

Analisi di una collezione di Tweet

```
In [4]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import ast
```

```
In [5]: data = pd.read_csv(r'data.csv', parse_dates=['created_at'])
df = pd.DataFrame(data)
```

Decoding del testo all'interno della colonna text .

All'interno della colonna text è presente una rappresentazione di un bytes sotto forma di stringa. Questa può essere valutata attraverso la funzione `ast.literal_eval(string)` e quindi decodificata secondo la codifica appropriata tramite il metodo `decode` .

```
In [6]: text_enc = []
for i in range(0, df.text.size):
    txt= ast.literal_eval(df.text[i]).decode('utf-8')
    text_enc.append(txt)

df['text_enc'] = text_enc
df
```

	favorite_count	source	text	is_retweet	created_at	retweet_count	text_en
0	1	Twitter for Android	b'La #monarqu\xc3\xada se burla a diario de lo...	False	2021-03-03 07:46:48+00:00	0	La #monarqu\xc3\xada se burla a diario de lo esp\u00e1nol.
1	0	Twitter for iPhone	b'@TitusNation the CON serving CONservative GO...	False	2021-03-03 17:22:21+00:00	0	@TitusNation thi CON serving CONservative GOP .
2	0	Twitter for Android	b'#Coronavirus\nAustria and Denmark are to sto...	False	2021-03-03 08:47:26+00:00	0	#Coronavirus\nAustri and Denmark are to stop .
3	0	Twitter Web App	b'BMC launches \xe2\x80\x98Mission Zero\xe2\x80...	False	2021-03-01 12:23:57+00:00	0	BMC launc 'Mission Zero' in Mumbai again to.
4	0	Twitter for iPhone	b'@Gadaffyduck @IsabelOakeshott @AmandaHolden ...	False	2021-03-04 20:49:11+00:00	0	@Gadaffyduck @IsabelOakeshot @AmandaHolden Du.
...
4163	4	Tweetbot for Mac	b'In other news, I have been jabbed with the f...	False	2021-03-02 19:18:52+00:00	0	In other news, I hav been jabbed with thi fir.
4164	5	Twitter Web App	b"So nearly 10% of today's new cases are the n...	False	2021-03-04 22:56:28+00:00	4	So nearly 10% o today's new cases ar the new.
4165	2	Twitter for Android	b"Read this\n\nThen read it again\n\nWritten, ...	False	2021-03-04 07:46:47+00:00	1	Read this\n\nThei read i again\n\nWritten, pr.
4166	8	Twitter Web App	b"\xe2\x80\xbc\xef\xbf\xbd\x8fUpdate\xe2\x80\xbc\x...	False	2021-03-04 12:39:09+00:00	4	!!Update!! Individual who file an IRS Forn 1.
4167	0	Buffer	b"As part NJ's efforts to expand access to men...	False	2021-03-01 13:26:15+00:00	1	As part NJ's efforts t expand access to menta.

4168 rows × 7 columns

Pulizia dei Tweet

Attraverso le Regular Expressions vengono rimossi elementi come:

- @menzioni
- Hashtag

- Link

Inoltre viene effettuata una pulizia del testo per accorpare termini simili tra loro

```
In [7]: import re

def cleanTxt(text):
    text=text.lower()
    text = re.sub('S*trumpS*', 'trump', text)
    text = re.sub('S*bidenS*', 'biden', text)
    text = re.sub('S*clintonS*', 'clinton', text)
    text = re.sub('@S+', '', text) #Rimuove le @menzioni
    text = re.sub('#', '', text) # Rimuove l'hashtag
    text = re.sub('https?:\/\/\S+', '', text) # Rimuove i Link
    text = re.sub('&', '', text) #Rimuove &
    text = re.sub('covid-19', 'covid', text ) #converte covid-19 in covid
    text = re.sub('covid19', 'covid', text ) #converte covid19 in covid
    text = re.sub('covid 19', 'covid', text ) #converte covid 19 in covid
    text = re.sub('coronavirus', 'covid', text)
    return text

def countMentions(text):
    return len(re.findall('@S+', text))

def countHashtags(text):
    return len(re.findall('#', text))

def countLinks(text):
    return len(re.findall('https?:\/\/\S+', text))

def extractHashtags(text):
    return

df.text_enc=df.text_enc.astype('str')
df['text_clean'] = df.text_enc.apply(cleanTxt)
df['mentions'] = df.text_enc.apply(countMentions)
df['hashtags'] = df.text_enc.apply(countHashtags)
df['links'] = df.text_enc.apply(countLinks)
```

Language Detection

il metodo `detect_langs` fornisce un vettore di probabilità riguardo la lingua del testo che sta analizzando. Vado a vedere se ci sono nel dataset testi ambigui (quindi quelli per cui la dimensione del dict ritornato da `detect_langs` è maggiore di 1).

Utilizzo `langdetect` che è abbastanza veloce per generare una previsione sommaria. Esso restituisce la probabilità per ciascuna lingua trovata quindi:

- se trovo una sola lingua tra quelle ammesse (en, es, fr) allora segno la lingua trovata e segno la detection come sicura
- se trovo più di una lingua oppure trovo come lingua più probabile una non ammessa segno la lingua e segno la detection come insicura/sbagliata (`unsure_wrong_detection=True`)

```
In [ ]: from langdetect import detect_langs

unsure_count = 0
wrong_count = 0
i=0
lang_detect = []
unsure_wrong = []

for twt in df.text_enc:
    try:
        detection = detect_langs(twt)
        lang = detection[0].lang

        lang_detect.append(lang)
        unsure_wrong.append(False)

        if((lang != "en") and (lang != "es") and (lang != "fr")):
            wrong_count = wrong_count +1
            unsure_wrong[i]=True
        if(len(detection)>1):
            unsure_wrong[i]=True
            unsure_count = unsure_count+1

    except:
        print("errore alla posizione ")
        i
        print("testo : " + twt)
        lang_detect.append('None')
        unsure_wrong.append(True)
```

```
i=i+1

df['langdetect']=lang_detect
df['unsure_wrong_detection'] = unsure_wrong

unsure_wrong_sum = df['unsure_wrong_detection'].sum()
unsure_wrong_sum
```

Check traduzione attraverso API Google

I tweet dei quali la traduzione non è sicura sono stati marcati con un `True` all'interno della colonna `unsure_wrong_detection`. Utilizzo questa informazione per passare questi tweet all'interno dell'API Google per la traduzione. Questo metodo non è stato utilizzato prima in quanto le possibili chiamate ai server Google sono limitate e molto lente.

```
In [ ]: from textblob import TextBlob

lang_detect_final = []
i = 0
for twt in df.text_clean:
    if not df['unsure_wrong_detection'][i]:
        lang_detect_final.append(df['langdetect'][i])
    else:
        blob = TextBlob(twt)
        lang = blob.detect_language()
        if((lang != "en") and (lang != "es") and (lang != "fr")):
            lang_detect_final.append('None')
        else:
            lang_detect_final.append(lang)
        df['unsure_wrong_detection'][i]=False
    i=i+1

df['lang_detect_final'] = lang_detect_final
```

```
In [10]: #numero totale di tweet di cui non è stata identificata La Lingua
df.unsure_wrong_detection.sum()
```

```
Out[10]: 34
```

Esportazione dei dati su formato Excel

Excel non supporta le date con le timezone quindi le ho dovute eliminare con il metodo `datetime.tz_localize()`

```
In [ ]: df_enc = df[['favorite_count', 'source', 'text_enc', 'text_clean', 'is_retweet', 'retweet_count', 'created_at', 'langdetect']]
df_enc.loc[:, 'created_at_ntz'] = df_enc.created_at.dt.tz_localize(None)

df_enc = df_enc.drop(columns='created_at')
df_enc.to_excel('data_dec.xlsx')
```

Inizio dell'analisi

Dopo aver salvato i dati ripuliti all'interno del file Excel, uso questo come checkpoint per partire con l'analisi. Questo perché non è sempre possibile performare di nuovo la traduzione a causa delle restrizioni dell'api di Google descritte sopra

Dal momento che i Tweet la cui lingua non è stata riconosciuta sono meno dell' 1%, questi vengono filtrati. Dopodiché il dataset viene splittato per lingua in tre `DataFrame`

```
In [11]: data=pd.read_excel(r'data_dec.xlsx')
df=pd.DataFrame(data)

df = df.drop(columns='langdetect')
df.loc[df['unsure_wrong_detection']==True].shape #controllo dei tweet la cui Lingua non è stata riconosciuta
df = df.loc[df['unsure_wrong_detection']==False] #filtraggio dei tweet non riconosciuti

#en
df_en = df.loc[df['lang_detect_final']=='en']
df_en = df_en.drop(columns=['lang_detect_final', 'unsure_wrong_detection'])
```

Preprocessing del testo

Attraverso la libreria `nltk` viene effettuato il preprocessing del testo. Ogni tweet viene tokenizzato in parole che vengono filtrate dalle stopwords e stemmate attraverso il `PorterStemmer`. I tweet, una volta stemmati vengono salvati nella colonna `text_stem` e quindi per comodità de-tokenizzati.

```
In [12]: from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))
porter = PorterStemmer()

df_en['text_clean']=df_en['text_clean'].astype('str')

text_stem = []
#df_en['text_clean']=df_en['text_clean'].astype('str')

for twt in df_en.text_clean:
    tweet = twt.lower()
    token_words = word_tokenize(tweet) #tokenizzazione
    token_filter_words = [w for w in token_words if not w in stop_words] #stopword filtering
    stem_sentence=[]
    for word in token_filter_words:
        stem_sentence.append(porter.stem(word)) #stemming
    stem_sentence.append(' ')
    text_stem.append(''.join(stem_sentence))

df_en['text_stem']=text_stem
```

Tf-Idf

A questo punto viene performata la Tf-Idf sui tweet stemmati al punto precedenti.

La Tf-idf viene effettuata utilizzando il parametro `min_df` che filtra le parole in base alla loro frequenza nel dataset: se `min_df=0.005` allora verranno filtrate tutte le parole che compaiono in meno del 0,5% dei tweet. Questo permette di ridurre drasticamente il numero di feature e l'analisi verrà effettuata utilizzando varie percentuali minime. Il parametro `max_df` è l'analogico per la frequenza massima.

Per comodità di visualizzazione viene creato un dataframe per visualizzare la matrice tf-idf e generare una classifica delle parole: lo score di ciascuna parola è la media del tf-idf della parola in tutti i tweet. Vengono visualizzati i termini con score maggiore.

```
In [13]: from sklearn.feature_extraction.text import TfidfVectorizer

def tfidf(data, mindf, lan='english'):

    tfidfVectorizer = TfidfVectorizer(min_df=mindf,
                                      stop_words=lan)
    tfidf_matrix = tfidfVectorizer.fit_transform(data['text_stem'])

    print("sono state prodotte", len(tfidfVectorizer.get_feature_names()), "parole nel processo di tf-idf con mindf=", mindf)
    ## Conversione TfIdf in Dataframe ##

    # tfidfVectorizer ritorna una matrice sparsa che non permette la visualizzazione agevole della matrice TfIdf
    tf_idf = pd.DataFrame(columns=tfidfVectorizer.get_feature_names(), index=data['text_clean'], dtype=float)

    M=tfidf_matrix.todense()
    for i in range(0, data.shape[0]):
        tf_idf.iloc[i, :]=M[i,:]

    ## classifica delle parole
    #score={}
    #for word in tfidfVectorizer.get_feature_names():
    #    score[word]= tf_idf_en[word].mean()

    #score_sroted = dict(sorted(score.items(), key=lambda item: item[1], reverse=True))
    #print("Score delle prime 10 parole: ", list(score_sroted.items())[0:10])

    return tf_idf, M
```

Scelta del numero di feature

Prosegua l'analisi con una mindf di 0.001

```
In [14]: tf_idf_en_01, tfidf_en_matrix_01 = tfidf(df_en, 0.001)
tf_idf_en_05, tfidf_en_matrix_05 = tfidf(df_en, 0.005)
```

```
tf_idf_en_10, tfidf_en_matrix_10 = tfidf(df_en, 0.01)
tf_idf_en_00, tfidf_en_matrix_00 = tfidf(df_en, 0)
```

sono state prodotte 1788 parole nel processo di tf-idf con mindf= 0.001
 sono state prodotte 412 parole nel processo di tf-idf con mindf= 0.005
 sono state prodotte 167 parole nel processo di tf-idf con mindf= 0.01
 sono state prodotte 8017 parole nel processo di tf-idf con mindf= 0

PCA vs min_df

Provo a ridurre le features non alla cieca come il parametro `min_df` della Tf Idf ma preservando le informazioni all'interno del dataset.
 La PCA è giustificata dalla non necessità di interpretare le feature.

```
In [15]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
def feature_redux(M):
    original_size=M.shape[1]
    #Standardizzo i dati
    scaler = StandardScaler()
    M_redux = scaler.fit_transform(M)

    #Applico la PCA mantenendo il 95% della varianza
    pca=PCA(0.90)
    M_redux = pca.fit_transform(M_redux)

    print("Il numero di feature è stato ridotto da ",original_size,"a", pca.n_components_)

    #scatter 3d dei punti prodotti
    from mpl_toolkits.mplot3d import Axes3D
    from matplotlib import pyplot

    figure = pyplot.figure()
    ax = Axes3D(figure)

    ax.scatter(M_redux[:,0], M_redux[:,1], M_redux[:,2])
    pca.explained_variance_
    pyplot.show()

    return M_redux
```

K-means Clustering

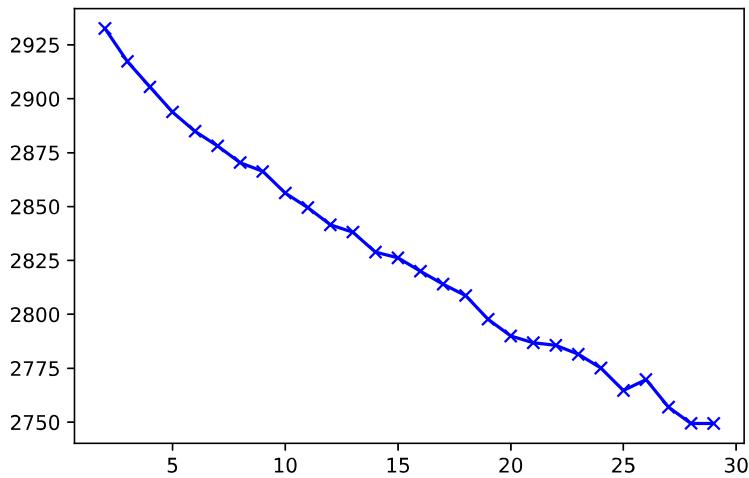
Viene formato un algoritmo di K-means Clustering, valutando i valori di K e piazzando la Sum of Square Differences.

```
In [18]: import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import time

def kmeans_ssd_plot(M):
    SSD = []
    K=range(2,30)
    t=time.perf_counter()
    for k in K:
        #print("Kmeans clustering con k = ", k)
        km=KMeans(n_clusters=k, max_iter=300, n_init=20)
        km = km.fit(M)
        SSD.append(km.inertia_)
    print("valutazione del kmeans clustering terminata in", time.perf_counter()-t)
    plt.plot(K, SSD, 'bx-')
    return SSD
```

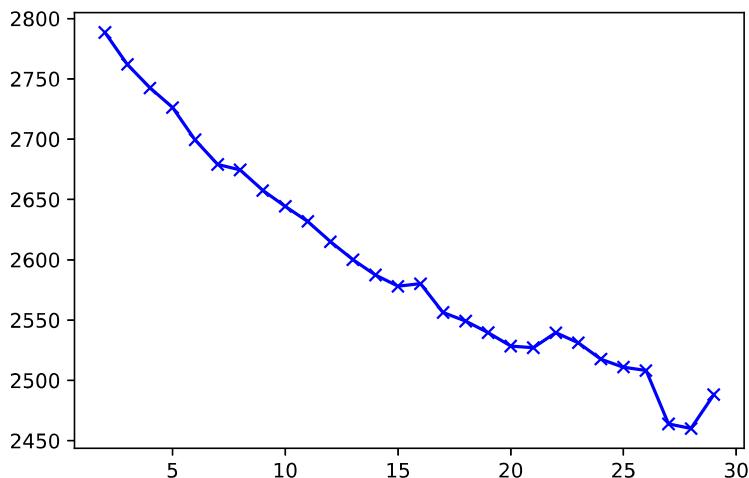
```
In [19]: SSD = kmeans_ssd_plot(tfidf_en_matrix_01)

valutazione del kmeans clustering terminata in 131.21417300000007
```



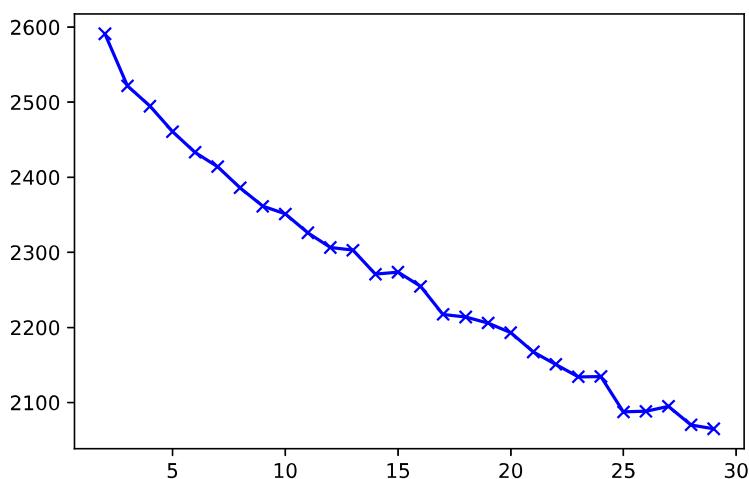
```
In [24]: SSD = kmeans_ssd_plot(tfidf_en_matrix_05)
```

valutazione del kmeans clustering terminata in 29.19181619999995



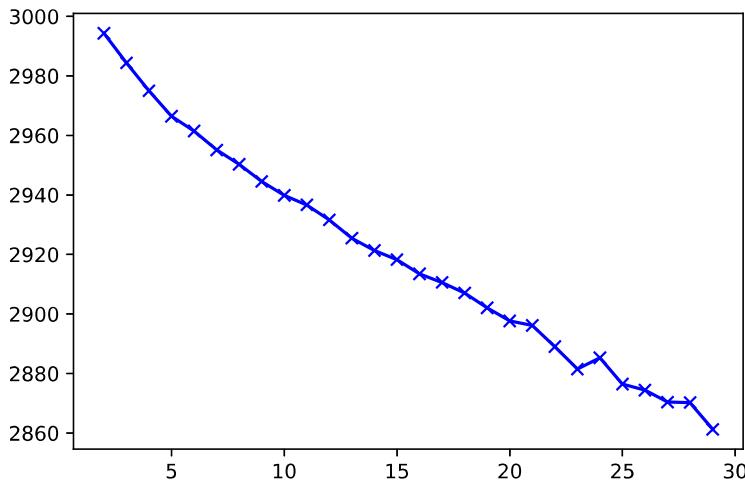
```
In [21]: SSD = kmeans_ssd_plot(tfidf_en_matrix_10)
```

valutazione del kmeans clustering terminata in 11.171770300000048



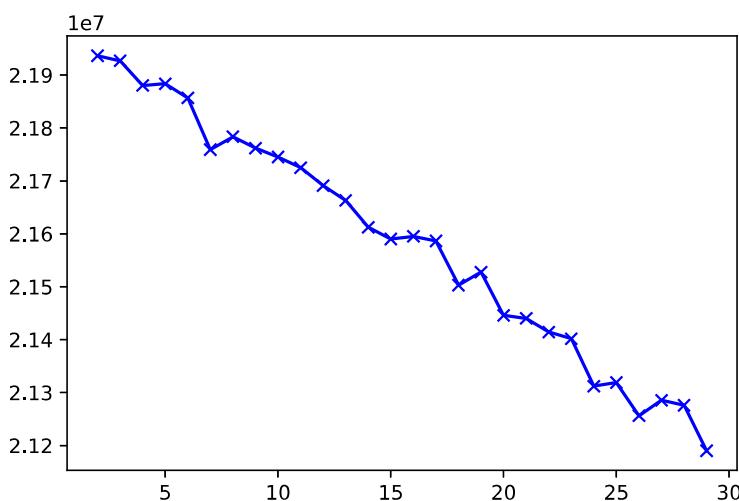
```
In [22]: SSD = kmeans_ssd_plot(tfidf_en_matrix_00)
```

valutazione del kmeans clustering terminata in 562.7450121



```
In [23]: SSD = kmeans_ssd_plot(tfidf_en_matrix_PCA)
```

valutazione del kmeans clustering terminata in 97.26917239999989



Si sceglie un valore di $k=15$, si performa il clustering e si aggiunge l'informazione di labeling al dataframe.

```
In [25]: k=15

model = KMeans(n_clusters=k, max_iter=300, n_init=20)
model.fit(tfidf_en_matrix_01)
df_en['label_kmeans_01']=model.labels_

model = KMeans(n_clusters=k, max_iter=300, n_init=20)
model.fit(tfidf_en_matrix_05)
df_en['label_kmeans_05']=model.labels_

model = KMeans(n_clusters=k, max_iter=300, n_init=20)
model.fit(tfidf_en_matrix_10)
df_en['label_kmeans_10']=model.labels_

model = KMeans(n_clusters=k, max_iter=300, n_init=20)
model.fit(tfidf_en_matrix_PCA)
df_en['label_kmeans_PCA']=model.labels_
```

Valutazione della qualità del clustering

Il clustering viene valutato attraverso un grafico che permette di visualizzare i valori di silhouette per ciascun cluster

```
In [26]: def cluster_eval(X, k):
    #####PER CLUSTER SSE#####
    model = KMeans(n_clusters=k, max_iter=300, n_init=20, random_state=3425).fit(X)
    cluster_centers = [X[model.labels_ == i].mean(axis=0) for i in range(k)]

    clusterwise_sse= [0 for x in range(k)] #VETTORE DI SSE PER OGNI CLUSTER
    for point, label in zip(X, model.labels_):
        clusterwise_sse[label] += np.square(point - cluster_centers[label]).sum()

    #-----SILHOUETTE-----
    from sklearn.metrics import silhouette_score, silhouette_samples
```

```

silhouette_avg = silhouette_score(X, model.labels_)
sample_silhouette_values = silhouette_samples(X, model.labels_)

print("La clusterizzazione ha prodotto cluster con coefficiente di silhouette pari a", silhouette_avg)

import matplotlib.cm as cm

fig, (ax1) = plt.subplots(1)
fig.set_size_inches(10,10)
ax1.set_xlim([-0.1, 0.3])
ax1.set_ylim([0, len(X) + (k + 1) * 10])

y_lower = 10
for i in range(k):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[model.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) / k)
    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("Plot della Silhouette per i vari cluster")
ax1.set_xlabel("Valori di silhouette")
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])

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

plt.show()

fig, ax = plt.subplots()
fig.set_size_inches(18,7)
ax.bar(range(k), clusterwise_sse)
ax.set_xlabel('Cluster')
ax.set_ylabel('SSE')
ax.set_title('SSE per cluster')
plt.show()
return sample_silhouette_values, clusterwise_sse

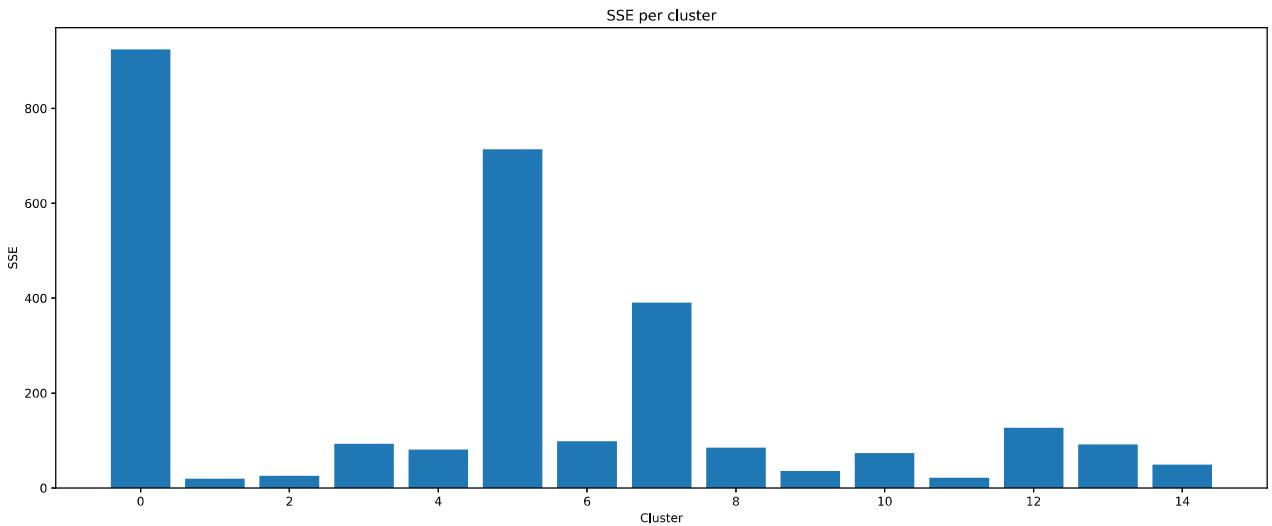
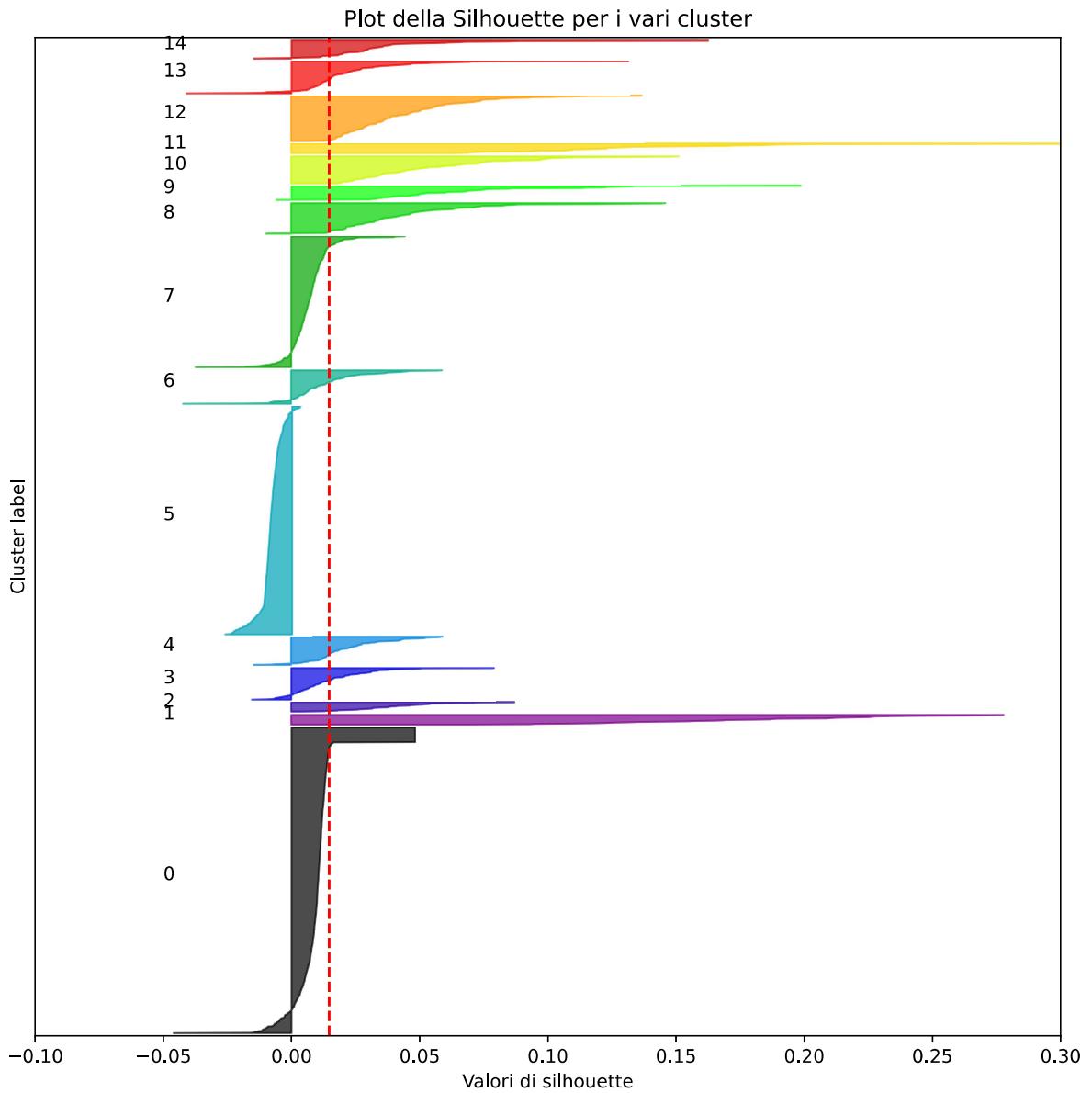
```

In [27]:

cluster_eval(tfidf_en_matrix_01, 15) #Il risultato migliore

La clusterizzazione ha prodotto cluster con coefficiente di silhouette pari a 0.01466543241351109

Silhouette analysis for KMeans clustering on sample data with n_clusters = 15



```
Out[27]: (array([ 0.00958639,  0.00703783, -0.01010757, ...,
   0.06154647,
   0.01129003,  0.01197971]),
 [923.8455827834381,
 19.286931218817486,
 25.58798176801619,
 93.04088727879964,
 80.48559176677736,
 713.2898376280255,
 97.93064510495913,
```

```
390.5037274784742,
84.46474341998194,
35.19796798657742,
73.21661536003283,
20.93517555896514,
126.90630921159607,
91.67886301793679,
48.938362043462995])
```

In []:

VAlutazione qualitativa dei cluster utilizzando una Wordcloud

Questo è il codice per generare Wordcloud

In [28]:

```
from wordcloud import WordCloud
def cloud(df, label, ncluster, text_index='text_clean'):

    frequent_words=['covid', 're']

    df_0 = df.loc[df[label]==ncluster]
    text = []
    for twt in df_0[text_index]:
        token_words = word_tokenize(twt) #tokenizzazione
        token_filter_words_1 = [w for w in token_words if not w in stop_words]
        token_filter_words_2 = [w for w in token_filter_words_1 if not w in frequent_words] #stopword filtering
        for w in token_filter_words_2:
            text.append(w)

    txt=' '.join([word for word in text])
    wordcloud = WordCloud(max_font_size=300, max_words=100, background_color="white", width=1920, height=1080).generate(txt)
    fig1 = plt.figure(figsize=(6,6))
    fig1.set_size_inches(18, 7)
    plt.imshow(wordcloud)
```

In []:

```
k=15

model = KMeans(n_clusters=k, max_iter=300, n_init=10, random_state=20)
#random state precedente = 5
model.fit(tfidf_en_matrix_01)
df_en['label_kmeans_01']=model.labels_

for i in range(k):
    cloud(df_en, 'label_kmeans_01', i)
```

Risultati di questo clustering:

- (2)-scuole
- (5)-vaccini
- (12)-texas

Codice per consolidare questo clustering:

In []:

```
df_en.to_excel('df_en_clustered.xlsx')
df_en[df_en.label_kmeans_01==2].to_excel('df_en_cluster_schools.xlsx')
df_en[df_en.label_kmeans_01==5].to_excel('df_en_cluster_vax.xlsx')
df_en[df_en.label_kmeans_01==12].to_excel('df_en_cluster_texas.xlsx')
```

Regole di associazione per ciascun cluster

In []:

```
df_en_school=pd.DataFrame(pd.read_excel('df_en_cluster_schools.xlsx'))
df_en_texas=pd.DataFrame(pd.read_excel('df_en_cluster_texas.xlsx'))
df_en_vax=pd.DataFrame(pd.read_excel('df_en_cluster_vax.xlsx'))
```

In []:

```
#genero una matrice one-hot encoded a partire dalla tf-idf completa

def onehot(r):
    if r==0:
        return 0
    else:
        return 1
```

```

from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth

def rules_fp_growth(X, min_sup=0.005):
    freq_itemsets = fpgrowth(X, min_sup, use_colnames=True)
    rules = association_rules(freq_itemsets, metric='confidence', min_threshold=0.3)
    return rules

def rules(df, minsup=0.01):
    M, tmp = tfidf(df, 0, maxdf=0.7)
    M=M.applymap(onehot)
    rules=rules_fp_growth(M, min_sup=minsup)
    return rules

```

```

In [ ]: rules(df_en_school).to_excel('rules_school.xlsx')
rules(df_en_texas, minsup=0.02).to_excel('rules_texas.xlsx')
rules(df_en_vax, minsup=0.02).to_excel('rules_vax.xlsx')

rules(df_en, minsup=0.005).to_excel('rules_allData.xlsx')

```

CODICE Sperimentale

Esaminazione dei cluster

```

In [ ]: #visualizzazione dei centroidi tramite PCA

M = model.cluster_centers_

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
#Standardizzo i dati
scaler = StandardScaler()
M = scaler.fit_transform(M)

#Applico la PCA mantenendo le prime 3 componenti
pca=PCA(n_components=3)
M = pca.fit_transform(M)

#scatter 3d dei centroidi prodotti
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot

figure = pyplot.figure()
ax = Axes3D(figure)

ax.scatter(M[:,0], M[:,1], M[:,2])
pca.explained_variance_

pyplot.show()

```

DB Scan Clustering

```

In [ ]: scaler = StandardScaler()
from sklearn.neighbors import NearestNeighbors
M=scaler.fit_transform(tfidf_en_matrix_00)

nbrs = NearestNeighbors(n_neighbors=5, radius=20, n_jobs=-1).fit(M)
distances, indices = nbrs.kneighbors(M)
distances_plot = [d[4] for d in distances]
#print(indices[0],distances[0])

```

```

In [ ]: distances_plot.sort()
plt.plot(range(M.shape[0]), distances_plot)

```

```

In [ ]: from sklearn.cluster import DBSCAN

clustering = DBSCAN(eps=100, min_samples=5).fit(M)

df_en['labels_DBSCAN']= clustering.labels_
df_en.loc[df_en['labels_DBSCAN']==0]

```

```

In [ ]: df_en.loc[df_en['labels_DBSCAN']==-1]

```

```
In [ ]: cloud(df_en, 'labels_DBSCAN', -1)
```