# Exercise 5

AMTH/CPSC 445a/545a - Fall Semester 2017

November 23, 2017

Compress your solutions into a single zip file titled `<lastname and initials>_assignment5.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment5.zip`. Include a single PDF titled `<lastname>_assignment5.pdf` and any MATLAB or Python scripts specified.

**Please include your name in the header of all submitted files (on every page of the PDF) and in a comment header in each of your scripts.**

Your homework should be submitted to Canvas before Friday, December 8, 2017 at 5:00 PM.

Programming assignments should use built-in functions in MATLAB or Python; In general, Python implementations may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of data mining algorithms - if a function exists that solves an entire problem (either in the MATLAB standard library or in the `scipy` stack), please consult with the TA before using it.

## Problem 1

1. Compute single-link and complete-link clustering, based on Euclidean distances between data points, for the data:

$$\begin{aligned} X = \{&(-31,-12),(-28,-18),(-27,-7),(-9,16),(-5,7),(-11,8),(-16,-8),\\ &(-10,-13),(-15,-19),(7,17),(8,6),(13,12),(18,6),(15,-3),(19,-5),\\ &(3,-10),(8,-17),(7,-8),(-11,-3),(-22,-13)\} \end{aligned}$$

   For each of the two clustering approaches, produce a dendrogram that clearly shows the order in which points/clusters are merged, and attach the two dendrograms to the submitted PDF.

2. To find $k$ clusters, we can use classic $k$-means, bisecting $k$-means, and agglomerative clustering using Ward's method, while stopping the dendrogram construction when we have exactly $k$ clusters. These three approaches are all based on the SSE as cluster quality measure. Determine which of them (if any) converges to a local minimum and which (if any) converges to a global minimum of total SSE over the clusters. Explain your answer.

3. Show that using BIRCH's clustering features one can compute the diameter

$$\text{diam}(C) = \sqrt{\frac{\sum_{x,y \in C} \|x-y\|^2}{|C|(|C|-1)}}$$

of a cluster $C$ (where $|C|$ is the number of points in the cluster), and the inter-cluster distance

$$\text{dist}(C_1, C_2) = \sqrt{\frac{\sum_{x \in C_1} \sum_{x \in C_2} \|x - y\|^2}{|C_1|\,|C_2|}}$$

between clusters $C_1$ and $C_2$ .

# Problem 2

*Notice:* for this question you are not required to submit code, but it is recommended to use a script when solving it. Consider a clustering setting where you are not getting the data points themselves as input, but just a $N \times N$ pairwise distance matrix $D$. To build an agglomerative hierarchical clustering scheme on this data we apply the following procedure:

**Initialize:**

- Set threshold $\tau \leftarrow 0$
- Create $N$ clusters at the bottom level of the dendrogram, each containing one point

**Repeat** the following steps:

- Increase $\tau$ until there is at least one new pair $x, y$ such that $D[x, y] \leq \tau$
- Create a new level in the dendrogram by merging together the clusters that are now connected by a threshold graph with edges $E = \{(x, y) \mid D_{[x,y]} \leq \tau\}$ and data points as vertices.

**Until** there is only one cluster left

1. Apply this approach to cluster data with the following distance matrix, which (for convenience) is also provided in the attached file `problem2_distances.csv`:

$$\begin{bmatrix}
0 & 90.6697 & 63.8905 & 50.3289 & 66.9104 & 30.0832 & 76.0592 & 22.4722 & 56.1427 & 14.0357 \\
90.6697 & 0 & 41.7253 & 56.4624 & 24.0000 & 61.0737 & 19.7990 & 78.4092 & 37.1618 & 76.6551 \\
63.8905 & 41.7253 & 0 & 56.8595 & 30.4138 & 41.8688 & 21.9317 & 62.0081 & 15.8114 & 51.0784 \\
50.3289 & 56.4624 & 56.8595 & 0 & 35.6090 & 26.5707 & 52.3450 & 29.5296 & 41.1461 & 40.0000 \\
66.9104 & 24.0000 & 30.4138 & 35.6090 & 0 & 37.1214 & 17.2047 & 54.5894 & 18.0278 & 52.9528 \\
30.0832 & 61.0737 & 41.8688 & 26.5707 & 37.1214 & 0 & 48.2701 & 20.2485 & 29.5466 & 16.5529 \\
76.0592 & 19.7990 & 21.9317 & 52.3450 & 17.2047 & 48.2701 & 0 & 67.6757 & 20.0250 & 62.1289 \\
22.4722 & 78.4092 & 62.0081 & 29.5296 & 54.5894 & 20.2485 & 67.6757 & 0 & 49.6488 & 18.1108 \\
56.1427 & 37.1618 & 15.8114 & 41.1461 & 18.0278 & 29.5466 & 20.0250 & 49.6488 & 0 & 42.2966 \\
14.0357 & 76.6551 & 51.0784 & 40.0000 & 52.9528 & 16.5529 & 62.1289 & 18.1108 & 42.2966 & 0
\end{bmatrix}$$

   sketch the generated dendrogram, where the vertical axis denotes the threshold $\tau$ at each level and add it to the PDF.

2. Use MDS to produce a 2D scatter plot of this data, and color it according to the top four level of the dendrogram.
   - These levels do not include the root with only one clusters, so the first level has (at least) two clusters.
   - Each of the levels should be presented in a separate figure, with the same MDS coordinates, but different colors.
   - Add these figures to the submitted PDF.

3. Does this method correspond to any of the discussed linkage models? justify your answer.

4. What if instead of adding connected components, the algorithm would only add cliques as clusters in the dendrogram? would this correspond to one of the learned linkage models? justify your answer.

# Problem 3

For this problem, you will implement a basic agglomerative clustering algorithm and compare it to the $k$-means algorithm from the previous exercise (*Note:* there is no need to resubmit your $k$-means even if you update it for this problem - you just need to submit the new agglomerative clustering code).

1. Implement single-link agglomerative clustering function `C = AggloClustering(X)`
   - The function gets as an input a $N \times n$ data matrix $X$.
   - It returns a dendrogram in a cell array $C$, where each cell corresponds to a level of the dendrogram, and should contain:
     (a) The indices of the two clusters from the previous level that were merged in this level
     (b) The single-link distance between these clusters
     (c) A $N \times 1$ vector that contains the clustering of data points at this level (i.e., by assiging cluster numbers to points)
   - The linkage should be based on Euclidean distances between data points.

2. Test the clustering on the Iris data:
   - Load the Iris dataset `iris_num.mat` (this is the same dataset used in the previous exercise)
   - Apply your agglomerative clustering to it, and use the clustering from the dendrogram level that provides three clusters.
   - Compute the accuracy of the obtained clustering. *Notice* that you first need to find the optimal mapping between cluster indices and class labels, since they might be permuted.
   - Report the obtained accuracy in the submitted PDF and compare it to the accuracies computed for k-means with this data in the previous exercise. *Notice:* the accuracy of $k$-means depends on its random initialization, as you should have seen in the previous exercise.
   - For $k$-means you were required to run the algorithm several times to assess its accuracy. Should this approach also be used with the agglomerative clustering in this exercise? Explain your answer.

3. Use your implementations of agglomerative clustering here, and $k$-means from the previous exercise to compare their runtimes:
   - Run each of the algorithms ten times on the Iris dataset.
   - For each iteration use measure the run time. For example, in MATLAB use `tic` and `toc`, or in Python use `time`.
   - Report the mean and variance of these ten runtimes for each algorithm in the submitted PDF (i.e., four numbers should be reported).
   - Which algorithm would be more efficient to use in practice (in terms of runtime)? (*hint:* in your answer, consider also the necessity for several re-executions of each algorithm to avoid bad initializations).

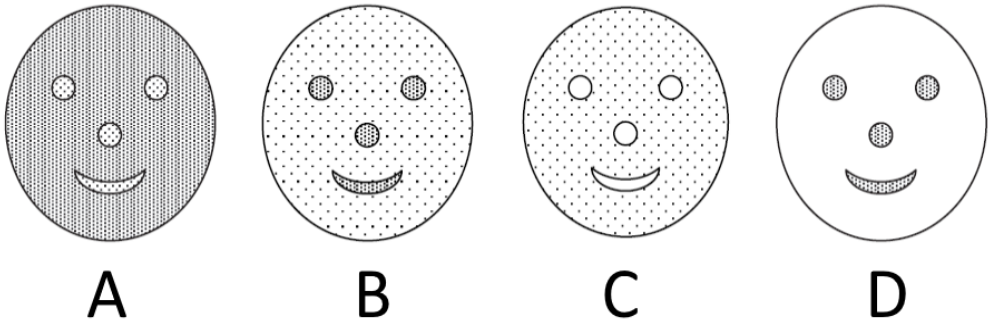
4. Compare the thoretical time complexity of the agglomerative clustering and $k$-means:
   - Compute (and report in the submitted PDF) the time complexity for the agglomerative clustering algorithm you implemented.

- Compare this complexity to the one you computed for $k$-means in the previous exercise, and explain the differences in complexity.
- Does this result agree with your measured runtime differences? Explain.

# Problem 4

1. Suppose your data is given in a sparse graph form, where edge weights represent distances, and suppose you have an efficient algorithm for finding minimal spanning trees on the data (notice that a minimal spanning tree might not be unique). Suggest a divisive (hierarchical) clustering algorithm based on this efficient algorithm. Explain your algorithm in details and justify it.

2. Consider the following four faces as two-dimensional datasets:




   where density is represented by the number of black dots (i.e., darker regions are denser) and solid lines only serve to distinguish regions (i.e., they do not rpresent data points).

   (a) For each of these dataset, determine whether $k$-means would be able to capture the patterns of the nose, eyes and mouth. Justify and explain your answer.

   (b) For each of these dataset, determine whether single link clustering would be able to capture the patterns of the nose, eyes and mouth. Justify and explain your answer.

   (c) What limitations, if any, prevent these clustering algorithms from working in cases where they cannot capture these patterns?

3. As mentioned in class, clustering is one of the tasks that is greatly affected by the curse of dimensionality. Typically, dimensiomality reduction methods are applied as preprocessing to alleviate this problem. Propose an alternative approach for applying intrinsic agglomerative clustering based on the manifold assumption and the principles used in isomap and Diffusion Maps.

# Problem 5

In this problem, you will implement isomap and test your implementation on several data sets.

1. Your isomap implementation should have the following calling sequence:

   $$\text{Y} = \text{isomap(X, epsilon, d)}$$

   where

   - $X$ is the $n \times m$ data matrix corresponding to $n$ points with $m$ attributes,

- *epsilon* is an anonymous function of the pairwise distance matrix used to set the neighborhood parameter. (The neighborhood graph should be formed by removing edges with length greater than *epsilon* from the complete distance weighted graph).
- $d$ is the output dimension parameter, and
- $Y$ is the $n \times d$ isomap embedding matrix.

2. Compare your isomap implementation to classical Multidimensional Scaling (MDS) on the `swiss_roll.mat` data set via the following steps:
   - Load `swiss_roll.mat` and create a three-dimensional scatter plot of the data $X$ colored by the vector $c$. Include the plot in your PDF.
   - Set the bandwidth function epsilon to be 3rd percentile of the positive entires of the pariwise distance matrix, i.e., in MATLAB,

     ```
     epsilon = @(D) prctile(D(D(:)>0),3);
     ```

     and in Python epsilon can be defined a similar way as lambda function using `numpy.percentile`.
   - Use classical MDS to create a two-dimensional embedding of $X$, and create a two-dimensional scatter plot of the embedding colored by the vector $c$. Include the plot in your PDF.
   - Use classical MDS to create a three-dimensional embedding of $X$, and create a three-dimensional scatter plot of the embedding colored by the vector $c$. Include the plot in your PDF.
   - Use your isomap implementation to create a two-dimensional embedding of $X$. Create a two-dimensional scatter plot of your isomap embedding colored by the vector $c$. Include the plot in your PDF.

3. Test your isomap implementation on the `torodial_helix` data set via the following steps:
   - Load `torodial_helix.mat` and create a three-dimensional scatter plot of the data $X$ colored by the vector $c$. Include this plot in your PDF.
   - Set the bandwidth function epsilon to be 3rd percentile of the positive entires of the pairwise distance matrix.
   - Use your isomap implementation to create a two-dimensional embedding of the Torodial Helix. Create a two-dimensional scatter plot of this embedding colored by the vector $c$. Include this plot in your PDF.

4. Test your isomap implementation on the `swiss_roll_hole.mat` data set via the following steps:
   - Load `swiss_roll_hole.mat` and create a three-dimensional scatter plot of the data $X$ colored by the vector $c$. Include this plot in your PDF.
   - Set the bandwidth function epsilon to be 3rd percentile of the positive entires of the pairwise distance matrix.
   - Use your isomap implementation to create a two-dimensional embedding of the data $X$. Create a two-dimensional scatter plot of this embedding colored by the vector $c$. Include this plot in your PDF.
   - Use your isomap implementation to create a three-dimensional embedding of the data $X$. Create a three-dimensional scatter plot of this embedding colored by the vector $c$. Include this plot in your PDF.

# Problem 6

In this problem you will implement the diffusion maps algorithm and test your implementation on several data sets.

1. Your diffusion maps function should have the following calling sequence:

   $$[\text{lams, Psi}] = \text{diffusionmap(X,alpha,epsilon)}$$

   where

   - X is an $n \times m$ real-valued data matrix corresponding to $n$ points with $m$ attributes,
   - *alpha* is the anisotropic normalization parameter,
   - *epsilon* is an anonymous function of the pairwise distance matrix $D$ used to set the bandwidth parameter,
   - *lams* is the $n$-dimensional vector of eigenvalues, and
   - *Psi* is the $n \times n$ matrix whose columns are the right eigenvectors of the Markov matrix underlying the diffusion maps algorithm.
   - **Notice** that the first entry of *lams* and the first column of *Psi* correspond to $\lambda_0 = 1$ and $\psi_0 = \vec{1}$ respectively.

2. Test your diffusion maps implementation on the `swiss_roll_hole.mat` data set via the following steps:

   - Load `swiss_roll_hole.mat` and create a three-dimensional scatter plot of the data $X$ colored by $c$. Include the plot in your PDF
   - Set the bandwidth function epsilon to be 0.75th percentile of the positive entires of the element-wise squared pairwise distance matrix, i.e., in MATLAB,

     ```
     epsilon = @(D) prctile(D(D(:)>0).^2,0.75);
     ```

     and in Python epsilon can be defined a similar way as lambda function using `numpy.percentile`.
   - Use your diffusion maps implementation with $alpha = 1$ to create a three-dimensional embedding of your data $X$ using $\lambda_1\psi_1$, $\lambda_2\psi_2$, and $\lambda_3\psi_3$. Create a three-dimensional scatter plot of this embedding colored by a vector $c$. Include the plot in your PDF.

3. Test your diffusion maps implementation on the `torodial_helix.mat` dataset via the following steps:

   - Load `torodial_helix.mat` and create a three-dimensional scatter plot of the data $X$ colored by the color parameter $c$. Include the plot in your PDF.
   - Set the bandwidth function epsilon to be 0.75th percentile of the positive entires of the element-wise squared pairwise distance matrix.
   - For $alpha = 0$ and $alpha = 1$ run you diffusion maps algorithm on the torodial helix data set and create a two dimensional scatter of the first two diffusion maps coordinates $\lambda_1\psi_1, \lambda_2\psi_2$ colored by the color parameter $c$. Include the two plots in your PDF.

4. Test your diffusion maps implementation on the `Bunny.mat` dataset via the following steps:

   - Load the `bunny.mat` data set, and plot the first Bunny. For example, in MATLAB

     ```
     imshow(reshape(X(1,:),sz));
     ```

     Include the plot in your PDF.
   - Set the bandwidth function epsilon to be 1.5th percentile of the positive entires of the element-wise squared pairwise distance matrix.
   - For $alpha = 0$ and $alpha = 1$ run you diffusion maps algorithm on the torodial helix data set and create a two dimensional scatter of the first two diffusion maps coordinates $\lambda_1\psi_1, \lambda_2\psi_2$ colored by the color parameter $c$. Include the two plots in your PDF.

## References

[1] "The scipy stack specification." [Online]. Available: https://www.scipy.org/stackspec.html