MATH 728D: Machine Learning Lab #9: Clustering and Similarity

John Burkardt

December 7, 2018

How can I arrange my large dataset into a small number of clusters?

In some cases, data comes with no classification, and it is the task of a machine learning algorithm to search for naturally-occuring groupings of the data.

Raw data comes without classification information. There are ways to try to discover classifications automatically. The simplest involves the idea of data similarity: we assume that we can compare any pair of data items and assign a measure of the difference between them; if the data is numeric vectors, then typically we use the Euclidean norm. Thus, in MATLAB, the difference between two data items would be:

```
diff = norm ( data(i,:) - data(j,:) );
```

1 Hierarchical Clustering

The **hierarchical clustering method** organizes the data one step at a time. At the beginning, each data item forms its own cluster. On each step, the two closest clusters are merged. The process terminates with a single cluster containing all the data. The merging process can be displayed as a tree diagram. In biological applications, this tree diagram can sometimes suggest a developmental process that explains how the data is related.

MATLAB has several commands that simplify the construction and display of a hierarchical clustering. We will take a file of data intended to study pollution and weather in 41 US cities, and concentrate on two measurements, average temperature and average precipitation.

Because the data file contains text and numbers, we need to use the **readtable()** function to read the file into a table, and use MATLAB's curly bracket notation to extract the city names and numeric data.

Exercise 1:

1. use `readtable()` to read the file *pollution_data.csv*, and save the information in a variable called `table`;
2. copy table{:,1} into the vector `names`;
3. copy table{:,[3,7]} into the vector `data`;
4. Normalize `data` so that both columns range between 0 and 1;
5. create the pairwise distance matrix `pd` by applying MATLAB's `pdist()` function to `data`;
6. create the clustering structure `cs` by applying MATLAB's `linkage()` function to `pd`; specify the `'average'` option;
7. display your dendrogram with this command:

```
dendrogram ( cs, 41, 'orientation', 'left', 'labels', names );
```

Can you explain some of the clusterings shown in the tree diagram?

2 Using Centers to Define Clusters

The **k-means algorithm** is used if we believe our m data values fall naturally into k separate groups, or *clusters*. To classify the data, we want to assign a cluster index `cluster(i)` to each data item `data(i,:)`. Assuming the clusters are roughly circular, then one way to do this is to pick a representative central point `center(j,:)` for each cluster and assign data items to the cluster with the nearest center, or, mathematically:

$$cluster(i) = \underset{1 \le j \le k}{\arg\min} \, ||data(i,:) - center(j,:)||$$

Given the centers, we can explain this classification as minimizing the cluster cost function:

$$E(data, center, cluster) = \sum_{i=1}^{m} ||data(i,:) - center(cluster(i),:)||$$

Exercise 2:

1. Use `csvread()` to read *faithful_data.csv*, skipping line 1 and column 1;
2. make a scatterplot of data, and notice that it seems to form two clusters;
3. define a pair of centers:

```
center = [ 2, 60;
           4, 70 ];
```

4. For each value `data(i,1:2)`, determine which center point `j` (1 or 2) is nearest, and define `cluster(i)=j`;
5. Cluster 1 points can be indexed by

```
c1 = find ( cluster == 1 );
```

6. Make a new scatterplot showing cluster 1 points in blue, and cluster 2 points in red; You will want commands like:

```
plot ( data(c1,1), data(c1,2), 'b.', 'Markersize', 15 );
```

7. Notice that some red points probably should be blue, and vice versa. This is because the data was not normalized, so horizontal and vertical distances in the graph are not the same. We will fix this in the next exercise.

3 Using K-Means To Find the Best Centers

Clearly, given our nearest-center-clustering rule, the cost function $E()$ depends on our choices for `center`, which we initialized arbitrarily. There is an iterative method, known as *k-means clustering*, which allows us to try to minimize $E()$ by improving the center values and adjusting the clustering, by replacing each cluster center by the average of its data items, and then recomputing the cluster assignments. The MATLAB function `kmeans()` will do this for us.

Exercise 3:

1. Use `csvread()` to read *faithful_data.csv*, skipping line 1 and column 1;
2. Normalize your data, so that both columns range between 0 and 1;
3. Make a scatterplot of data;
4. Request that your data be divided into two clusters by this command:

```
cluster = kmeans ( data, 2 );
```

5. Identify the data for each cluster. Cluster 1 is indexed by:

```
c1 = find ( cluster == 1 );
```

6. Compute the cluster centers. Cluster 1 center is:

```
center(1,1:2) = mean ( data(xn(c1,1:2) );
center(2,1;2) = ?;
```

7. Make a new scatterplot showing cluster 1 points in blue, and cluster 2 points in red; You will want commands like:

```
plot ( data(c1,1), data(c1,2), 'b.', 'Markersize', 15 );
```

8. Include in your scatterplot center 1 (in cyan) and center 2 (in magenta). For instance

```
plot ( center(1,1), center(1,2), 'c.', 'Markersize', 35 );
```

If your calculation and plot are done correctly, then the centers should be in the centers of their clusters, and the points in each cluster should look like they "belong" to their center.

4 Replacing Clustered Data by Cluster Centers

In the process of grouping our m pieces of data, the k-means algorithm has identified k cluster centers; every data item is relatively close to one of these centers. This suggests that the cluster centers could be take as *representatives* or *substitutes* for the original data values.

For example, we can think of a color photograph as an $m \times n \times 3$ array of colors, where we may have hundreds (8 bit), thousands (16 bit), or millions (24 bit) of colors to choose from. One way to compress an image is to reduce the number of colors used, but replacing one color by another must be done carefully to avoid ruining the image.

MATLAB has a build-in function `kmeans()` that makes our work easy; the hard part is rearranging our data so that `kmeans()` accepts it, and then un-rearranging it so that it looks like an image again!

MATLAB reads images, storing them as an $m \times n \times 3$ array of a special arithmetic type called `uint*` for "unsigned *bit integers", where "*" is 8, 16, 32 or 64. Our example will use 8 bits. The `kmeans()` function expects to operate on a 2 dimensional array of type `double`. So we need to convert our 3 dimensional array using the `double()` and `reshape()` commands, before we can operate.

Then, we want to use the information from `kmeans` to replace the original colors by their cluster representatives. Then we need to convert the resulting 2 dimensional double array back into a 3 dimensional uint8 array before trying to display it.

Exercise 4:

1. Use the `imread()` command to read *swim.jpg* into the array `swim`;
2. Use the `size()` command to determine the sizes m and n of `swim`;
3. use the `double()` function to convert `swim` to a real array;
4. use the `reshape()` function to convert `swim` to have the shape (m*n,3);
5. use the `kmeans()` function, requesting 4 clusters:

```
[ cidx, ctrs ] = kmeans ( swim, 4 );
```

6. Replace the colors by their representatives:

```
swim = ctrs(cidx,1:3);
```

7. Use the `reshape()` function to convert `swim` to have shape (m,n,3);
8. Use the `uint8()` function to convert `swim` to 8 bit integers;
9. Use the `imshow()` function to display the image, which should only use 4 colors now;