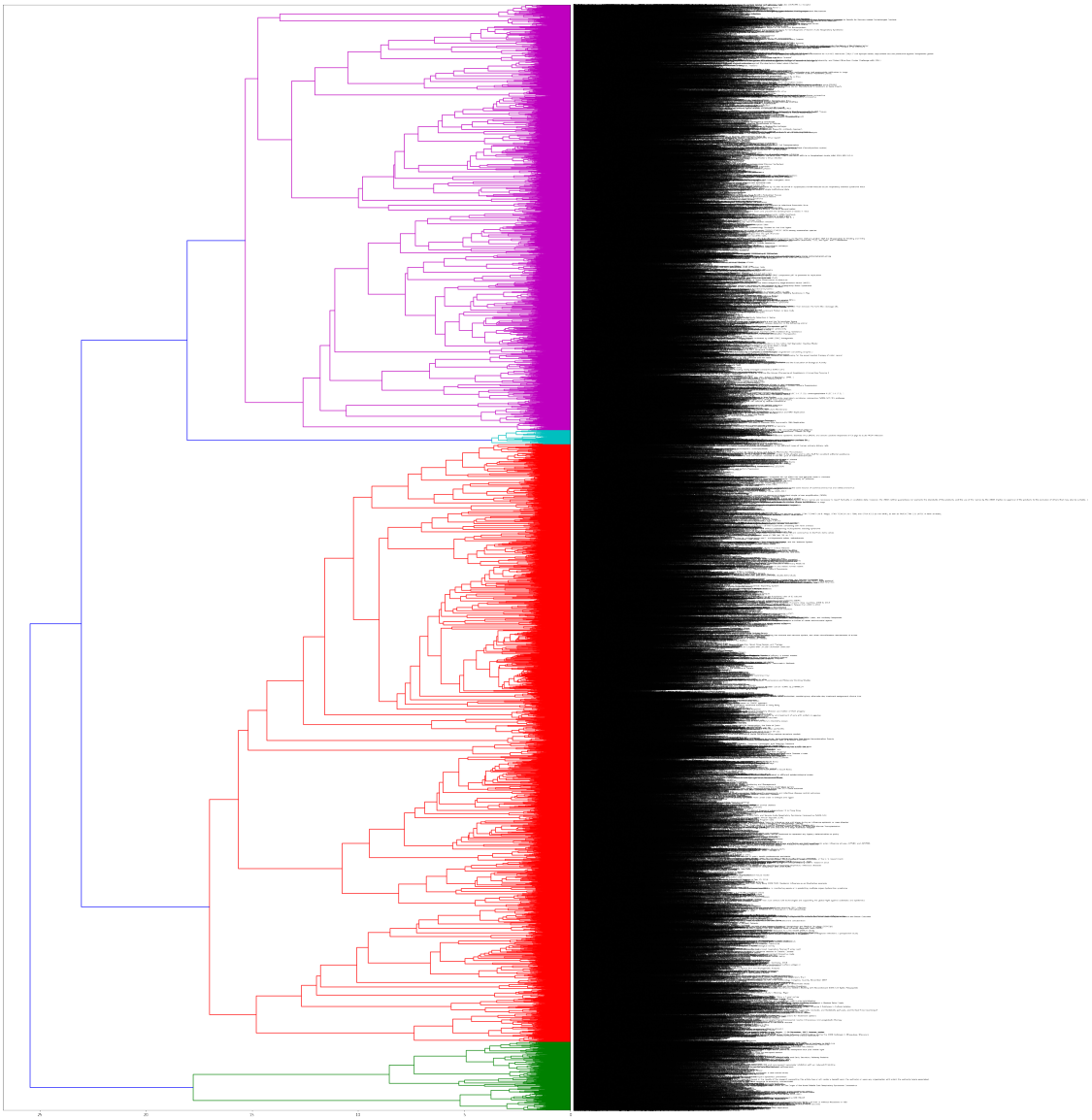
[15]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
x_matrix = tfidf_matrix.todense()
pca.fit(x_matrix)
X = pca.transform(x_matrix)


fig, ax = plt.subplots(figsize=(25, 15))
ax.scatter(X[:, 0], X[:, 1], alpha=0.2)

#for i, txt in enumerate(df['title'].to_list()):
#    ax.annotate(txt, (X[:, 0][i], X[:, 1][i]))
```

[15]:
```
<matplotlib.collections.PathCollection at 0x152c83290>
```



[17]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.manifold import MDS
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

#  The distance function can be ‘braycurtis’, ‘canberra’,
#‘chebyshev’, ‘cityblock’, ‘correlation’, ‘cosine’, ‘dice’, ‘euclidean’,␣
 ↪‘hamming’, ‘jaccard’,
# ‘jensenshannon’, ‘kulsinski’, ‘mahalanobis’, ‘matching’, ‘minkowski’,␣
 ↪‘rogerstanimoto’, ‘russellrao’,
# ‘seuclidean’, ‘sokalmichener’, ‘sokalsneath’, ‘sqeuclidean’, ‘yule’.

#dist = 1 - cosine_similarity(tfidf_matrix)
#MDS()
#mds = MDS(n_components=2, dissimilarity="precomputed", random_state=1)
#pos = mds.fit_transform(tfidf_matrix.todense())  # shape (n_components,␣
 ↪n_samples)

#linked = linkage(X, 'average', 'cosine')
linked = linkage(x_matrix, 'ward', 'euclidean')

labelList = df['title'].to_list()

plt.figure(figsize=(25, 100))
dendrogram(linked,
```

```
            orientation='left',
            labels=labelList,
            distance_sort='descending',
            show_leaf_counts=True)
plt.show()
```



[29]:
```
from sklearn.cluster import KMeans

num_clusters = 4

km = KMeans(n_clusters=num_clusters)
```

```
%time clusters=km.fit_predict(X)
#%time clusters=km.fit_predict(x_matrix)

#clusters = km.labels_.tolist()

df['cluster'] = clusters

df.head()
```

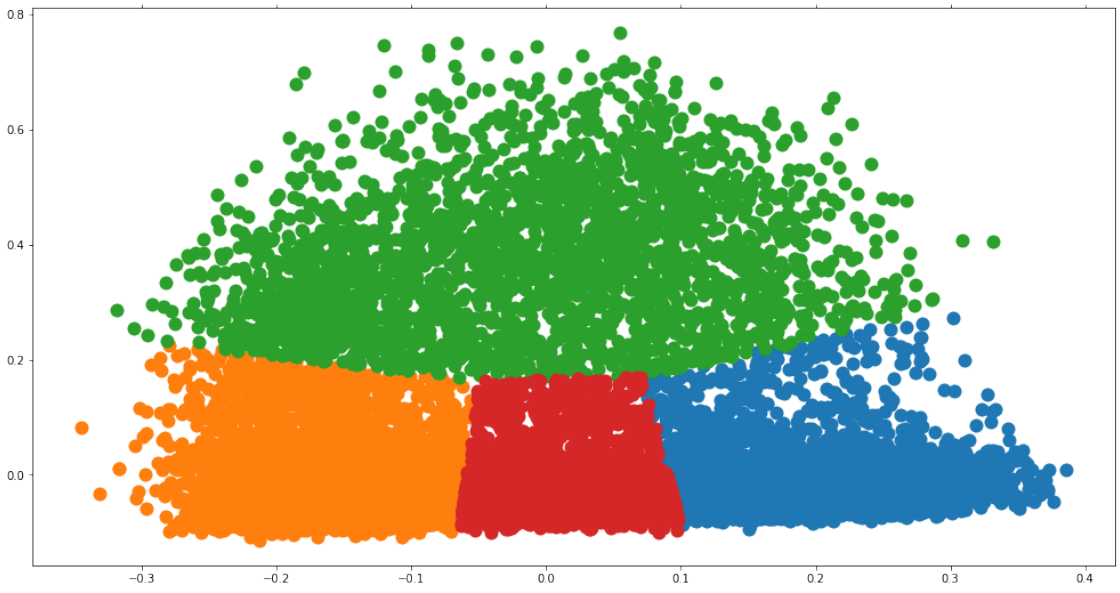
```
CPU times: user 954 ms, sys: 161 ms, total: 1.12 s
Wall time: 291 ms
```

[29]:

	title	doi	abstract	publish_time	publish_year	lang	size	cluster
0	SIANN: Strain Identification by Alignment to N…	10.1101/001727	[generation, sequencing, increasingly, study, …	2014-01-10	2014	en	154	0
1	Spatial epidemiology of networked metapopulati…	10.1101/003889	[emerge, disease, infectious, epidemic, cause,…	2014-06-04	2014	en	115	3
2	Sequencing of the human IG light chain loci fr…	10.1101/006866	[germline, variation, immunoglobulin, gene, IG…	2014-07-03	2014	en	168	0
3	Bayesian mixture analysis for metagenomic comm…	10.1101/007476	[deep, sequencing, clinical, sample, establish…	2014-07-25	2014	en	191	3
4	Mapping a viral phylogeny onto outbreak trees …	10.1101/010389	[develop, method, reconstruct, transmission, h…	2014-11-11	2014	en	191	3

[27]:
```
#some ipython magic to show the matplotlib plots inline
%matplotlib inline

xs, ys = X[:, 0], X[:, 1]

#create data frame that has the result of the MDS plus the cluster numbers and␣
 ↪titles
df2 = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=df['title'].
 ↪to_list()))

#group by cluster
groups = df2.groupby('label')

# set up plot
```

```
fig, ax = plt.subplots(figsize=(17, 9)) # set size
ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling

#iterate through groups to layer the plot
#note that I use the cluster_name and cluster_color dicts with the 'name'␣
 ↪lookup to return the appropriate color/label
for name, group in groups:
    #ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,
    #        label=cluster_names[name], color=cluster_colors[name],
    #        mec='none')
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,
            mec='none')
    ax.set_aspect('auto')
    ax.tick_params(\
        axis= 'x',          # changes apply to the x-axis
        which='both',      # both major and minor ticks are affected
        bottom='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelbottom='off')
    ax.tick_params(\
        axis= 'y',         # changes apply to the y-axis
        which='both',      # both major and minor ticks are affected
        left='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelleft='off')

#ax.legend(numpoints=1)  #show legend with only 1 point

#add label in x,y position with the label as the film title
#for i in range(len(df)):
    #ax.text(df.ix[i]['x'], df.ix[i]['y'], df.ix[i]['title'], size=8)



plt.show() #show the plot
```

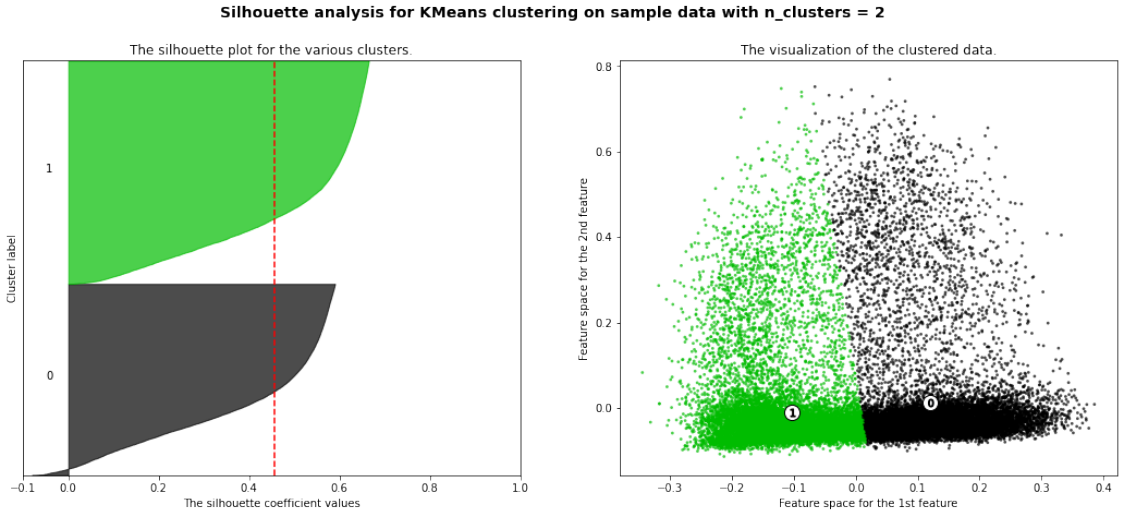


[28]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)
```

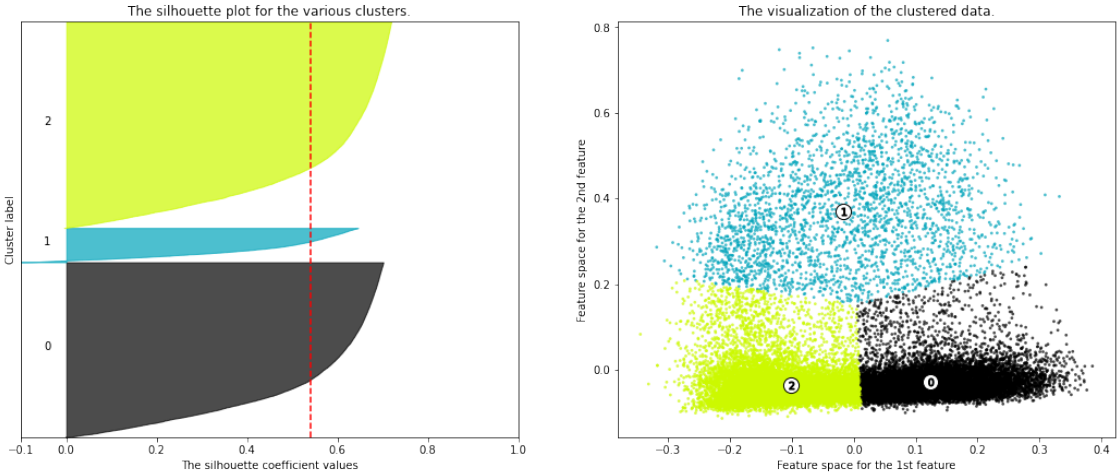
```
    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')
```

```
    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

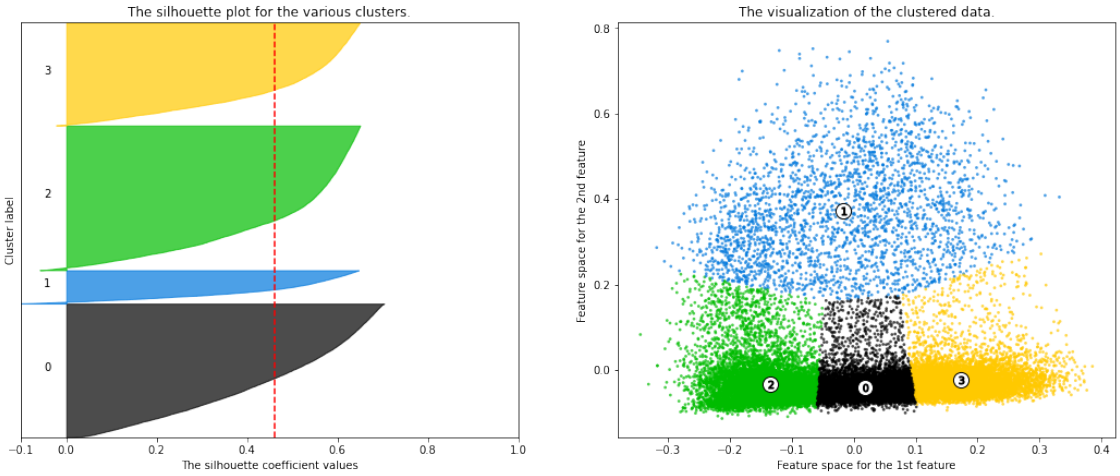
```
For n_clusters = 2 The average silhouette_score is : 0.4566245288917496
For n_clusters = 3 The average silhouette_score is : 0.5406117622416885
For n_clusters = 4 The average silhouette_score is : 0.4608330202987449
For n_clusters = 5 The average silhouette_score is : 0.4658664338972714
For n_clusters = 6 The average silhouette_score is : 0.4232161758542394
```

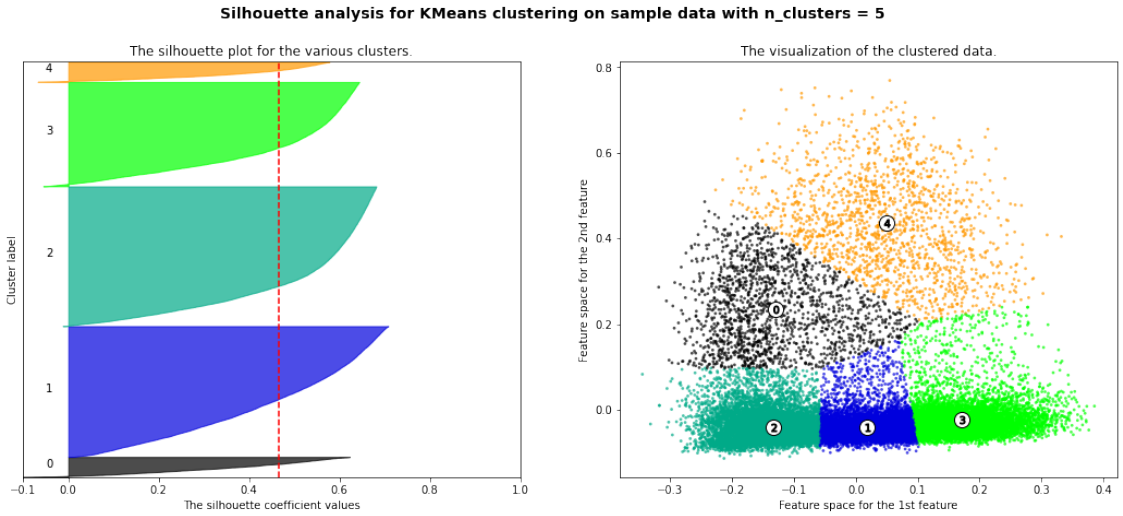


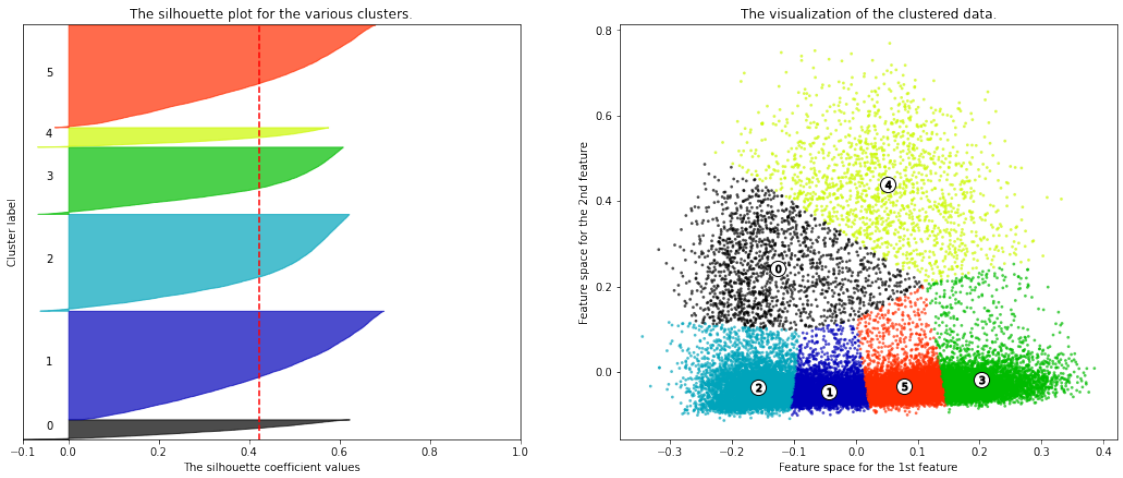
Silhouette analysis for KMeans clustering on sample data with n_clusters = 3



Silhouette analysis for KMeans clustering on sample data with n_clusters = 4







[ ]:
```
from gensim.models import ldaseqmodel
from gensim.corpora import Dictionary
from gensim.test.utils import get_tmpfile
import numpy
from gensim.matutils import hellinger
import  csv

df.sort_values(by=['publish_year'], inplace=True)

year_slice_df = df.groupby(['publish_year']).size().reset_index(name='counts')

tmp_fname = get_tmpfile(base_directory+"/dictionary.txt")
```

```
dictionary = Dictionary(df['abstract'])

# Filter out words that occur less than 50 documents, or more than 50% of the␣
 ↪documents.
dictionary.filter_extremes(no_below=20, no_above=0.5)

dictionary.save_as_text(tmp_fname)
print("dictionary saved at",tmp_fname)

corpus = [dictionary.doc2bow(text) for text in df['abstract']]

with open(base_directory+"/corpus.csv","w") as f:
    wr = csv.writer(f)
    wr.writerows(corpus)
print("corpus saved")

print('Number of unique tokens: %d' % len(dictionary))
print('Number of documents: %d' % len(corpus))
print(year_slice_df)
```

[ ]:
```
from gensim.models import ldaseqmodel
from gensim.test.utils import datapath

number_of_topics = 4
time_slice = year_slice_df['counts'].tolist()
print(time_slice)
%time ldaseq = ldaseqmodel.LdaSeqModel(
    corpus=corpus,
    id2word=dictionary,
    time_slice=time_slice,
    num_topics=number_of_topics,
    alphas=0.01,
    chain_variance = 0.005,
    passes=5,
    lda_inference_max_iter=25,
    em_min_iter=6,
    em_max_iter=20,
    chunksize=100
)
print("model created")
```

[ ]:
```
from IPython.display import display

def get_topic_df(topic_id,num_words):
  topics = ldaseq.print_topic_times(topic=topic_id,top_terms=num_words)
  topic_slice = 0
  data = {}
```

```
  for year in year_slice_df['publish_year']:
    topic_words = topics[topic_slice]
    word_list = []
    for word,score in topic_words:
      word_list.append(word)
    data[year]=word_list
    topic_slice += 1
  return pd.DataFrame(data)

# Create DataFrame

for topic in range(0,number_of_topics):
  print("Topic",topic)
  display(get_topic_df(topic,10))
```

[ ]:
```
temp_file = get_tmpfile(base_directory+"/model")
ldaseq.save(temp_file)
print("model saved")
```

[ ]:
```
with open(base_directory+"/corpus-embeddings.csv","w") as f:
    wr = csv.writer(f)
    for doc in corpus:
        embedding = ldaseq[doc]
        wr.writerow(embedding)
print("corpus embeddings saved")
```