Connor Malley

Task 1 Suppose we have 10 college football teams X1 to X10. We want to cluster them into 2 groups. For each football team, we have two features: One is # wins in Season 2016, and the other is # wins in Season 2017.

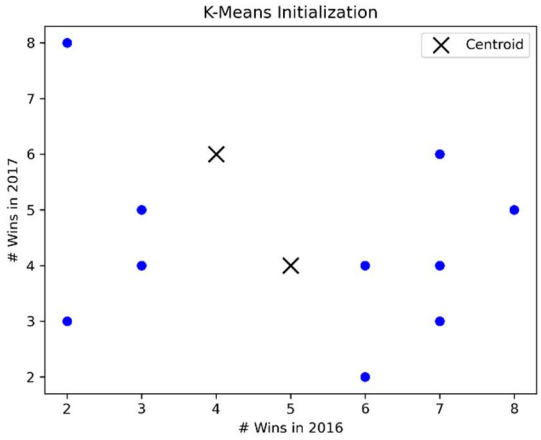
Team	# wins in Season 2016 (x-axis)	# wins in Season 2017 (y-axis)
X1	3	5
X2	3	4
X3	2	8
X4	2	3
X5	6	2
X6	6	4
X7	7	3
X8	7	4
X9	8	5
X10	7	6

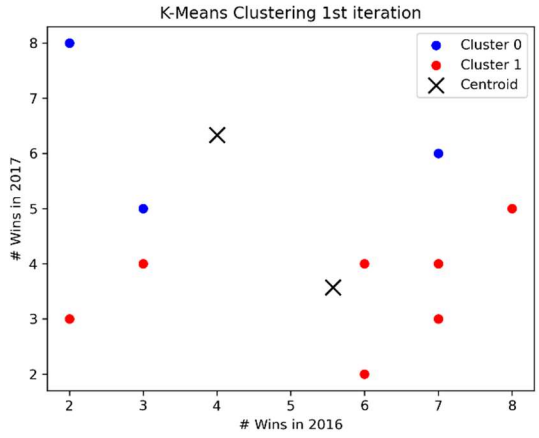
Connor Malley

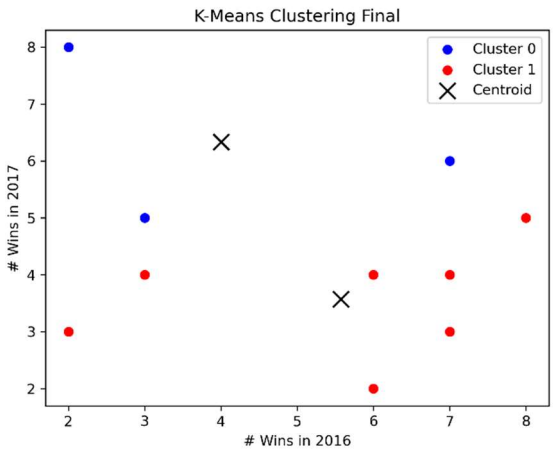
I just used my implementation of kmeans in Python for this task w/ manhattan and Euclidean distance, since I needed to implement it for task 2 anyways.

(1) Initialize with two centroids, (4, 6) and (5, 4). Use Manhattan distance as the distance metric. First, perform one iteration of the K-means algorithm and report the coordinates of the resulting centroids. Second, please use K-Means to find two clusters.

Centroids: [[4. 6.33333333], [5.57142857 3.57142857]]



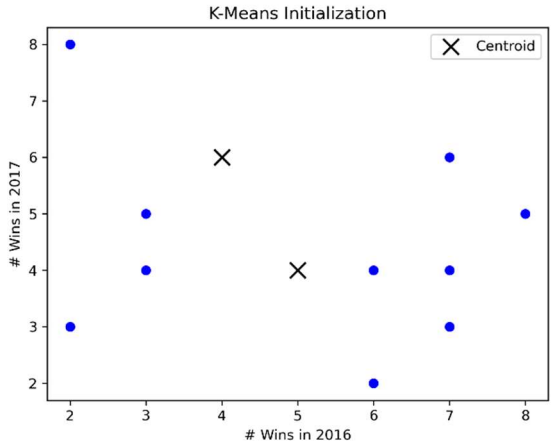


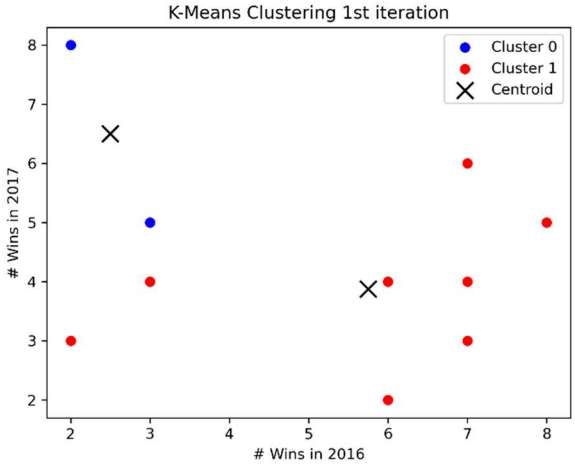


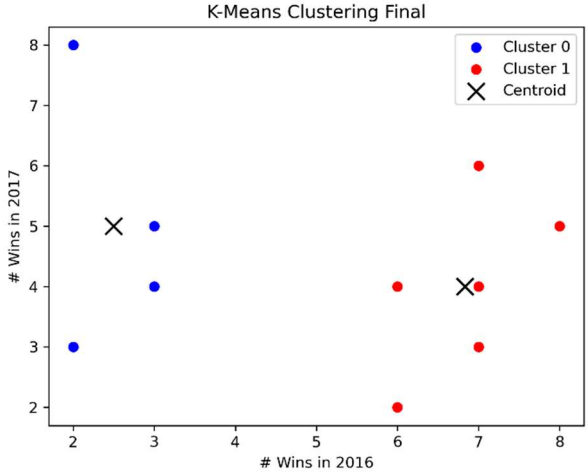
Connor Malley

(2) Initialize with two centroids, (4, 6) and (5, 4). Use Euclidean distance as the distance metric. First, perform one iteration of the K-means algorithm and report the coordinates of the resulting centroids. Second, please use K-Means to find two clusters.

Centroids: [[2.5 6.5], [5.75 3.875]]

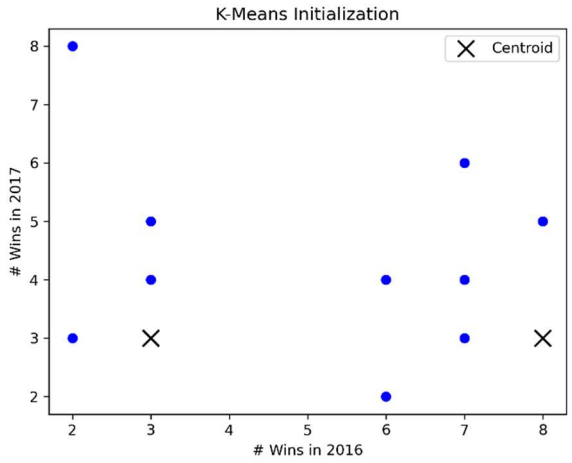


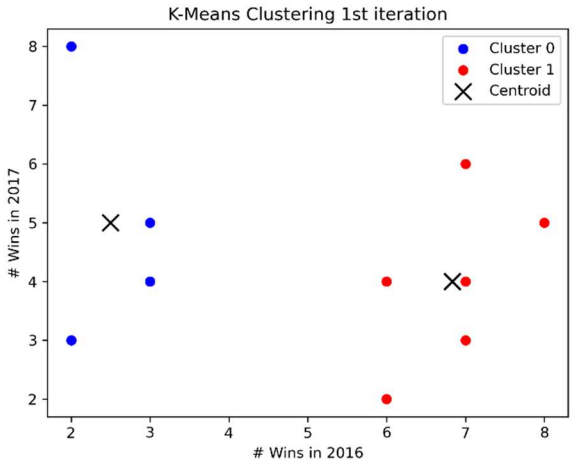


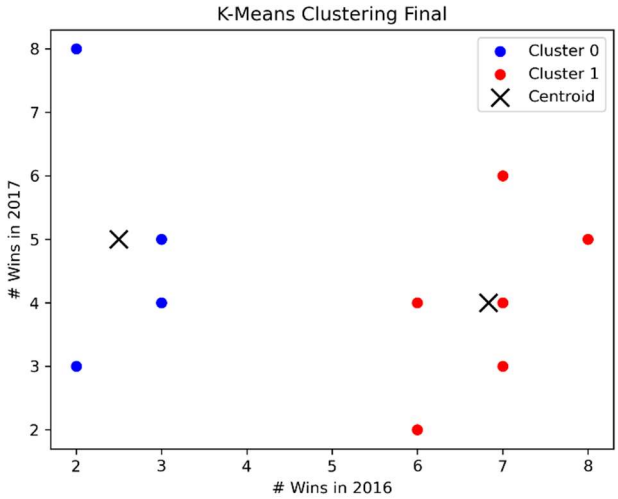


(3) Initialize with two centroids, (3, 3) and (8, 3). Use Manhattan distance as the distance metric. First, perform one iteration of the K-means algorithm and report the coordinates of the resulting centroids. Second, please use K-Means to find two clusters.

Centroids: [[2.5 5.], [6.83333333 4.]]

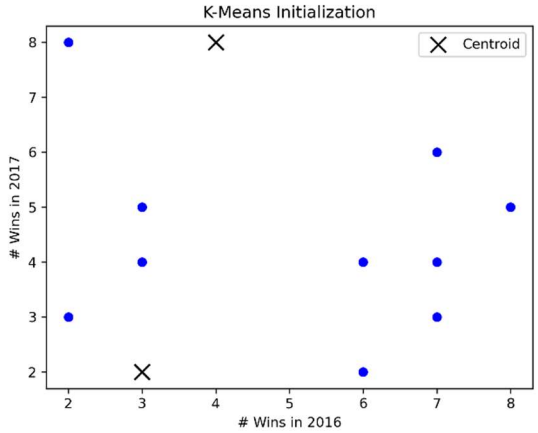


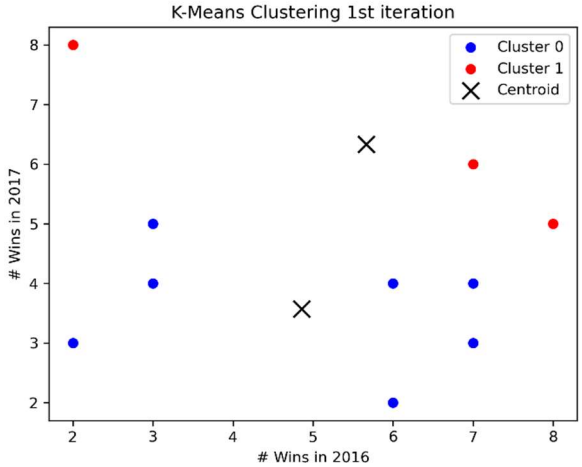


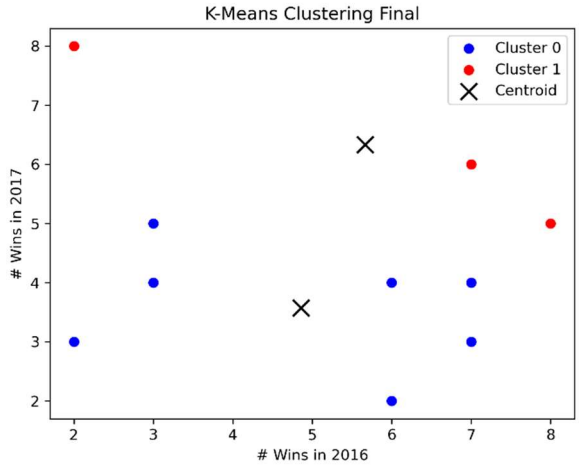


(4) Initialize with two centroids, (3, 2) and (4, 8). Use Manhattan distance as the distance metric. First, perform one iteration of the K-means algorithm and report the coordinates of the resulting centroids. Second, please use K-Means to find two clusters.

Centroids: [[4.85714286 3.57142857], [5.66666667 6.33333333]]







Connor Malley

Task 2 K-Means Clustering with Real World Dataset

First, download the Iris data set from: https://archive.ics.uci.edu/ml/datasets/Iris. Then, implement the K-means algorithm. K-means algorithm computes the distance of a given data point pair. Replace the distance computation function with Euclidean distance, 1- Cosine similarity, and 1 – the **Generalized** Jarcard similarity (https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/jaccard.htm).

Q1: Run K-means clustering with Euclidean, Cosine and Jarcard similarity. Specify K= the number of categorical values of y (the variable of label). Compare the SSEs of Euclidean-K-means Cosine-K-means, Jarcard-K-means. Which method is better?

Euclidean SSE: 7.122750

Cosine SSE: 10.663956

Jaccard SSE: 7.514386

Euclidean distance showed the lowest SSE, therefore Euclidean is the best in this case.

Used only euclidean distance in the SSE calculation. (It doesn't make sense to compare SSEs that were calculated using different distance measures. Therefore, I used the correct distances when running kmeans (if that stopping condition was selected), and only Euclidean distances when calculating the SSE after.

Q2: Compare the accuracies of Euclidean-K-means Cosine-K-means, Jarcard-K-means. First, label each cluster with the label of the highest votes. Later, compute the accuracy of the K-means with respect to the three similarity metrics. Which metric is better?

Accuracy of Euclidean: 0.880000

Accuracy of Cosine: 0.693333

Accuracy of Jaccard: 0.866667

Therefore Euclidean had the highest accuracy in this case, and is the better metric for this particular dataset

Q3: Which of Euclidean-K-means, Cosine-K-means, Jarcard-K-means requires more iterations and times?

Euclidean Iterations: 10

Cosine Iterations: 10

Jaccard Iterations: 7

Cosine and Euclidean required the most iterations, but Euclidean also was the most accurate

Q4: Compare the SSEs of Euclidean-K-means Cosine-K-means, Jarcard-K-means with respect to the following three terminating conditions:

- when there is no change in centroid position

  Iterations: 10

  Iterations: 10

  Iterations: 7

  Euclidean SSE: 7.122750

  Cosine SSE: 10.663956

  Jaccard SSE: 7.514386
- when the SSE value increases in the next iteration

  Iterations: 10
  Iterations: 4
  Iterations: 7
  Euclidean SSE: 7.122750
  Cosine SSE: 10.793332
  Jaccard SSE: 7.514386
- when the maximum preset value (100) of iteration is complete

  Iterations: 100
  Iterations: 100
  Iterations: 100
  Euclidean SSE: 7.122750
  Cosine SSE: 10.663956
  Jaccard SSE: 7.514386
- Which method requires more time or more iterations?

From the above it is clear that the preset 100 iterations required the most time, while the SSE increase required the least. In this case, the Euclidean and Jaccard iterations were equal for the centroid and SSE methods, but the cosine SSE method converged much more quickly than the cosine centroid method. Thefore , the best method, which is Euclidean, also required the most iterations in this case.

3, Understanding K-Means:

- Please give a scenario in which K-means cluster may not work very well?

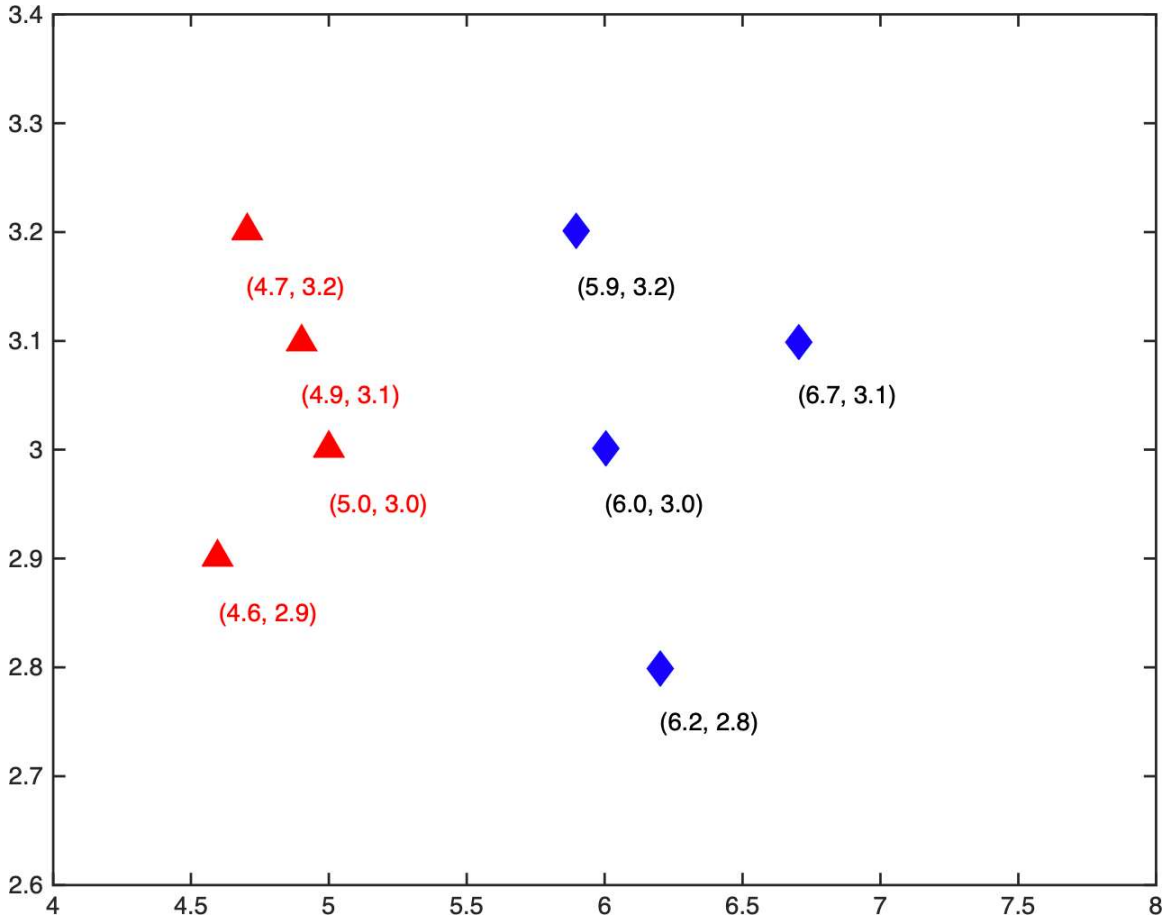
  When there are largely differing densities between the clusters. If there are two very dense clusters close together, depending on the initialization, the kmeans algorithm will most likely mark the two close and dense regions as 1 cluster. If using random initialization, the algorithm may also break down if the initialized centroids are all very far from any underlying clusters.

  (Referenced from the powerpoint)

- The classic K-means algorithm randomly initializes K centers. Is there any better strategy for selecting K initial centers?

  We could use a down-top or a top-down hierarchical clustering algorithm to initialize the centroids for k-means, giving a more accurate starting point than random initialization.

4, There are two clusters A (red) and B (blue), each has four members and plotted in Figure. The coordinates of each member are labeled in the figure. Compute the distance between two clusters using Euclidean distance.



A. What is the distance between the two farthest members? (round to four decimal places here, and next 2 problems);

See code for work

Points with max distance
[[4.6, 2.9], [6.7, 3.1]]
Max distance: 2.1095

B. What is the distance between the two closest members?

Points with min distance

[[5.0, 3.0], [5.9, 3.2]]

Min distance: 0.9220

C. What is the average distance between all pairs?

Average distance between pairs: 1.4129

D, Discuss which distance (A, B, C) is more robust to noises in this case?

Max is more robust to noise in this case since it tends to prefer globular shapes, and here we have two very well separated clusters (no outliers). (Easily linearly separable in this case)

Please submit a PDF report. In your report, please answer each question with your explanations, plots, results in brief. DO NOT paste your code or snapshot into the PDF. At the end of your PDF, please include a website address (e.g., Github, Dropbox, OneDrive, GoogleDrive) that can allow the TAto read your code

Github Link: https://github.com/malleyconnor/cap5610/tree/master/hw5