Project 4: Unsupervised Learning

This project utilizes K-means clustering to classify data. In K-means clustering, the goal is to identify K number of clusters in the data. In the K-means clustering algorithm, the desired number of centers, K, is given, and the algorithm calculates the location of the centers based on the training data. This is done by calculating the nearest points to each center. The average of these clusters of points is calculated and used to relocate the center. Once the centers have all been moved to the average of the points closest to them, the points closest to the newly located centers are used to calculate a new average that will be used to again relocate the centers until the points closest to each center no longer change. This algorithm is illustrated in Figure 1. First, a set of N (in this case N = 6) training examples is given (a). K (here K = 2) training examples are randomly selected to be a set of initial cluster means (centers) (b). The distance between each training example and each of the K cluster means is determined, and each training example is assigned to the closest cluster mean (c). The average of the training examples assigned to each cluster mean is calculated, creating a new mean. The centers are moved to these new means (d). The distances between each training example and each of the K cluster means (these have different locations than they previously did) are again calculated, and each training example is assigned to the closest cluster mean. This process is repeated until all training examples are assigned to the same cluster mean as on the previous iteration (i.e., the cluster means do not change).

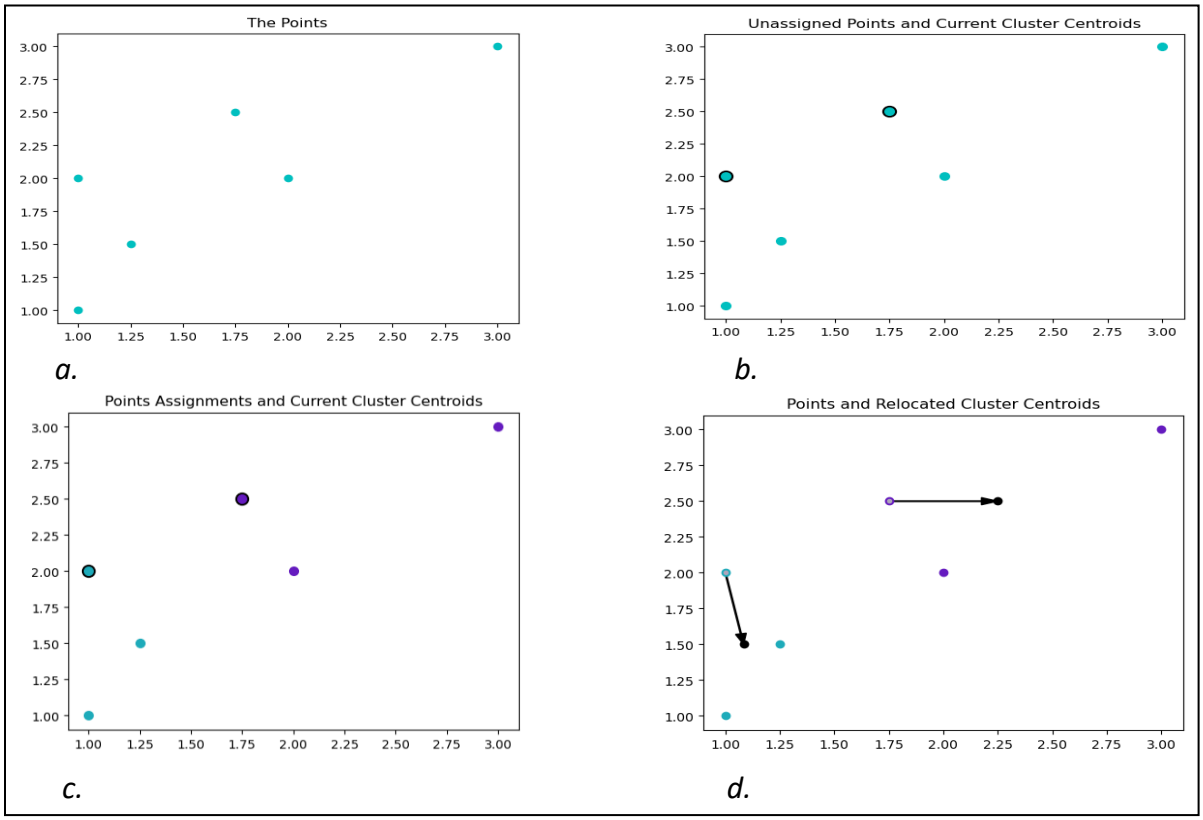

Figure 1: *a.* is a plot of the data points; *b.* shows the initial random centers; *c.* shows the initial point assignments (colored blue and purple); *d.* shows the relocated centers.

The code I developed implements this algorithm. The program takes three command-line arguments: the number of clusters (K), the filename of the training data, and the filename of the validation data. The program initializes K centroid vectors as the first K examples from the training data (the training data file is randomly shuffled so that this method allows for random selection). If the number of clusters is larger than the number of examples provided in the training data, the number of clusters is set to the maximum number of examples in the training data. Values of K less than one are treated as one, and it is assumed that the training data is not empty. The program then determines the closest centroid vector to each training example (using Euclidean distance). Links for information on how I accomplished this are as follows: Calculating Euclidean distance with NumPy - Stack Overflow, Understanding np.linalg.norm() - Sparrow Computing. Vectors are assigned to centers based on the closest distance. Links with information I used to accomplish this (including how I rearranged my distances data) are as follows: Element-wise array maximum function NumPy (more than two arrays) - Stack Overflow, Reshape list to ndim array with NumPy - Stack Overflow. The new locations of the centers are calculated based on the average of the vectors assigned to each centroid. An edge case in this part of the process occurs when no vectors are assigned to a centroid. In this case, the centroid retains its original location. This process is repeated until all the training examples are assigned to the same centers on two consecutive iterations. After the final centers have been calculated, a class label will be assigned to each cluster (which is represented by each center). This is done by finding which classification occurs the most in each cluster and using that classification to represent that cluster. Information I used to accomplish this can be found via the following links: Find the most frequent number in a NumPy array - Stack Overflow, Find the most frequent value in an array using NumPy - Projectpro. If there is a center without any vector assigned to it, it will be excluded. This edge case results in a fewer number of centers then was originally specified. This method of handling the edge case works well in cases where the specified K value is larger than the number of centers needed to represent the data. Here is a link to information I used in handling this edge case: Remove None values from a list - Stack Overflow. Additional links I used in this entire stage of the project are as follows: Empty list of a certain size - Stack Overflow, Last element of each list in a list of lists - Stack Overflow. After the clusters have been calculated and a classification has been assigned to each, the performance of the program is determined based on the validation data. The closest center to each validation example is calculated, and the classifications of the cluster associated with the center and the validation example's classification are compared. If they match, this counts as a correct classification. The program outputs the total number of validation examples correctly classified.

I then used my program to calculate the mean performance and standard error of the K-means classifier. I did this by using leave-10-out repeated random subsampling cross-validation. In other words, I tested my program for different values of K and consistently trained the program with all but ten of the data examples, then used the ten excluded examples to test the performance of the program. I did this for all numbers of clusters, from one to the total number of examples minus ten ($K = [1, 2, 3, ..., n - 10]$). For each value of K, I ran one hundred random

shuffles of the data with a training set size of n – 10 and a validation set size of 10, where n is the total number of examples in the data set. Since the program uses the first K examples as the initial cluster centers, shuffling the data ensures the cluster centers will change. I then plotted the mean performance (as a percentage) versus the number of clusters (K), including error bars of +/- 1.96 standard errors away from the mean. I performed this process on two separate datasets, one on iris flower classification and one on cancer classification. The iris data includes four attributes (sepal length, sepal width, petal length, and petal width) and the actual classification of the iris (Iris Setosa, Iris Versicolour, or Iris Virginica), which are labeled zero through two. The cancer dataset is focused on breast cancer. It includes nine attributes and a final breast cancer classification (these classifications are labeled from zero to five). The iris dataset has one hundred and fifty rows (150 examples), and the cancer dataset has one hundred and five rows (105 examples). The plot for the iris data is in Figure 2 and the plot for the cancer data is in Figure 3.

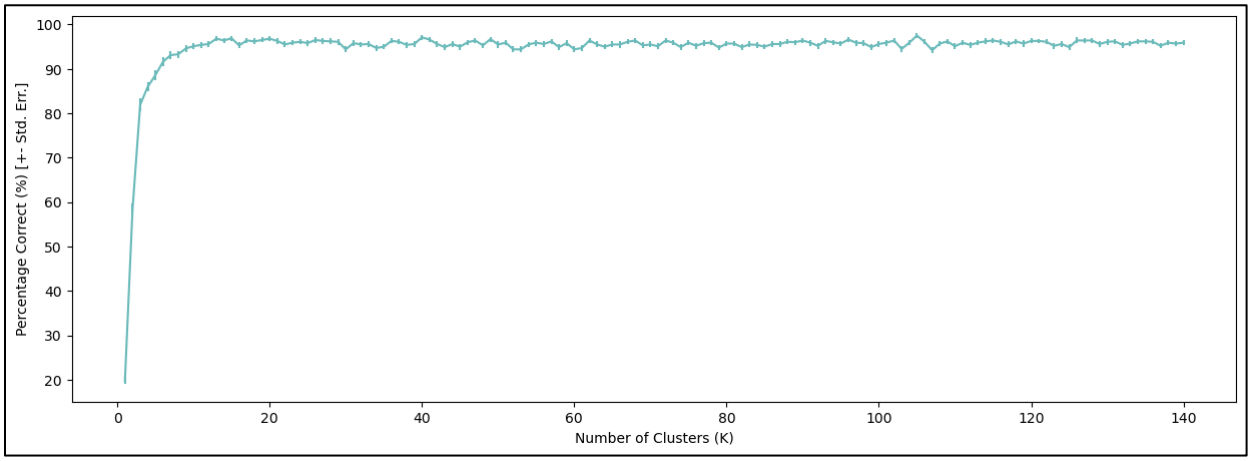

Figure 2: Performance of K-means clustering for iris data.

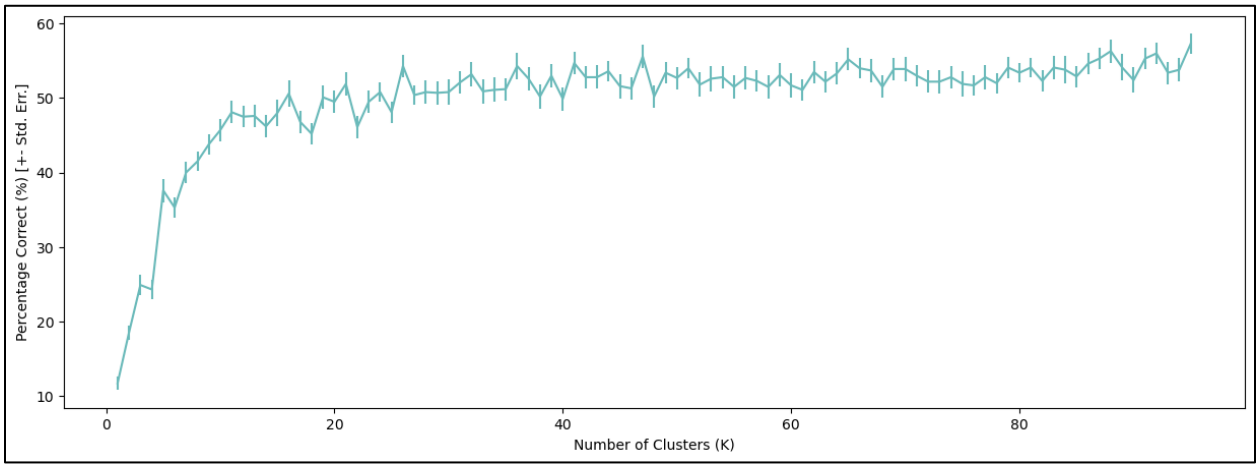

Figure 3: Performance of K-means clustering for cancer data.

For the iris data, one can see that the correctness of the classification per number of clusters evens out fairly quickly. This is helpful, as it tells us that we do not need a large number of clusters to classify the data. This allows for data compression. For example, we could discard

excess data examples and just keep the necessary number of clusters to classify future examples. In fact, since the model does fairly well at classifying the data with K = 8, we could keep only eight centers and discard the rest of the data. This results in a compression ratio (calculated as [Before / After]) of 18.75 (150/8) and a compressed percentage (calculated as [After/Before * 100]) of 5.33% (8/150 * 100). This means that keeping only 8 vectors instead of 150 results in a compression ratio of almost 19:1 (or a little over 5% of the original data size). This is greatly beneficial, as one would need to store a significantly smaller amount of data. On the other hand, the cancer data has no clear demarcation for the required number of clusters to achieve maximum correctness of classification. The percentage of correctly classified examples continues to increase for the entirety of the plot. As a result, one cannot specify a certain value of K which is sufficient to represent the data. Additional domain knowledge is necessary in order to constrain K and make K-means clustering an effective means of classification.