## Background

When working on problems in artificial intelligence, it is not always possible to have data with attributes that are labeled. Supervised learning techniques will not necessarily work on unlabeled data. This develops the need for algorithms that employ an approach called unsupervised learning. Unsupervised learning techniques often use parametric and non-parametric models to fit data and develop conclusions while relying on statistical properties of the training data (Russell). There are a few methods for doing this: clustering, manifold learning, regression, etc. (3 Phillips). A particular method of the first category was used in this open lab.

## K-Means Algorithm

The algorithm used in open lab four is a non-parametric clustering method called K-means. For explanation of the process's sake, say that we are given a large collection of two-dimensional unlabeled data that can be plotted on an XY grid. If the x and y data are good indicators of things such as a classification, behavior, or appearance, one can deduce that the data points that are close to each other will be similar. If the data was partitioned into a lesser number of clusters/groups that are each centered around a centralized point, one could make conclusions about the data or new data points without as many points. K-Means seek to iteratively find these central points. In the figure below, data points are in blue and centroids are in red. Using the K-means technique, the number of data points was effectively compressed from seven to three. (Slide 9 Phillips).

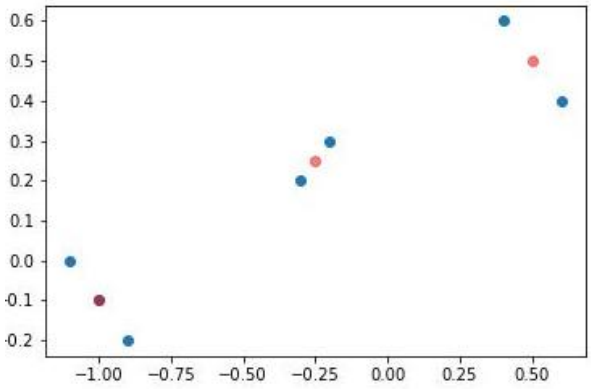


*Figure 1 K Means on HW9 dataset with K=3*

When testing a new data point, all one has to do is check the distance between the new data point and each centroid. The new point will belong to the centroid that has the shortest distance to it and will therefore belong to the category represented by it. This approach can be extended to higher dimensions and the number of clusters can be varied as well. The primary task at hand for open lab four was to test how accurately validation data could be classified with varying numbers of clusters on two datasets.

## Approach

One python file needed to be implemented in this project: kmeans.py. In terms of coding, it wasn't too complicated. Three parameters were required: an integer representing the number of *K* clusters, a training data file string, and a validation data file string. The script reads in the data

from the two files and stores them into a kmeans object in the form of numpy arrays. The first step in the process is to choose the first K points from the training data file. This training file is assumed to be shuffled, so this is basically like selecting K random points. These are initialized as the first centers. Next, a function is called that assigns a cluster parent to each point in the training data array. This is stored in a one-dimensional numpy array. After this, a new cluster point is created by iterating over each cluster, choosing all points that belong to said cluster, and finding the average of all of them together to create a centroid. This process is repeated until the previous iteration's cluster ownership array is identical to the current i.e. none of the points changed clusters and the current centroid points are as good as they are going to get. Before testing, each cluster does a majority vote to determine what attribute it represents e.g., if most of the points are category 3, then that cluster now represents category three. (Phillips OL4)

Once the clusters are created, ten points from the validation data file are tested against the clusters and each time a point is correctly categorized-the point's category is equal to the cluster-a counter is incremented. Finally, this counter is printed to stdout.

## Results

The code was ran on two different datasets: the iris set and the cancer set. For the iris set, the cluster values one through one hundred-forty were tested, and for the cancer set the cluster values one through ninety-five were tested. Each cluster case was ran one hundred times which basically equated to one hundred different testing cases and the results were placed into files. At the end, the mean and the standard error of each cluster case was calculated and plotted.

Figure **N** shows the results of the iris data. As one can see, it follows a rather logarithmic pattern as the cluster size increases. The peak accuracy is recorded in green, and associated errors for each case are represented by the bars at each point. As expected, the lowest performance was at the smallest cluster size. It also appears that as the number of clusters increased, the associated error tends to decrease and stabilize. As a rule of thumb, it seems that everything past ten clusters tends to be above ninety percent accuracy. One can see that the returns are diminishing with the increasing cluster sizes as for the given processing power, the accuracy and error obtained are miniscule. This is made even more apparent by the peak accuracy not being found near the highest cluster sizes, but instead closer to the 75 percent mark.

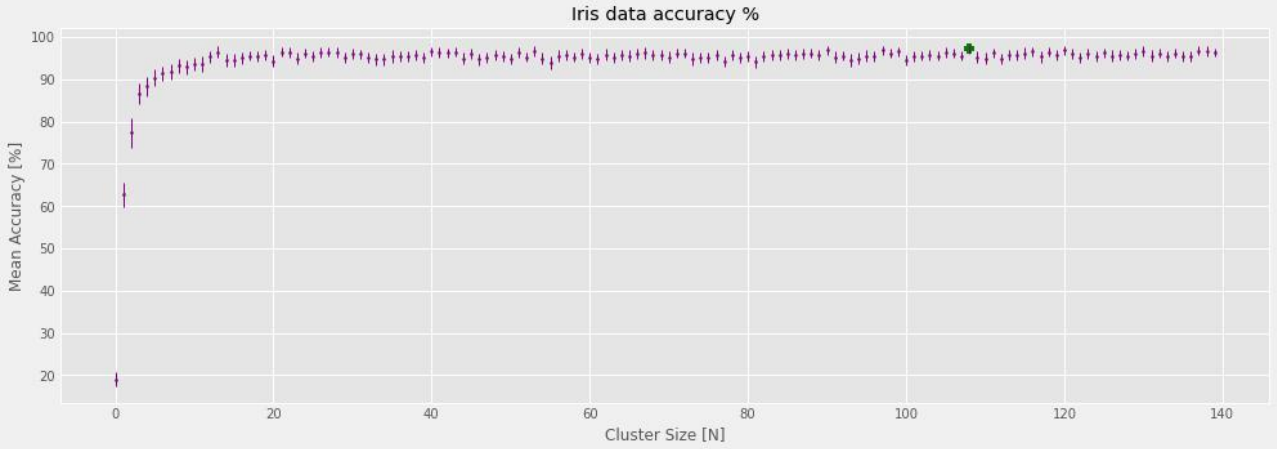


*Figure 2 Iris data set with error bars, peak mean in green*

Figure **N** shows the results of the cancer data. The largest difference between it and the iris data is the error bars are much wider, and the best performance occurs at 94 clusters which is one away from the highest number of clusters possible. The average accuracy of these cases was much smaller in comparison to the Iris data, with none of them breaking sixty percent. This is somewhat expected as the cancer data is much more complicated than the iris data and has more categories. Unlike the iris data, it appears that it would be beneficial to have a larger test and cluster size.

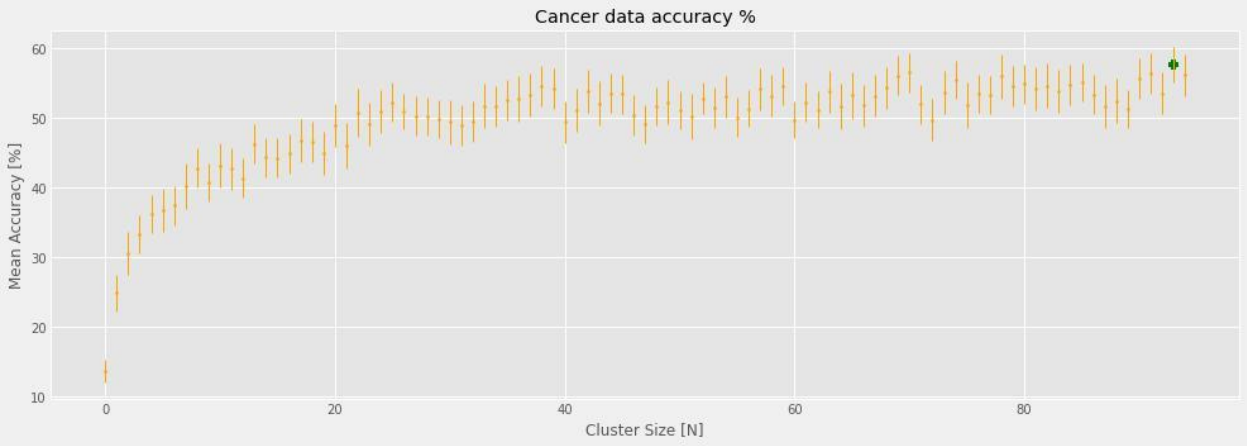


*Figure 3 Cancer data set with error bars, peak mean in green*

## Conclusion/Considerations

While working on the kmeans program, there was only one real snag. Because of a lack of foresight, the original centroid calculation routine was actually calculating centroids with the category being treated as a discrete value. To fix this, I had to pass in an array slice that took each row and chopped off the last column, however this introduced yet another problem. After trying to run a case, I had an error pop up indicating that I was attempting to take the mean of an empty slice. After some debugging, this case occurred whenever a cluster had all its values taken from it-to fix it, all I had to do was just check to ensure that the size of the slice was not equal to zero. For speed, I attempted to utilize Numba but found that the JIT compilation was not worth it for the time/number of calculations and the code ran faster without it.

It is worth mentioning that the results obtained are somewhat stochastic in nature. As K means is an iterative algorithm, the choice of the starting state of centroids can heavily affect the final result and possibly lead to better or worse results than what I have. To circumvent this, we ran each case one hundred times, but it is still worth keeping in mind. Better performance could possibly be obtained by “smartly” choosing the starting centroids rather than randomly sampling them. Another consideration is that both datasets are somewhat small and can be subject to overfitting or underfitting.

## References

Phillips, Joshua Dr. (2022 November 14). Open Lab 4: Unsupervised Learning. Department of Computer Science. Middle Tennessee State University. https://jupyterhub.cs.mtsu.edu/azuread/services/csci4350-materials/private/OLA4-4350.pdf

Phillips, Joshua Dr. (2022, November 09). Unsupervised Learning. [2,3,10]. Department of Computer Science. Middle Tennessee State University. https://jupyterhub.cs.mtsu.edu/azuread/services/csci4350-materials/private/2022-11-09_Chapter_18b.pdf

Russell, Stuart J. (Stuart Jonathan). (2020). Artificial intelligence : a modern approach. Upper Saddle River, N.J. :Prentice Hall, Chapter 21.7.1 Unsupervised Learning.