CSC 535 Data Mining

Assignment 4 Report Collection

Submitted to:

Dr. Jamil Saquer

Author(s):

*Chase Dickerson*

*Jacob Schaum*

**Report**

**Introduction**

For this assignment we implemented the K means algorithm for clustering. The dataset we used was called synthetic\_2D.txt and was provided for us to use for this assignment specifically. The data set contained 500 entries. Each entry consisted of an x value, a y value, and cluster that entry should belong to. The cluster was included so we could use it to test the accuracy of our algorithm.

**Background**

The K means algorithm is used for partitional clustering. For our implementation, we randomly selected the “k” (in our case 3) initial means. We then iteratively move items among the set of clusters until all three means are the same. We used Euclidean Distance to find the distance between points. The distances are used to determine which cluster to put the points in.

The K means algorithm has a running time of O(tkn) were t is the number of iterations needed to find the same means, k is the number of clusters, and n is the number of objects to cluster. K means usually terminates at a local optimum.

**Implementation**

For our implementation, we got three random means to use as the initial means. We then implemented a loop with a condition of seeing if our new means matched the previous ones. Within out loop, we found the clusters (Figure 1), got the means of each cluster, and then checked the termination condition. We determined clusters by finding the smallest distances using the Euclidean Distance formula.

About the only thing we struggled with was matching the output required in the assignment and finding the accuracy. We struggled to see if a cluster was in the correct cluster but eventually figured it out.

**def** clusterData**(**mean1**,** mean2**,** mean3**,** dataset**):**

# Using euclidean distance with the data's x,y and mean's x,y

clusters **=** **{**

0**:** **[],**

1**:** **[],**

2**:** **[]**

**}**

# Get distance from data to mean1,2,3 and add the data to the cluster

# with the smallest distance.

**for** data **in** dataset**:**

distance **=** **{**

0**:** euclideanDistance**(**mean1**,** data**),**

1**:** euclideanDistance**(**mean2**,** data**),**

2**:** euclideanDistance**(**mean3**,** data**)**

**}**

clusters**[**min**(**distance**,**key**=**distance**.**get**)].**append**(**data**)**

**return** clusters

Figure 1: Finding the clusters

**Experimental Setup and Results**

Our results were pretty consistent. We got around 88% accuracy on correct clusters which we were happy with. We tested using different random seeds and different variations of initial random means and still got the same results. Our output is provided below:

Initial k means are

mean[0] is ((8.61, 13.0), 2)

mean[1] is ((4.11, 9.5), 0)

mean[2] is ((10.93, 8.03), 1)

=====================

Cluster 0

Size of cluster 0 is 167

Cluster label 2

Number of objects misclustered in the cluster is 28

((4.49, 17.14), 0)

((6.8, 15.44), 0)

((4.84, 13.17), 0)

((5.77, 14.46), 0)

((5.06, 14.86), 0)

((3.68, 13.54), 0)

((5.52, 12.82), 0)

((3.98, 14.4), 0)

((4.54, 13.46), 0)

((5.2, 13.67), 0)

((4.72, 13.48), 0)

((4.99, 14.87), 0)

((6.35, 14.32), 0)

((4.5, 13.5), 0)

((5.15, 13.84), 0)

((6.87, 12.35), 0)

((4.63, 13.31), 0)

((7.21, 12.63), 1)

((7.07, 13.03), 1)

((8.24, 12.68), 1)

((7.55, 16.65), 1)

((10.37, 15.49), 1)

((9.39, 13.35), 1)

((9.2, 13.64), 1)

((10.84, 14.67), 1)

((10.82, 15.06), 1)

((9.73, 14.3), 1)

((8.06, 12.82), 1)

((7.1, 13.76), 2)

((7.03, 13.02), 2)

((6.44, 17.19), 2)

((6.62, 14.34), 2)

((7.17, 13.48), 2)

((6.35, 18.57), 2)

((7.06, 14.84), 2)

((9.5, 17.72), 2)

((6.83, 13.93), 2)

((6.69, 15.43), 2)

((6.61, 13.08), 2)

((9.9, 15.52), 2)

((6.83, 15.2), 2)

((9.61, 14.25), 2)

((6.49, 13.81), 2)

((6.93, 16.8), 2)

((7.3, 14.56), 2)

((7.47, 14.81), 2)

((7.8, 17.98), 2)

((7.55, 14.37), 2)

((7.58, 16.77), 2)

((7.05, 14.2), 2)

((7.85, 15.25), 2)

((6.03, 15.34), 2)

((8.49, 14.69), 2)

((7.32, 14.68), 2)

((8.47, 16.37), 2)

((6.98, 15.4), 2)

((7.25, 14.58), 2)

((7.89, 14.39), 2)

((8.82, 19.94), 2)

((6.3, 17.04), 2)

((8.49, 16.97), 2)

((7.39, 15.45), 2)

((7.72, 17.09), 2)

((8.42, 14.22), 2)

((7.76, 15.97), 2)

((7.81, 12.2), 2)

((8.68, 20.18), 2)

((7.76, 14.97), 2)

((8.12, 13.04), 2)

((6.74, 13.68), 2)

((6.06, 15.84), 2)

((9.33, 14.36), 2)

((7.47, 14.14), 2)

((8.42, 12.95), 2)

((7.43, 19.36), 2)

((7.21, 16.18), 2)

((7.85, 13.33), 2)

((7.34, 16.08), 2)

((5.94, 16.09), 2)

((7.89, 15.02), 2)

((9.24, 16.63), 2)

((8.03, 13.24), 2)

((8.6, 14.88), 2)

((7.35, 13.87), 2)

((7.8, 13.15), 2)

((7.78, 18.36), 2)

((8.89, 15.65), 2)

((8.49, 13.64), 2)

((5.58, 18.59), 2)

((8.65, 15.5), 2)

((7.13, 16.72), 2)

((7.65, 18.07), 2)

((6.0, 14.51), 2)

((8.24, 13.89), 2)

((7.64, 19.04), 2)

((7.25, 17.17), 2)

((7.99, 12.73), 2)

((8.55, 13.39), 2)

((8.9, 13.9), 2)

((8.96, 15.3), 2)

((8.3, 13.78), 2)

((7.15, 14.09), 2)

((6.33, 16.12), 2)

((7.22, 19.57), 2)

((8.72, 15.79), 2)

((7.75, 15.84), 2)

((8.3, 15.2), 2)

((6.23, 14.7), 2)

((7.99, 16.1), 2)

((6.74, 15.16), 2)

((7.32, 16.87), 2)

((7.56, 14.81), 2)

((8.0, 12.97), 2)

((7.6, 14.71), 2)

((7.05, 17.31), 2)

((7.42, 13.65), 2)

((6.75, 16.08), 2)

((7.15, 17.05), 2)

((9.09, 13.52), 2)

((6.61, 18.41), 2)

((6.11, 14.51), 2)

((5.53, 13.59), 2)

((7.31, 14.33), 2)

((8.54, 14.15), 2)

((8.5, 12.96), 2)

((8.7, 14.64), 2)

((8.61, 13.0), 2)

((6.55, 14.59), 2)

((8.66, 15.01), 2)

((7.08, 14.97), 2)

((6.09, 14.47), 2)

((7.84, 13.89), 2)

((7.57, 15.27), 2)

((8.71, 13.37), 2)

((7.17, 14.47), 2)

((6.85, 13.75), 2)

((6.93, 13.21), 2)

((6.91, 14.67), 2)

((8.36, 20.3), 2)

((9.1, 14.28), 2)

((6.66, 15.66), 2)

((7.21, 17.5), 2)

((7.31, 12.48), 2)

((7.38, 19.62), 2)

((8.19, 17.36), 2)

((8.23, 13.79), 2)

((8.68, 14.43), 2)

((7.32, 12.03), 2)

((8.29, 18.55), 2)

((7.84, 16.45), 2)

((5.81, 13.94), 2)

((6.1, 15.0), 2)

((6.85, 14.73), 2)

((7.87, 16.04), 2)

((8.18, 16.82), 2)

((5.95, 14.44), 2)

((6.69, 13.76), 2)

((8.56, 13.75), 2)

((6.02, 15.55), 2)

((8.98, 15.09), 2)

((8.69, 14.99), 2)

((6.57, 15.38), 2)

((7.34, 15.46), 2)

((7.35, 18.24), 2)

((7.54, 18.27), 2)

((7.77, 17.17), 2)

((8.93, 14.27), 2)

=====================

Cluster 1

Size of cluster 1 is 208

Cluster label 0

Number of objects misclustered in the cluster is 25

((5.94, 8.54), 0)

((3.6, 10.52), 0)

((4.32, 11.34), 0)

((5.37, 12.05), 0)

((3.98, 7.44), 0)

((4.93, 10.21), 0)

((5.18, 10.14), 0)

((4.17, 5.85), 0)

((3.69, 8.27), 0)

((5.19, 10.06), 0)

((5.99, 6.44), 0)

((4.35, 10.75), 0)

((4.67, 12.59), 0)

((6.65, 7.41), 0)

((4.44, 12.93), 0)

((7.4, 9.44), 0)

((3.47, 9.38), 0)

((5.8, 9.1), 0)

((3.0, 4.18), 0)

((4.4, 7.54), 0)

((6.5, 11.74), 0)

((6.22, 9.44), 0)

((4.1, 11.27), 0)

((4.55, 9.97), 0)

((5.08, 9.87), 0)

((3.74, 11.08), 0)

((5.55, 7.8), 0)

((7.23, 10.82), 0)

((3.64, 10.58), 0)

((3.02, 7.59), 0)

((5.29, 6.72), 0)

((4.88, 8.96), 0)

((4.95, 12.31), 0)

((4.81, 9.34), 0)

((4.01, 10.5), 0)

((5.67, 9.66), 0)

((3.68, 12.17), 0)

((6.17, 11.05), 0)

((5.01, 6.16), 0)

((5.5, 11.62), 0)

((4.45, 11.8), 0)

((4.08, 8.92), 0)

((6.8, 8.5), 0)

((5.47, 11.12), 0)

((6.21, 8.84), 0)

((5.19, 10.57), 0)

((7.61, 10.07), 0)

((5.36, 8.93), 0)

((3.97, 7.68), 0)

((5.43, 8.44), 0)

((2.68, 10.49), 0)

((4.88, 9.69), 0)

((5.98, 9.98), 0)

((5.8, 6.89), 0)

((4.66, 8.23), 0)

((3.79, 7.38), 0)

((5.49, 12.34), 0)

((3.85, 8.36), 0)

((6.32, 8.27), 0)

((4.69, 11.79), 0)

((4.08, 10.7), 0)

((4.43, 12.57), 0)

((4.19, 11.14), 0)

((4.44, 10.06), 0)

((4.26, 8.23), 0)

((4.62, 13.06), 0)

((5.24, 9.64), 0)

((5.6, 8.22), 0)

((3.89, 9.58), 0)

((4.05, 9.6), 0)

((4.57, 12.92), 0)

((5.1, 9.51), 0)

((2.96, 11.94), 0)

((6.7, 8.41), 0)

((4.11, 9.5), 0)

((6.82, 8.53), 0)

((3.65, 10.2), 0)

((4.04, 11.2), 0)

((4.75, 10.38), 0)

((4.78, 9.24), 0)

((4.23, 9.89), 0)

((5.73, 8.54), 0)

((3.2, 11.02), 0)

((6.06, 9.21), 0)

((4.18, 10.47), 0)

((6.25, 11.9), 0)

((4.59, 8.77), 0)

((3.6, 12.96), 0)

((5.45, 5.4), 0)

((7.23, 8.73), 0)

((4.95, 11.9), 0)

((5.11, 9.34), 0)

((4.6, 8.0), 0)

((4.12, 8.96), 0)

((4.25, 13.09), 0)

((3.59, 7.23), 0)

((5.07, 8.64), 0)

((6.62, 8.61), 0)

((5.47, 9.55), 0)

((5.25, 10.77), 0)

((4.14, 10.34), 0)

((5.19, 10.0), 0)

((4.58, 9.29), 0)

((5.94, 12.0), 0)

((6.23, 10.1), 0)

((6.25, 7.83), 0)

((5.37, 9.74), 0)

((6.04, 7.35), 0)

((4.34, 10.19), 0)

((4.9, 9.1), 0)

((4.79, 8.11), 0)

((4.1, 5.96), 0)

((4.12, 9.76), 0)

((3.08, 8.29), 0)

((7.04, 10.79), 0)

((5.05, 11.56), 0)

((5.62, 8.72), 0)

((4.84, 5.87), 0)

((5.49, 7.13), 0)

((5.48, 10.9), 0)

((5.74, 11.48), 0)

((2.77, 11.37), 0)

((6.04, 10.16), 0)

((5.15, 10.57), 0)

((5.62, 11.32), 0)

((5.46, 9.55), 0)

((5.26, 10.81), 0)

((6.42, 11.06), 0)

((4.63, 8.19), 0)

((5.51, 11.74), 0)

((7.4, 11.13), 0)

((6.77, 9.88), 0)

((5.29, 7.49), 0)

((5.9, 11.68), 0)

((4.78, 11.68), 0)

((3.1, 8.37), 0)

((5.53, 11.53), 0)

((4.7, 9.12), 0)

((3.78, 7.65), 0)

((2.7, 10.02), 0)

((6.4, 9.35), 0)

((5.54, 8.82), 0)

((5.52, 8.92), 0)

((4.24, 9.11), 0)

((4.99, 7.25), 0)

((4.42, 12.97), 0)

((5.22, 7.81), 0)

((3.99, 11.11), 0)

((5.18, 10.25), 0)

((3.92, 9.85), 0)

((4.65, 11.45), 0)

((5.87, 7.79), 0)

((5.4, 9.28), 0)

((4.35, 6.44), 0)

((4.85, 10.77), 0)

((4.94, 10.34), 0)

((4.95, 10.39), 0)

((4.1, 9.58), 0)

((5.78, 10.09), 0)

((5.23, 10.57), 0)

((7.47, 10.37), 0)

((4.51, 9.43), 0)

((5.66, 11.04), 0)

((3.46, 11.74), 0)

((4.07, 12.33), 0)

((3.96, 9.54), 0)

((4.79, 11.92), 0)

((4.03, 8.62), 0)

((3.75, 7.31), 0)

((5.74, 8.65), 0)

((5.65, 11.14), 0)

((5.01, 6.83), 0)

((5.64, 9.8), 0)

((4.37, 11.13), 0)

((4.26, 9.24), 0)

((5.92, 7.36), 0)

((4.09, 9.4), 0)

((4.13, 11.05), 0)

((5.52, 7.96), 0)

((4.69, 10.51), 0)

((4.01, 8.08), 0)

((4.84, 6.66), 0)

((4.74, 7.98), 0)

((6.95, 10.76), 1)

((6.02, 11.62), 1)

((7.67, 6.5), 1)

((7.43, 11.33), 1)

((5.88, 9.88), 1)

((7.46, 9.92), 1)

((5.75, 11.39), 1)

((7.83, 10.57), 1)

((7.88, 6.63), 1)

((7.8, 10.7), 1)

((5.71, 11.03), 1)

((7.76, 10.6), 1)

((8.02, 9.48), 1)

((7.69, 10.49), 1)

((7.82, 10.15), 1)

((6.78, 10.41), 1)

((7.51, 11.45), 1)

((7.11, 9.84), 1)

((6.88, 11.27), 1)

((6.5, 9.95), 2)

((6.83, 12.1), 2)

((7.19, 11.85), 2)

((7.01, 11.27), 2)

((6.74, 11.8), 2)

((5.94, 12.35), 2)

=====================

Cluster 2

Size of cluster 2 is 125

Cluster label 1

Number of objects misclustered in the cluster is 5

((10.87, 12.31), 1)

((11.27, 10.45), 1)

((10.0, 10.36), 1)

((8.47, 9.19), 1)

((9.93, 11.65), 1)

((13.63, 11.66), 1)

((11.59, 10.78), 1)

((11.59, 9.25), 1)

((9.4, 12.13), 1)

((11.99, 9.26), 1)

((12.65, 11.48), 1)

((11.09, 12.6), 1)

((11.15, 6.18), 1)

((11.18, 11.89), 1)

((10.33, 11.46), 1)

((9.41, 9.46), 1)

((12.03, 8.15), 1)

((10.47, 9.91), 1)

((10.2, 8.91), 1)

((9.39, 8.25), 1)

((13.41, 9.63), 1)

((14.79, 11.63), 1)

((11.46, 12.36), 1)

((9.53, 12.39), 1)

((10.41, 12.74), 1)

((8.87, 9.51), 1)

((10.21, 9.25), 1)

((12.71, 11.03), 1)

((10.56, 11.12), 1)

((9.35, 10.13), 1)

((9.31, 8.36), 1)

((9.44, 11.08), 1)

((12.32, 8.2), 1)

((10.25, 10.67), 1)

((8.27, 10.39), 1)

((8.57, 8.83), 1)

((8.78, 9.53), 1)

((8.66, 7.96), 1)

((12.15, 11.37), 1)

((12.13, 14.37), 1)

((13.07, 14.01), 1)

((13.16, 13.86), 1)

((11.36, 12.49), 1)

((12.1, 8.4), 1)

((10.89, 10.26), 1)

((7.75, 11.56), 1)

((10.29, 10.97), 1)

((8.74, 12.63), 1)

((10.43, 11.14), 1)

((10.09, 10.56), 1)

((10.19, 11.45), 1)

((9.23, 9.35), 1)

((14.48, 9.68), 1)

((11.29, 13.08), 1)

((12.79, 12.07), 1)

((8.99, 12.77), 1)

((8.74, 10.4), 1)

((13.08, 8.49), 1)

((12.05, 9.74), 1)

((12.66, 8.98), 1)

((9.34, 9.65), 1)

((9.67, 11.63), 1)

((10.92, 7.53), 1)

((11.16, 12.86), 1)

((14.15, 10.19), 1)

((9.21, 10.36), 1)

((10.46, 9.15), 1)

((10.49, 10.08), 1)

((12.14, 10.08), 1)

((10.42, 9.37), 1)

((9.81, 8.96), 1)

((9.54, 11.35), 1)

((14.48, 7.95), 1)

((9.86, 9.79), 1)

((11.71, 12.57), 1)

((10.81, 11.62), 1)

((11.82, 10.98), 1)

((10.34, 10.34), 1)

((15.4, 10.67), 1)

((10.34, 10.69), 1)

((10.32, 6.88), 1)

((9.31, 11.75), 1)

((11.07, 13.44), 1)

((9.15, 11.16), 1)

((12.87, 12.63), 1)

((10.95, 9.72), 1)

((9.61, 7.97), 1)

((11.92, 10.89), 1)

((10.09, 10.8), 1)

((12.69, 13.69), 1)

((10.42, 11.71), 1)

((9.0, 10.99), 1)

((11.4, 10.0), 1)

((11.19, 7.94), 1)

((11.02, 11.49), 1)

((11.86, 10.78), 1)

((9.6, 11.92), 1)

((8.36, 9.26), 1)

((10.28, 11.44), 1)

((14.65, 12.03), 1)

((9.16, 12.71), 1)

((9.04, 9.71), 1)

((10.93, 8.03), 1)

((9.96, 12.02), 1)

((8.98, 9.96), 1)

((9.43, 10.84), 1)

((13.86, 12.83), 1)

((10.58, 11.49), 1)

((8.36, 9.08), 1)

((10.15, 11.69), 1)

((13.35, 9.14), 1)

((9.47, 11.25), 1)

((12.02, 12.3), 1)

((11.02, 13.47), 1)

((10.31, 10.02), 1)

((10.8, 12.89), 1)

((10.1, 11.61), 1)

((11.87, 11.08), 1)

((10.93, 11.84), 1)

((8.97, 11.63), 1)

((8.38, 11.44), 2)

((8.01, 11.95), 2)

((9.36, 12.6), 2)

((9.01, 12.31), 2)

((9.08, 12.64), 2)

88.4%

**Conclusion**

In conclusion, we learned a lot about the K Means algorithm and how it can be used to cluster objects. It is a very popular clustering algorithm for good reason because it is reliable and efficient.

**Code**

# Program: HW4.py

# Developer: Chase Dickerson

# Date: 11/14/2019

# Purpose: Implment the K means algorithm

**from** \_\_future\_\_ **import** division

**import** random

**from** collections **import** OrderedDict

**import** collections

**import** math

**import** time

**import** operator

**global** correct

correct **=** 0

**def** euclideanDistance**(**cluster**,** data**):**

**return** math**.**sqrt**(((**float**(**cluster**[**0**])-**float**(**data**[**0**]))\*\***2**)+(**float**(**cluster**[**1**])-**float**(**data**[**1**]))\*\***2**)**

**def** clusterData**(**mean1**,** mean2**,** mean3**,** dataset**):**

# Using euclidean distance with the data's x,y and mean's x,y

clusters **=** **{**

0**:** **[],**

1**:** **[],**

2**:** **[]**

**}**

# Get distance from data to mean1,2,3 and add the data to the cluster

# with the smallest distance.

**for** data **in** dataset**:**

distance **=** **{**

0**:** euclideanDistance**(**mean1**,** data**),**

1**:** euclideanDistance**(**mean2**,** data**),**

2**:** euclideanDistance**(**mean3**,** data**)**

**}**

clusters**[**min**(**distance**,** key**=**distance**.**get**)].**append**(**data**)**

**return** clusters

**def** calculateMean**(**clusters**):**

mean1 **=** **None**

mean2 **=** **None**

mean3 **=** **None**

# Loop through x and y and get new mean

xTotal1 **=** 0

yTotal1 **=** 0

**for** data **in** clusters**[**0**]:**

xTotal1 **+=** float**(**data**[**0**])**

yTotal1 **+=** float**(**data**[**1**])**

mean1 **=** **[**xTotal1**/**len**(**clusters**[**0**]),** yTotal1**/**len**(**clusters**[**0**]),** 0**]**

# Loop through x and y and get new mean

xTotal2 **=** 0

yTotal2 = 0

for data in clusters[1]:

xTotal2 += float(data[0])

yTotal2 += float(data[1])

mean2 = [xTotal2/len(clusters[1]), yTotal2/len(clusters[1]), 1]

# Loop through x and y and get new mean

xTotal3 = 0

yTotal3 = 0

for data in clusters[2]:

xTotal3 += float(data[0])

yTotal3 += float(data[1])

mean3 = [xTotal3/len(clusters[2]), yTotal3/len(clusters[2]), 2]

return mean1, mean2, mean3

############

def getAccuracy(clusters):

# Loop through each cluster, tally if the data is in right cluster

correct = 0

total = 0

for i in range(3):

for data in clusters[i]:

total += 1

if int(data[2]) == i:

correct += 1

accuracy = int(correct) / int(total)

print("Accuracy: " + str(accuracy\*100) + "%")

def misclustered(clusters, i):

incorrect = 0

for data in clusters:

if int(data[2]) != i:

incorrect += 1

return incorrect

#############

def calcAccuracy(clusters):

# Print majority cluster, size, and incorrect

# Loop through each cluster

clusterDict = {

'0': 0,

'1': 0,

'2': 0

}

# Loop through each cluster and get total

for data in clusters:

clusterDict[data[2]] += 1

maxCluster = max(clusterDict.items(), key=operator.itemgetter(1))[0]

numCorrect = clusterDict[maxCluster]

global correct

print("Cluster label", maxCluster)

correct += numCorrect

total = clusterDict['0'] + clusterDict['1'] + clusterDict['2']

print("Number of objects misclustered in the cluster is", str(total - numCorrect))

def displayContent(clusters):

for i in range(3):

print("=====================")

print("Cluster", i)

print("Size of cluster", i, "is", len(clusters[i]))

calcAccuracy(clusters[i])

for data in clusters[i]:

print("((" + str(data[0]) + ", " + str(data[1]) + "), " + str(data[2]) + ")")

def kMeans(dataset):

# Get 3 random values for starting means

print("Initial k means are")

mean1 = dataset[random.randint(0,500)]

mean2 = dataset[random.randint(0,500)]

mean3 = dataset[random.randint(0,500)]

print("mean[0] is ((" + str(mean1[0]) + ", " + str(mean1[1]) + "), " + str(mean1[2]) + ")")

print("mean[1] is ((" + str(mean2[0]) + ", " + str(mean2[1]) + "), " + str(mean2[2]) + ")")

print("mean[2] is ((" + str(mean3[0]) + ", " + str(mean3[1]) + "), " + str(mean3[2]) + ")")

noMatch = True

# Save previous means (may not be needed)

prevMean1 = mean1

prevMean2 = mean2

prevMean3 = mean3

# While means haven't changes from previous iter:

while noMatch:

# Organize the data into cluster with closest mean

clusters = clusterData(mean1, mean2, mean3, dataset)

# Recalculate the mean around the newly clustered data

# Add all X's and Y's in each cluster to create:

# (mean\_x, mean\_y, cluster)

mean1, mean2, mean3 = calculateMean(clusters)

# Check if these means are same as previous means

# If so, break

# Else, continue

if mean1 == prevMean1 and mean2 == prevMean2 and mean3 == prevMean3:

#print("Unchanged means: ", mean1, mean2, mean3)

noMatch = False

else:

prevMean1 = mean1

prevMean2 = mean2

prevMean3 = mean3

# Get accuracy here for output

# Loop through all classes and see if they match the cluster they're in

displayContent(clusters)

print()

print(str(correct/len(dataset)\*100) + "%")

def main():

dataset = []

with open("synthetic\_2D.txt", "r") as file:

for line in file:

dataset.append(line.split())

kMeans(dataset)

if \_\_name\_\_ == "\_\_main\_\_":

main()