K-Means Clustering

By: Adris Azimi (100704571), Khayam Moradi (100699466), Zachary Carman (100697844)

April 4th, 2020

Project Description

The goal of this project was to create an efficient algorithm to perform K-Means clustering. K-Means clustering is an important technique used to cluster data. One example of K-Means clustering is photo compression. A K-Means clustering algorithm can detect the many different colours in a photo and compress it down to a few colours.

Our program is a proof of concept. The program generates a static “n” amount of random points and stores them in an ArrayList. The K for our algorithm is 4 and this can not be changed. This means 4 centroids will be generated to compare the points too.

Background

K-Means clustering is one of the most simple yet efficient algorithms used in machine learning applications. K-Means is what is known as an unsupervised algorithm. This means that the ultimate outcome is unknown from the start.[1] Overall, the purpose of K-Means is to discover underlying patterns. In the most basic sense, this is done by taking an “n” number of data points and segmenting them into “k” number of “clusters” or in an application, that would be the number of different subgroups defined by the programmer.[1] These subgroups or “clusters” represent the similarities between the data points within each cluster.[2]

A K-Means algorithm first starts with a randomly placed number of centroids which each represent their own set cluster. The algorithm assigns a relationship between every data point and the nearest centroid to complete the initial clusters. The algorithm then goes through an iterative process of calculations to shorten the distance between each data point in relation to every centroid. This is done by moving each centroid around until there is none or almost no noticeable difference from their past locations.

The benefit of the K-Means clustering algorithm is that it is very versatile and can be adapted to many different applications. One of the most common applications includes customer segmentation.[3] This application of K-Means allows a business to take raw data of their customers and use that data to divide their customers into different groups so marketing can be made basically autonomous. In this application, the K-Means algorithm serves as a great cost-effective and time-saving solution as opposed to performing the task manually without K-Means.

Another common application of the K-Means algorithm is image segmentation.[4] In this case, the K-Means algorithm would look at every individual pixel in the image and group them into clusters based on their similarities which in this case would be the RGB values. An image would go through the K-Means algorithm process and come out as a simpler looking version of the original image with all the edges in the picture being emphasized. Overall the K-Means algorithm makes it easier to discern the difference between objects or also easier to see the outline of an object itself.

Pseudocode

```
createArrayList()
**Input:** null
**Output:** sets a global Arraylist for points
For the amount of points(n)
Add an arraylist for each index.
Add random value to the index of points arraylist. This value will be the x value
Add another value to the index. This value will be the y value

createCentroids()
**Input:**null
**Output:** sets a global Arraylist for centroids
For when i = 4
Add index for each index.
Add random value to the index of the arraylist. This value will be the x value
Add another value to the index. This value will be the y value


computedistance()
**Input:**null;
**Output:** Fills an global array with distance values
For number of points(n) //index L
For number of centroids(4) //index i
x← PointArraylist at index .get(0) //point x value
y← PointArraylist at index .get(1) //point y value
w← CentroidArraylist at  second index .get(0) //centroid x value
z← CentroidArraylist at  second index .get(1) //centroid y value
Calculate the Euclidean distance between the point and the centroid
distance[i][L] = Euclidean distance
//The first index is the centroid the second is the points so each point will have 4 distances calculated

clusterAssign()
**Input:** null
**Output:** Fills sorted ArrayList // Sorted will have 4 index one for each centroid filled with the points closest to that centroid
For number of points(n) //index L
if distance at index 0 is the smallest
```

```
Add point at index L for point arraylist to index 0 of sorted
If distance at index 1 is the smallest
Add  point at index L for point arraylist to index 1 of sorted
If distance at index 2 is the smallest
Add point at index L for point arraylist to index 2 of sorted
If distance at index 3 is the smallest
Add point at index L for point arraylist to index 3 of sorted
```

MoveCluster()
Input:null
Output: Changes the position of Arraylist Centroid to be the average all points assigned to it.

```
var.Empty() // a simple function to empty the Centroid arraylist
For number of centroids //index i
If sorted at index i is empty
Break
For number of points(n)
If sorted at index i is empty
Break
Else
Y += sorted at first index and the second value
X += sorted at first index and the first value
Remove index 0 at sorted i
Increment a temp value // this value tracks how many points are sorted to each centroid
//outside second for loop
newx← X/temp //average x values
newy← Y/temp //average y values
Add newx and newy and centroid arraylist at index i
```

nextCheck()
Input:null
Output: final sorted clusters

```
While endcheck>0.5 and endcheck <-0.5
var.EmptySorted() // simple function
var. computedistance()
var.culsterAssign()
var.MoveCluster
One ← Value() //function to get sum of the distance
Iterationcount ++ //keeps track for the iteration
 var.EmptySorted() // simple function
var. computedistance()
var.culsterAssign()
var.MoveCluster
```

```
Two← Value()
Endcheck ← one-two
Iterationcount ++
//the function stops when the distances don’t change anymore or move very slightly
```

Time Complexity

The time complexity of the project is O(n^2), however, a time complexity O(n) could have been achieved by using recursion. Instead, nested for loops were used throughout the algorithm as this was the easiest method of developing iterative functions for the algorithm. The possibility of moving from nested for loops to recursion was explored, but due to time constraints this change could not be implemented. The two methods with the O(n^2) time complexity are computedistance() and MoveCluster(). The other methods such as createCentroids(), clusterAssign(), and EmptyCent() all have O(n) as they either contain an arraylist, n indexes, or go through an arraylist of n indexes.

Major Classes

As the K-Means clustering algorithm requires multiple steps to complete the clustering process, multiple classes needed to be written. The first major classes built were the createArrayList() and createCentroids(). The createArraylist class stores the x and y values for every different point whereas the createCentroids class stores the x and y values for every centroid. These x and y points have random values between 0 and 50.

The next major class is the computedistance() class which uses the euclidean formula to calculate the distance between every point and every centroid. Once the distances are calculated, they are then stored inside a 2d array for comparison in the clusterAssign() class. The clusterAssign class, compares the point distances obtained from every centroid and then assigns the points to the closest. Once this is done, The MoveCluster() class is then called to move the x and y points of the centroid to the average x and y values of the points assigned to that centroid. Once the centroids have moved, the points need to have their distances to the centroids calculated again to maintain the shortest distance between the points and centroids, this is done in the nextCheck class. The nextCheck() method empties the arraylist for sorted and calls the computedistance(), clusterAssign(), and moveCluster() classes again. The new distances are calculated, the clusters are moved again and the points are reassigned. These classes are called twice to compare the previous iteration to the new one in order to determine when the program should stop. To do this, a while loop was built to run as long as the difference between the new and previous iteration is less than 0.5 or greater than -0.5.

Data Structures

The Structure chosen for this project was arraylists. Arraylists were chosen as they are a dynamic structure that can potentially store 2 values inside the same memory location. A dynamic structure was important as there was no way of determining how many points would be assigned to each centroid. In addition, an arraylist could store another arraylist which was important as storing multiple values to a single list was needed throughout this project. Furthermore there is a built in library that allows the values to be chosen with the .get() method for x values and .get.get() for y values.

Other data structures such as trees, queues, and hashing were considered as well. Using trees ultimately made coding the algorithm too complex, for example differentiating the x and y values inside the tree as well as taking the values to be computed were two problems encountered that made it an impractical solution. In short, trees were not chosen due to the uncertainty in the capability of being able to assign and extract values into and from the tree. Queues were also considered, but once proven that they could not store 2 values inside 1 memory location, the idea was abandoned. Hashing was the last data structure explored. Hashing could store 2 values or more in the same memory location, but a key would also need to be outputted. This was not ideal, therefore arraylists were studied as a final way to store 2 values inside 1 memory location.

Team Tasks

Zachary Carman

- Wrote pseudocode
- Helped with report
- Coded
    - computeDistance
    - NextCheck
    - Movecluster
    - clusterAssigned
    - CreateArrayList
    - createCentroids

Khayam Moradi

- Early testing with different Data Structures
- Analyzed time complexity of program
- Project report
- Coded
    - computeDistance
    - createArrayList

    - createCentroids
    - clusterAssign

Adris Azimi

- Preliminary project research
- Explored coding with other data structures including recursion
- Project report
    - Wrote various parts
    - Final edits
- Helped with createSorted, createdCentroids and other minor functions
- Moral Support™

Conclusion

The goal that was set out in the start of this project was to create an efficient enough K-Means clustering algorithm. Ultimately, this goal was met and the algorithm proved to be a very efficient K-Means clustering algorithm even when testing it with large amounts of data. Even with its already proven efficiency, our algorithm could still benefit from the use of other data structures such as recursion. Our algorithm could have also benefited from allowing the number of centroids to be a dynamic number.

References

1. M. J. Garbade, “Understanding K-means Clustering in Machine Learning,” Medium, 12-Sep-2018. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1. [Accessed: 05-Apr-2020].

2. C. Piech, “K Means,” CS221, 2012. [Online]. Available: https://stanford.edu/~cpiech/cs221/handouts/kmeans.html. [Accessed: 05-Apr-2020].

3. P. Sharma, “The Most Comprehensive Guide to K-Means Clustering You'll Ever Need,” Analytics Vidhya, 12-Mar-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/. [Accessed: 05-Apr-2020].

4. N. S. Chauhan, “Introduction to Image Segmentation with K-Means clustering,” Medium, 01-Mar-2020. [Online]. Available: https://towardsdatascience.com/introduction-to-image-segmentation-with-k-means-clustering-83fd0a9e2fc3. [Accessed: 05-Apr-2020].