
Task 1

Completed in knn_classify.py

This file uses the same uci_data.py file as last assignment. It is included in the zip and required for knn_classify.py to read the data from the file.

Pendigits:

K=1: classification accuracy=0.9743

K=3: classification accuracy=0.9750

K=3: classification accuracy=0.9763

All other datasets match the sample accuracies.

Task 1b:

Function declaration:

knn_classify(training_file, test_file, k, dist_opt)

I made two changes: an option for a distance function along with considering distance when choosing a class.

The first change is straightforward; there is a new parameter in the function declaration that determines whether the L1 or L2 distance is used. As expected this either improves the accuracy or decreases it depending on the dataset. I noticed a decrease in accuracy for the pendigits and satellite datasets with an increase in accuracy for the yeast dataset.

Secondly, and more interesting, I decided to weigh the k nearest data-points by their inverse distance. This way closer points have a larger weight towards the chosen class. This effect causes extremely close neighbors to almost always be chosen even if k is large.

No Weights Just Counts:

dataset	k = 1	k = 3	k = 5	k = 7	k = 9
pendigits	97.43	97.50	97.63	97.48	97.30
satellite	89.35	90.42	90.45	90.32	89.78
yeast	49.59	51.79	55.14	54.41	55.51

With Weights:

dataset	k = 1	k = 3	k = 5	k = 7	k = 9
pendigits	~~97.43~~	97.46	97.66	97.48	97.34
satellite	~~89.35~~	90.60	90.65	90.35	90.05
yeast	~~49.59~~	52.89	55.58	56.61	56.40

As you can see, There is a fairly consistent small increase in the total accuracy for each data set across multiple k values. K=1 values are crossed out because weights need to be relative to another value in order to change the result.

Different weight function could also be used such as the squared inverse or negative natural log. Functions that satisfy $\lim_{x \to 0^+} f(x) = \infty$ should work best.

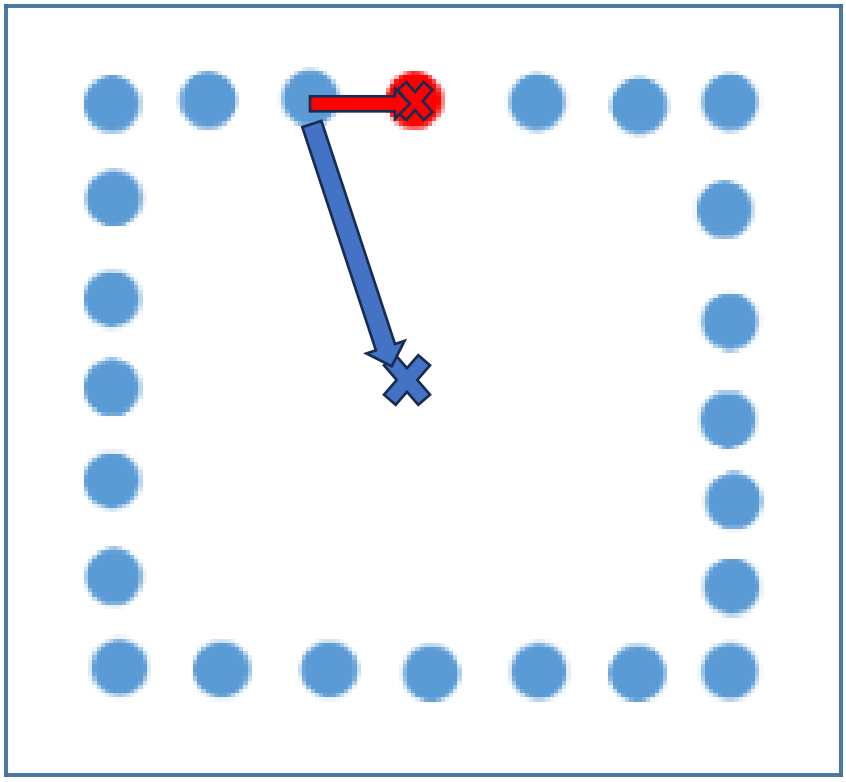
Task 2

Completed in k_means.py

The given drawing.py file is used. Comment out the last two lines if this is undesired.

Task 3

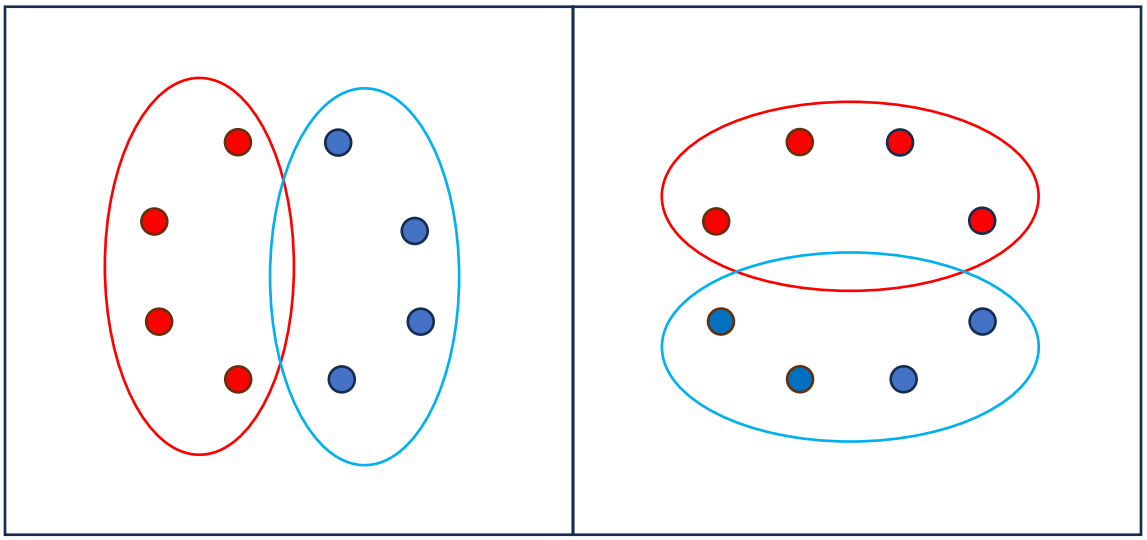
No.



The picture show above shows the rough means of each cluster along with the distances one data-point is from each cluster mean. As shown, a data-point in the blue cluster is closer to the red mean so on the next iteration it would change to a red data-point. Because changes are still occurring, the k-means algorithm would not exit. The k-means algorithm only stops when there are no new assignments.

Task 4

a. The EM algorithm can give different results depending on the random initialization of the gaussians. An example for this would be a 2-dimensional dataset where the data-points are arranged to be a symmetric circle. Because of the symmetry of the data-points, there are multiple ‘valid’ results that give the same log likelihood. Two examples are shown below for k=2. The left example initializes the gaussians horizontally; the right example initializes the gaussians vertically. Example resultant gaussians are shown, but these are rough estimates of what the iteration process would end up with.



Otherwise, if the initialization of the gaussians are kept constant, it will give the same result because EM iteration is deterministic. Each iteration can only have one result because no ‘random’ choices are made, but the result of the EM algorithm is dependent on the initial configuration of the gaussians.

b. When applied to the same dataset, agglomerative clustering will always give the same result. As long as there are no ties, there is no situation where a random choice has to be made. Every iteration of combining clusters is dependent on the min distance between contained data-points which does not change if the dataset does not change.

Task 5

a. Clusters: (2) (4) (7) (11) (16) (22) (29) (37)
   Min_min_dist: (2, 4) -> 2
   Clusters: (2, 4) (7) (11) (16) (22) (29) (37)
   Min_min_dist: (4, 7) -> 3
   Clusters: (2, 4, 7) (11) (16) (22) (29) (37)
   Min_min_dist: (7, 11) -> 4
   Clusters: (2, 4, 7, 11) (16) (22) (29) (37)
   Min_min_dist: (11, 16) -> 5
   Clusters: (2, 4, 7, 11, 16) (22) (29) (37)
   Min_min_dist: (16, 22) -> 6
   Clusters: (2, 4, 7, 11, 16, 22) (29) (37)
   Min_min_dist: (22, 29) -> 7
   Clusters: (2, 4, 7, 11, 16, 22, 29) (37)
   Min_min_dist: (29, 37) -> 8
   Clusters: (2, 4, 7, 11, 16, 22, 29, 37)

b. Clusters: (2) (4) (7) (11) (16) (22) (29) (37)
   Min_max_dist: (2, 4) -> 2
   Clusters: (2, 4) (7) (11) (16) (22) (29) (37)
   Min_max_dist: (7, 11) -> 4
   Clusters: (2, 4) (7, 11) (16) (22) (29) (37)
   Min_max_dist: (16, 22) -> 6
   Clusters: (2, 4) (7, 11) (16, 22) (29) (37)
   Min_max_dist: (29, 37) -> 8
   Clusters: (2, 4) (7, 11) (16, 22) (29, 37)
   Min_max_dist: (2, 11) -> 9
   Clusters: (2, 4, 7, 11) (16, 22) (29, 37)
   Min_max_dist: (2, 22) -> 20
   Clusters: (2, 4, 7, 11, 16, 22) (29, 37)
   Min_max_dist: (2, 37) -> 35
   Clusters: (2, 4, 7, 11, 16, 22, 29, 37)