James Simmons
Fall 2021
Machine Learning
Homework 4

Task 1:

(1)

Initial Centroids:	(4, 6),	(5,4)
Distance Metric:	Manhattan	
Final Clusters:	[X1, X3, X10],	[X2, X4, X5, X6, X7, X8, X9]
Final Centroids:	(4.0, 6.3333),	(5.5714, 3.5714)
One Iter Centroids:	(4.0, 6.3333),	(5.5714, 3.5714)

Converged after 1 iteration

(2)

Initial Centroids:	(4, 6),	(5, 4)
Distance Metric:	Euclidean	
Final Clusters:	[X1, X2, X3, X4],	[X5, X6, X7, X8, X9, X10]
Final Centroids:	(2.5, 5.0),	(6.8333, 4.0)
One Iter Centroids:	(2.5, 6.5),	(5.75, 3.875)

Converged after 2 iterations

(3)

Initial Centroids:	(3, 3),	(8, 3)
Distance Metric:	Manhattan	
Final Clusters:	[X1, X2, X3, X4],	[X5, X6, X7, X8, X9, X10]
Final Centroids:	(2.5, 5.0),	(6.8333, 4.0)
One Iter Centroids:	(2.5, 5.0),	(6.8333, 4.0)

Converged after 1 iteration

(4)

Initial Centroids:	(3, 2),	(4, 8)
Distance Metric:	Manhattan	
Final Clusters:	[X1, X2, X4, X5, X6, X7, X8],	[X3, X9, X10]
Final Centroids:	(4.8571, 3.5714),	(5.6666, 6.3333)
One Iter Centroids:	(4.8571, 3.5714),	(5.6666, 6.3333)

Converged after 1 iteration

Task 2

To answer the questions for this task, I ran k-means with each method 20 times (60 total runs). For each run, I picked a random point to be the first centroid, the point farthest from that point to be the second, the point with the greatest total distance from each of the first two to be the third, and so on to select ten reasonably distant points as the initial centroids. I then collected the accuracies, SSEs, iterations needed to terminate, and the termination condition for each run.

I ran the full 60-run tests a total of 3 times, changing the termainting condidtions. In the main set, it could terminate due to exceeding 500 iterations, the SSE increased, or the centroids didn't change. In another set, SSE termination was disallowed. In the thrid, centroid termination was disallowed. The information presented in Q1, Q2, and Q3 uses the main set. Q4 explores their differences.

Q1) Compare SSEs across methods.

To calculate SSE, we need to find the difference between a point and the center. Problem: which difference metric do we use? Since I couldn't decide this, I just did all 9 combinations. Below are the averages across the 20 runs.

		K-means Optimizing For:		
		Euclidean	Cosine	Jaccard
SSE Using:	Euclidean	$2.5485 * 10^{10}$	$2.5512 * 10^{10}$	$2.5464 * 10^{10}$
	Cosine	725.8	687.0	687.7
	Jaccard	3770.2	3671.3	3668.6

We would expect that the best SSE for a particular metric would be with the k-means that optimized for it, and that it wouldn't be close, but this is not the case. Curiously, all three metrics had a euclidean SSE within 0.5% of each other, and jaccard actually did better at euclidean SSE than euclidean did! Indeed, cosine and jaccard both had very similar results for all three SSEs, and euclidean was by far the worst at both cosine SSE and jaccard SSE. Note that this is the average over 20 runs; any close values are in fact within each other's margins of deviation. Therefore, either jaccard or cosine are the best methods.

Q2) Compare accuracies across methods.

The min, max, and average accuracies for each of the methods was as follows.

Euclidean:	**min 53.13%**	**max 61.12%**	**average 59.13%**
Cosine:	**min 53.63%**	**max 63.46%**	**average 59.85%**
Jaccard:	**min 54.04%**	**max 62.11%**	**average 59.97%**

Overall, the accuracies are extremely similar, and well within each other's margins of deviation.

The average accuracy was also around 59-60% for both the run where the terminating condition was restricted to SSE only and the run where it was restricted to centroid only (see Q4).

Q3) Which method requires more iterations to converge?

The min, max, and average number of iterations needed to terminate for each of the methods was as follows.

Euclidean:	**min 30**	**max 134**	**average 58.40**
Cosine:	**min 13**	**max 125**	**average 42.70**
Jaccard:	**min 12**	**max 62**	**average 31.90**

None of the methods reached the iteration limit of 500. Had the limit been 100, euclidean would have reached it 2 times (10%), cosine one time (5%) and jaccard still 0 times. Euclidean terminated primarily due to centroids, 18 times (90%), with the other 2 being SSE (10%). Cosine was the opposite, with 18 times SSE and 2 times centroid. Jaccard terminated via SSE every time.

Q4) Which termination condition did each method use?

The SSE used as a terminating condition was the same one used for distance. i.e. euclidean k-means used euclidean SSE, cosine k-means used cosine SSE, etc.

For the data presented in Q1, Q2, and Q3, the program could terminate on SSE, centroid, or iteration. I also ran the full 60 run set again two more times; once where it wasn't allowed to terminate on centroid, and once where it wasn't allowed to terminate on SSE. In all cases, the iteration cap was 500.

Iterations to converge using centroid only:

Euclidean:	**min 30**	**max 97**	**average 56.30**
Cosine:	**min 32**	**max 127**	**average 67.75**
Jaccard:	**min 26**	**max 121**	**average 75.85**

All methods terminated on centroid before the iteration limit of 500. Had the iteration limit been 100 instead, it would have terminated cosine 2 times (10%) and jaccard 7 times (35%).

Iterations to converge using SSE only:

Euclidean:	**min 32**	**max 500**	**average 366.55**
Cosine:	**min 17**	**max 500**	**average 65.30 (42.42 if you ignore the 1 500 outlier)**
Jaccard:	**min 19**	**max 55**	**average 32.5**

Euclidean reached the iteratation limit of 500 14 times (70%), and exceeded 100 iterations only those same 14 times. Cosine reached the iteration limit only once (5%) and also didn't exceed 100 any other time. Jaccard always terminated before 100 iterations.

As can be seen by these tests, and also inferred by the termination-cause splits in Q3 when all three conditions were allowed, the preference order is as follows.

Euclidean:	**Centroid > Iteration > SSE**	**(SSE works well or not at all, no in-between)**
Cosine:	**SSE > Centroid > Iteration**	
Jaccard:	**SSE > Centroid > Iteration**	**(same order, but more extreme than cosine)**

Q5) Summary observations / takaways

All three batches with varying terminating conditions still had an overall accuracy of about 60%, and were always well within each other's margins of deviation. We would expect the fastest runtimes are with the fewest iterations, and the fewest iterations are with the most terminating conditions. When all terminating conditions are enabled, jaccard has the fewest iterations, then cosine, then euclidean is the most. Jaccard and cosine were practically tied for SSE, and all three were practically tied for accuracy, so the best method overall is jaccard.

Interestingly, it is not the case that fewer iterations results in faster runtimes. Since jaccard requires max and min operations, which are slow, it actually took about 25% longer than cosine to compute the 20 runs (15:46 vs 12:42), despite having 25% fewer iterations on average (31.9 vs 42.7). Based on real-world runtime, not number of iterations, cosine is actually the best.

Task 3

A) Max Distance:

The farthest members are (4.6, 2.9) and (6.7, 3.1), with a distance of 2.105

B) Min Distance:

The closest members are (5.0, 3.0) and (5.9, 3.2), with a distance of 0.9220

C) Group Average Distance:

The average distance between all pairs is 1.4129

D) Robustness to noise:

Max Distance is sometimes completely unaffected by noise (i.e. when the distance between a blue noise point is closer to each red point than the max distance without noise), but when it is affected, it totally changes distance value. That said, it can only make the value bigger, so it still functions as an upper bound.

Min Distance follows the exact same logic as Max Distance. Sometimes unaffected, but when it is affected it makes a big difference. Still functions as a lower bound.

The Group Average Distance is always affected by noise, but so long as the true points vastly outnumber the noise points (which they typically do), the affect should be fairly minor. For this reason, Group Average Distance is the most robust to noise overall.

<u>All Distances</u>

(5.0, 3.0) - (5.9, 3.2): 0.922
(5.0, 3.0) - (6.0, 3.0): 1.0
(4.9, 3.1) - (5.9, 3.2): 1.005
(4.9, 3.1) - (6.0, 3.0): 1.1045
(4.7, 3.2) - (5.9, 3.2): 1.2
(5.0, 3.0) - (6.2, 2.8): 1.2166
(4.7, 3.2) - (6.0, 3.0): 1.3153
(4.9, 3.1) - (6.2, 2.8): 1.3342
(4.6, 2.9) - (5.9, 3.2): 1.3342
(4.6, 2.9) - (6.0, 3.0): 1.4036
(4.7, 3.2) - (6.2, 2.8): 1.5524
(4.6, 2.9) - (6.2, 2.8): 1.6031
(5.0, 3.0) - (6.7, 3.1): 1.7029
(4.9, 3.1) - (6.7, 3.1): 1.8
(4.7, 3.2) - (6.7, 3.1): 2.0025
(4.6, 2.9) - (6.7, 3.1): 2.1095

Additional Questions

Hours Spent: 15

Greatest Challenge: My initial programmatic implementation of k-means was unusably slow for a dataset the size of task 2, so I had to reimplement the whole thing from scratch *again*, using numpy to speed up the calculations. A large part of my new implementation relies heavily on numpy matrix broadcasting in order to do “all-pairs” calculations, such as finding the distance between each and each center. There are definitely some optimizations that could be made (namely that I have to iterate over each of the 10 clusters and work with them individually), but I got it working acceptably fast after much research and trial and error.

Most Enjoyed: I found it really funny that jaccard k-means had a better euclidean SSE than euclidean k-means did. I also had a variety of small revelations along the way. Nothing especially major stood out, though.

Desired Changes: I think task 2 would be much more interesting if it were real data, rather than a bunch of arbitrary numbers. Being able to connect with what your code is doing is a big part of feeling like you’re doing it correctly.

Code:

All code available at: https://github.com/Mesaj2000/Machine_Learning_5610/tree/master/hw4

This includes the console output from the 20 runs for task 2, for each of the three sets of terminating conditions.