ShriDattaMadhira_Assignment5

October 17, 2021

[2]:
```
# (c) 2014 Reid Johnson
#
# Modified from:
# (c) 2013 Mikael Vejdemo-Johansson
# BSD License
#
# SciPy function to compute the gap statistic for evaluating k-means clustering.
#
# The gap statistic is defined by Tibshirani, Walther, Hastie in:
#  Estimating the number of clusters in a data set via the gap statistic
#  J. R. Statist. Soc. B (2001) 63, Part 2, pp 411-423

import scipy as sp
import scipy as sp
import scipy.cluster.vq
import scipy.spatial.distance
import scipy.stats
import sklearn.cluster
import numpy as np

import pylab as pl

dst = sp.spatial.distance.euclidean

def gap_statistics(data, refs=None, nrefs=20, ks=range(1,11)):
    """Computes the gap statistics for an nxm dataset.

    The gap statistic measures the difference between within-cluster dispersion␣
 ↪on an input
    dataset and that expected under an appropriate reference null distribution.

    Computation of the gap statistic, then, requires a series of reference␣
 ↪(null) distributions.
    One may either input a precomputed set of reference distributions (via the␣
 ↪parameter refs)
    or specify the number of reference distributions (via the parameter nrefs)␣
 ↪for automatic
```

```
    generation of uniform distributions within the bounding box of the dataset␣
↪(data).

    Each computation of the gap statistic requires the clustering of the input␣
↪dataset and of
    several reference distributions. To identify the optimal number of clusters␣
↪k, the gap
    statistic is computed over a range of possible values of k (via the␣
↪parameter ks).

    For each value of k, within-cluster dispersion is calculated for the input␣
↪dataset and each
    reference distribution. The calculation of the within-cluster dispersion␣
↪for the reference
    distributions will have a degree of variation, which we measure by standard␣
↪deviation or
    standard error.

    The estimated optimal number of clusters, then, is defined as the smallest␣
↪value k such that
    gap_k is greater than or equal to the sum of gap_k+1 minus the expected␣
↪error err_k+1.

    Args:
      data ((n,m) SciPy array): The dataset on which to compute the gap␣
↪statistics.
      refs ((n,m,k) SciPy array, optional): A precomputed set of reference␣
↪distributions.
        Defaults to None.
      nrefs (int, optional): The number of reference distributions for␣
↪automatic generation.
        Defaults to 20.
      ks (list, optional): The list of values k for which to compute the gap␣
↪statistics.
        Defaults to range(1,11), which creates a list of values from 1 to 10.

    Returns:
      gaps: an array of gap statistics computed for each k.
      errs: an array of standard errors (se), with one corresponding to each␣
↪gap computation.
      difs: an array of differences between each gap_k and the sum of gap_k+1␣
↪minus err_k+1.

    """
    shape = data.shape
```

```
    if refs==None:
        tops = data.max(axis=0) # maxima along the first axis (rows)
        bots = data.min(axis=0) # minima along the first axis (rows)
        dists = sp.matrix(np.diag(tops-bots)) # the bounding box of the input␣
↪dataset


        # Generate nrefs uniform distributions each in the half-open interval␣
↪[0.0, 1.0)
        rands = sp.random.random_sample(size=(shape[0],shape[1], nrefs))

        # Adjust each of the uniform distributions to the bounding box of the␣
↪input dataset
        for i in range(nrefs):
            rands[:,:,i] = rands[:,:,i]*dists+bots
    else:
        rands = refs

    gaps = sp.zeros((len(ks),))   # array for gap statistics (lenth ks)
    errs = sp.zeros((len(ks),))   # array for model standard errors (length ks)
    difs = sp.zeros((len(ks)-1,)) # array for differences between gaps (length␣
↪ks-1)

    for (i,k) in enumerate(ks): # iterate over the range of k values
        # Cluster the input dataset via k-means clustering using the current␣
↪value of k
        try:
            (kmc,kml) = sp.cluster.vq.kmeans2(data, k)
        except np.linalg.LinAlgError:
            kmeans = sklearn.cluster.KMeans(n_clusters=k).fit(data)
            (kmc, kml) = kmeans.cluster_centers_, kmeans.labels_

        # Generate within-dispersion measure for the clustering of the input␣
↪dataset
        disp = sum([dst(data[m,:],kmc[kml[m],:]) for m in range(shape[0])])

        # Generate within-dispersion measures for the clusterings of the␣
↪reference datasets
        refdisps = sp.zeros((rands.shape[2],))

        for j in range(rands.shape[2]):
            # Cluster the reference dataset via k-means clustering using the␣
↪current value of k
            try:
                (kmc,kml) = sp.cluster.vq.kmeans2(rands[:,:,j], k)
```

```
            except np.linalg.LinAlgError:
                kmeans = sklearn.cluster.KMeans(n_clusters=k).fit(rands[:,:,j])
                (kmc, kml) = kmeans.cluster_centers_, kmeans.labels_

            refdisps[j] = sum([dst(rands[m,:,j],kmc[kml[m],:]) for m in␣
↪range(shape[0])])

        # Compute the (estimated) gap statistic for k
        gaps[i] = sp.mean(sp.log(refdisps) - sp.log(disp))

        # Compute the expected error for k
        errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
                              for refdisp in refdisps)/float(nrefs)) * sp.
↪sqrt(1+1/nrefs)

    # Compute the difference between gap_k and the sum of gap_k+1 minus err_k+1
    difs = sp.array([gaps[k] - (gaps[k+1]-errs[k+1]) for k in␣
↪range(len(gaps)-1)])

    #print "Gaps: " + str(gaps)
    #print "Errs: " + str(errs)
    #print "Difs: " + str(difs)

    return gaps, errs, difs

def plot_gap_statistics(gaps, errs, difs):
    """Generates and shows plots for the gap statistics.

    A figure with two subplots is generated. The first subplot is an errorbar␣
↪plot of the
    estimated gap statistics computed for each value of k. The second subplot␣
↪is a barplot
    of the differences in the computed gap statistics.

    Args:
      gaps (SciPy array): An array of gap statistics, one computed for each k.
      errs (SciPy array): An array of standard errors (se), with one␣
↪corresponding to each gap
        computation.
      difs (SciPy array): An array of differences between each gap_k and the␣
↪sum of gap_k+1
        minus err_k+1.

    """
    # Create a figure
    fig = pl.figure(figsize=(16, 4))
```

```
    pl.subplots_adjust(wspace=0.35) # adjust the distance between figures

    # Subplot 1
    ax = fig.add_subplot(121)
    ind = range(1,len(gaps)+1) # the x values for the gaps

    # Create an errorbar plot
    rects = ax.errorbar(ind, gaps, yerr=errs, xerr=None, linewidth=1.0)

    # Add figure labels and ticks
    ax.set_title('Clustering Gap Statistics', fontsize=16)
    ax.set_xlabel('Number of clusters k', fontsize=14)
    ax.set_ylabel('Gap Statistic', fontsize=14)
    ax.set_xticks(ind)

    # Add figure bounds
    ax.set_ylim(0, max(gaps+errs)*1.1)
    ax.set_xlim(0, len(gaps)+1.0)

    # Subplot 2
    ax = fig.add_subplot(122)
    ind = range(1,len(difs)+1) # the x values for the difs

    max_gap = None
    if len(np.where(difs > 0)[0]) > 0:
        max_gap = np.where(difs > 0)[0][0] + 1 # the k with the first positive␣
 ↪dif
#     if len(np.where(difs > 0)[0]) > 0:
#         max_gap = np.where(difs == max(difs))[0][0] + 1

    # Create a bar plot
    ax.bar(ind, difs, alpha=0.5, color='g', align='center')

    # Add figure labels and ticks
    if max_gap:
        ax.set_title('Clustering Gap Differences\n(k=%d Estimated as Optimal)'␣
 ↪% (max_gap), \
                     fontsize=16)
    else:
        ax.set_title('Clustering Gap Differences\n', fontsize=16)
    ax.set_xlabel('Number of clusters k', fontsize=14)
    ax.set_ylabel('Gap Difference', fontsize=14)
    ax.xaxis.set_ticks(range(1,len(difs)+1))

    # Add figure bounds
    ax.set_ylim(min(difs)*1.2, max(difs)*1.2)
```

```
    ax.set_xlim(0, len(difs)+1.0)

    # Show the figure
    pl.show()

# (c) 2014 Reid Johnson
# BSD License
#
# Function to compute the sum of squared distance (SSQ) for evaluating k-means␣
↪clustering.

import numpy as np
import scipy as sp
import sklearn.cluster
from scipy.spatial.distance import cdist, pdist

import pylab as pl

def ssq_statistics(data, ks=range(1,11), ssq_norm=True):
    """Computes the sum of squares for an nxm dataset.

    The sum of squares (SSQ) is a measure of within-cluster variation that␣
↪measures the sum of
    squared distances from cluster prototypes.

    Each computation of the SSQ requires the clustering of the input dataset.␣
↪To identify the
    optimal number of clusters k, the SSQ is computed over a range of possible␣
↪values of k
    (via the parameter ks). For each value of k, within-cluster dispersion is␣
↪calculated for the
    input dataset.

    The estimated optimal number of clusters, then, is defined as the value of␣
↪k prior to an
    "elbow" point in the plot of SSQ values.

    Args:
      data ((n,m) SciPy array): The dataset on which to compute the gap␣
↪statistics.
      ks (list, optional): The list of values k for which to compute the gap␣
↪statistics.
        Defaults to range(1,11), which creates a list of values from 1 to 10.

    Returns:
      ssqs: an array of SSQs, one computed for each k.
```

```python
    """
    ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)

    #n_samples, n_features = data.shape # the number of rows (samples) and columns (features)
    #if n_samples >= 2500:
    #    # Generate a small sub-sample of the data
    #    data_sample = shuffle(data, random_state=0)[:1000]
    #else:
    #    data_sample = data

    for (i,k) in enumerate(ks): # iterate over the range of k values
        # Fit the model on the data
        kmeans = sklearn.cluster.KMeans(n_clusters=k, random_state=0).fit(data)

        # Predict on the data (k-means) and get labels
        #labels = kmeans.predict(data)

        if ssq_norm:
            dist = np.min(cdist(data, kmeans.cluster_centers_, 'euclidean'), axis=1)

            tot_withinss = sum(dist**2) # Total within-cluster sum of squares
            totss = sum(pdist(data)**2) / data.shape[0] # The total sum of squares
            betweenss = totss - tot_withinss # The between-cluster sum of squares
            ssqs[i] = betweenss/totss*100
        else:
            # The sum of squared error (SSQ) for k
            ssqs[i] = kmeans.inertia_

    return ssqs

def plot_ssq_statistics(ssqs):
    """Generates and shows plots for the sum of squares (SSQ).

    A figure with one plot is generated. The plot is a bar plot of the SSQ computed for each
    value of k.

    Args:
      ssqs (SciPy array): An array of SSQs, one computed for each k.

    """
    # Create a figure
```

```
    fig = pl.figure(figsize=(6.75, 4))

    ind = range(1,len(ssqs)+1) # the x values for the ssqs
    width = 0.5 # the width of the bars

    # Create a bar plot
    #rects = pl.bar(ind, ssqs, width)
    pl.plot(ind, ssqs)

    # Add figure labels and ticks
    pl.title('Clustering Sum of Squared Distances', fontsize=16)
    pl.xlabel('Number of clusters k', fontsize=14)
    pl.ylabel('Sum of Squared Distance (SSQ)', fontsize=14)
    pl.xticks(ind)

    # Add text labels
    #for rect in rects:
    #    height = rect.get_height()
    #    pl.text(rect.get_x()+rect.get_width()/2., 1.05*height, '%d' %␣
 ↪int(height), \
    #            ha='center', va='bottom')

    # Add figure bounds
    pl.ylim(0, max(ssqs)*1.2)
    pl.xlim(0, len(ssqs)+1.0)

    pl.grid(True)
    pl.show()
```

[3]:
```
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('./shopping-data.csv')
df.head()
```

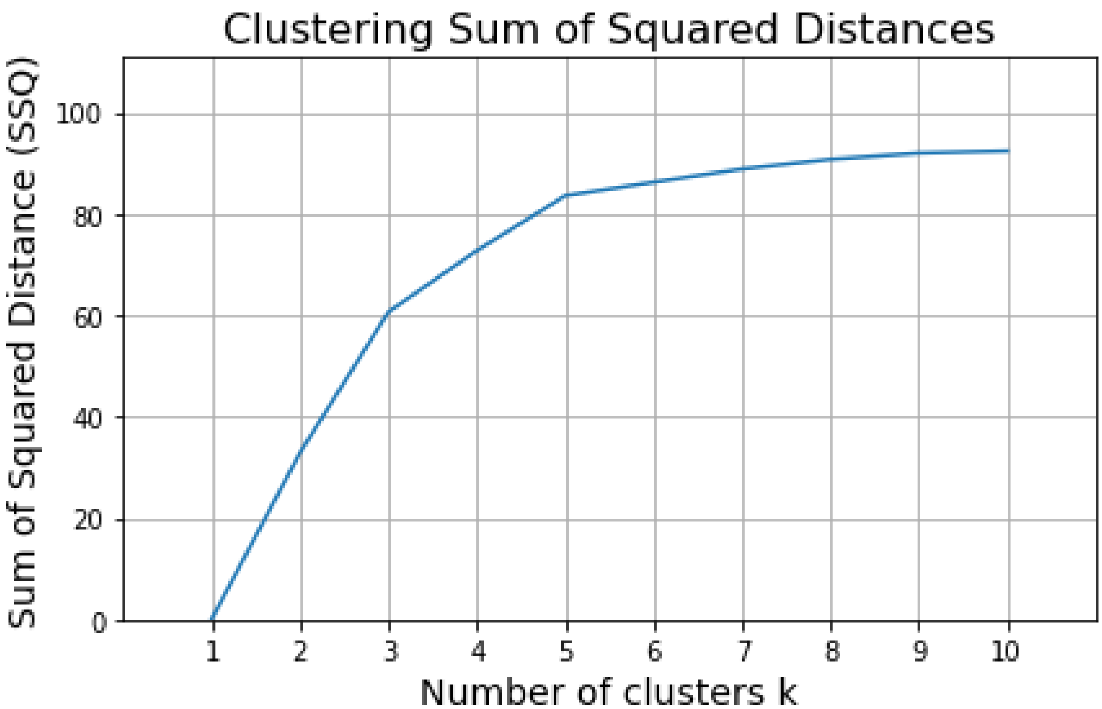
[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[4]:
```
df.drop('CustomerID', 1, inplace=True)
df.drop('Genre', 1, inplace=True)
df.drop('Age', 1, inplace=True)
```

[5]:

```python
# Sum of Squared Deviations(SSQ)
ssqs = ssq_statistics(df, ks=range(1,11))
# print(ssqs)
plot_ssq_statistics(ssqs)
```



[13]:

```python
# Gap Statistic
from sklearn.preprocessing import StandardScaler

s_scaler = StandardScaler()
df_s = s_scaler.fit_transform(df)

gaps, errs, difs = gap_statistics(df_s.astype(float), nrefs=20, ks=range(1,11))
```

[7]:

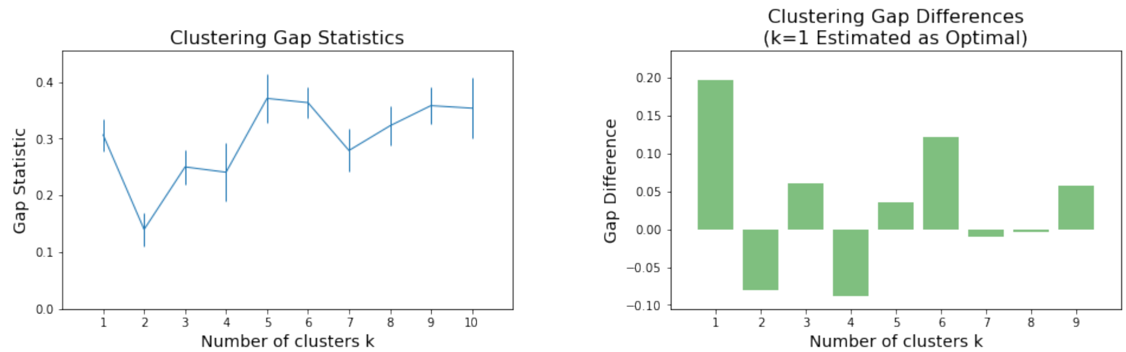
```python
print(gaps)
print("====================================================")
print(errs)
print("====================================================")
print(difs)

# print(np.where(difs>0)[0])
# print(np.where(difs == max(difs))[0][0])
```

```
[0.30680243 0.14008508 0.25028758 0.24090546 0.37109559 0.363583
 0.27924146 0.32270755 0.35832763 0.35384366]
```

```
==================================================
[0.0281036  0.02955172 0.03034732 0.05144172 0.04252948 0.02802046
 0.03784266 0.03428021 0.03190271 0.0527273 ]
==================================================
[ 0.19626908 -0.07985518  0.06082383 -0.08766066  0.03553305  0.1221842
 -0.00918588 -0.00371737  0.05721127]
```

[8]:
```
plot_gap_statistics(gaps, errs, difs)
```



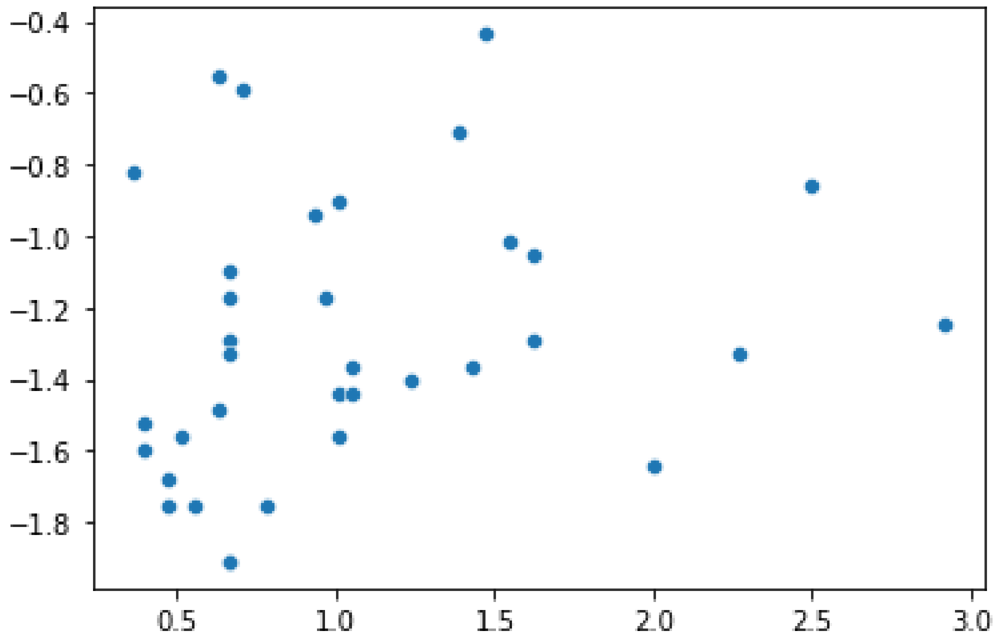
[9]:
```
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

# 5 clusters
kmeans = KMeans(n_clusters=5)
clusters = kmeans.fit_predict(df_s)

cluster0 = df_s[clusters == 0]

sns.scatterplot(cluster0[:,0] , cluster0[:,1])
```

[9]:
```
<AxesSubplot:>
```

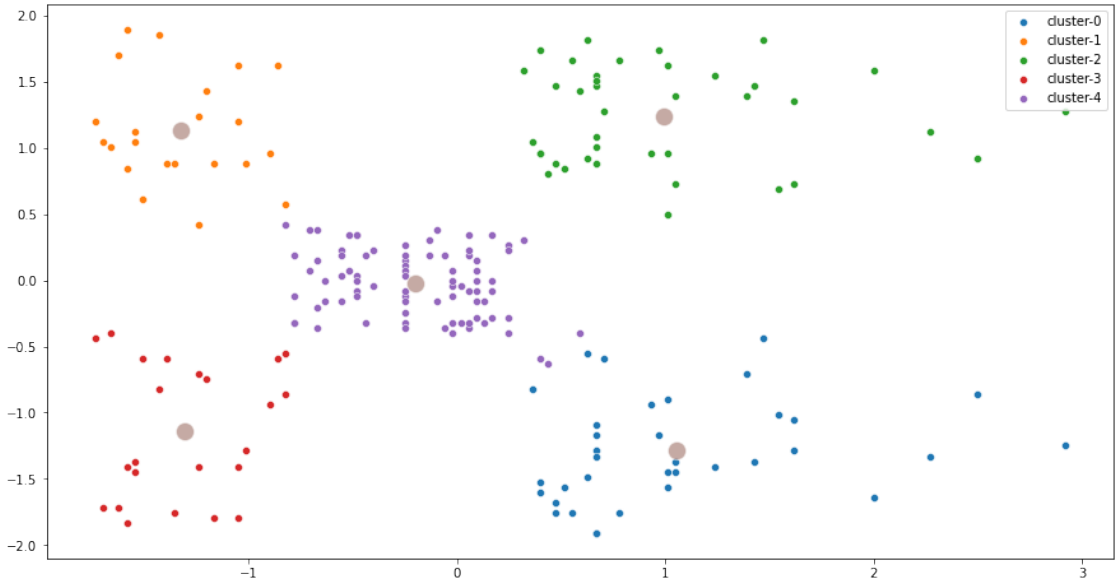


[10]:
```
#Getting unique labels
u_clusters = np.unique(clusters)
print(u_clusters)

#plotting the results
plt.figure(figsize=(15,8))
for i in u_clusters:
    sns.scatterplot(df_s[clusters == i , 0] , df_s[clusters == i , 1] , label =␣
↪'cluster-'+str(i))

# for i in u_clusters:
centers = kmeans.cluster_centers_
sns.scatterplot(centers[:, 0], centers[:, 1], palette='black', s=200, alpha=0.
↪5);
```

```
[0 1 2 3 4]
```



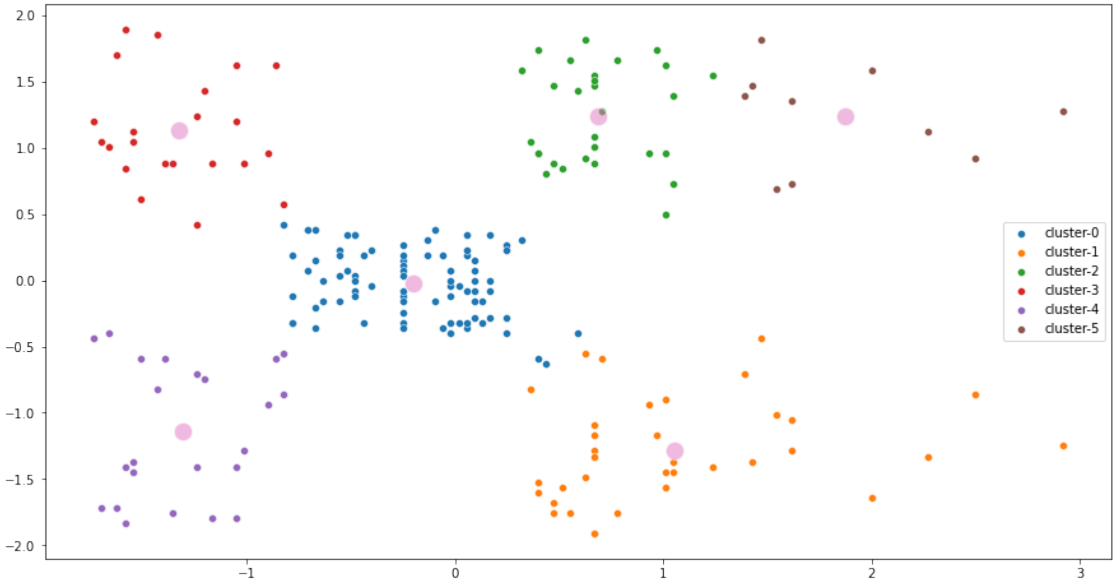
[14]:
```
# 6 clusters
kmeans = KMeans(n_clusters=6)
clusters = kmeans.fit_predict(df_s)

#Getting unique labels
u_clusters = np.unique(clusters)
print(u_clusters)

#plotting the results
plt.figure(figsize=(15,8))
for i in u_clusters:
    sns.scatterplot(df_s[clusters == i , 0] , df_s[clusters == i , 1] , label =␣
↪'cluster-'+str(i))

# for i in u_clusters:
centers = kmeans.cluster_centers_
sns.scatterplot(centers[:, 0], centers[:, 1], palette='black', s=200, alpha=0.
↪5);
```

```
[0 1 2 3 4 5]
```



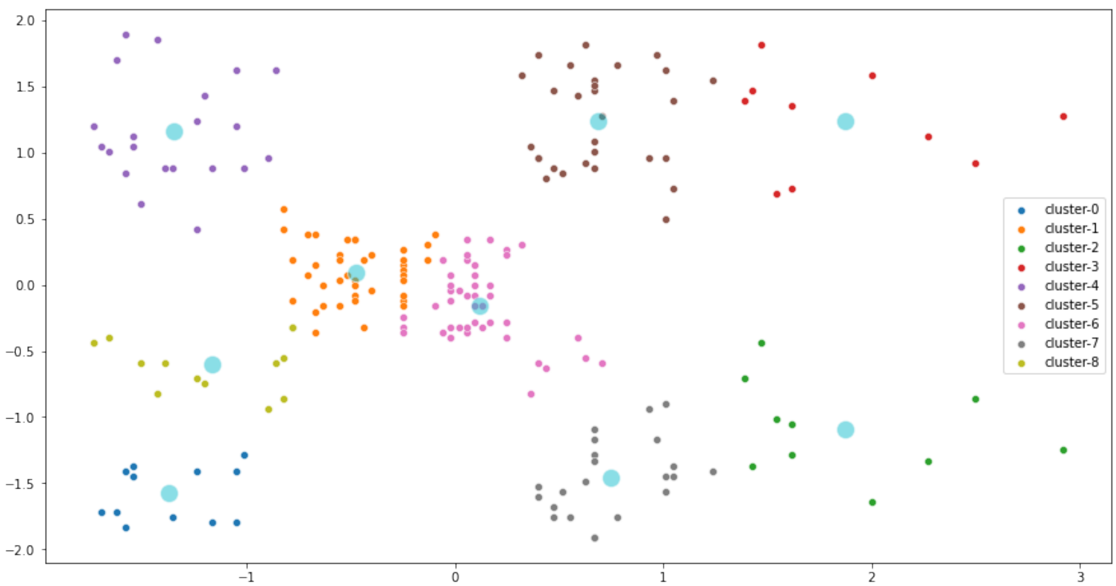
[15]:
```python
# 9 clusters
kmeans = KMeans(n_clusters=9)
clusters = kmeans.fit_predict(df_s)

#Getting unique labels
u_clusters = np.unique(clusters)
print(u_clusters)

#plotting the results
plt.figure(figsize=(15,8))
for i in u_clusters:
    sns.scatterplot(df_s[clusters == i , 0] , df_s[clusters == i , 1] , label =␣
 ↪'cluster-'+str(i))

# for i in u_clusters:
centers = kmeans.cluster_centers_
sns.scatterplot(centers[:, 0], centers[:, 1], palette='black', s=200, alpha=0.
 ↪5);
```

```
[0 1 2 3 4 5 6 7 8]
```



0.1 QUESTION-1

The estimated elbow point according to the sum of squared (SSQ) statistic is between k=5 to k=8. There is a significant deviation between k=4 and k=5 and it kept increasing till k=8 after which there is a flat line. So, I believe the elbow points should be inbetween k=5 to k=8.

The gap statistic typically estimated the optimal k value to be k=5. After running the gap statistic multiple times, plotting the clusters, and analysing the inter-cluster distances, I decided k=5 is the optimal value for number of clusters.

0.2 QUESTION-2

After plotting the clustered data for different values of k, I reached a conclusion that k=5 is the optimal value for number of clusters for this dataset. From the above plot we can see that the data is segmented nearly perfectly into 5 different clusters. The intra-cluster distance is less and inter-cluster distance is maximized to the extent possible.

Seperating the customers into 5 different groups depending on their Annual Income and Spending Score. This will give us a better understanding on what type of customer is safe and who is someone that can cause an issue.

0.3 QUESTION-3

Out of sum of squared(SSQ) statistics and gap statistics, SSQ is more consistent in choosing the value of k. Gap statistics outputted a k value that ranged from 4 to 9, with multiple runs suggesting multiple values in that range.