retina_for_pca_kmeans_opt_nonneg

April 4, 2022

1 Results for non-negative direct optimization method

```
[ ]: from scipy.io import loadmat
     import numpy as np
     from sklearn.cluster import KMeans
     from sklearn.decomposition import PCA
     from sklearn.metrics import silhouette_score
     import matplotlib.pyplot as plt
```

1.1 quick example with 2 clusters

```
[ ]: F = loadmat('factors_opt_nonneg.mat')['F']
     pca = PCA(35)
     X = pca.fit_transform(F)
```

```
[ ]: F.shape
```

```
[ ]: plt.plot(F[:,0])
```

```
[ ]: plt.plot(F[:,4])
```

```
[ ]: plt.plot(pca.explained_variance_)
     plt.title('Principal Values')
```

```
[ ]: X.shape
```

```
[ ]: kmeans = KMeans(init="random", n_clusters=6, n_init=10, max_iter=5000,␣
      ↪random_state=42)
     kmeans.fit(X)
     centroids = kmeans.cluster_centers_
     clusters = kmeans.labels_
```

```
[ ]: clusters.shape
```

```
[ ]: neuron_labels = np.array(clusters)
[ ]: import plotly.graph_objects as go
     import plotly.express as px

     fig = go.Figure()
     traces = []
     colors_palette = px.colors.qualitative.Dark24
     data = X

     for i, label in enumerate(set(neuron_labels)):
         mask = (neuron_labels == label)
         print(label, sum(mask))
         traces.append(go.Scatter3d(
             x=data[mask,0],
             y=data[mask,1],
             z=data[mask,2],
             mode='markers',
             marker=dict(
                 size=2.5,
                 color=colors_palette[i],
                 opacity=1,

                 #showscale= True,
             )))

     for trace in traces:
         fig.add_trace(trace)

     fig.update_layout(margin=dict(l=0, r=0, b=0, t=0),showlegend=True,)
     fig.show()
```

```
[ ]: ax = plt.axes(projection='3d')
     ax.scatter(X[:, 0], X[:,1], X[:,2], c=clusters, s=50, cmap='viridis')
```

```
[ ]: scores = []
     from sklearn.metrics import davies_bouldin_score
     from sklearn.metrics import calinski_harabasz_score

     for i in range(2,15):
         kmeans = KMeans(init="random", n_clusters=i, n_init=10, max_iter=5000,␣
      ↪random_state=42)
         kmeans.fit(X)
         centroids = kmeans.cluster_centers_
         clusters = kmeans.labels_
         scores.append(calinski_harabasz_score(X, clusters))
     plt.plot(scores)
```

1.2 find the optimal number of clusters, k

Solutions: 1. gap statistics https://towardsdatascience.com/k-means-clustering-and-the-gap-statistics-4c5d414acd29 2. elbow method https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad I chose to use the elbow method which seemed to be the most popular one.

Documentation for yellowbrick: https://www.scikit-yb.org/en/latest/api/cluster/elbow.html

[ ]:
```
# Elbow Method for K means
# Import ElbowVisualizer
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
# k is range of number of clusters.
visualizer = KElbowVisualizer(model, k=(2,10), metric='distortion', timings=␣
 ↪False)
visualizer.fit(X)        # Fit data to visualizer
visualizer.show()        # Finalize and render figure
```

From the above plot, we observed that the elbow value is at K = 32, showing that the optimal number of clusters is 32. Thus we re-run the kmeans clustering with K = 32.

[ ]:
```
K = 6
kmeans = KMeans(init="random", n_clusters=K, n_init=10, max_iter=300,␣
 ↪random_state=42)
kmeans.fit(X)
centroids = kmeans.cluster_centers_
clusters = kmeans.labels_
```

[ ]:
```
ax = plt.axes(projection='3d')
ax.scatter(X[:, 0], X[:,1], X[:,2], c=clusters, s=30, cmap='viridis')
```

[ ]:
```
retina_original_data =␣
 ↪loadmat('retina-201205_bg_bothDs_1_3b50subMeanSclStimsDel142.mat')['X']
```

[ ]:
```
retina_original_data.shape
```

1.3 test plot for the second cluster

[ ]:
```
faces_clusters =  [[] for k in range(K)]
for i in range(698):
    faces_clusters[clusters[i]].append(retina_original_data[i,1,:])

nfaces = np.zeros(K,dtype=int)
## number of faces in the kth cluster:
nfaces[2] = len(faces_clusters[2])
nrow = int(nfaces[2]/4)
fig, axs = plt.subplots(nrow, 4, figsize = (12,4))
```

```
for i , ax in enumerate(axs.flatten()):
    img = faces_clusters[2][i].reshape((8,33),order='F')
    ax.imshow(img, cmap = 'viridis')
    ax.axis('off')
I = i + 1

fig, axs2 = plt.subplots(1, nfaces[2] - nrow * 4, figsize = (4,20))
for _,ax in enumerate(axs2.flatten()):
    img = faces_clusters[2][I].reshape((8,33),order='F')
    ax.imshow(img, cmap = 'viridis')
    ax.axis('off')
    I = I + 1
plt.tight_layout()
```

```
[ ]: nfaces = np.zeros(K,dtype=int)
     for k in range(0,K):
         ## number of faces in the kth cluster:
         nfaces[k] = len(faces_clusters[k])
         ncol = 3
         nrow = int(nfaces[k]/ncol)
         fig, axs = plt.subplots(nrow, ncol, figsize = (12,6))
         for i , ax in enumerate(axs.flatten()):
             img = faces_clusters[k][i].reshape((8,33),order='F')
             ax.imshow(img, cmap = 'viridis')
             ax.axis('off')
         fig.suptitle('Given Stimuli Type 1: PSTH in Neuron Cluster ' + str(k + 1),␣
      ↪fontsize=16)

         if nfaces[k] - nrow * ncol != 0:
             I = i + 1
             if nfaces[k] - nrow * ncol == 1:
                 img = faces_clusters[k][I].reshape((8,33),order='F')
                 plt.imshow(img, cmap = 'viridis')
                 plt.axis('off')
             else:
                 fig, axs = plt.subplots(1, nfaces[k] - nrow * ncol, figsize = (6,4))
                 for i,ax in enumerate(axs):
                     img = faces_clusters[k][I].reshape((8,33),order='F')
                     ax.imshow(img, cmap = 'viridis')
                     ax.axis('off')
                     I = I + 1
```

1.4 hashing for more efficient storage

1.5 evaluate the performance of clustering

[ ]:
```
silhouette_score(X, clusters)
```

[ ]:
```
from sklearn.metrics import davies_bouldin_score
davies_bouldin_score(X, clusters)
```

[ ]:
```
from sklearn.metrics import calinski_harabasz_score
calinski_harabasz_score(X, clusters)
```

[ ]:
```

```