CS/INFO 3300; INFO 5100
Homework 6
Due 4:59pm Wednesday 4/13 (NOTE THE **AFTERNOON** DEADLINE)

K-means clustering; interaction

Before break we worked with the k-means clustering algorithm. One of the common complaints about this algorithm is that it requires us to specify a number of clusters, k. Here we'll look at the Kulis-Jordan algorithm, which relaxes this assumption and tries to make the number of clusters fit the data better.

The idea is simple: start with one cluster, and add more as necessary. Each cluster is defined by a centroid, whose *x* and *y* coordinates are the mean *x* and mean *y* coordinates of all the data points assigned to that cluster. As before, we'll consider each data point in turn and find its nearest cluster centroid. If the nearest cluster centroid is further than some distance *ε* from the point we will add a new cluster with a centroid at that data point. If clusters become empty, leave their position unchanged.

You should have two arrays, `points` and `centroids`. The elements of the points array should be the data points from the example on 3/25: the log number-of-doctors and %GDP health spending for various countries.

1. Implement this algorithm. You are encouraged to start with the code from the notes for 3/21 or 3/25, but make the following changes. You will separate the clustering algorithm from the SVG display code. As always, mention any sources of code or inspiration that you used.

a. In the callback function for the `d3.csv()` function, instead of creating a `circle` for each country and cluster, create a `text` element. Initialize to a single cluster located at a randomly selected point, and assign all points to that cluster. (5 pts)

b. Create a function `iterate(threshold)`. This function implements the clustering algorithm, but only touches the data, not the display. The function should both reassign points to clusters *and* then update the cluster centroids to their new values. Use the threshold parameter to determine when to add new clusters. Do *not* change any SVG display variables in this function. To calculate distances, use data coordinates, not pixel coordinates. The function should return the number of data points whose cluster assignment has changed in this iteration. (15 pts)

c. Create an `updateDisplay()` function that creates SVG `<text>` elements for new clusters as necessary, removes unneeded elements, updates the position of cluster centroid SVG elements, and updates the text value of the SVG elements for data points. Each point and cluster centroid should be represented by a `<text>` element. The value of the text element should be a unicode character unique to a cluster. You may wish to use the `String.fromCharCode()` function. The "Geometric Shapes" Unicode region (x25A0-x25FF) might be a good choice, but be creative! Hint: try finding the char code for the first character in a range (like 65 for "A") and then add the cluster ID to that char code. The text for each data point should match the character for its current cluster. Use color, size, or other visual characteristics to distinguish data points from cluster centroids. Note that many clusters may have the same position as data points, we want to be able to see both. (20 pts)

d. Create a function `cluster(threshold, display)`. The value of `threshold` is a number, the value of `display` is a function. The `cluster` function should "reset" the cluster state: create a new `centroids` array with one centroid, and assign all data points to that cluster. The position of the new centroid should be a randomly selected data point. Run your `iterate` function with the specified `threshold` value until the cluster assignments do not change. After each call to `iterate`, call `display`. Return the number of non-empty clusters in the final converged solution. (10 pts)

2. Create a 300x300 `<svg>` element and a `range` input (i.e. a slider) with values from 0.5 to 10.0. You can create these elements dynamically or hard code them. The `updateDisplay` function should add new elements and update the display in this `<svg>` element. Each time the user moves *and releases* the slider, run your `cluster` function with the selected threshold and your `updateDisplay` function as parameters. (Use a standard HTML `<input type="range">` element --- no jQuery.) (25 pts)

3. Use the interactive interface in Problem 2 to select four interesting values of the threshold variable. For each of these four different values of `threshold`:
a. Run your cluster function ten times and store the returned cluster counts in an array. Display these values in a table, with one row for each value of the threshold, and one column for each repetition. You may do this dynamically or do it manually and hard code the result. (15 pts)
b. In a new <p> tag, describe the effect of the threshold parameter. To what extent does it control the number of clusters? If someone says that this algorithm allows the data to "speak for itself" without human intervention, would you believe them? (10 pts)