k-Means Programming Exercises (Java JUnit Edition)

For these exercises you will begin with the partial implementation KMeansClusterer.java which processes command line options, reads the data set input format from System.in, and writes the clustered output format on System.out.

Data Set Input Format

On the standard input, the first two lines of input data are:

```
% {number1} dimensions
% {number2} points
```

Each of the `{number2}` lines following has `{number1}` space-separated numbers, where `{number2}` and `{number1}` are positive integers.

One can and should visualize 2D input data in Matlab or Octave using the `data2d.m` script.

Clustered Output Format

On the standard output, the first four lines of output cluster data are:

```
% {number1} dimensions
% {number2} points
% {number3} clusters/centroids
% {number4} within-cluster sum of squares
```

Each of the next `{number3} + {number2}` lines is space-separated and contains point coordinates of `{number1}` dimensions *preceded by* that point's associated 0-based cluster number.
In the first `{number3}` lines following the first 4 lines above, the coordinates are of the centroids, ordered by ascending centroid cluster number.
In the next `{number2}` lines, the coordinates are from the original data in the same order.
The value `{number4}` is the computed within-cluster sum of squares (WCSS).

One can and should visualize 2D output data in Matlab or Octave using the `cluster2d.m` script.

Exercises

For all exercises, assume the Standard Algorithm with Forgy initialization.

k-Means with given *k*

Implement KMeansClusterer.kMeansCluster() so that when kMin == kMax, it computes *kMin* clusters using Forgy initialization and the standard k-means algorithm.

From a UNIX command line (e.g. terminal window, or Cygwin bash terminal on Windows), your command would have the form:

```
java KMeansClusterer –k {k}  <  {input file}  >  {output file}
```

Example:

```
java KMeansClusterer –k 2 < bullseye2.dat > bullseye2-out.dat
```

To see just the first 4 lines of output from a file, use the UNIX "head" command, e.g.:

```
head -4 bullseye2-out.dat
```

To see these first 4 lines of output directly without generating an output file in a UNIX environment, one can "pipe" the output to the head command, e.g.:

```
java KMeansClusterer –k 2 < bullseye2.dat | head -4
```

Using GNU Octave or MATLAB, one can visualize the 2D input/output data files as follows:

- Copy the script files data2d.m and cluster2d.m into the working directory with your data.
- Start GNU Octave or MATLAB and use the "cd" command to change your directory to the same working directory.
- Visualizing input: Use the command `data2d('<filename>')`.
- Visualizing cluster output: Use the command `cluster2d('<filename>')`.

Questions:

1. What are the *assumptions* the *k*-means algorithm makes of the input data?
2. Each input file has the number of original generating clusters in the filename before the ".dat" extension. For the files your instructor provides, run *k*-means as described 3 times, observing the within-cluster sum of squares (WCSS) value for each run, and visualizing the output of for the lowest WCSS value.
    a. What is your median and minimum WCSS value for each problem?
    b. For which files does *k*-means clustering appear to succeed always/sometimes/never in your sample of 3 runs?
3. For those problems where *k*-means appears to always fail, which assumptions (if any) of the k-means algorithm are violated?
4. Is it possible for *k*-means to fail if no assumptions are violated? Why or why not?

Iterated *k*-Means with given *k*

Create an augmented version of the first program that, for `iter` > 1 performs `iter` independent runs of the k-means algorithm and outputs only the clustering result with the lowest WCSS value.
From a UNIX command line (e.g. terminal window, or Cygwin bash terminal on Windows), your command would have the form:

```
java KMeansClusterer –k {k} –iter {i} < {input file} > {output file}
```
Example:
```
java KMeansClusterer –k 2 –iter 10 < bullseye2.dat > bullseye2-out.dat
```

Use 10 iterations for answering the following questions.

Questions:

(1) Why should iterated use of the *k*-means algorithm help in some cases with the quality of output clusters?
(2) For the files your instructor provides, run iterated k-means one time each. For which files does k-means clustering appear to succeed in your iterated sample run?
(3) For which types of clustering problems does/doesn't this iterated approach help?

k-Means using the Gap Statistic to choose *k*

Create an augmented version of the second program that has all of the previous functionality, but additionally chooses the best k value in a range of k values using a simplified gap statistic. The range of k values is specified via variables kMin and kMax.
From the command line, these are supplied through options "-kmin" and "-kmax" respectively.
From a UNIX command line (e.g. terminal window, or Cygwin bash terminal on Windows), your command would have the form:
```
java KMeansClusterer –kmin {min k} –kmax {max k} –iter {i} < {input file} > {output file}
```
Example:
```
java KMeansClusterer –kmin 2 –kmax 9 –iter 10 < easygaussian7.dat > easygaussian7-out.dat
```

One very difficult problem concerns how one can automatically determine the number of clusters in a data set. There are many, many suggested techniques, e.g.:

- http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set
- http://stackoverflow.com/questions/15376075/cluster-analysis-in-r-determine-the-optimal-number-of-clusters/15376462#15376462

Perhaps the simplest technique is to plot the lowest WCSS found for each *k* and look for an "elbow", that is, a *k* after which reductions in WCSS are not significant as *k* increases.

- http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set#The_Elbow_Method

The difficulty with Elbow Method, as it is called, is that the most significant elbow bend is not always easily discernable from this graph. What one would like is some means of comparing the gap between the expected WCSS on uniformly-distributed random data and the WCSS obtained on the given data to see which *k* yields the most significant gap. This measure is what is known as the Gap Statistic:

- http://web.stanford.edu/~hastie/Papers/gap.pdf

Here we specify a simplification of the simplest alternative presented in the paper above.

Create a program that

(1) for each *k* from *kMin* to *kMax* (inclusive):
    a. computes the log of the minimum WCSS from *iter* iterations of *k-means* clustering as in the previous exercise,
    b. generates 100 data sets of an equal number of data points randomly distributes within the [minimum, maximum] range for each dimension of the given data, computes the k-means algorithm for each data set (for 1 iteration), and computes the average log of the WCSS for each data set, and
    c. computes the gap statistic for this *k* by subtracting the value of (a) above from the value of (b); and,
(2) retains the minimum WCSS clustering for the *k* yielding the maximum gap statistic.

Put another way, for a given *k* we compute the log of the best (least) WCSS we can find over *iter* iterations of *k*-means, denoted $\log(W_k)$. Next we want to get a sense of what the expected value of $\log(W_k)$ would have been had we been working with uniformly distributed data. This value is denoted $E^*_n\{\log(W_k)\}$. One way of estimating this value is to uniformly generate a number of data sets in the range of values seen in the input data. We do this for 100 uniformly generated data sets, taking the log of each resulting WCSS value, and averaging all of these values to estimate $E^*_n\{\log(W_k)\}$, our expected WCSS value on our "null reference distribution". Then the gap statistic for the given *k* is simply $E^*_n\{\log(W_k)\} - \log(W_k)$.
Having then determined the *k* yielding the maximum gap statistic, we finally output our best (lowest WCSS) *k*-means clustering results for that *k*.

Questions:

(1) Each input file has the number of original generating clusters in the filename before the ".dat" extension. For the files your instructor provides, run your program 10 times with a maximum *k* value of 10, recording from each run the number of clusters *k* that yielded the maximum gap statistic.
    a. For which data set(s) does this technique consistently succeed in discerning the correct number of clusters?
    b. For which data set(s) does the discerned number of clusters vary from run to run?
    c. For which data set(s) does this technique consistently return the *incorrect* number of clusters? What do you observe about the nature of the data in these case(s)?