CLUSTERING : KMEANS

```
#Ruchi Bhavsar

# (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 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(sp.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 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 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

    # 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.

    """
    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.show()
```

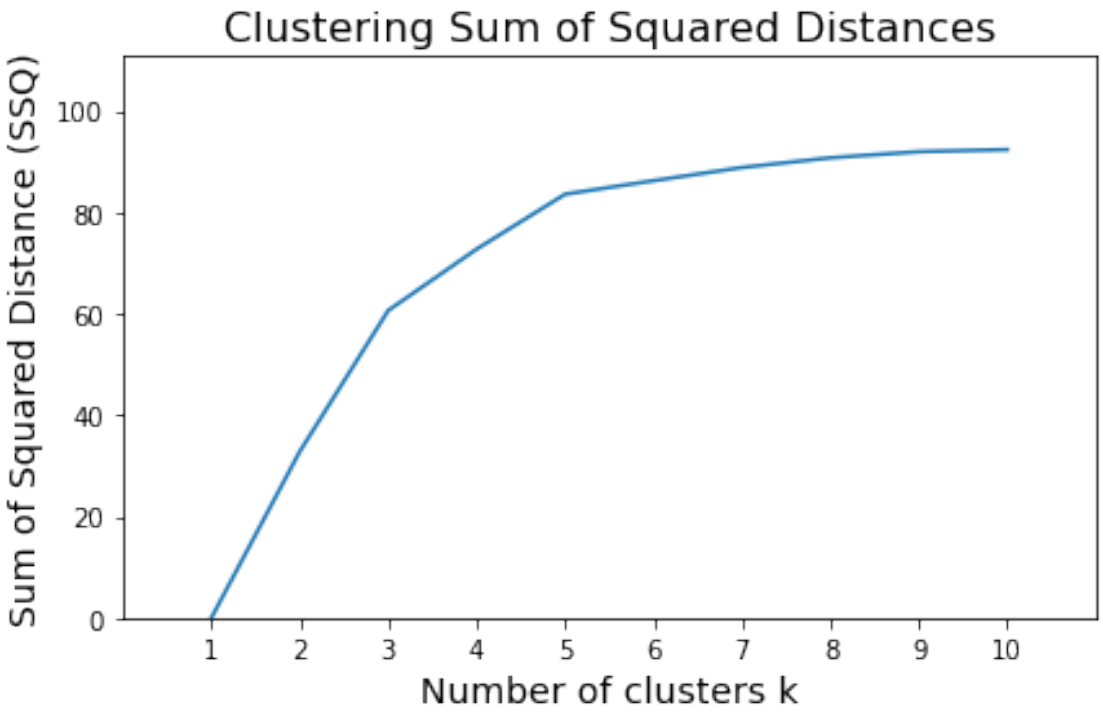
Loading the dataset

```
from operator import delitem
import csv

def load_dataset(filename):
  with open(filename, 'r') as file:
    d = csv.reader(file, delimiter=',', quotechar='"')
    data = [data for data in d]
    d_a = np.asarray(data)
  t = []
  for i in range(1, len(d_a)):
    t.append([d_a[i][3], d_a[i][4]])
  dataset = np.asarray(t).astype(float)
  return dataset



import pandas as pd

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

#data = df.drop(['CustomerID', 'Genre', 'Age'], axis = 1)
data = load_dataset('/content/shopping-data.csv')
```

Generating and Plotting SSQ statisic

```
ssqs = ssq_statistics(data, ks=range(1,11))
plot_ssq_statistics(ssqs)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:36:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
```

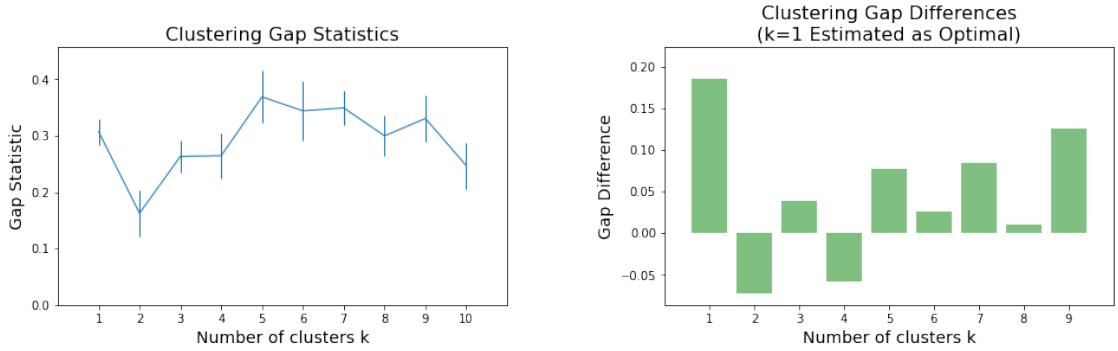


Generating and Plotting gap statistics

```
gaps, errs, difs = gap_statistics(data, nrefs=20, ks=range(1, 11))
plot_gap_statistics(gaps, errs, difs)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:67:
DeprecationWarning: scipy.diag is deprecated and will be removed in
SciPy 2.0.0, use numpy.diag instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:78:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:79:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:80:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:94:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:106:
DeprecationWarning: scipy.log is deprecated and will be removed in
SciPy 2.0.0, use numpy.lib.scimath.log instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:106:
DeprecationWarning: scipy.mean is deprecated and will be removed in
SciPy 2.0.0, use numpy.mean instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:109:
DeprecationWarning: scipy.log is deprecated and will be removed in
SciPy 2.0.0, use numpy.lib.scimath.log instead
```

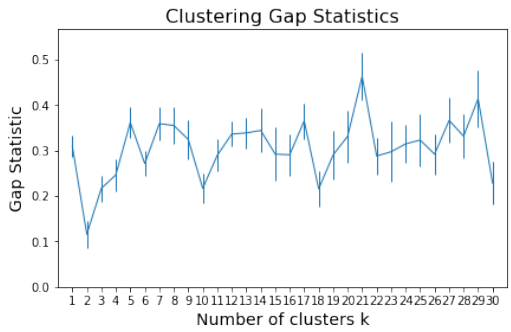
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:109:
DeprecationWarning: scipy.mean is deprecated and will be removed in
SciPy 2.0.0, use numpy.mean instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:109:
DeprecationWarning: scipy.sqrt is deprecated and will be removed in
SciPy 2.0.0, use numpy.lib.scimath.sqrt instead
/usr/local/lib/python3.7/dist-packages/scipy/cluster/vq.py:579:
UserWarning: One of the clusters is empty. Re-run kmeans with a
different initialization.
  warnings.warn("One of the clusters is empty. "
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:112:
DeprecationWarning: scipy.array is deprecated and will be removed in
SciPy 2.0.0, use numpy.array instead
```

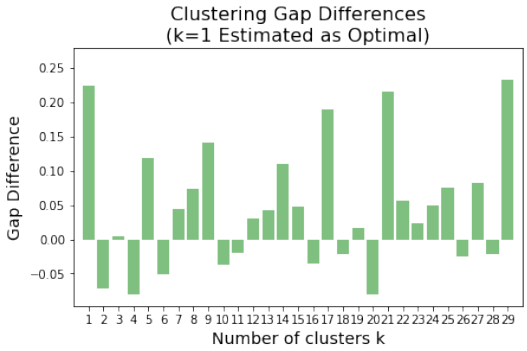



```
gaps, errs, difs = gap_statistics(data, nrefs=20, ks=range(1, 31))
plot_gap_statistics(gaps, errs, difs)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:67:
DeprecationWarning: scipy.diag is deprecated and will be removed in
SciPy 2.0.0, use numpy.diag instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:78:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:79:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:80:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:94:
DeprecationWarning: scipy.zeros is deprecated and will be removed in
SciPy 2.0.0, use numpy.zeros instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:106:
DeprecationWarning: scipy.log is deprecated and will be removed in
SciPy 2.0.0, use numpy.lib.scimath.log instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:106:
DeprecationWarning: scipy.mean is deprecated and will be removed in
SciPy 2.0.0, use numpy.mean instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:109:
```

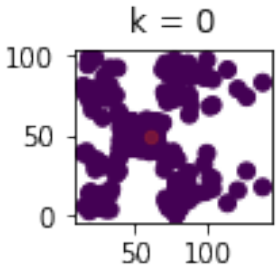
```
DeprecationWarning: scipy.log is deprecated and will be removed in
SciPy 2.0.0, use numpy.lib.scimath.log instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:109:
DeprecationWarning: scipy.mean is deprecated and will be removed in
SciPy 2.0.0, use numpy.mean instead
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:109:
DeprecationWarning: scipy.sqrt is deprecated and will be removed in
SciPy 2.0.0, use numpy.lib.scimath.sqrt instead
/usr/local/lib/python3.7/dist-packages/scipy/cluster/vq.py:579:
UserWarning: One of the clusters is empty. Re-run kmeans with a
different initialization.
  warnings.warn("One of the clusters is empty. "
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:112:
DeprecationWarning: scipy.array is deprecated and will be removed in
SciPy 2.0.0, use numpy.array instead
```

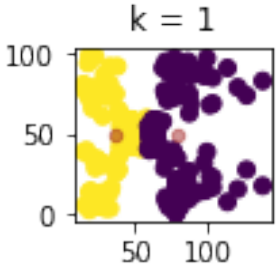


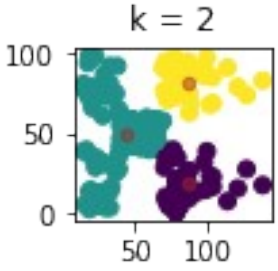


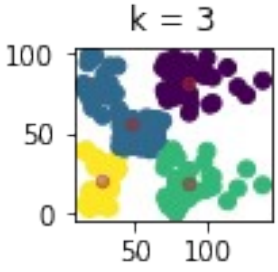
Scatter Plot

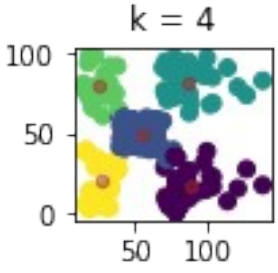
```
from sklearn.cluster import KMeans
def draw_plot(data, ks):
  pl.figure()
  for(i, k) in enumerate(ks):
    kmeans = KMeans(n_clusters=k, random_state=0)
    y_pred = kmeans.fit_predict(data)
    pl.subplot(3, 4, i+1)
    pl.title('k = '+ str(i))
    pl.scatter(data[:, 0], data[:, 1], c=y_pred)
    centeroids = kmeans.cluster_centers_
    pl.scatter(centeroids[:, 0], centeroids[:, 1], c='brown', s=20,
alpha=0.5);
    pl.show()

draw_plot(data,ks=range(1,11))
```

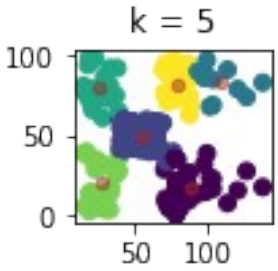

k = 0
100
50
0
50
100

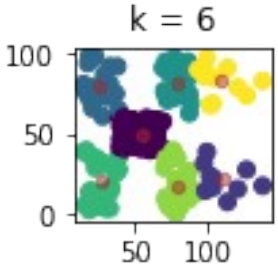

k = 1
100
50
0
50
100

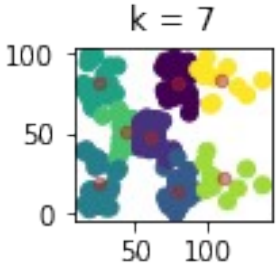

k = 2
100
50
0
50
100

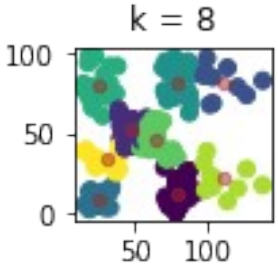

k = 3
100
50
0
50
100

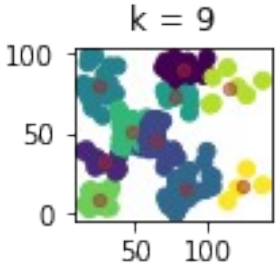

k = 4
100
50
0
50
100


k = 5
100
50
0
50
100


k = 6
100
50
0
50
100


k = 7
100
50
0
50
100


k = 8
100
50
0
50
100


k = 9
100
50
0
50
100

Questions

1. Where did you estimate the elbow point to be (between what values of k)? What value of k was typically estimated as optimal by the gap statistic? To adequately answer this question, consider generating both measures several (atleast 5) times, as there may be some amount of variation in the value of k that they each estimate as optimal.

- We observe the elbow point produces k value between 2 and 5 for SSQ and produces a k value to be anywhere between 4 and 5 foe the gap statistics.

2. Based on the scatter plot of the clustered data, what makes most sense? Give logical interpretation from visually inspecting the clusters.

- As the k increases we see that the number of clusters also increase, this shows that the data will be able to scale to large datasets and adapts to new samples easily.
- For smaller values of k , we can lessen the dependence by repeating the algorithm multiple times for different initial values and then choosing the best outcome. But as k increases, we will need k-means to choose better values of the initial centroids(k-means seeding)

3. Between SSQ and Gap Statistics, does one measure seem to be a consistently better criterion for choosing the value of k than the other? Why or why not?

- Gap Statistics is a better criterion for choosing the k value. We can easily spot the k values that corresponds to the largest gap staistic.
- The basic idea of the Gap Statistics is to choose the number of K, where the biggest jump in within-cluster distance occurred, based on the overall behavior of uniformly drawn samples.
- The larger the gap, the better will be the clustering effect. If the value of k increases extremely, the gap maitains a straight line and shows no category relationship.

References

- https://stats.stackexchange.com/questions/88550/using-the-gap-statistic-to-compare-algorithms
- https://towardsdatascience.com/k-means-clustering-and-the-gap-statistics-4c5d414acd29
- https://www.pinecone.io/learn/k-means-clustering/
- https://github.com/NUOEL/cs6220/blob/master/M05-A01%20-%20functions.ipynb