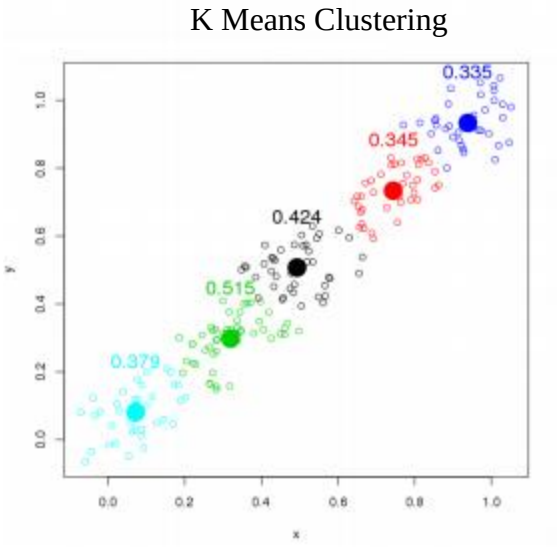
Brandon Goode

K Means Clustering



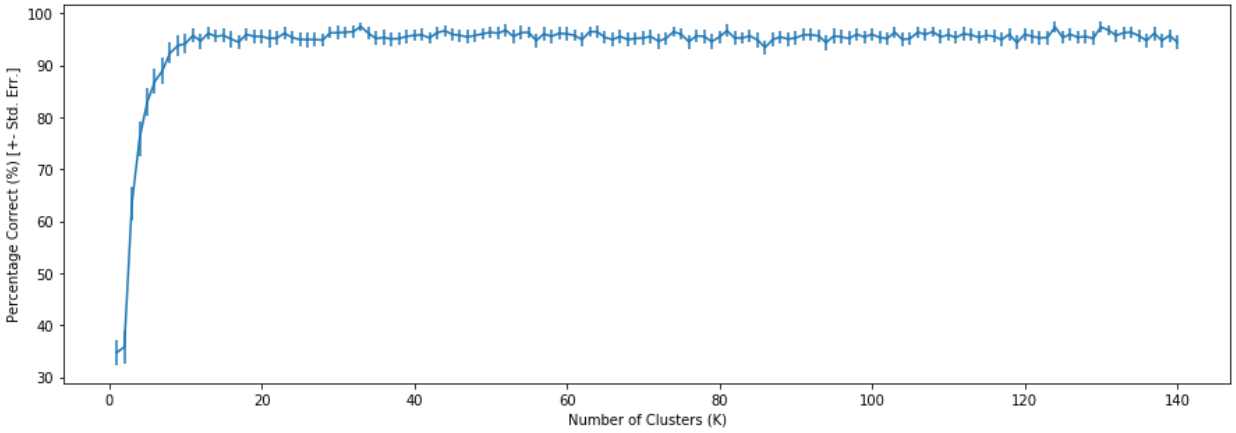
When classifying data, there are two common approaches to accomplishing the goal: supervised learning and unsupervised learning. In supervised learning algorithms, the class labels are used to determine what category the data is going into. In unsupervised learning algorithms, the class labels are not used when creating the categories as the categories are created by finding the trends in the data itself. These areas where data is heavily concentrated are called clusters and one of the unsupervised learning algorithms that help to utilize these clusters is called K Means Clustering.

As an unsupervised learning algorithm, the K Means Clustering algorithm(shown above) starts by taking the data and splitting it into one training set and one testing set making sure to not include any samples from the training set in the testing set and vice versa. After the data sets have been created, some number (K) of centroids are initialized. The purpose of a centroid is to be able to classify all the data points that are closest to that centroid as whatever class the centroid belongs to. The centroids are first chosen by taking K random samples from the training set and placing a centroid at each of them. After the centroids have been created, the next step is to create an average for all the data points closest to each centroid. Once the averages have been determined, the centroid is then shifted to the location of that average. The distances from each point are then calculated from the new average and the algorithm continues until it reaches convergence, i.e. none of the averages change after two consecutive iterations.

In my implementation of the K Means Clustering algorithm, I started by taking in four command line arguments: a number seed, the number of centroids to create, the name of a training set, and the name of a testing set. I then split my data file into a training set and a testing set that were unique from one another and passed them in as the third and fourth argument. After the split, I read in both files and stored them as NumPy arrays. I then created a new numpy array and copied the contents of the training set into it so that I could keep track of which centroids I pulled from the data set and delete it from the set so they did not get chosen more than once. After I had my array of centroids, I created a list in parallel to the training set to hold which centroid was nearest to which data point. I created a loop that used the Euclidean distance of all the attributes added together to check each data point against each centroid to find out which

centroid was closest to each point and appended it to the list once the loop terminated. This allowed me to have a list where each index corresponded to the data point of the same index in the training array to be able to keep track of which data point went with each centroid. I then created a loop that checked every point associated with each centroid and averaged them together by calculating the mean. The new means that got created replaced the original centroids. After the new centroids got created, I created a loop that calculated that distance from each data point to each new centroid and then took the average of each point nearest to that centroid. The loop terminates after none of the new averages are different from the previous averages. After the loop terminates, the centroids are now where they are supposed to be and have reached convergence. Now that the centroids have been placed in their final position, I created a loop that traverses the list of centroids nearest to each point for each centroid and calculated how many classes existed for each centroid. I then created a classification list that existed in parallel with the list of centroids to show which class each centroid belonged to. Finally, I used the classification list in a loop to test against each data point in the testing set and incremented a counter by one every time the centroid class was the same as the class for the data point. I output this number as the number of items the algorithm was able to guess correctly.

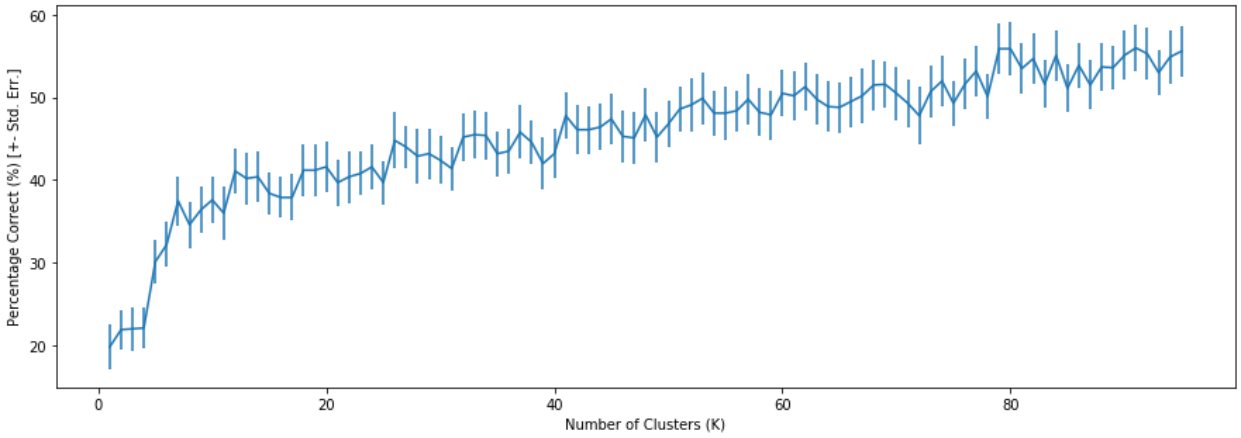
K Means ran with Iris Data



When I ran the K Means algorithm on the iris data set, the number of items guessed correctly went up drastically as the number of centroids increased up until about 8-10 centroids and then it almost completely leveled out. After a K of 10+ the algorithm guessed correctly about 95% of the time. At this point, the standard error bars consistently overlap and therefore conclude that the difference is not statistically significant. The difference is significantly significant up until about that point of K=8.

Brandon Goode

K Means ran with Cancer Data



When I ran the K Means algorithm on the cancer data set, the number of items guessed correctly gradually went up as K increased. The problem with the algorithm's performance on the cancer data set is that after K is higher than 4, the error bars overlap for every sample after that. This means that there is not a significant difference in the percentage of correct guesses after K is greater than approximately 4.