K-means Clustering Algorithm

By: Tahvanh Lucero

Disclaimer: This is not an official paper. Its purpose is to summarize the findings of this project, inspired by my curiosity.

Outline

This paper will first introduce and explain the concept of K-means clustering. Then it will show step-by-step how to write the algorithm. The Iris flower dataset will be used throughout the paper as an example for implementing the algorithm. Finally, the paper will show results and discuss applications of K-mean clustering.

What is K-means Clustering?

K-means clustering is an algorithm that sorts data into meaningful categories by grouping data points with similar properties. It does so by observing which points are particularly closer to each other than others. In other words, it looks for the densest locations in the dataset and then groups all of the nearby points into one cluster. The center of a cluster is called a centroid. The goal of the algorithm is to move the centroids (initially in random positions) to the center of each cluster. You get to determine the K-value, which indicates the number of clusters in a dataset.

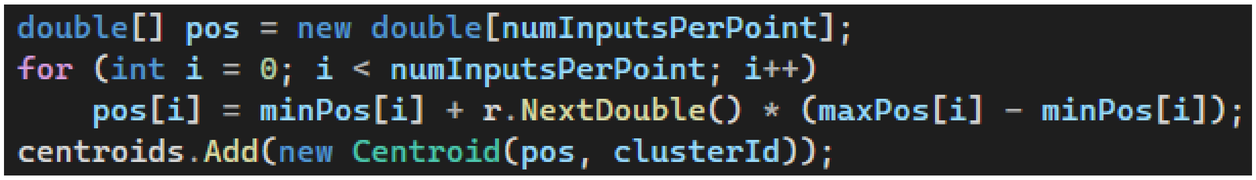
Setup

Each data point should have quantifiable characteristics that can be used to identify it. For example, an Iris flower can be identified by measuring its sepal length, sepal width, petal length, and petal width. K-means clustering finds the densest locations, which means that it looks for the points in "space" with the highest number of points per unit "space." To better work with the idea of data points in a "space," we will be using a coordinate grid system.

Imagine a coordinate grid system where each axis represents a metric used to identify a data point. For example, if an Iris flower has a sepal length of 5.1 cm, a sepal width of 3.5 cm, a petal length of 1.4 cm, and a petal width of 0.2 cm, it would have the coordinates (5.1, 3.5, 1.4, 0.2). Do this for all data points. *Programming tip: Create a DataPoint class where you can store the data point's coordinates.*

Next, you will need to define your K-value. Later, you will find the optimal K-value for your dataset, but for now, pick a small number like 2 or 3. Create K number of centroids. Initially, the centroids' coordinates should be randomized between the lowest and highest coordinates possible (based on your dataset).

*Programming Tip: The formula for randomization I used was: minimumPosition + [random number between 0 and 1] * (maximumPosition - minimumPosition)*



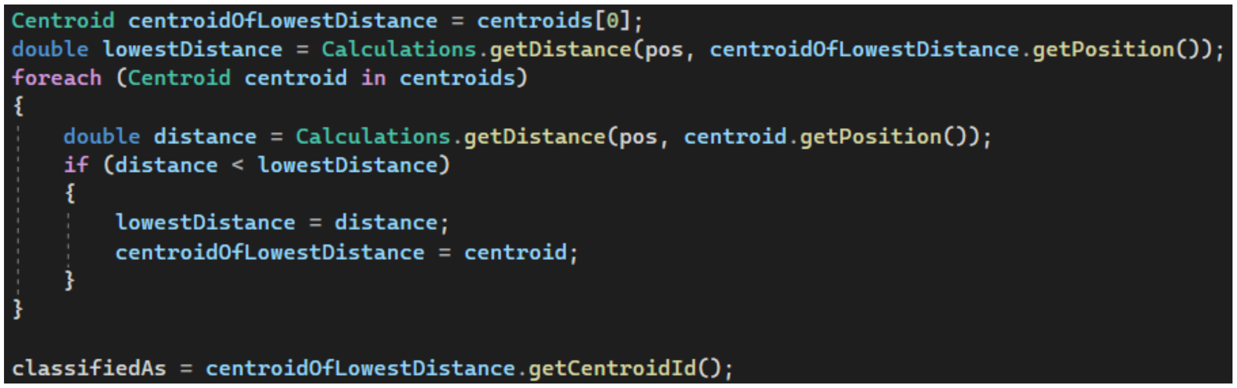
```
double[] pos = new double[numInputsPerPoint];
for (int i = 0; i < numInputsPerPoint; i++)
    pos[i] = minPos[i] + r.NextDouble() * (maxPos[i] - minPos[i]);
centroids.Add(new Centroid(pos, clusterId));
```

Algorithm

Two steps:

1) Assign each data point a cluster based on the nearest centroid, using the euclidean distance formula.

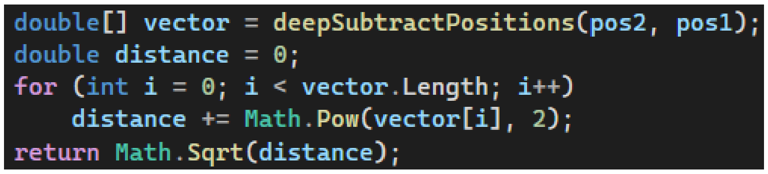
Programming Tip: Initially set the lowest distance to a centroid's position. Iterate through the centroids and get the distance between the data point and each centroid. If a data point is closer to a different centroid, set the lowest distance to that centroid's position.



```
Centroid centroidOfLowestDistance = centroids[0];
double lowestDistance = Calculations.getDistance(pos, centroidOfLowestDistance.getPosition());
foreach (Centroid centroid in centroids)
{
    double distance = Calculations.getDistance(pos, centroid.getPosition());
    if (distance < lowestDistance)
    {
        lowestDistance = distance;
        centroidOfLowestDistance = centroid;
    }
}

classifiedAs = centroidOfLowestDistance.getCentroidId();
```

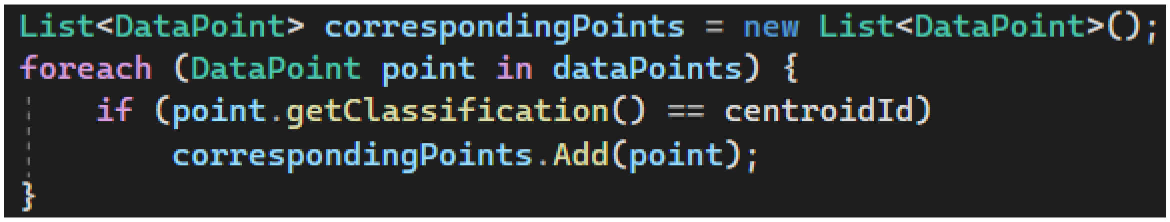
Programming Tip: To find the distance between a data point and a centroid, create an array with each element being the difference between the positions of the data point and the centroid. Square each element in the array, add them up and take the square root of the sum.



```
double[] vector = deepSubtractPositions(pos2, pos1);
double distance = 0;
for (int i = 0; i < vector.Length; i++)
    distance += Math.Pow(vector[i], 2);
return Math.Sqrt(distance);
```

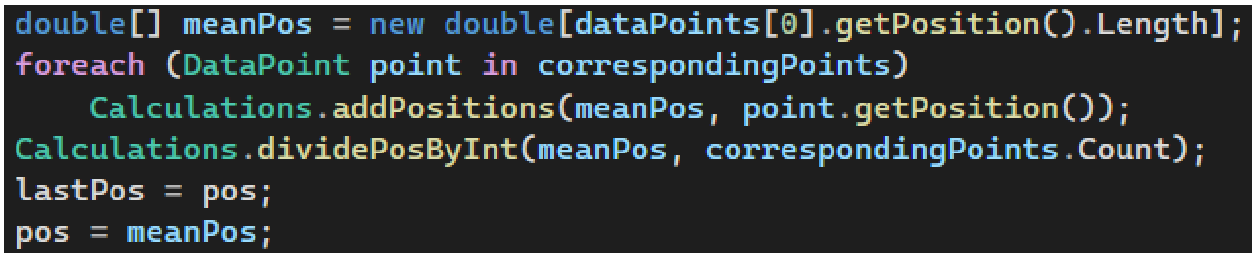
2) For each cluster, recalculate its centroid's position by getting the average position of all the data points that are assigned to that cluster.

Programming Tip: Have a list of your data points. Iterate through that list and get the data points with the same classification as this cluster's centroid. Those will be the cluster's/centroid's corresponding data points.



```
List<DataPoint> correspondingPoints = new List<DataPoint>();
foreach (DataPoint point in dataPoints) {
    if (point.getClassification() == centroidId)
        correspondingPoints.Add(point);
}
```

Programming Tip: Add up the positions of all the corresponding data points and divide that by the number of corresponding data points. That's the new position of the centroid.



```
double[] meanPos = new double[dataPoints[0].getPosition().Length];
foreach (DataPoint point in correspondingPoints)
    Calculations.addPositions(meanPos, point.getPosition());
Calculations.dividePosByInt(meanPos, correspondingPoints.Count);
lastPos = pos;
pos = meanPos;
```

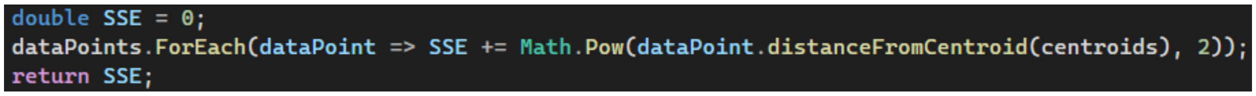
Repeat these two steps until an end condition is met. A good end condition is when every centroid stops moving.

Programming Tip: Have a "last position" variable in the centroid class. Every time you update a centroid's position, make sure to first update the last position variable to its current position. Then update the centroid's position. Take the difference between the last position and its current position, and you'll have how much the centroid has moved. Repeat the two steps until the sum of the change in position of each centroid is equal or close to zero.

Finding the Optimal K-value

To test the effectiveness of the K-value, we will calculate the Sum of Squares Error (SSE) between each point and its corresponding centroid. The SSE is equal to the sum of the squares of the distances between each point and its corresponding centroid.

Programming Tip: For each point, get the distance between it and its corresponding centroid → square that distance → add it to SSE



```
double SSE = 0;
dataPoints.ForEach(dataPoint => SSE += Math.Pow(dataPoint.distanceFromCentroid(centroids), 2));
return SSE;
```

Due to the initial randomization of each centroid's position, the SSE will not be the same each time you run the program. To account for this random error, run the program many times and get the median SSE of the runs.* That will be your official SSE for that K-value. *Programming Tip: Run the program 1000 or 2000 times and store each SSE result in an array. Sort the array in ascending order and find the median SSE.*

*It could be better to find the mean SSE. However, I first used the mean and found that the outlier SSEs were skewing the data away from the optimal K. The idea behind using the median was expecting the true SSE to appear over 50% of the time. I could be wrong, so use whatever measure you think is best.

Run the program with different K-values and plot them with their SSEs. Figure 1 below shows the plot graphed for the Iris flower dataset.

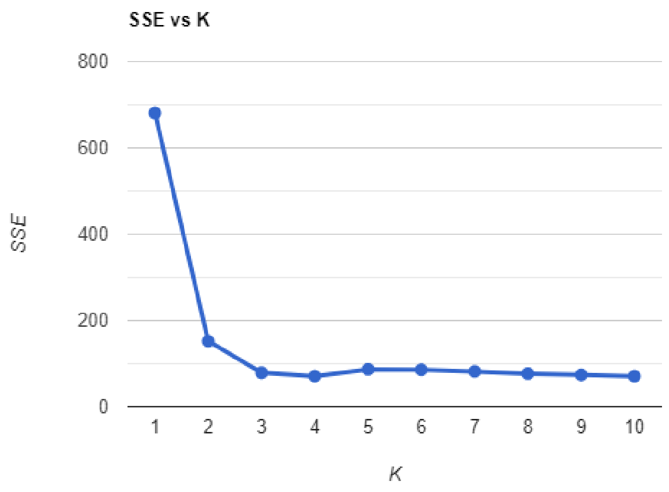


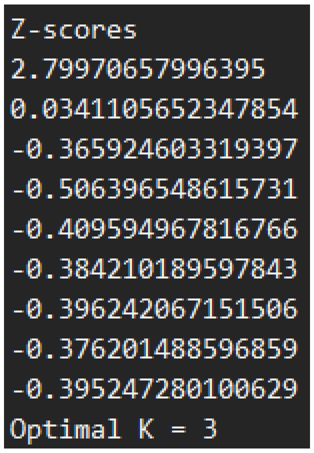
Figure 1: Plot of SSE vs K for each K-value. I cannot explain why the graph starts increasing after K = 4. It could potentially mean a flaw in my code, or it could mean some hypothetical "repelling/spreading effect" that comes with more centroids, overcome by adding even more centroids to account for the faraway points. Either way, the algorithm gave a pretty consistent and accurate prediction, so it works out in the end.

Apply the "elbow rule" which says that the optimal K-value is found at the "elbow" of the curve, AKA the point at which incrementing K will not have a huge effect on SSE. Once at the "elbow," do not keep incrementing K, even if SSE values are lower, because you would be dividing your data so much that each category becomes too niche to use. In other words, the categories lose meaning.

For the Iris flower dataset, we find that the "elbow" of the curve is at K = 3.

(Optional) Algorithm for finding optimal K-value

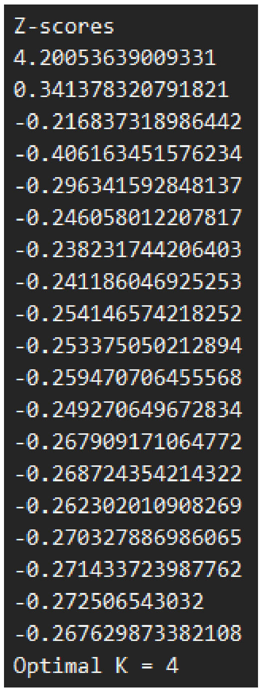
I could not find an algorithm online for finding the optimal K-value besides the "elbow rule" so I tried to make my own. I calculated the change in SSE between each K-value and put them in an array called "deltas." A delta at index 0 corresponds to an SSE at index 1. The SSE for K = 1 (index 0) does not have a corresponding delta, as it is the first point. I then calculated the z-scores of each delta (given the array "deltas"). The delta with the z-score closest to zero corresponds to the optimal K. For example, if the delta with the z-score closest to zero had index 1, then the optimal K would be at index 2 of the SSE array, which corresponds to a K-value of 3.



```
Z-scores
2.79970657996395
0.0341105652347854
-0.365924603319397
-0.506396548615731
-0.409594967816766
-0.384210189597843
-0.396242067151506
-0.376201488596859
-0.395247280100629
Optimal K = 3
```

Figure 2: z-scores of the deltas for a domain of K-values that range from 1-10

This algorithm is not perfect. When tested against a large domain of K-values such as one ranging from 1-20, the mean delta greatly decreased, influencing the z-scores of each delta and overestimating the optimal K-value.



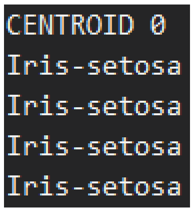
```
Z-scores
4.20053639009331
0.341378320791821
-0.216837318986442
-0.406163451576234
-0.296341592848137
-0.246058012207817
-0.238231744206403
-0.241186046925253
-0.254146574218252
-0.253375050212894
-0.259470706455568
-0.249270649672834
-0.267909171064772
-0.268724354214322
-0.262302010908269
-0.270327886986065
-0.271433723987762
-0.272506543032
-0.267629873382108
Optimal K = 4
```

Figure 3: z-scores of the deltas for a domain of K-values that range from 1-20

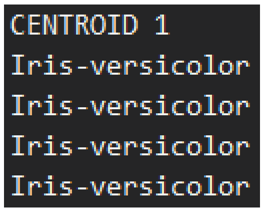
Testing Accuracy
Once you have found the optimal K-value, proceed to run the algorithm once more with the correct K-value. If you get an SSE that doesn't reflect the median SSE from earlier, run the program again until you get a more accurate SSE.

For each cluster, get its corresponding data points →
For each data point gathered, compare its classification (what the cluster represents) with its true classification (known from the dataset).

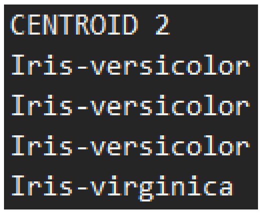
Programming Tip: When first initializing the data points, pass their true classification through the class's constructor and store it in the class.



```
CENTROID 0
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
```



```
CENTROID 1
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-versicolor
```



```
CENTROID 2
Iris-versicolor
Iris-versicolor
Iris-versicolor
Iris-virginica
```

Figures 4-6: Each centroid/cluster along with the classification of its corresponding points. Not every point's classification is shown because it would not fit on this page.

Results
Centroid 1 (setosa) - 50 setosa
Centroid 2 (versicolor) - 47 versicolor, 14 virginica
Centroid 3 (virginica) - 36 virginica, 3 versicolor

Total accuracy: 133/150 = *89%*

Conclusion
K-means algorithm can segment any dataset, which is why it can be used in almost any industry. In software that analyzes images, it can separate objects and identify them. In marketing, it can identify different types of consumers in a market (customer segmentation). In a field of Iris flowers, it can identify their species.

K-means clustering is a part of unsupervised learning, which is ideal for identifying trends and organizing data.

If you'd like to see my implementation of the algorithm in code, it is public at https://github.com/TacoL/K-means-Iris-Flower-Sorting/tree/main/K-means%20Iris%20Flower%20Sorting/K-means%20Iris%20Flower%20Sorting