# Constructing a Classifier Using $k$-means

Ryan Florida

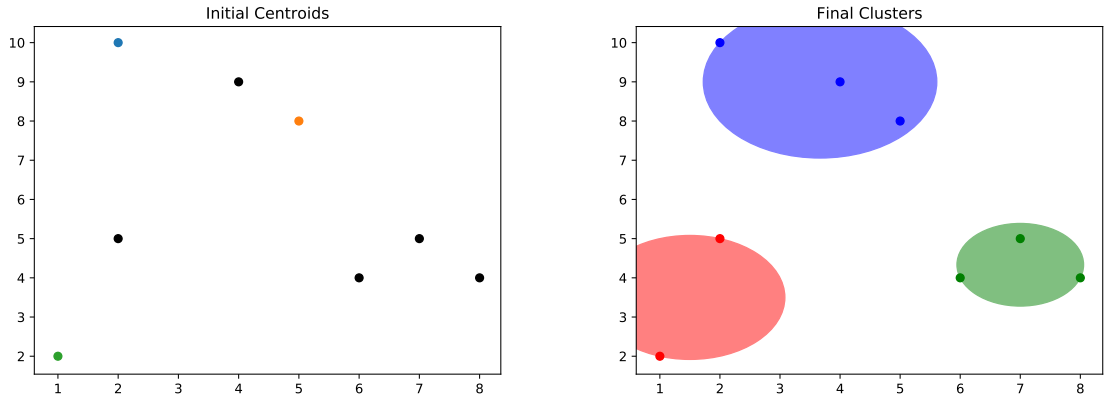
The purpose of this project was to construct a data classifier, which is formed by way of the $k$-means clustering algorithm. The concept of $k$-means clustering is relatively simple and can be summarized as follows:

i Choose $k$ random centroids from a training dataset—it should be noted that the programmer chooses $k$, however there are methods that will suggest an appropriate $k$ value but these are beyond the scope of this assignment.

ii Form initial clusters around the $k$ centroids based upon a chosen metric; in our case this metric is the Euclidean distance.

iii Average the feature values of all data points in each cluster and update the centroids with these averages.

iv Repeat steps (ii) and (iii) until there are no centroid updates.




(a) Initial Random Centroids (b) Complete Clusters

Figure 1: Illustrative Example of $k$-means.

Once the clusters have been formed (see Figure 1), we can use them to classify unseen (by the algorithm) test data for which we know the true labels. Thus, the goal of this project is to create an implementation of the classifier described above and then test its effectiveness on two datasets provided to us—these are the Iris and the breast cancer datasets collected by the University of California Irvine.

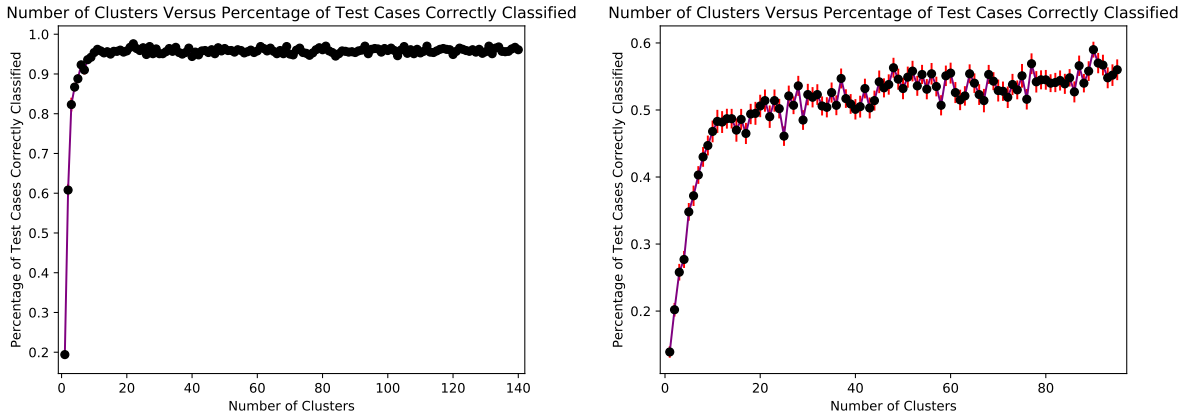
The code I developed followed the basic algorithm detailed above. For constructing the classifier, I chose to store each of the training data points in a user-defined data structure that I called *Data*. The *Data* structure has two attributes associated with it: *features* and *label*. The *features* attribute is a vector that stores doubles and, as its name implies, it holds the feature values associated with each dimension of feature space. The *label* attribute stores the class label, which is determined by the class to which the data object is known to belong. I also had to overload the comparison operator for the *Data*

structure in order to test the equality between two centroids. I made this comparison solely dependent on the *features* attribute solely because I only had one use for it, which was to compare two centroids.

For storing the actual data points, I used a vector of *Data* objects since vectors are very efficient for traversal and have the benefit of random access. Though insertion into a vector brings with it a bit of an overhead, I only had to store the objects once so this was not really a concern. My initial thought for storing the centroids and clusters was to use an unordered map in which the keys would be the centroids and the values would be the clusters associated with those centroids—this idea was inspired by a Python implementation of $k$-means that I created, which was also used to generate Figure 1. This did not translate so well to C++, so I ended up using two vectors, *centroids* and *clusters*, which were maintained in parallel. I chose vectors again because of their random access and their ability to be quickly traversed. Though I did clear and re-populate the *clusters* vector frequently, I always just pushed the new entries to the back since the ordering did not matter; this slightly reduced the overhead of adding a new element to the vector. In order to determine the cluster labels, I used an unordered map (both keys and values were integers) to count the number of labels per cluster—this enabled me to determine the majority label in the cluster and, coincidentally, assign that label as the cluster label.

For the actual classification portion of the project, I stored the test data in a vector of *Data* object. After storing the test data, I found the distance between each centroid and test data member. This lead to me being able to determine the minimal distance between a given cluster and test point, then assign the appropriate cluster label.

After finishing the programming portion of the project, I ran the algorithm on both of the provided datasets. As mentioned earlier, a drawback of using $k$-means is that the programmer must choose the $k$ value, which is not always an easy choice to make. Another drawback of $k$-means is that if the data is not radially separable (see Figure 1b for example), then we may not be able to form meaningful clusters—in this case we would have to project the data into a higher-dimensional feature space and reattempt clustering in that space. This being said, our values for $k$ ranged from $k = 1$ to $k = M$, where $M$ is the number of total training data points we had. For each value of $k$ I ran the program 100 times and then computed the mean percentage of correctly classified test data points against the number of clusters used. The results of this effort can be seen in Figure 2.




(a) Iris Dataset

(b) Breast Cancer Dataset

Figure 2: Effectiveness of $k$-means on two different datasets.

With respect to the Iris dataset (Figure 2a), we see that we need a relatively low number of clusters in order to achieve a high percentage $\approx 96\%$ of correctly classified test cases. It is also worth noting that the standard error associated with a given number of clusters is very small in the Iris dataset. This seems to imply that the results from successive classifications of the test data will not deviate significantly from the observed trend.

The breast cancer dataset seems to tell a different story. We see that only about 52% to 58% of test cases were classified correctly. We also see that the standard error associated with each value of $k$ is quite significant. Thus, results from successive runs of the testing data will more than likely deviate from the current results in a noticeable way.