06_AM_NLP_Analysis

July 13, 2021

[1]:
```
import pandas as pd
import numpy as np
from gensim.models import word2vec
from sklearn.cluster import MiniBatchKMeans
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
import plotly.express as px
import plotly.offline as po
import matplotlib.pyplot as plt
```

1 NLP - Text Analytics

[2]:
```
df = pd.read_csv('../data/processed/vaccine_tweets_with_sentiment.csv',␣
↪index_col=0)
```

[3]:
```
df.head()
```

[3]:

	id	created_at	user	geo	full_text
0	1382113874439192576	2021-04-13 23:30:00+00:00	{'id': 758414676480946200, 'id_str': '75841467…	NaN	"Safe and Effective, Safe and Effective, Safe …
1	1362299993504145408	2021-02-18 07:16:43+00:00	{'id': 160763636, 'id_str': '160763636', 'name…	NaN	Hey dear friends my #COVAXIN is done and feel …
2	1375483502616010752	2021-03-26 16:23:16+00:00	{'id': 1356008062620991500, 'id_str': '1356008…	NaN	We will be giving thousands of doses of the in…
3	1367114490890752000	2021-03-03 14:07:49+00:00	{'id': 143025857, 'id_str': '143025857', 'name…	NaN	Better efficacy than Oxford/covishield's 62%. …
4	1395429383582724096	2021-05-20 17:21:05+00:00	{'id': 19386982, 'id_str': '19386982', 'name':…	NaN	Huge thank you to the wonderful @HSELive staff…

```
                                                hashtags              user_id  \
0  ['mrna', 'covidvaccine', 'johnsonandjohnson', …   758414676480946200
1           ['covaxin', 'covid19', 'vaccinemaitri']           160763636
2     ['oxfordastrazeneca', 'passover', 'covidjab']  1356008062620991500
3                                     ['covaxin']           143025857
4                                     ['moderna']            19386982

   PfizerBiontech  SputnikV  Sinopharm  Sinovac  Moderna  AstraZeneca  \
0               1         0          0        1        1            1
1               0         0          0        0        0            0
2               0         0          0        0        0            1
3               0         0          0        0        0            0
4               0         0          0        0        1            0

   Covaxin  JandJ user_location              coordinates  \
0        0      1           NaN                      NaN
1        1      0         India  [22.3511148, 78.6677428]
2        0      0           NaN                      NaN
3        1      0           NaN                      NaN
4        0      0           NaN                      NaN

                                              corpus  \
0  safe effective safe effective safe effective s…
1  hey dear friend covaxin done feel good thanks …
2  giving thousand dos incredibly safe powerfully…
3  better efficacy oxford covishield best inactiv…
4  huge thank wonderful staff race got cancellati…

                                           sentiment  sentiment_compound
0  {'neg': 0.0, 'neu': 0.091, 'pos': 0.909, 'comp…              0.9953
1  {'neg': 0.0, 'neu': 0.311, 'pos': 0.689, 'comp…              0.9876
2  {'neg': 0.0, 'neu': 0.282, 'pos': 0.718, 'comp…              0.9842
3  {'neg': 0.0, 'neu': 0.325, 'pos': 0.675, 'comp…              0.9837
4  {'neg': 0.0, 'neu': 0.295, 'pos': 0.705, 'comp…              0.9813
```

Drop empty corpus:

[4]:
```
df = df.dropna(subset=["corpus"]).reset_index(drop= True)
```

---

[5]:
```
import plotly.graph_objects as go
```

1.1 Most talked about vaccines in the discourse about delta variant

[6]:
```
df[df["corpus"].str.contains("delta") == True].sum()[7:-5].index
```

[6]:
```
Index(['PfizerBiontech', 'SputnikV', 'Sinopharm', 'Sinovac', 'Moderna',
       'AstraZeneca'],
      dtype='object')
```

[7]:
```
fig = go.Figure()
fig.add_trace(
            go.Pie(labels=df[df["corpus"].str.contains("delta") == True].
 ↪sum()[7:-5].index, values=df[df["corpus"].str.contains("delta") == True].
 ↪sum()[7:-5], title="Most talked about vaccines in discourse about Delta␣
 ↪Variant")
       )
```

The delta variant appeared first in India thus it makes sense that the most talked about vaccines is an indian one.

---

2 Needed Functions for text analysis

Tokenize Corpus: - split every tweet into its words - words are by definition letters or numbers surrounded by whitespace

[8]:
```
def tokenize_corpus(corpus):
    token_list = []
    for i in range(len(corpus)):
        corpus[i] = str(corpus[i])
        token_list.append(corpus[i].split())
    return token_list
```

Elbow method to get an estimation of the sse depenending on clusters: - The point where there isn't a significant decrease in SSE is the suggested cluster - If there isn't an elbow (e.g. straight line), then the dataset/method doesn't provide a good enough basis for clustering

[9]:
```
def elbow_method(tokens, k_cluster, title):

    #sum of squared errors
    sse = []
    for k in range(2, k_cluster+1):
        sse.append(MiniBatchKMeans(n_clusters=k, init_size=1024,␣
 ↪batch_size=2048, random_state=20).fit(tokens).inertia_)

    #plot curve:
    f, ax = plt.subplots(1, 1)
    ax.plot(range(2, k_cluster+1), sse, marker='o')
    ax.set_xlabel('Clusters')
    ax.set_xticks(range(2, k_cluster+1))
    ax.set_xticklabels(range(2, k_cluster+1))
    ax.set_ylabel('SSE')
```

```
    ax.set_title('SSE by Cluster amount for ' + title)
```

Runs K-Means clustering algorithm for a set of tokens with a given cluster number (usually the suggested from elbow method):

[10]:
```
def model_kmeans(tokens, n_cluster):

    kmeans = KMeans(n_clusters = n_cluster, init = 'k-means++', random_state =␣
↪42)
    y_kmeans = kmeans.fit_predict(tokens)

    return y_kmeans
```

Print the relevant words for each cluster (Doc2vec):

[11]:
```
def cluster_keywords_average(model, tokens, y_kmeans, tags):
    matrix = sparse.csr_matrix(tokens)
    df = pd.DataFrame(matrix.todense()).groupby(y_kmeans).mean()

    for i in range(len(df)):
        wordlist = {}
        mean_document = model.docvecs.most_similar(positive = [np.
↪asarray(list(df.iloc[i]))], topn=1)[0][0]

        for word in set(tags[int(mean_document)-1][0]):
            wordlist[word]=tags[int(mean_document)-1][0].count(word)

        sorted(wordlist.values(), reverse=True)[:5]
        sort_orders = sorted(wordlist.items(), key=lambda x: x[1], reverse=True)

        print(f"Cluster {i}: ", sort_orders[:5])
```

[12]:
```
def cluster_keywords_total(tags, y_kmeans):
    taglist = [[] for i in set(y_kmeans)]

    for i in set(y_kmeans):
        #for every cluster
        for j in range(len(tags)):
            #check if a document belongs to that cluster
            if y_kmeans[j] == i:
                #if document is part of cluster, add Words to vocabulary of␣
↪that Cluster
                for word in tags[j][0]:
                    taglist[i].append(word)

    for i in set(y_kmeans):
        #for every cluster
        wordlist = {}
```

```
        for word in set(taglist[i]):
            #set() turns the Wordlist into a unique vocabulary and counts the␣
↪term frequency
            wordlist[word] = taglist[i].count(word)

        sort_orders = sorted(wordlist.items(), key=lambda x: x[1], reverse=True)

        print(f'Cluster {i}: ', end = ""), print(', '.join(r[0] for r in␣
↪sort_orders[:6]))
```

Prints the relevant words for each cluster(TF-IDF):

[13]:
```
def cluster_keywords_tfidf(model, data, clusters, n_terms):
    labels = model.get_feature_names()
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()

    for i,r in df.iterrows():
        print('Cluster {}: '.format(i), end = ""), print(', '.join([labels[t]␣
↪for t in np.argsort(r)[-n_terms:]]))
```

Model t-Sne:

[14]:
```
def model_tsne(tokens):

    tsne_model = TSNE(perplexity=50, n_components=2, init='pca', n_iter=5000,␣
↪random_state=38)
    new_values = tsne_model.fit_transform(tokens)

    return new_values
```

Data tagging: - Needed for Doc2Vec - Maps an index/tag to every tweet corpus

[15]:
```
def data_tagging(data, corpus):
    token = tokenize_corpus(corpus)
    tagged_data = []

    for i in range(len(data)):
        tagged_data.append(
            TaggedDocument(words=token[i], tags=[str(data.index[i])]))

    return tagged_data
```

Visualize results:

[16]:
```
def plot_2d(values, y_kmeans, labels, title):
    labelColorMap = {
        0: "cluster 0",
        1: "cluster 1",
```

```
        2: "cluster 2",
        3: "cluster 3",
        4: "cluster 4",
        5: "cluster 5",
        6: "cluster 6",
        7: "cluster 7",
        8: "cluster 8",
        9: "cluster 9",
        10: "cluster 10",
        11: "cluster 11",
        12: "cluster 12",
        13: "cluster 13",
        14: "cluster 14",
        15: "cluster 15",
        16: "cluster 16",
        17: "cluster 17",
        18: "cluster 18",
        19: "cluster 19"}

    labelColor = [labelColorMap[i] for i in y_kmeans]

    fig = px.scatter(
        values, x=0, y=1,
        hover_name = labels,
        color= labelColor,
        title= "Embedding w/ " + title)

    #po.plot(fig, filename="../reports/figures/VaccineText/"+title.replace(" ",␣
↪"_")+'.html')
    fig.write_html("../reports/figures/VaccineText/"+title.replace(" ","_")+".
↪html")

    fig.show()
```

---

3 Word2Vec

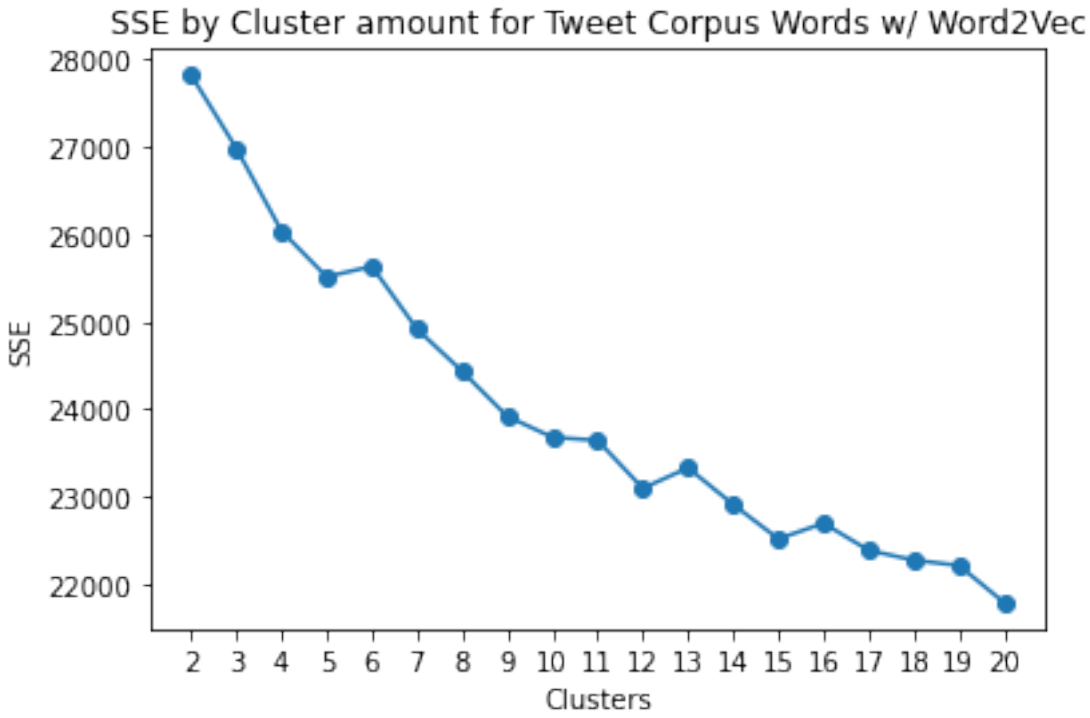
in short: Word2Vec creates a unique vector for every word. The interesting part about this is that the vectors reflect the context of a word. The most common example is that the wordvector of the word “King” minus the vector for “Man” plus the vector for the word “Female” leads to a vector that is close to the Word “Queen”.

[17]:
```
def word2vec_model(corpus, algorithm):

    #create tokens
    corpus_token = tokenize_corpus(corpus)
```

```
    #Initiate model
    model = word2vec.Word2Vec(corpus_token, size = 100, window=10, min_count =
↪35, workers = 4, sg = algorithm)

    labels = []
    tokens = []


    for word in model.wv.vocab:
        tokens.append(model.wv[word])
        labels.append(word)

    return model, labels, tokens
```

[18]:
```
df_model, df_labels, df_tokens = word2vec_model(df.corpus[:],1)
```

[19]:
```
elbow_method(df_tokens, 20, "Tweet Corpus Words w/ Word2Vec")
```



Elbow method doesn't yield an optimal result. That means that the data doesn't provide a good basis for clustering.

Testing out 8 Clusters:

[20]:
```
df_y = model_kmeans(df_tokens, 8)
```

Running dimensionality reduction:

[21]:
```
df_tsne= model_tsne(df_tokens)
```

Visualize Word2Vec results:

[22]:
```
plot_2d(df_tsne, df_y, df_labels, "Vaccine Tweet Corpus")
```

[23]:
```
df_model.wv.most_similar("delta")
```

[23]:
```
[('deltavariant', 0.8551425933837891),
 ('alpha', 0.8102719783782959),
 ('variant', 0.7920688390731812),
 ('beta', 0.7807437181472778),
 ('alphavariant', 0.7415281534194946),
 ('deltaplus', 0.739632248878479),
 ('neutralize', 0.7220360636711121),
 ('detected', 0.7157915234565735),
 ('effectively', 0.7102565765380859),
 ('neutralises', 0.7101479768753052)]
```

[25]:
```
df_model.wv.most_similar("billgates")
```

[25]:
```
[('greatreset', 0.8271923065185547),
 ('antivaxxers', 0.7966406941413879),
 ('genocide', 0.795373797416687),
 ('plandemic', 0.771074116230011),
 ('drfauci', 0.7487349510192871),
 ('vaccinepassports', 0.7146542072296143),
 ('vaccinatie', 0.6953786611557007),
 ('maga', 0.6903533339500427),
 ('merck', 0.6849798560142517),
 ('criminal', 0.6788644790649414)]
```

[26]:
```
df_model.wv.most_similar(negative=["drfauci","bad"])
```

[26]:
```
[('ensuring', 0.07949046790599823),
 ('rajiv', 0.04241025447845459),
 ('gmp', 0.01358264684677124),
 ('consignment', 0.00033785775303840637),
 ('international', -0.006062516942620277),
 ('initiated', -0.006336908787488937),
 ('bengaluru', -0.01014620065689087),
 ('capacity', -0.010990476235747337),
 ('registration', -0.011032816022634506),
 ('bangalore', -0.013618746772408485)]
```

[27]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

4 TF-IDF

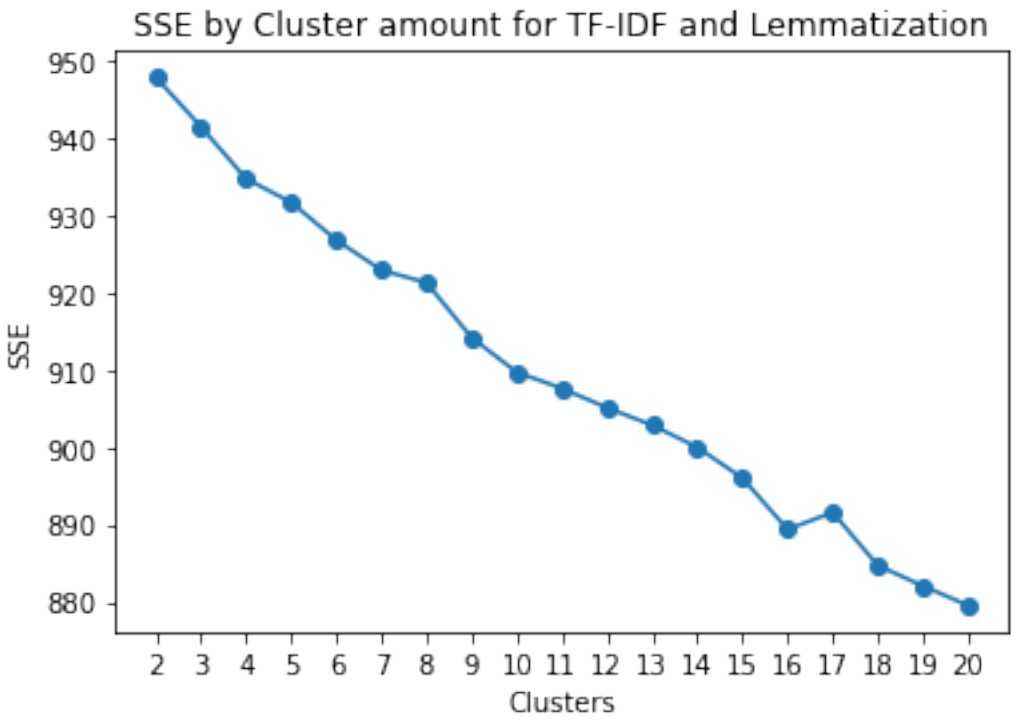
Term frequency - inverse document frequency: - yields to preserve high information value within a text - That means: - Words that appear in many tweets are likely irrelevant (think stop-words or “vaccine”) - Words that appear only in a few are also likely to be irrelevant (think names/misspellings)

[28]:
```
def tfidf_model(corpus, data):

    tfidf = TfidfVectorizer(min_df = 5, max_df = 0.95, max_features = 10000,␣
↪stop_words = 'english').fit(corpus)
    text_matrix = tfidf.transform(corpus)

    labels = data.index.tolist()

    return tfidf, text_matrix, labels
```

[29]:
```
tfidf_model_fitted, df_data, df_labels = tfidf_model(df["corpus"][:1000], df[:
↪1000])
```

Elbow method:

[30]:
```
elbow_method(df_data, 20, "TF-IDF and Lemmatization")
```



The elbow method yielded poor results for the sample size.

Clustering with 9 clusters:

[31]:
```
df_y = model_kmeans(df_data, 9)
```

Keyword extraction:

[32]:
```
print("TF-IDF keywords: ")
cluster_keywords_tfidf(tfidf_model_fitted, df_data, df_y, 9)
```

```
TF-IDF keywords:
Cluster 0: free, dose, covid, today, got, fully, moderna, thanks, vaccinated
Cluster 1: dos, people, vaccination, vaccinated, hope, covid, sinopharm, china,
sinovac
Cluster 2: centre, jab, vaccination, staff, volunteer, friendly, super,
efficient, oxfordastrazeneca
Cluster 3: getting, covid, science, got, love, best, shot, moderna, thank
Cluster 4: got, great, thank, covid, dose, amazing, today, grateful,
pfizerbiontech
Cluster 5: stronger, uk, coronavirus, sputnikv, covid, covaxin, variant, safe,
effective
Cluster 6: sputnik, india, win, great, best, vaccine, russian, russia, sputnikv
Cluster 7: second, like, great, good, pfizer, dose, covid, feeling, moderna
Cluster 8: good, indian, covishield, best, great, pm, covid, india, covaxin
```

Dimensionality reduction:

[33]:
```
tfidf_tsne = model_tsne(df_data.todense())
```

Plot:

[34]:
```
plot_2d(tfidf_tsne, df_y, df_labels, "TF-IDF and Lemmatization")
```

Note: Elbow method doesn't show a perfect elbow, that means that kmeans wasn't able to detect good enough clusters.

---

5 Doc2Vec

Doc2Vec works similarily to Word2Vec, only difference is that it takes in whole documents/text instead of single words. It creates a document vector which can be used to identify similar documents. This could be useful to identify similar tweets.

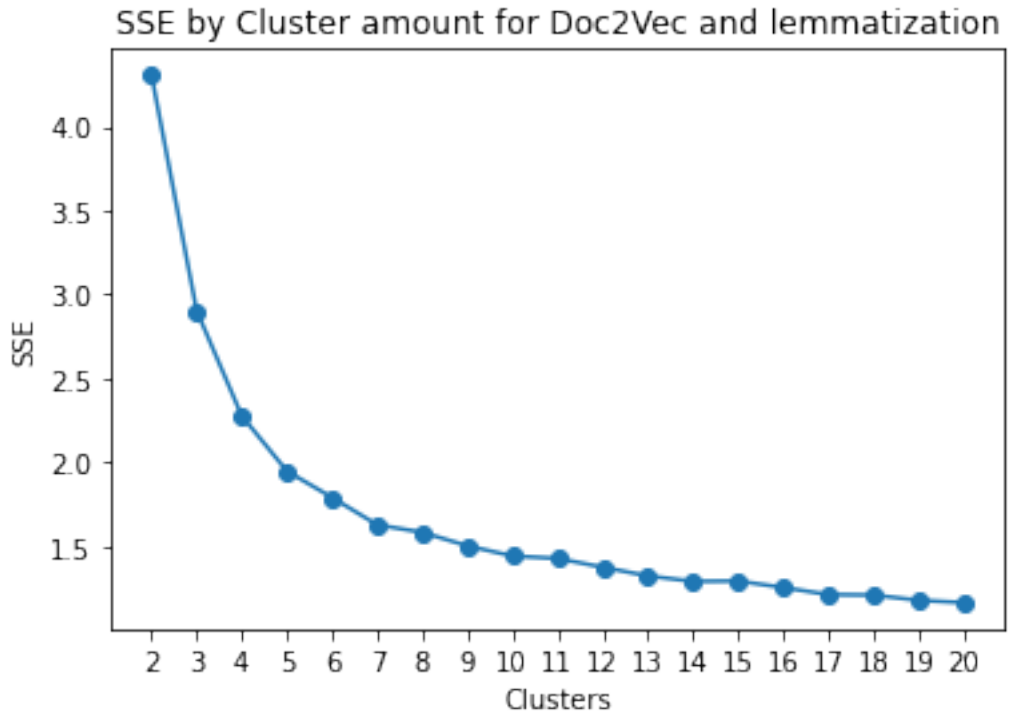
[35]:
```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
from scipy import sparse
```

[36]:
```
def doc2vec_model(tagged_data, max_epochs, data):

    model = Doc2Vec(vector_size=100, min_count=35, workers = 6, dm = 1)#The␣
 ↪Algorithms here are called "distributed memory" and "distributed bag of␣
 ↪words"

    model.build_vocab(tagged_data)

#    for epoch in range(max_epochs):
    model.train(tagged_data, total_examples=model.corpus_count,␣
 ↪epochs=max_epochs)
    model.alpha -= 0.0002
    model.min_alpha = model.alpha

    tokens = [model.docvecs[i] for i in range(len(model.docvecs))]
    labels = [data.index[i] for i in range(len(model.docvecs))]


    #model.save("../models/doc2vec")
    return model, tokens, labels
```

[37]:
```
doc2vec_model, df_tokens, df_labels = doc2vec_model(data_tagging(df[:1000],␣
 ↪df["corpus"][:1000]), 30, df)
```

Elbow method:

[38]:
```
elbow_method(df_tokens, 20, "Doc2Vec and lemmatization")
```



The elbow method looks ideal. The suggested cluster should be around 6.

Clustering with 7 clusters:

[39]:
```
doc2vec_y = model_kmeans(df_tokens, 7)
```

Keywords by cluster:

[40]:
```
print("Keywords (averaged): ")
cluster_keywords_average(doc2vec_model, df_tokens, doc2vec_y, data_tagging(df[:
↪1000], df["corpus"][:1000]))
```

```
Keywords (averaged):
Cluster 0:  [('fightcorona', 1), ('today', 1), ('rosamma', 1), ('hospital', 1),
('honourable', 1)]
Cluster 1:  [('tackle', 1), ('problem', 1), ('feel', 1), ('grateful', 1),
('make', 1)]
Cluster 2:  [('fightcorona', 1), ('today', 1), ('rosamma', 1), ('hospital', 1),
('honourable', 1)]
Cluster 3:  [('fightcorona', 1), ('today', 1), ('rosamma', 1), ('hospital', 1),
('honourable', 1)]
Cluster 4:  [('thank', 2), ('spring', 1), ('moderna', 1), ('optimism', 1),
('family', 1)]
```

```
Cluster 5:  [('fightcorona', 1), ('today', 1), ('rosamma', 1), ('hospital', 1),
('honourable', 1)]
Cluster 6:  [('fightcorona', 1), ('today', 1), ('rosamma', 1), ('hospital', 1),
('honourable', 1)]
```

[41]:
```
print("Keywords for lemmatized corpus: ")
cluster_keywords_total(data_tagging(df[:100], df.corpus[:100]), doc2vec_y)
```

```
Keywords for lemmatized corpus:
Cluster 0: moderna, happy, thankful, hope, great, thank
Cluster 1: thank, moderna, effective, shot, grateful, great
Cluster 2: moderna, great, news, thanks, get, pfizerbiontech
Cluster 3: acceptance, atmanirbharbharat, public, towards, truly, covaxin
Cluster 4: safe, effective, got, friend, today, thank
Cluster 5: covaxin, work, life, moderna, congratulation, hope
Cluster 6: great, thank, moderna, covid, best, people
```

Dimensionality Reduction:

[42]:
```
doc2vec_tsne = model_tsne(df_tokens)
```

Results of sample size in Doc2Vec:

[43]:
```
plot_2d(doc2vec_tsne, doc2vec_y, df_labels, "Doc2Vec and lemmatization")
```

[44]:
```
doc2vec_model.docvecs.most_similar("12")
```

[44]:
```
[('665', 0.9911946058273315),
 ('32', 0.9911589026451111),
 ('499', 0.9905409216880798),
 ('160', 0.9904836416244507),
 ('855', 0.9904360771179199),
 ('378', 0.9903075695037842),
 ('149', 0.9903039932250977),
 ('211', 0.9900524616241455),
 ('701', 0.989974319934845),
 ('448', 0.9899573922157288)]
```

We ran both Doc2Vec with tf-idf in a python script and exportet the results. The models are under `./models/`

6 Doc2Vec (pre-computed)

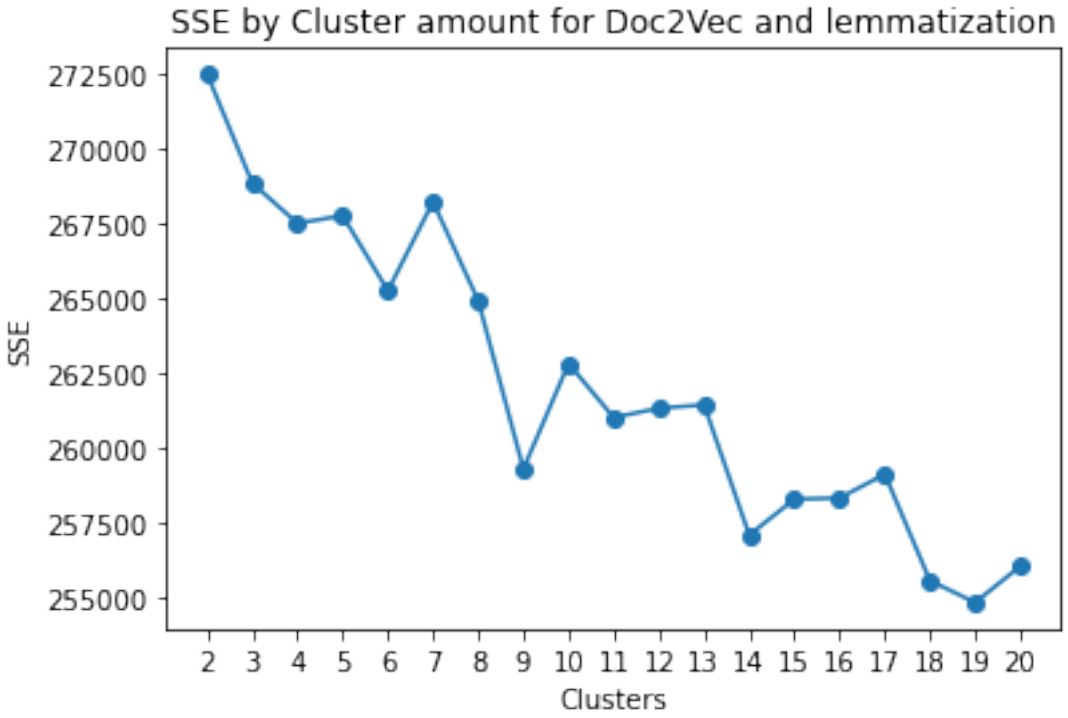
Import model:

[45]:
```
doc2vec_model = Doc2Vec.load("../models/Doc2Vec_model/doc2vec")
df_tokens = [doc2vec_model.docvecs[i] for i in range(len(doc2vec_model.
 ↪docvecs))]
```

```
df_labels = [df.index[i] for i in range(len(doc2vec_model.docvecs))]
doc2vec_tsne = np.load("../models/Doc2Vec_model/doc2vec_tsne.npy")
```

Elbow method:

[46]:
```
elbow_method(df_tokens, 20, "Doc2Vec and lemmatization")
```



The results are rather suboptimal. That likely means that the data doesn't yield good clusters.

Clustering with 9 clusters:

[47]:
```
doc2vec_y = model_kmeans(df_tokens, 9)
```

[48]:
```
df = df.dropna(subset=["corpus"]).reset_index(drop=True)
df = df.dropna(subset=["user_location"]).reset_index(drop=True)
```

Keywords by cluster:

[49]:
```
print("Keywords (averaged): ")

cluster_keywords_average(doc2vec_model, df_tokens, doc2vec_y, data_tagging(df[:
↪], df["corpus"][:]))
```

```
Keywords (averaged):
Cluster 0:  [('efficacy', 2), ('covaxin', 2), ('great', 1), ('domestic', 1),
('released', 1)]
```

```
Cluster 1:  [('delhi', 2), ('modi', 2), ('aiims', 2), ('narendramodi', 1),
('bharatbiotech', 1)]
Cluster 2:  [('moderna', 1), ('vaccinated', 1), ('fully', 1)]
Cluster 3:  [('st', 1), ('status', 1), ('dose', 1), ('city', 1), ('covaxin', 1)]
Cluster 4:  [('ocgn', 2), ('bharatbiotech', 1), ('vaccine', 1), ('roadmaps', 1),
('committee', 1)]
Cluster 5:  [('dose', 4), ('slot', 2), ('free', 1), ('availability', 1),
('covaxin', 1)]
Cluster 6:  [('cr', 2), ('get', 1), ('afghanistan', 1), ('asia', 1), ('dos', 1)]
Cluster 7:  [('following', 2), ('india', 2), ('event', 2), ('report', 1),
('ruchibhatia', 1)]
Cluster 8:  [('researcher', 1), ('vaccine', 1), ('illegal', 1), ('moderna', 1),
('cybersecurity', 1)]
```

Visualize results:

[50]:
```
plot_2d(doc2vec_tsne, doc2vec_y, df_labels, "Doc2Vec and lemmatization")
```

[51]:
```
df.full_text[30000]
```

[51]:
```
'@CPHO_Canada Who is eating alone today because Doug Ford and Trudeau Toronto
Ontario Canada forced you to stay home #lockdown no family, no business, no
church, no hope, no life #MentalHealthMatters Order Russia #SputnikV Get out of
Syria #COVID19 #spring #Cuba https://t.co/DSHOzCyIta https://t.co/A8X5yEHBZf'
```

[52]:
```
new_vector = doc2vec_model.infer_vector(["30000"])
sims = doc2vec_model.docvecs.most_similar([new_vector])
```

[53]:
```
sims
```

[53]:
```
[('18445', 0.36457157135009766),
 ('29381', 0.3368401527404785),
 ('4624', 0.33177727460861206),
 ('25312', 0.32824602723121643),
 ('1458', 0.326931357383728),
 ('13176', 0.32390666007995605),
 ('2421', 0.32356274127960205),
 ('25565', 0.3228134512901306),
 ('7688', 0.3217652142047882),
 ('17665', 0.3209880292415619)]
```

[54]:
```
df.full_text[24458]
```

[54]:
```
'@SuryaHospitals Mumbai is giving #Covaxin to selected ones!!\nDenied to give me
2nd dose of on the ground that I had been vaccinated earlier in New Delhi so I
had to go from Mumbai to New Delhi for the purpose?\nShould I take flight in
this covid-19 era from Mumbai to Delhi ?? https://t.co/gQqU6OBJKu'
```

these tweets have a similarity of 35%, which explains why they don't seem to be related at all.

---

7 tf-idf (pre-computed)

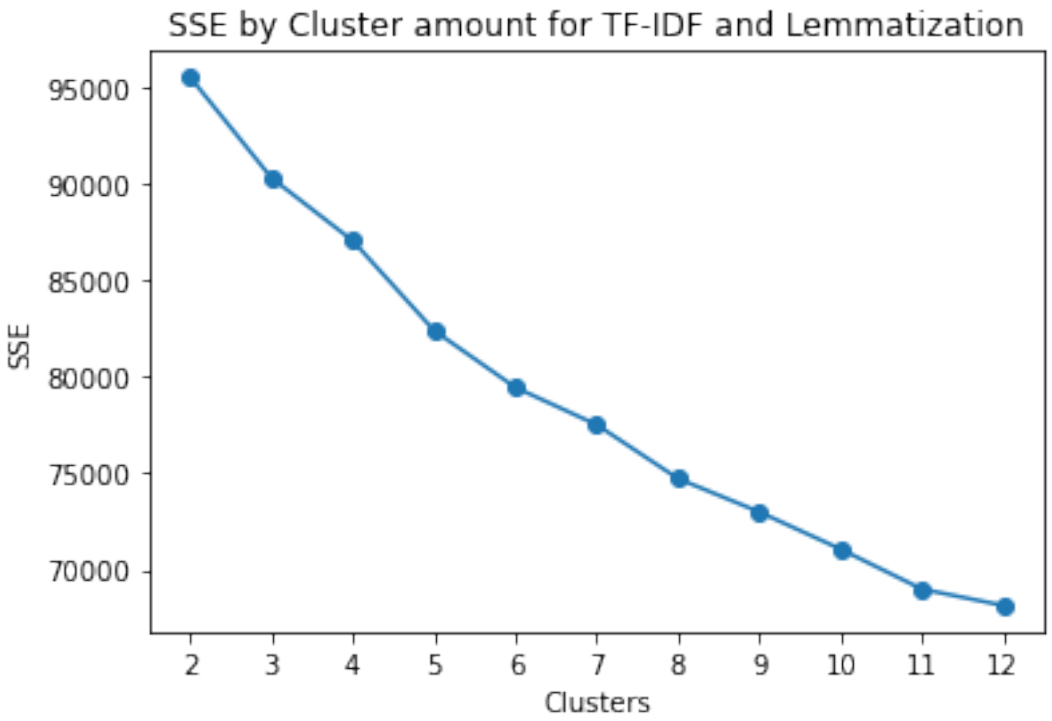
Import model:

[55]:
```
import pickle
```

[56]:
```
tfidf_model_fitted = pickle.load(open('../models/tf-idf_model/tfidf_model.
↪pickle','rb'))
tfidf_matrix = pickle.load(open('../models/tf-idf_model/tfidf_matrix.pickle',␣
↪'rb'))
tfidf_labels = pickle.load(open('../models/tf-idf_model/tfidf_labels.
↪pickle','rb'))
tfidf_tsne = np.load('../models/tf-idf_model/tfidf_tsne.npy')
```

Elbow method:

[57]:
```
elbow_method(tfidf_matrix, 12, "TF-IDF and Lemmatization")
```



Elbow method doesn't yield definit results which leads to the assumption that the datasat might not offer good clusters.

Clustering with 8 clusters:

[58]:
```
df_y = model_kmeans(tfidf_matrix, 8)
```

Extracting keywords for each cluster:

[59]:
```
print("TF-IDF keywords: ")
cluster_keywords_tfidf(tfidf_model_fitted, tfidf_matrix, df_y, 9)
```

```
TF-IDF keywords:
Cluster 0: dos, use, vaccination, sputnikv, covishield, vaccine, covid, covaxin,
india
Cluster 1: trial, vaccinated, hospital, vaccination, covishield, covid, vaccine,
dose, covaxin
Cluster 2: today, second, vaccinated, dose, got, shot, covid, vaccine, moderna
Cluster 3: vaccinated, coronavirus, vaccination, covidvaccine, astrazeneca,
covid, vaccine, moderna, pfizer
Cluster 4: coronavirus, country, case, canada, sputnik, covid, vaccine, russia,
sputnikv
Cluster 5: astrazeneca, pfizer, vaccination, dose, covidvaccine, covid, vaccine,
oxfordastrazeneca, pfizerbiontech
Cluster 6: vaccination, dose, vaccinated, dos, covid, china, vaccine, sinopharm,
sinovac
Cluster 7: india, vaccination, health, dose, hospital, covaxin, age, bbmp, slot
```

Visualizing results:

[60]:
```
plot_2d(tfidf_tsne, df_y, tfidf_labels, "TF-IDF and Lemmatization")
```

TF-IDF yields a better visualization than doc2vec. However it isn't perfect yet. To further explore the text of the dataset, we suggest to try out different hyper paramters for tuning the vectorization and dimensionality reduction