Assignment3_45731365_StanleyEpuna_COMP8210

October 29, 2021

0.0.1 Importing / installing libraries

[1]:
```
#Use the following code to install any required libraries that are missing
#!pip install -U pip setuptools wheel
#!pip install -U spacy
#!python -m spacy download en_core_web_sm
#!pip install --upgrade gensim
#!pip install --user torch torchvision torchaudio
#!pip install --user transformer
#!pip install --user pyldavis
#!pip install --user wordcloud
#!pip install --user -U nltk
from pprint import pprint
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import torch
from transformers import BertTokenizer
import gensim.models.phrases as gen
from gensim.models.coherencemodel import CoherenceModel
from wordcloud import WordCloud
import seaborn as sns
import pyLDAvis
import pyLDAvis.sklearn
import re
from gensim import models,corpora
import nltk
nltk.download('stopwords')
import string
string.punctuation
stopwords = nltk.corpus.stopwords.words('english')
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
from sklearn.manifold import TSNE from sklearn.decomposition
import LatentDirichletAllocation from nltk.tokenize import
word_tokenize from nltk.stem.snowball import SnowballStemmer from
sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer from sklearn.feature_extraction.text import
CountVectorizer, TfidfVectorizer from tqdm._tqdm_notebook import
tqdm_notebook,tnrange,tqdm from collections import
Counter,OrderedDict from nltk.stem.wordnet import
WordNetLemmatizer lmtzr = WordNetLemmatizer() import warnings
import pyLDAvis.gensim_models import re, nltk, spacy, gensim from
sklearn.decomposition import LatentDirichletAllocation,
TruncatedSVD from sklearn.model_selection import GridSearchCV
import warnings warnings.filterwarnings('ignore')
```

```
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:34: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  method='lar', copy_X=True, eps=np.finfo(np.float).eps,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:164: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  method='lar', copy_X=True, eps=np.finfo(np.float).eps,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:281: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_Gram=True, verbose=0,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:865: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:1121: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
```

```
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:1149: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, positive=False):
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:1379: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:1621: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\linear_model\_least
_angle.py:1755: DeprecationWarning: `np.float` is a deprecated alias
for the builtin `float`. To silence this warning, use `float` by
itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
eps=np.finfo(np.float).eps, copy_X=True, positive=False):
C:\Users\abuwa\anaconda3\lib\site-
packages\sklearn\decomposition\_lda.py:28: DeprecationWarning:
`np.float` is a deprecated alias for the builtin `float`. To silence
this warning, use `float` by itself. Doing this will not modify any
behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations EPS
= np.finfo(np.float).eps
```

```
C:\Users\abuwa\anaconda3\lib\sitepackages\sklearn\feature_extraction\
image.py:172: DeprecationWarning: `np.int` is a deprecated alias for
the builtin `int`. To silence this warning, use `int` by itself.
Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to
specify the precision. If you wish to review your current use, check
the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  dtype=np.int):
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\abuwa\AppData\Roaming\nltk_data…
[nltk_data] Package stopwords is already up-to-date!
<ipython-input-1-06ecb96179f3>:43: TqdmDeprecationWarning: This
function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.*` instead of
  `tqdm._tqdm_notebook.*` from tqdm._tqdm_notebook import
  tqdm_notebook,tnrange,tqdm
```

0.1 Task 1 - What is the distribution of journals per source?

Due to large dataset, limited memory and performance availability, a portion of the dataset it used (220,000 rows). The 'nrows' value can be easily changed for a larger dataset if more memory and performance is available.

```
[2]: df = pd.read_csv("metadata.csv",nrows=220000)
```

Getting the journal count by source and visualizing the data via table.

```
[3]: new_df = df.value_counts(subset=['source_x', 'journal']).
     ↪reset_index(name="count")
     new_df.head(20)
```

```
[3]:   source_x                                    journal count
     0   Medline                                        BMJ  2862
     1   Medline                          Journal of virology 2154
     2   BioRxiv                                    bioRxiv  2036
     3       PMC                           Reactions Weekly 1602
     4   Medline                                     Nature  1301
     5   Medline                                       JAMA 1041
     6       PMC                                   PLoS One  1004
     7   Medline                          Surgical endoscopy   974
     8   Medline                                    Science   789
     9   Medline            The New England journal of medicine
                                                              766
   10      Medline Proceedings of the National Academy of Science…
                                                              665
             11      Medline   The Journal of general virology  607
                                                  12      PMC Sci Rep     572
```

```
  13     Medline   Journal of neurointerventional surgery 536


14 Medline     AJNR. American journal of neuroradiology  532
15 Medline                         The Veterinary record  527
16 Medline                                      BMJ open   512
17 Medline                                      PloS one   504
18 Medline Advances in experimental medicine and biology  504
19  PMC Computational Science and Its Applications - I… 500
```

Getting the top 10 most frequent journals that appear in the dataset.

```
[4]: df['journal'].value_counts()[:10]
```

```
[4]:  BMJ                   3155
   Journal of virology   2154
    bioRxiv              2043
    Reactions Weekly     1620
    Nature               1608
    PLoS One             1276
    JAMA                 1134
   Surgical endoscopy     974
    Science               878
    Lancet                875
  Name: journal, dtype: int64
```

Extracting the top 10 most frequent journals from the dataset

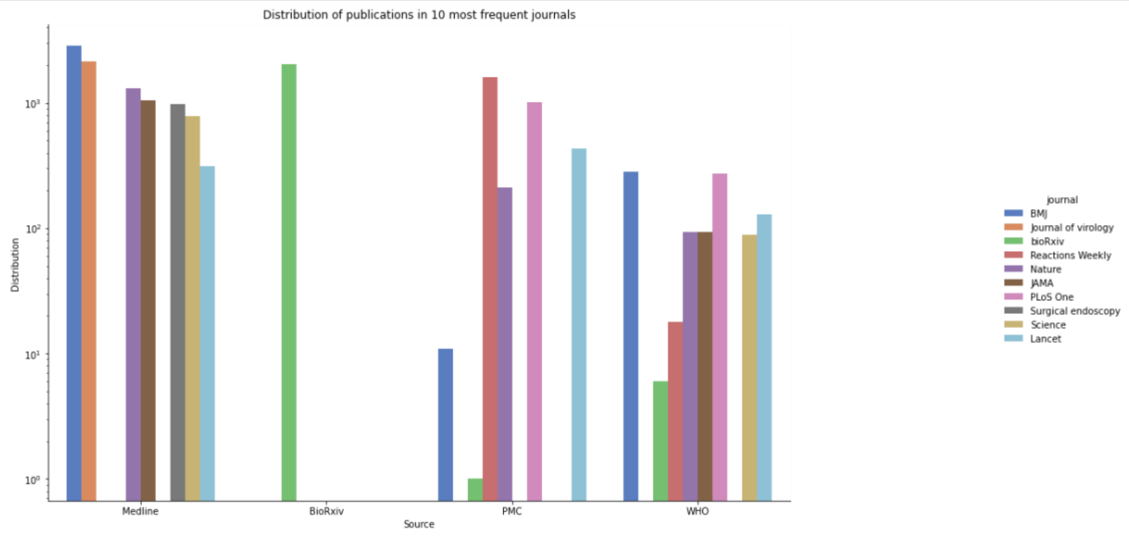
```
[5]: top_ten_journals = ['BMJ','Journal of virology','bioRxiv','Reactions␣
     ↪Weekly','Nature','PLoS One','JAMA','Surgical
      endoscopy','Science','Lancet' ]
```

Visualizing the dataset via a table after extracting top 10 most frequent journals

```
[6]: new_df = new_df[new_df.journal.isin(top_ten_journals)]
```

Distribution of journals by source via clustered column chart. Plotting is conducted using seaborn catplot.

```
[7]: # Visualization of a barplot
     g = sns.catplot(x="source_x",  =new_df['count'], hue="journal",
                         data=new_df, kind="bar",
                         height=5, palette="muted", legend=True, log=True)

     g.fig.set_figwidth(18)
     g.fig.set_figheight(8)
     plt.title('Distribution of publications in 10 most frequent journals ')
     plt.xlabel('Source')
     plt.ylabel('Distribution')
     plt.show()
```



Outcome: Although 7 sources exist in total, the clustered column chart shows only 4 sources. This may be because the remaining sources do not have any journal distribution or they do not possess enough journals that could be appropriately visualized in the chart.

0.1.1 Task 2 - What are the main clusters?

```
[8]: df = pd.read_csv("metadata.csv",nrows=1500)
```

Due to limited memory and performance availability, a portion of the dataset will be used (1500 rows). The ‘nrows’ value can be easily changed for a larger dataset if more memory and performance is available. A larger dataset will always yield a more accurate representation of the entire dataset.

Dropping all empty rows in the abstract column and extract the column for further analysis.

```
[9]: df = df.dropna(subset=['abstract'])
     df['abstract'] = df['abstract'].astype(str)
     df = pd.DataFrame(df['abstract'])
     df
```

```
[9]:                                              abstract
     0     OBJECTIVE: This retrospective chart review des…
     1     Inflammatory diseases of the respiratory tract…
     2     Surfactant protein-D (SP-D) participates in th…
     3     Endothelin-1 (ET-1) is a 21 amino acid peptide…
     4     Respiratory syncytial virus (RSV) and pneumoni…
     …                                                  …
     1494 Glycyrrhizic acid (GA) is a triterpene glycosi…
     1495 Background: Upper respiratory tract infections…
     1496 BACKGROUND: Various pathways have been implica…
     1497 INTRODUCTION: There is a hyperoxidative state …
     1498 INTRODUCTION: The aim of this study was to inv…

     [1435 rows x 1 columns]
```

Verifying that no null values exist in the abstract column.

```
[10]: df.isnull().sum()
```

```
[10]: abstract  0
     dtype: int64
```

Text preprocessing is conducted. We'll also add our own custom stopwords which include some common words used in research papers.

```
[11]: custom_stop_words = [
         'reserved','peer','CZI', 'reviewed', 'org' , 'et', 'author',
         'figure',
    'rights', 'permission','Elsevier','biorxiv','https', 'copyright',␣
     ↪'medrxiv', 'license','preprint','fig', 'fig.',
         'al.', 'al', 'PMC', ,'doi'
     ]

     stopwords.extend(custom_stop_wo

     rds)

     # A function to prepare the
     text def clean(text):
         regex = re.compile('[' + re.escape(string.punctuation) + '0-
         9\\r\\t\\n]') text = regex.sub(" ", text.lower()) words =
         text.split(" ") words = [re.sub('\S*@\S*\s?', '', sent) for
sent in words] words = [re.sub('\s+', ' ', sent) for sent in
words] words = [re.sub("\'", "", sent) for sent in words] words
= [w for w in words if not len(w) < 2] words = [w for w in
words if w not in stopwords] words = [lmtzr.lemmatize(w) for w
in words] words = ' '.join([str(w) for w in words]) return
words
```

Applying above function to the abstract column.

```
[12]: df["clean_abstract"] = df["abstract"].apply(clean)
```

Tokenizing the text in abstract column with BertTokenizer [13]: def

```
sen_to_vec(sentence):
tokenizer=BertTokenizer.from_pretrained('bert-base-uncased')
tokens=tokenizer.tokenize(sentence)   tokens   =   ['[CLS]']   +
tokens + ['[SEP]']
    T=624
    padded_tokens=tokens +['[PAD]' for _ in range(T-len(tokens))]
    attn_mask=[ 1 if token != '[PAD]' else 0 for token in
    padded_tokens ]
    seg_ids=[0 for _ in range(len(padded_tokens))]
    sent_ids=tokenizer.convert_tokens_to_ids(padded_toke
    ns) return np.array(sent_ids)
```

After tokenizaton is completed, it is stored in a new data frame ‘final_df’ for further analysis.

```
[14]: df["array"] =
     df["clean_abstract"].apply(sen_to_vec)
     final_df = pd.DataFrame(df["array"]) final_df
     = final_df.pop('array').apply(pd.Series)
     final_df = final_df.fillna(0)
     final_df.head(3)
```

```
[14]:   0      1      2      3      4      5      6      7     8      9    … \
     0  101 7863 15354 3673  3319  5577  4958  5178 4328  6779 …
     1  101 201874295 16464 12859 4141  3378  8319 2537  9152 …
     2  101 14175 18908      4630  5250 11867 17257 25605 3433 15938 …

        614  615  616  617  618  619  620  621  622  623
      0    0    0    0    0    0    0    0    0    0    0
      1    0    0    0    0    0    0    0    0    0    0
      2    0    0    0    0    0    0    0    0    0    0
     [3 rows x 624 columns]
```

Using PCA to reduce the dimensionality of tokenized array data to 2 so that it can be visualized efficiently.

```
[15]: from sklearn.decomposition import PCA

      pca = PCA(n_components = 2)
      X_PCA=pca.fit_transform(final_df)

      print(final_df.shape)
      print(X_PCA.shape)
```

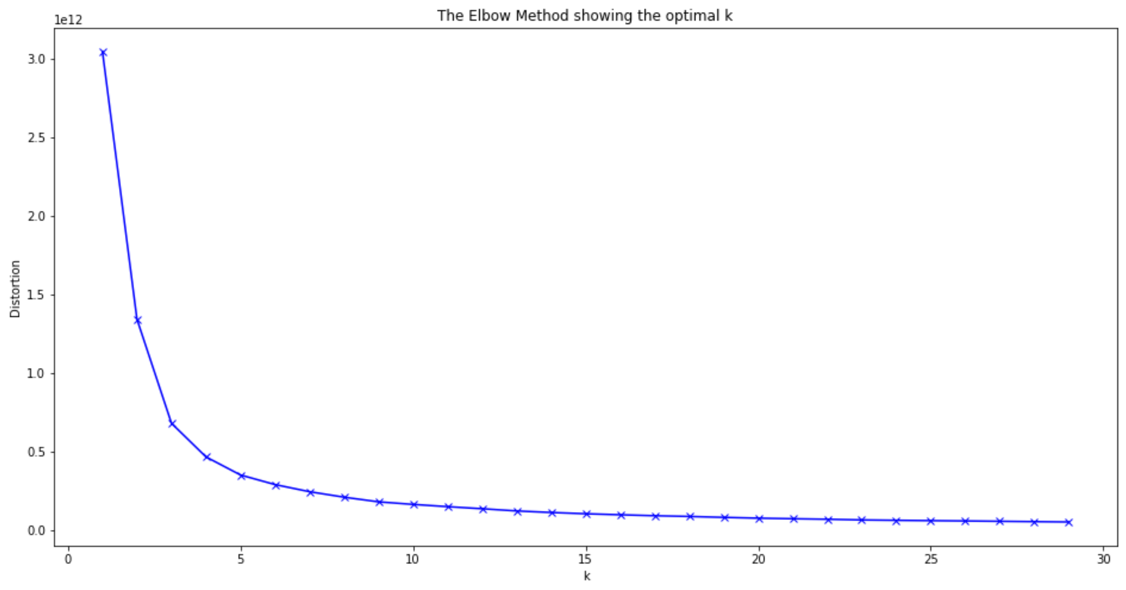
```
(1435, 624)
(1435, 2)
```

Finding the best K value for our Kmeans model.

```
[16]: distortions = []
      K = range(1,30)
      for k in K:
          kmeanModel = KMeans(n_clusters=k)
          kmeanModel.fit(X_PCA)
          distortions.append(kmeanModel.inertia_)
```

Finding the best K value by using Elbow method visualization.

```
[17]: plt.figure(figsize=(16,8))
      plt.plot(K, distortions, 'bx-')
      plt.xlabel('k')
      plt.ylabel('Distortion')
      plt.title('The Elbow Method showing the optimal k ')
      plt.show()
```



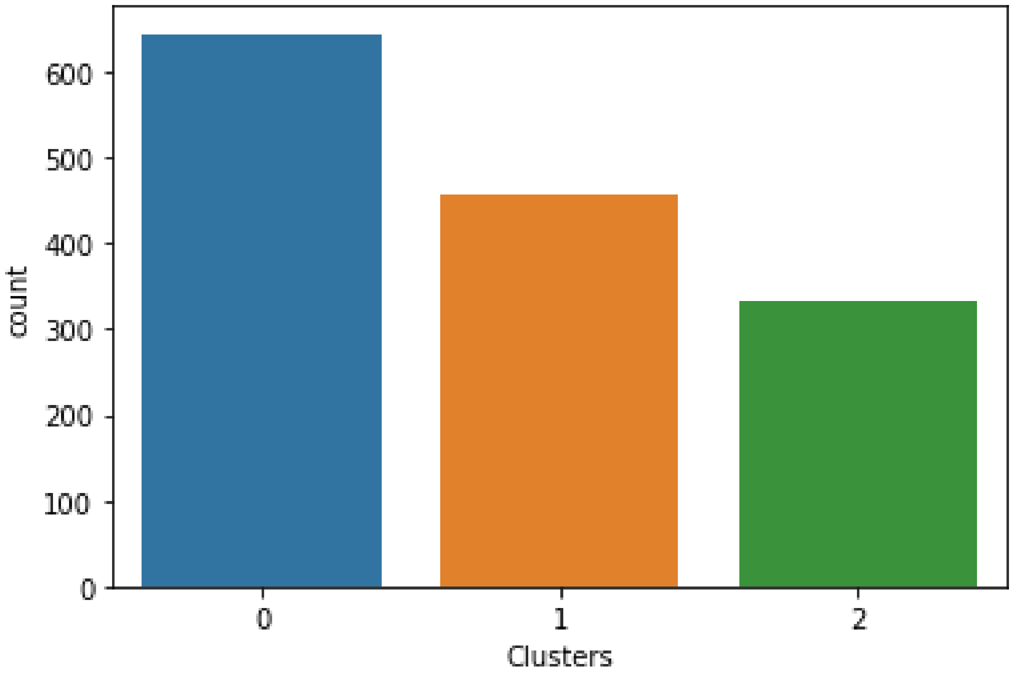
By analyzing above graph, we can deduce k = 3.

```
[18]: k = 3
      kmeans = KMeans(n_clusters=k, random_state=42)
      y_pred = kmeans.fit_predict(X_PCA)
      df['Clusters'] = y_pred
```

Vizualisation of clusters.

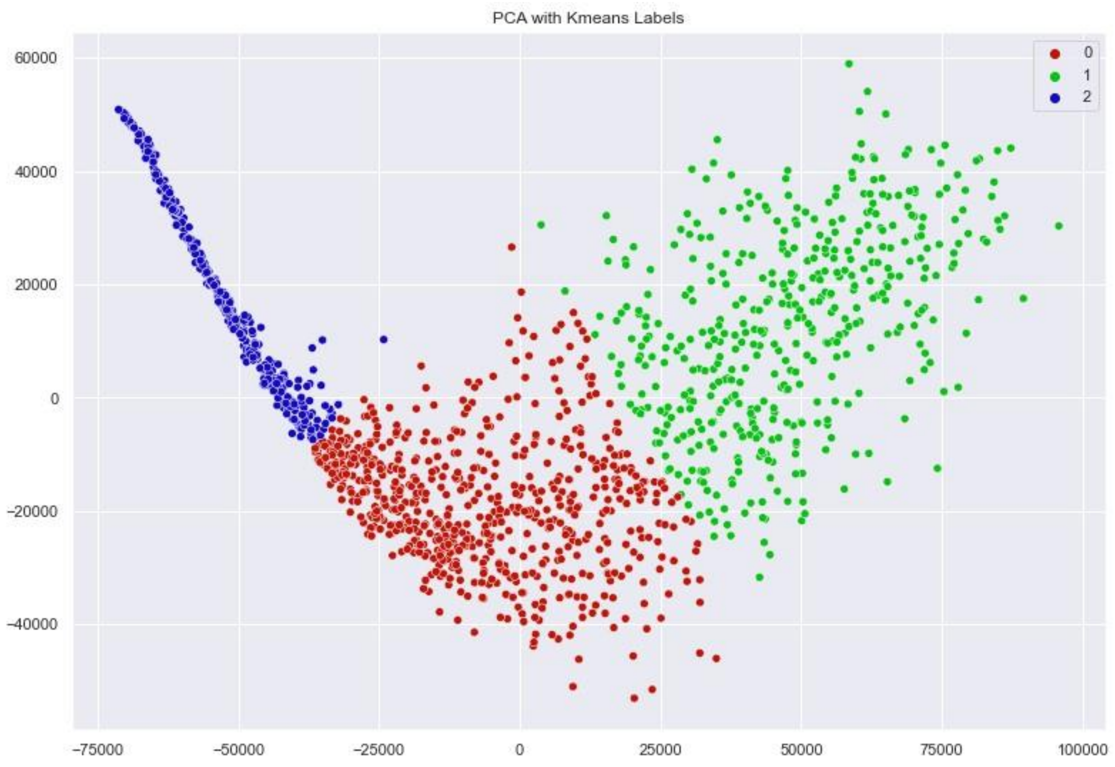
```
[19]: sns.countplot(df['Clusters'])
```

```
[19]: <AxesSubplot:xlabel='Clusters', ylabel='count'>
```



2D representation of each document in its corresponding clusters. The clustered are represented by its following colours.

```
[20]: # sns settings
      sns.set(rc={'figure.figsize':(13,9)})

      # colors
      palette = sns.hls_palette(3,  =.4,  =.9)

      # plot
      sns.scatterplot(X_PCA[:,0], X_PCA[:,1], hue=y_pred, legend='full',␣
       ↪palette=palette)
      plt.title('PCA with Kmeans Labels ')
      plt.savefig("clusterRevised.png")
      plt.show()
```



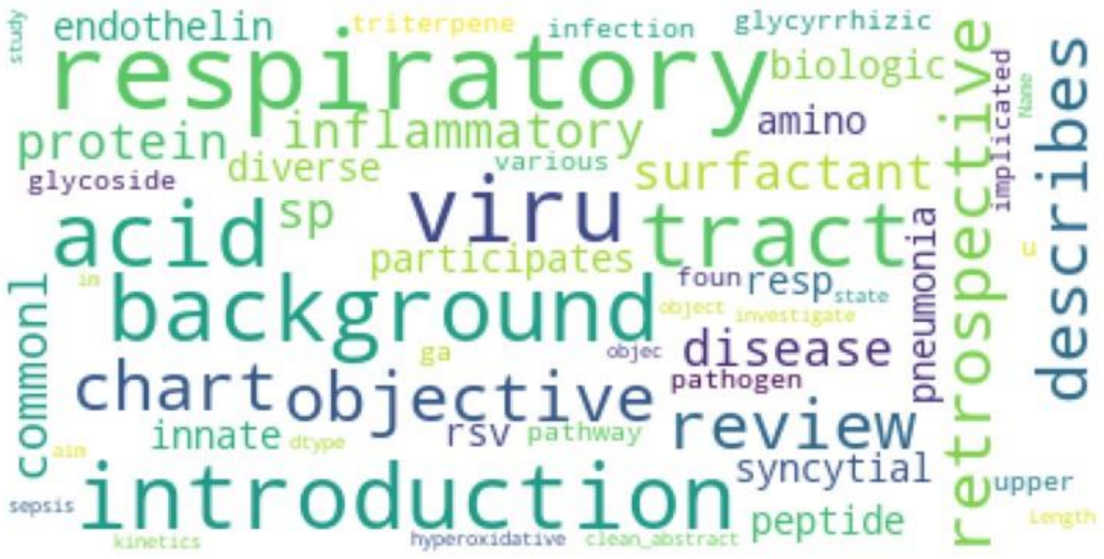
0.1.2 Task 3 - For each cluster, what are the most representative words?

[21]:
```
wordcloud =␣
↪WordCloud(background_color='white',stopwords=stopwords,max_words=200,max_font_size=40,rando
↪generate(str(df['clean_abstract']))
```

[22]:
```
print(wordcloud)
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
fig.savefig("word1.png", dpi=900)
```

We will use cleaned abstract data obtained earlier to produce the wordcloud.

```
<wordcloud.wordcloud.WordCloud object at 0x000001905114BAF0>
```



Note: Upon analyzing the above Wordcloud, we can see that even more meaningful results can be obtained if more custom stopwords are added to our list such as “introduction”, “background”, “objective”. This can easily be done by adding these words to our already created list above “custom_stop_words”. However, as the program takes many hours to run for even small datasets, it was decided to not perform this operation again.

0.1.3 Task 4 - What are the most common topics?

Preprocessing of data using gensim.

```
[23]: data =
df.clean_abstract.values.tolist() def
sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence),
        deacc=True))  #␣
 ↪deacc=True removes punctuations
data_words =
list(sent_to_words(data))
print(data_words[:1])
```

```
[['objective', 'retrospective', 'chart', 'review', 'describes',
'epidemiology',
'clinical', 'feature', 'patient', 'culture', 'proven', 'mycoplasma',
'pneumoniae', 'infection', 'king', 'abdulaziz', 'university',
'hospital',
'jeddah', 'saudi', 'arabia', 'method', 'patient', 'positive',
'pneumoniae',
'culture', 'respiratory', 'specimen', 'january', 'december',
'identified',
'microbiology', 'record', 'chart', 'patient', 'result', 'patient',
'identified',
'required', 'admission', 'infection', 'community', 'acquired',
'infection',
'affected', 'age', 'group', 'common', 'infant', 'pre', 'school',
'child',
'occurred', 'year', 'round', 'common', 'fall', 'spring', 'three',
'quarter',
'patient', 'comorbidities', 'twenty', 'four', 'isolates',
'associated',
'pneumonia', 'upper', 'respiratory', 'tract', 'infection',
'bronchiolitis',
'cough', 'fever', 'malaise', 'common', 'symptom', 'crepitation',
'wheeze',
'common', 'sign', 'patient', 'pneumonia', 'crepitation', 'bronchial',
'breathing', 'patient', 'likely', 'non', 'patient', 'present',
'pneumonia',
'versus', 'patient', 'pneumonia', 'uneventful', 'recovery',
'recovered',
'following', 'complication', 'died', 'pneumoniae', 'infection',
'died', 'due',
'underlying', 'comorbidities', 'patient', 'died', 'pneumoniae',
'pneumonia',
'comorbidities', 'conclusion', 'result', 'similar', 'published',
'data',
'except', 'finding', 'infection', 'common', 'infant', 'preschool',
'child',
'mortality', 'rate', 'pneumonia', 'patient', 'comorbidities',
'high']]
```

Lemmatization

```
[24]: def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB',
'ADV']):␣
    ↪#'NOUN', 'ADJ', 'VERB',
       'ADV' texts_out = []
       for sent in texts:
           doc = nlp(" ".join(sent))
           texts_out.append(" ".join([token.lemma_ if token.lemma_ not
           in␣
    ↪['-PRON-'] else '' for token in doc if token.pos_ in
       allowed_postags])) return texts_out
```

```
[25]: # Initialize spacy 'en' model, keeping only tagger component (for
efficiency)
      # Run in terminal: python -m spacy download en
      nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner']) #
      Do lemmatization keeping only Noun, Adj, Verb, Adverb
      data_lemmatized = lemmatization(data_words,
      allowed_postags=['NOUN', 'ADJ',␣
       ↪'VERB', 'ADV']) #select noun and verb
      print(data_lemmatized[:2])
```

```
['objective retrospective chart review describe epidemiology clinical
feature patient culture prove mycoplasma infection king method
patient positive pneumoniae culture respiratory speciman identify
microbiology record chart patient result patient identify require
admission infection community acquire infection affect age group
common infant pre school child occur year round common fall spring
quarter patient comorbiditie isolate associate upper respiratory
tract cough fever malaise common symptom crepitation wheeze common
sign patient pneumonia bronchial breathing patient likely non patient
present pneumonia patient pneumonia uneventful recovery recover
follow complication die pneumoniae infection die due underlie
comorbiditie patient die pneumonia comorbiditie conclusion result
similar publish datum find infection common infant preschool child
mortality rate pneumonia patient comorbiditie high', 'inflammatory
disease respiratory tract commonly associate elevated production
nitric oxide increase index dependent oxidative stress know anti
microbial anti inflammatory anti oxidant property various line
evidence support contribution lung injury several disease model basis
biochemical evidence often presume dependent oxidation due formation
oxidant peroxynitrite alternative mechanism involve phagocyte derive
heme protein myeloperoxidase peroxidase operative condition
inflammation overwhelm literature generation activity respiratory
tract scope commentary review area comprehensively instead focus
recent evidence concept presume contribution inflammatory disease
lung']
```

Data is vectorized using CountVectorizer

```
[26]: vectorizer = CountVectorizer(analyzer='word',
                                 min_df=10,
      # minimum reqd occurences of a word
                                 stop_words='english',
      # remove stop words
                                 lowercase=True,
      # convert all words to lowercase
                                 token_pattern='[a-zA-Z0-9]{3,}')

      data_vectorized = vectorizer.fit_transform(data_lemmatized)
```

Buildng an LDA Model.

Since we had 3 clusters we will stick to 3 topics, to be able to make a comparison.

```
[27]: # Build LDA Model
      lda_model = LatentDirichletAllocation(n_components=3,                # Number of␣
       ↪topics
                                            max_iter=10,
      # Max learning iterations
                                            learning_method='online',
                                            random_state=100,
      # Random state
                                            batch_size=128,
      # n docs in each learning iter
                                            evaluate_every = -1,
      # compute perplexity every n iters, default: Don't
                                            n_jobs = -1,
      # Use all available CPUs
                                           )
      lda_output = lda_model.fit_transform(data_vectorized)
      print(lda_model)  # Model attributes
```

```
LatentDirichletAllocation(learning_method='online', n_components=3,
                          n_jobs=-1, random_state=100)
```

For each topic, get the top n keywords.

```
[29]:  # Show top n keywords for each topic def
      show_topics(vectorizer=vectorizer, lda_model=lda_model,
      n_words=20):
          keywords = np.array(vectorizer.get_feature_names())
          topic_keywords = [] for topic_weights
          in lda_model.components_:
              top_keyword_locs = (-
          topic_weights).argsort()[:n_words]
          topic_keywords.append(keywords.take(top_keyword_locs))
          return topic_keywords
```

```
topic_keywords = show_topics(vectorizer=vectorizer,
 lda_model=lda_model,␣ ↪n_words=15)

# Topic - Keywords Dataframe df_topic_keywords =
pd.DataFrame(topic_keywords) df_topic_keywords.columns = ['Word
'+str(i) for i in range(df_topic_keywords.
↪shape[1])] df_topic_keywords.index = ['Topic '+str(i) for i in
range(df_topic_keywords.
↪shape[0])]
df_topic_keywords
```

[29]:
```
           Word 0    Word 1    Word 2    Word 3      Word 4    Word 5  \
Topic 0      use   disease influenza    health       virus sequence
Topic 1  patient infection     study influenza respiratory clinical
Topic 2     cell   protein     virus      gene   infection    viral

              Word 6 Word 7   Word 8 Word 9   Word 10  Word 11   Word 12  \
Topic 0     method  study   result  model     datum pandemic  analysis
Topic 1       high    day   severe result treatment disease mortality
Topic 2 expression    use activity  human     study   induce  response

              Word 13    Word 14
Topic 0 transmission population
Topic 1        virus       case
Topic 2       result       host
```

Adding appropriate labels for the 3 topics.

[30]:
```
Topics = ["Research","Disease and Treatment","Virus Behaviour"]

df_topic_keywords["Topics"]=Topics
df_topic_keywords
```

[30]:
```
           Word 0    Word 1    Word 2    Word 3      Word 4    Word 5  \
Topic 0      use   disease influenza    health       virus sequence
Topic 1  patient infection     study influenza respiratory clinical
Topic 2     cell   protein     virus      gene   infection    viral

              Word 6 Word 7   Word 8 Word 9   Word 10  Word 11   Word 12  \
Topic 0     method  study   result  model     datum pandemic  analysis
Topic 1       high    day   severe result treatment disease mortality
Topic 2 expression    use activity  human     study   induce  response

              Word 13    Word 14                Topics
Topic 0 transmission population   Research Topic 1
 virus case Disease and Treatment
```

```
Topic 2       result       host       Virus Behaviour
```

Defining a function to predict topic for a given text document. Furthermore, as an example, topic is inferred for a given text. This can be seen in the result below.

```
[36]: nlp = spacy.load('en_core_web_sm', disable=['parser',
'ner']) def predict_topic(text, nlp=nlp):
    global sent_to_words
    global lemmatization
# Step 1: Clean with simple_preprocess
    mytext_2 = list(sent_to_words(text))
# Step 2: Lemmatize mytext_3 = lemmatization(mytext_2,
 allowed_postags=['NOUN', 'ADJ', 'VERB',␣ ↪'ADV'])
# Step 3: Vectorize transform mytext_4
    = vectorizer.transform(mytext_3)
# Step 4: LDA Transform topic_probability_scores =
 lda_model.transform(mytext_4) topic =
 df_topic_keywords.iloc[np.argmax(topic_probability_scores), 1:14].
↪values.tolist()

    # Step 5: Infer Topic
    infer_topic =
    df_topic_keywords.iloc[np.argmax(topic_probability_scores),␣
↪-1]

                                                        #topic_guess =
        df_topic_keywords.iloc[np.argmax(topic_probability_scores),␣
↪Topics] return infer_topic, topic,
    topic_probability_scores
# Predict the topic mytext = ["initial coronavirus study
promising we will now finish final␣
↪analysis"] infer_topic, topic, prob_scores =
predict_topic(text = mytext) print(infer_topic)
```

```
Research
```

0.1.4 Task 5 - What are the most common topics in each cluster?

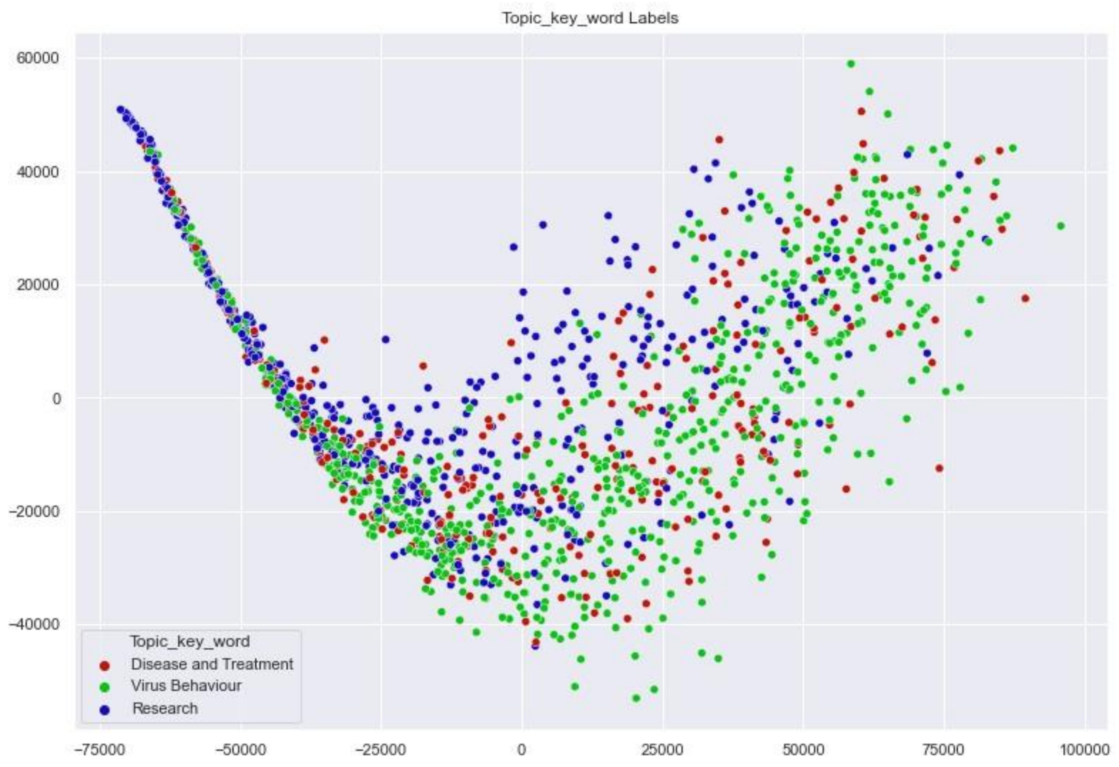
Information from the topics in task 4 is used to characterise the topics in each cluster and adding it to the dataframe.

```
[37]: def apply_predict_topic(text):
 text = [text]
 infer_topic, topic, prob_scores = predict_topic(text = text)
 return(infer_topic)
df["Topic_key_word"]= df['abstract'].apply(apply_predict_topic)
df.head()
```

```
[37]:                                                  abstract \
     0  OBJECTIVE: This retrospective chart review des…
     1  Inflammatory diseases of the respiratory tract…
     2  Surfactant protein-D (SP-D) participates in th…
     3  Endothelin-1 (ET-1) is a 21 amino acid peptide…
     4  Respiratory syncytial virus (RSV) and pneumoni…

                                          clean_abstract \
     0  objective retrospective chart review describes…
     1  inflammatory disease respiratory tract commonl…
     2  surfactant protein sp participates innate resp…
     3  endothelin amino acid peptide diverse biologic…
     4  respiratory syncytial virus rsv pneumonia viru…

                                                     array Clusters \
     0 [101, 7863, 15354, 3673, 3319, 5577, 4958, 517…  0
     1 [101, 20187, 4295, 16464, 12859, 4141, 3378, 8…  2
     2 [101, 14175, 18908, 4630, 5250, 11867, 17257, …  1
     3 [101, 2203, 14573, 18809, 13096, 5648, 25117, …  2
     4 [101, 16464, 26351, 22123, 4818, 7865, 12667, …  2
               Topic_key_word
     0       Disease and Treatment
     1       Virus Behaviour
     2       Virus Behaviour
     3       Virus Behaviour
     4       Virus Behaviour
```

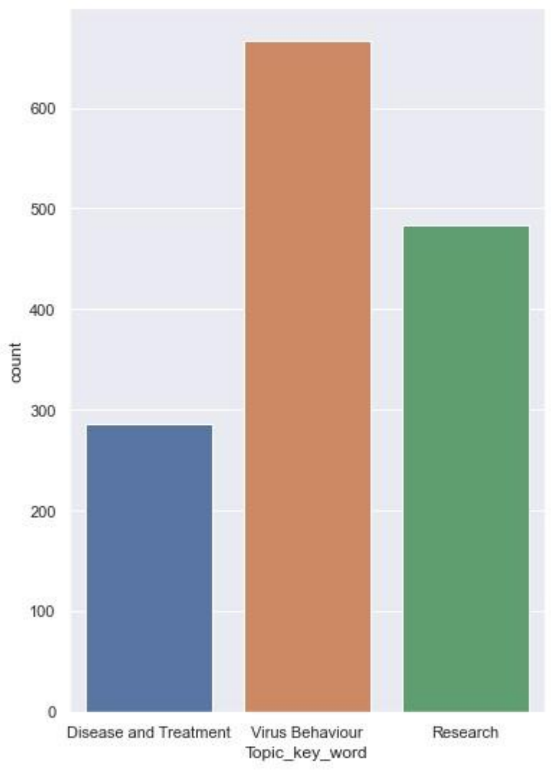
Visualization of top 3 common topics.

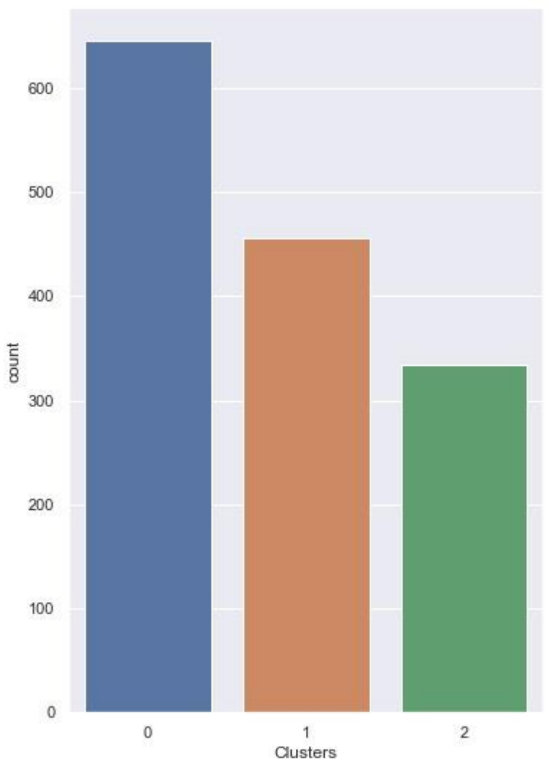
```
[38]: # sns settings
      sns.set(rc={'figure.figsize':(14,10)})

      # colors
      palette = sns.hls_palette(3,  =.4,  =.9)

      # plot
      sns.scatterplot(X_PCA[:,0], X_PCA[:,1], hue=df['Topic_key_word'],␣
       ↪legend='full', palette=palette)
      plt.title('Topic_key_word Labels')
      plt.savefig("clusterRevised.png")
      plt.show()
```



Comparison of Kmeans clusters and Topics from Lda

[39]:
```
fig, ax =plt.subplots(1,2)
sns.countplot(df['Topic_key_word'], ax=ax[0])
sns.countplot(df['Clusters'], ax=ax[1])
fig.show()
```


count
0
100
200
300
400
500
600
Disease and Treatment
Virus Behaviour
Research
Topic_key_word


count
0
100
200
300
400
500
600
0
1
2
Clusters