This notebook became relatively quickly antiquated when we had switched to faiss.kmeans(), which turned out being a more powerful and less computationally expensive method.

In [1]:
```
import pandas as pd
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import cm

from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import silhouette_samples
from sklearn.decomposition import TruncatedSVD

from scipy.sparse import csr_matrix, coo_matrix, hstack
```

In [2]:
```
df = pd.read_csv('final.csv', low_memory=False)
```

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

Out[3]:

	Unnamed: 0	Unnamed: 0.1	text	favorite_count	user_id	mentions
0	0	0	earth order survive must stop global warming ...	14116.0	UCmERzF_P0BZWGGjr2wGGnMQ	NaN
1	1	1	phase 4 moon declares independence tired eart...	12898.0	UCRgqsjV2VMb11prjm_bIC8Q	NaN
2	2	3	let get straight guy astronaut great public s...	10670.0	UCwrM8ulAgp_QiA2VgdJeJRA	NaN
3	3	5	walk spider web australia thats called assist...	9282.0	UC_m10vuJcLOosqYT5oOAKvg	NaN
4	4	6	love video send existentialist crisis others ...	6820.0	UCcnv-fzEfAhmRyWC60HFSSg	NaN

In [4]:
```
csvs = ['final_clean_4_word.csv', 'final_clean_5_word.csv', 'final_clean_6_word.csv']
```

In [5]:
```
scaler = MinMaxScaler()
df[['favorite_count']] = scaler.fit_transform(df[['favorite_count']])
df[['repost_count']] = scaler.fit_transform(df[['repost_count']])
```

In [6]:
```
df.drop(['Unnamed: 0', 'Unnamed: 0.1', 'user_id', 'mentions', 'post_id'
], axis=1, inplace=True)
```

In [7]:
```
df.head()
```

Out[7]:

	text	favorite_count	repost_count
0	earth order survive must stop global warming ...	0.248788	0.0
1	phase 4 moon declares independence tired eart...	0.227322	0.0
2	let get straight guy astronaut great public s...	0.188054	0.0
3	walk spider web australia thats called assist...	0.163591	0.0
4	love video send existentialist crisis others ...	0.120200	0.0

In [8]:
```
def vectorizer_and_hstacker(dataframe):
    # create the TDIDF transform
    vect = TfidfVectorizer().fit_transform(dataframe['text'])
    print('tfidf vectorizing complete')
    # determine components, here I'm reducing dimensionality by a factor
of 100, and making sure our minimum is at least 100
    components = max(vect.shape[0]//100, 100)
    # doing some PCA to reduce our dimensions to make this faster and ea
sier to work with
    svd = TruncatedSVD(n_components=components, n_iter=5, random_state=0
)
    print('dimensionality reduction complete')
    # transforming our text matrix and transforming it back to a matrix
 after transforming
    vect = csr_matrix(svd.fit_transform(vect))
    #turning our other x variables into a matrix so we can combine all o
ur features into one
    x_mtx = MinMaxScaler().fit_transform(csr_matrix(dataframe[['favorite
_count', 'repost_count']]))
    # stacking our matricies into a single matrix to model off of
    haystack = hstack([x_mtx, vect])
    print('matrix complete')
    return haystack
```

In [ ]:
```
vectorizer_and_hstacker(df)
```

```
tfidf vectorizing complete
dimensionality reduction complete
```

In [ ]:
```
distortions = []
ScoreList   = []
maxNumberOfClusters=20

for i in range(3, maxNumberOfClusters):
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=0)
    km.fit_predict(haystack)
    distortions.append(km.inertia_)
    ScoreList.append(-km.score(haystack))
```

In [9]:
```
def normalizer(dataframe):
    fav_std = dataframe['favorite_count'].std()
    fav_mean = dataframe['favorite_count'].mean()
    repo_std = dataframe['repost_count'].std()
    repo_mean = dataframe['repost_count'].mean()

    for i in range(len(dataframe)):
        dataframe.at[i, 'favorite_count'] = (dataframe.at[i, 'favorite_c
ount'] - fav_mean)/fav_std
        dataframe.at[i, 'repost_count'] = (dataframe.at[i, 'repost_coun
t']- repo_mean)/repo_std
```

In [10]:
```
def clusterizer(dataframe, haystack, num_clusters):
    kmeans = KMeans(n_clusters=num_clusters,
                    init='k-means++',
                    n_init=10,
                    max_iter=300,
                    tol=1e-04,
                    random_state=0)

    fit_predict = kmeans.fit_predict(haystack)
    dataframe['cluster'] = fit_predict
    return kmeans, fit_predict
```

In [11]:
```
def arrayer(dataframe):
    cluster_array = dataframe['cluster'].to_frame().to_numpy()
    return cluster_array
```

In [12]:
```
def make_silhouette(haystack, cluster, array, csv):
    cluster_labels = np.unique(fit_predict)
    n_clusters = cluster_labels.shape[0]
    silhouette_vals = silhouette_samples(haystack, fit_predict, metric=
'euclidean')
    y_ax_lower, y_ax_upper = 0, 0
    yticks = []
    for i, c in enumerate(cluster_labels):
        c_silhouette_vals = silhouette_vals[fit_predict == c]
        c_silhouette_vals.sort()
        y_ax_upper += len(c_silhouette_vals)
        color = cm.jet(float(i) / n_clusters)
        plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, heigh
t=1.0,
                 edgecolor='none', color=color)
        yticks.append((y_ax_lower + y_ax_upper) / 2.)
        y_ax_lower += len(c_silhouette_vals)
    silhouette_avg = np.mean(silhouette_vals)
    plt.axvline(silhouette_avg, color="red", linestyle="--")
    plt.yticks(yticks, cluster_labels + 1)
    plt.ylabel('Cluster')
    plt.xlabel('Silhouette coefficient')
    print(csv[12:19].replace('_', ' ') + ' ' + str(n_clusters) + ' ' + s
tr(round(cluster.inertia_, 2)))
    plt.savefig(str(round(cluster.inertia_/1000, 0)) + 'k_' + csv[12:] +
'_' + str(n_clusters) + '_clst.png', dpi=300)
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
for csv in csvs:
    dataframe = pd.read_csv(csv, low_memory=False)
    normalizer(dataframe)
    stack = vectorizer_and_hstacker(dataframe)
    for i in range(3,12,2):
        cluster, fit_predict = clusterizer(dataframe, stack, i)
        cluster_array = arrayer(dataframe)
        make_silhouette(stack, cluster, cluster_array, csv)
```