An Improved K-Means Clustering Algorithm for One Dimensional Data

Ryan Froese
Dept. of Computer Science
University of Manitoba
Winnipeg, MB, Canada
froeser5@myumanitoba.ca

James Klassen
Dept. of Computer Science
University of Manitoba
Winnipeg, MB, Canada
klass167@myumanitoba.ca

Tyler Loewen
Dept. of Computer Science
University of Manitoba
Winnipeg, MB, Canada
loewent4@myumanitoba.ca

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—Clustering algorithms, Cluster analysis, k-means algorithm, Data analysis**

1 Introduction

Clustering is a data mining technique used to arrange large amounts of data into distinct clusters. It is used in many fields including image processing, pattern recognition, bioinformatics, spatial data analysis, and more. The idea of a clustering algorithm is to create clusters containing data points from a dataset where every data point in the same cluster is more similar to one another than to the data points in other clusters (i.e., maximizing the intra-class similarities within a cluster and minimizing inter-class similarities between other clusters). Thus, creating a set of data points in a cluster that are locally more similar to each other than to data points in other clusters. Clustering can help researchers better understand the distribution and structure of large datasets. In Section 2 we introduce the partitioning approach to clustering and describe the k-means clustering algorithm along with some of its shortcomings. We then discuss some commonly used approaches to improving the k-means algorithm. In Section 3 we propose our improved k-means algorithm. We include pseudo code for our algorithm, a formal proof of correctness, and the time complexity of a single iteration. In Section 4 we analyze the experimental results of our algorithm and compare both artificial datasets of different sizes and distributions, and a real-world dataset. In Section 5 we use a real-world dataset containing oceanic and atmospheric data combined with our clustering algorithm to visualize geospatial data. We also discuss any new findings and relationships that become apparent when visualizing the data. Lastly, Section 6 provides a summary of our findings, visualizations of our dataset and potential improvements to our algorithm in the future.

2 Background

2.1 Clustering

There are three major techniques for clustering: partitioning methods, hierarchical methods, and density-based methods. The partitioning method is used by the original k-means algorithm and our algorithm. The partitioning method takes a dataset of size n and partitions it into some k number of partitions (clusters) where $k \leq n$. Each cluster contains at least one data point, and each data point belongs to exactly one cluster (i.e., distinct clusters). The similarities between items in each cluster are then evaluated using a distance function through an iterative process until all data points are in their current cluster. The centre of a cluster can be calculated using the centroid or medoid methods. The centroid of a cluster is determined by calculating the average of every data point in the cluster while the medoid of a cluster is determined by calculating the median data point of the cluster.

2.2 k-Means Clustering Algorithm

The k-means algorithm is a widely used clustering algorithm that is easy to understand and implement and can be used across a large variety of problems [1, 2, 3, 4, 5]. We have included the pseudo code of the k-means algorithm as Algorithm 1 for reference. There are two phases in the k-mean algorithm. During the initial phase, k random values are chosen from the dataset and assigned as the initial centroids of each of the clusters where k is the number of clusters [1]. In the second phase, the distance from each data point in a cluster to the centroid of every other cluster is calculated. If a data point is closer to a different cluster than the one it currently belongs to, the data point is moved to the closest cluster. After one iteration is complete and all data points have been moved, the centroids for each cluster are recomputed. This process continues until none of the data points move to a different cluster which occurs when the centroids of each cluster do not change. One of the issues with the k-means algorithm is the random selection of initial cluster centroids. This random selection has two main effects on the behaviour of the algorithm. First, it makes the algorithm non-deterministic in the final clusters it produces [1]. This means the number of data points in each cluster can vary greatly depending on the initial centroids chosen. Second, for large datasets, some random selections of initial centroids can greatly reduce the performance of the algorithm as more iterations are required to move data points between clusters [1, 2].

Algorithm 1: k-means clustering algorithm

```
Input:
    • $D = \{d_1, d_2, \ldots, d_n\}$: Set of $n$ data points in any order.
    • $k$: Number of clusters.
Output:
    • A set of $k$ clusters.
Steps:
    1) Randomly select $k$ data points from $D$ as initial centroids.
Repeat:
    2) Assign each data point $d_i$ to the cluster with the closest centroid.
    3) Calculate the new centroid of each cluster.
Stop when values of centroids do not change.
```

2.3 Related Work

There have been many improvements to the k-means clustering algorithm [1, 2, 3]. The global k-means clustering algorithm modifies the k-means algorithm by minimizing the distance between data points within the same cluster using the k-means algorithm as a local search procedure [2]. In this algorithm, the local k-means algorithm starts with a single cluster and calculates the centroid of this cluster. The global k-means algorithm then applies the k-means algorithm sequentially on an increasingly large number of clusters to solve a local problem [2]. At the start of each local search, the cluster centroids are initially placed at the currently most optimal position, which is determined by the centroids calculated in the previous iteration's local search [2]. This improves performance by reducing the number of data points that have to be moved between clusters. The global k-means algorithm produces deterministic results because it starts with a single cluster and computed centroid. It also has improved performance over the k-means algorithm because it always passes the most recently calculated centroid(s) of the current iteration to the next iteration, reducing the number of data points that are moved between clusters [2].

The improved k-means algorithm proposed by K. Nazeer and M. P. Sebastian [1] modifies the original k-means algorithm using a technique to systematically determine the initial centroids of each cluster with better accuracy than when selecting them randomly [1]. The clustering process is then applied to the initial clusters using a heuristic approach for improved efficiency [1]. In this heuristic approach, the distances from each data point in a cluster to their nearest cluster are saved and passed to the next iteration. In the next iteration the distances from the previous iteration are compared to the new distances from each data point in the cluster to the cluster they are closest to. If the distance calculated in the current iteration is less than or equal to the distance calculated in the previous iteration then the data point stays in its current cluster. Otherwise, the distance from the data point to every other cluster is calculated to find the new closest cluster to the data point.

3 Modified Approach

3.1 f-Clustering algorithm

In this paper, we propose the f-cluster algorithm, a modified version of the original k-means algorithm optimized for single-dimensional data. The f-cluster algorithm has two phases. In the first phase, we use Algorithm 2 to determine the initial cluster centroids which are used in Algorithm 3. To get the initial cluster centroids, Algorithm 2 sorts the data points in ascending order in step 1 and assigns a centroid value to each k clusters by choosing k data points evenly distributed across the dataset in steps 2 and 3. In step 4, we iterate over each data point exactly once and assign it to the cluster closest to it. The initial centroids are recalculated in step 5 to account for data points that have changed clusters in step 4. Lastly, step 6 returns the list of initial centroids, a list of the sum of data point values in each cluster, and the number of data points in each cluster.

In the second phase, we use Algorithm 3 which takes the output of Algorithm 2 and determines the final clusters. Step 1 calculates the positions of the cluster borders by using the number of data points in each cluster as seen in Eq.(1). Once the cluster borders are calculated we calculate the initial average of the border in Step 2 using the centroids of the two clusters neighbouring the cluster border. Step 3 is an iterative looping process that moves the cluster borders one position at a time until they are in the correct position. Step 3 is executed on each cluster border sequentially, only moving the next cluster border when the current cluster border is in the correct position. There are three cases to consider in Step 3. In case 3a, the cluster border continues to move one position to the left as long as the data point to the left of the cluster border is greater or equal to the cluster border average. In case 3b the cluster border continues to move one position to the right as long as the data point to the right of the cluster border is less than the cluster border average. In both case 3a and case 3b, the cluster border will no longer be moved when case 3c is true. In case 3c the cluster border is in the correct position because the data point directly to the left of the border is less than the cluster border average and the data point immediately to the right of the cluster border is greater or equal to the

Algorithm 2: Determine initial centroids

```
Input:
  • $D = \{d_1, d_2, \ldots, d_n\}$: Set of $n$ data points in any order.
  • $k$: Number of clusters.
Output:
  • $C = \{c_1, c_2, \ldots, c_k\}$: Set of initial cluster centroids in ascending order.
  • $S = \{s_1, s_2, \ldots, s_k\}$: The sum of data point values in each cluster.
  • $N = \{n_1, n_2, \ldots, n_k\}$: The number of data points in each cluster.
Steps:
  1) Sort $D$ in ascending order.
  2) Define the first data point $d_1$ as min and the last data point $d_n$ as max.
  3) Define $C$ as a linear interpolation between the min and the max with $k$ data points. (i.e. $c_1$ = min ($d_1$) , $c_k$ = max ($d_n$), etc)
  4) For each data point $d_i$:
       • Determine the cluster centroid closest to $d_i$.
       • Add the value of $d_i$ to the appropriate element in $S$.
       • Increment the count of the appropriate element in $N$.
  5) Re-calculate each centroid $c_i \in C$ as $\frac{s_i}{n_i}$.
  6) Return $C$, $S$, and $N$.
```

cluster border average. When this is the case, the algorithm continues to the next cluster border. When case 3c is true for every cluster border, then all data points are in the correct clusters; this is the terminating condition of Algorithm 3. When Step 3 has terminated, any clusters containing no data points are removed in Step 4. Then in Step 5, the centroid of each cluster is updated using the cluster sums and cluster counts lists. Since each cluster sum and count is updating continuously in Step 3, calculating the centroids in Step 5 is trivial and does not require iterating over every data point. Finally, Step 6 outputs the list of centroids, sums of each cluster, and the number of items in each cluster.

Algorithm 3: f-cluster algorithm

```
Input:
  • $D = \{d_1, d_1, \ldots, d_k\}$: Set of n data points in ascending order.
  • $k$: Number of clusters.
  • $C = \{c_1, c_2, \ldots, c_k\}$: Current set of cluster centroids in ascending order.
  • $S = \{s_1, s_2, \ldots, s_k\}$: The sum of data points values in each cluster.
  • $N = \{n_1, n_2, \ldots, n_k\}$: The number of data points in each cluster.
Output:
  • $C = \{c_1, c_2, \ldots, c_k\}$: Updated centroids for each cluster.
  • $S = \{s_1, s_2, \ldots, s_k\}$: Updated sums of data points values in each cluster.
  • $N = \{n_1, n_2, \ldots, n_k\}$: Updated numbers of data points in each cluster.
Steps:
  1) Determine the indexes that divide each cluster using $N$.
  2) Compute the cluster border averages by taking the average of the cluster centroids adjacent to each cluster border.
  3) Repeat for each cluster border $b_i$:
       • If the data point immediately before $b_i$ is greater than or equal to the corresponding adjacent cluster border average, move $b_i$ to the left until this is no longer the case. Update the appropriate elements in $S$ and $C$ for each cluster.
       • If the data point immediately after $b_i$ is less than the corresponding adjacent cluster border average, move $b_i$ to the right until this is no longer the case. Update the appropriate elements in $S$ and $C$ for each cluster.
       • If the data point immediately before $b_i$ is less than the corresponding adjacent cluster border average, and the data point immediately after the cluster border is greater than or equal to the corresponding adjacent cluster border average, $b_i$ is in the correct position.
  4) Delete any empty clusters.
  5) Update each $c_i \in C$ as $\frac{s_i}{n_i}$.
  6) Return $C$, $S$, $N$.
```

3.2 Algorithm Overview

Algorithm 2: Determine initial centroids
Input:

- $D = \{d_1, d_2, \ldots, d_n\}$: Set of n data points in any order.
- k: Number of clusters.

Output:

- $C = \{c_1, c_2, \ldots, c_k\}$: Set of initial cluster centroids in ascending order.
- $S = \{s_1, s_2, \ldots, s_k\}$: The sum of data point values in each cluster.
- $N = \{n_1, n_2, \ldots, n_k\}$: The number of data points in each cluster.

Steps:

1) Sort D in ascending order.
2) Define the first data point d_1 as min and the last data point d_n as max.
3) Define C as a linear interpolation between the min and the max with k data points. (i.e. c_1 = min (d_1) , c_k = max (d_n), etc)
4) For each data point d_i:
    - Determine the cluster centroid closest to d_i.
    - Add the value of d_i to the appropriate element in S.
    - Increment the count of the appropriate element in N.
5) Re-calculate each centroid c_i in C as $\frac{s_i}{n_i}$.
6) Return C, S, and N.

Algorithm 1: f-cluster algorithm

- Inputs
    - Dataset: All data points in ascending order
    - K: Number of clusters
    - Input means: Current set of cluster means in ascending order
    - Cluster sums: The sum of data points in each cluster
    - Cluster counts: The number of data points in each cluster
- Outputs
    - Output means: Updated means for each cluster
    - Cluster sums: Updated sums of data points in each cluster
    - Cluster counts: Updated numbers of data points in each cluster

Steps:

1) Determine the indexes that divide each cluster using the Cluster counts array.
2) Compute the average of the cluster means adjacent to the cluster borders (e.g. (Input means[2]+Input means[3])/2)
3) Repeat
4) If the data point immediately before the cluster border is greater than or equal to the corresponding adjacent cluster average, move the cluster border to the left until that is no longer the case. Update the sums and counts for each cluster as you do this. Note: We do not update the mean of each cluster until the end.
5) If the data point immediately after the cluster border is less than the corresponding adjacent cluster average, move the cluster border to the right until that is no longer the case. Update the sums and counts for each cluster as you do this.
6) If the data point immediately before the cluster border is less than the corresponding adjacent cluster average, AND the data point immediately after the cluster border is greater than or equal to the corresponding adjacent cluster average, do not move the cluster border and skip to the next cluster border.

For each cluster border:

1) Calculate the updated mean of each cluster as the updated sum divided by the updated count as Output means.
2) Delete any clusters that have zero items
3) Return Output means, Cluster sums, Cluster counts.

3.3 Proof of Correctness

This algorithm needs a couple requirements in order to function properly:

1) There must be no duplicate cluster centroids
2) Every cluster must contain at least one item in it

Note: if a data point is the same distance from two cluster centroids, this algorithm favours the cluster with the larger centroid. This case is ignored in this proof, as the chance of this happening is vanishingly small in real world datasets due to floating point precision.

To prove correctness of the algorithm, we must prove the following properties

Property 1. *Each item in the dataset will be assigned to the cluster with the closest centroid, and*

Property 2. *The cluster borders cannot cross each other (even if the algorithm is done in parallel), as this would cause sum/count calculations to be incorrect.*

3.3.1 Algorithm inputs

- $D = \{d_{11}, d_{12}, \ldots, d_{1n_1}, \ldots, d_{kn_k}\}$ where the dataset D contains elements d_{ij} where i is the cluster and j is the index within the cluster.
- k, the number of cluster where $k \geq 2$

- $C = \{c_1, \dots, c_k\}$ where C is a set containing the centroids of each cluster where each c is unique and $c_1 < c_2 < \dots < c_k$. This sequence is constant until the very end of the algorithm.
- $S = \{s_1, \dots, s_k\}$ where S is a set containing the sums of data points in each cluster
- $N = \{n_1, \dots, n_k\}$ where N is a set containing the number of data points in each cluster
- $N_{\text{sum}} = n_1 + \dots + n_k$ where N_{sum} is the total number of data points in D

3.3.2

In this proof we use the concept of cluster borders separating the data set into its clusters. This is an integral concept as the algorithm leans heavily on it. In order to better visualize the intuition of the algorithm, we model the data and associated clusters as a sorted list of data points in clusters separated by cluster borders, i.e. the first border separates clusters 1 and 2, etc.. The set of all cluster borders positions is $B = b_1, \dots, b_{k-1}$. Visually, this looks like the following diagram:

$$D = \underbrace{d_{11}, d_{12}, \dots, d_{1x_1}}_{\substack{s_1 = d_{11} + \dots + d_{1x_1} \\ c_1 = \frac{s_1}{x_1}}} \overbrace{\;|\;}^{\substack{b_1 = \\ x_1}} \underbrace{d_{21}, d_{22}, \dots, d_{2x_2}}_{\substack{s_2 = d_{21} + \dots + d_{2x_2} \\ c_2 = \frac{s_2}{x_2}}} \overbrace{\;|\;}^{\substack{b_2 = \\ x_1 + x_2}} \dots \overbrace{\;|\;}^{\substack{b_{k-1} = \\ x_1 + x_2 + \dots + x_{k-1}}} \underbrace{d_{k1}, d_{k2}, \dots, d_{kx_k}}_{\substack{s_k = d_{k1} + \dots + d_{kx_k} \\ c_k = \frac{s_k}{x_k}}} \tag{1}$$

3.3.3 Proof outline for Property 1
Proof outline for Property 1

1.

Property 1.1. *Data points can only be closest to one of the clusters they're adjacent to or inside of, i.e. d_{ij} is closest to either c_{i-1}, c_i or c_{i+1}*

Definition 1.2. *Definition: a cluster border b_i is in the correct location if all data points before b_i are closer to c_i than c_{i+1}, and all data points after b_i are closer to c_{i+1} than c_i.*

Property 1.3. *An equivalent statement: If a cluster border b_i is in the correct location, the item x immediately before b_i satisfies $x < (c_i + c_{i+1})/2$, and the item y immediately after b_i satisfies $y > (c_i + c_{i+1})/2$*

Property 1.4. *Once the first i cluster borders have been put in the correct location, it follows that all data points before b_i have been assigned to the correct cluster*

Property 1.5. *Once the last cluster border b_{k-1} has been put in the correct location, all data points after b_{k-1} have been correctly assigned to the last cluster.*

Conclusion: By properties 1.4 and 1.5, once the algorithm terminates, all data points have been assigned to the correct cluster.

Proof of Property 1
Recall that:

1) $c_1 < c_2 < \dots < c_k$
2) $d_{11} \le d12 \le \dots \le d_{1n_1} \le \dots \le d_{k1} \le \dots \le d_{kn_k}$ Due to the properties of means, we know that for any cluster i: $d_{i1} \le c_i \le din_i$

Property 1.1 We are now going to prove that data points can only be closest to one of the clusters they're adjacent to or inside of, i.e. d_{ij} is closest to either c_{i-1}, c_i, or c_{i+1}.

Given an item d_{ij} in a cluster i where j is the index of the item inside the cluster, which cluster centroids could d_{ij} conceivably be closest to? If $i = 1$, then it should make sense that d_{ij} is either closest to c_1 or c_2. It wouldn't make sense for it to be closer to c_3 (or any centroid higher than that), since $c_3 > c_2$. Putting it more formally: We want to minimize the L2 Norm in 1 dimension $|d_{1i} - c_a|$ over the variable a. By the algebraic definition of absolute values, $|d_{1i} - c3| = c_3 - d_{1i}$ because $c_3 > d_{1i}$ and $|d_{1i} - c_2| = c_2 - d_{1i}$ because $c_2 \ge d_{1i}$. Then it follows that: $|d_{1i} - c_3| > |d_{1i} - c_2| \iff c_3 - d_{1i} > c_2 - d_{1i} \iff c_3 > d_2$ which is one of our given properties.

This same argument applies to any cluster i: Data points can only be closest to one of the clusters they're adjacent to or inside of i.e. d_{ij} is closest to either c_{i-1}, c_i, or c_{i+1}

Definition 1.2 For a cluster border b_i to be in the correct position, all data points to the left of b_i are closer to ci than c_{i+1}, and all data points to the right of bi are closer to c_{i+1} than ci.

Property 1.3 For convenience, define $m_i = (c_i + c_{i+1})/2$ as the mean of two adjacent cluster centroids.

Lemma: if an arbitrary element d_{ij} is less than m_i, then it's closer to c_i than c_{i+1}. Additionally, if an arbitrary element d_{ij} is greater than m_i, then it's closer to c_{i+1} than c_i.

- If $d_{ij} \leq c_i$ (and hence automatically closer to c_i than c_{i+1}):
  - Then $d_{ij} < m_i$ since $m_i > c_i$
- If $d_{ij} > c_i$:
  - Then $|d_{ij} - c_i| = d_{ij} - c_i$ and $|d_{ij} - c_{i+1}| = c_{i+1} - d_{ij}$ (since $d_{ij} < c_{i+1}$ must be true), after which it follows that: $(d_{ij} - c_i) < (c_{i+1} - d_{ij}) \iff 2d_{ij} < c_i + c_{i+1} \iff d_{ij} < (c_i + c_{i+1})/2 \iff d_{ij} < m_i$.
  - This also applies to $(d_{ij} - c_i) > (c_{i+1} - d_{ij})$:
  - $(d_{ij} - c_i) > (c_{i+1} - d_{ij}) \iff 2d_{ij} > c_i + c_{i+1} \iff d_{ij} > (c_i + c_{i+1})/2 \iff d_{ij} > m_i$

This extends without loss of generality to an arbitrary element $d_{(i+1)j}$ in cluster $i + 1$. Then by the lemma, our condition for a cluster border being in the correct position is then that the data point x immediately before the border satisfies $x < m_i$, and the data point y immediately after the border satisfies $y > m_i$, ensuring that both elements have the shortest distance to the centroid of the cluster it belongs to. Then by the transitive property, all elements before x are also closer to c_i than c_{i+1}, and all elements after y are also closer to c_{i+1} than c_i.

Property 1.4 We will now prove that once the first i cluster borders have been put in the correct location, it follows that all data points before border b_i have been assigned to the correct cluster

Our algorithm starts with the first cluster border at position b_1, moving it left/right until it's in the right place. Since every element before b_1 is closer to c_1 than c_2, and each item must be closest to either c_1 or c_2, we can conclude each item before b_1 has been assigned to the correct cluster, i.e. the first cluster.

We then move onto the second cluster border at position b_2, and adjust that to the left/right until it's in the right place. The same argument as before applies; since every item before b_2 is closer to c_2 than c_1 or c_3, and since every item before b_2 must be closest to either c_1, c_2 or c_3, everything before b_2 must be assigned to the correct cluster. We can continue this process for i cluster borders, resulting in every data point before b_i being assigned to the correct cluster.

Property 1.5 We will now prove that once the last cluster border b_{k-1} has been moved into the correct position, all data points after b_{k-1} have been assigned to the correct cluster.

Following from property 1.1 and definition 1.2, all data points after b_{k-1} must be closer to ck than ck-1, and all data points after b_{k-1} must be closer to either ck or ck-1. Therefore, once the last cluster border b_{k-1} has been moved into the correct position, all data points after b_{k-1} have been assigned to the correct cluster.

Conclusion: Therefore, by properties 1.4 and 1.5, once the algorithm terminates, all data points have been assigned to the correct cluster.

Proof outline for Property 2

2.

Property 2.1. *As a cluster border is being adjusted, it cannot cross an unadjusted border to the left or right.*

Property 2.2. *As a cluster border is being adjusted, it cannot cross an already-adjusted border to the left or right.*

Conclusion: By properties 2.1 and 2.2, cluster borders cannot cross each other, even if the f-cluster algorithm is executed in parallel.

Proof of Property 2 Property 2.1: Let us start with an un-adjusted cluster border bi surrounded by unadjusted cluster borders b_{i-1} and b_{i+1}. If bi is moved to the left to get to its correct position, it cannot cross to the left of d_{i1} (the data point just to the right of b_{i-1}), since $d_{i1} \leq c_i < c_{i+1}$. This means that d_{i1} is closer to c_i than c_{i+1}, but if b_i were to cross to the left of d_{i1}, that would imply it was closer to c_{i+1}. This also means that b_i can move at most n_{i-1} data points to the left.

If b_i is instead moved to the right to get to its correct position, it cannot cross to the right of $d_{i+1n_{i+1}}$ (the data point just to the left of b_{i+1}). This is because $d_{i+1n_{i+1}} >= c_{i+1} > c_i$, meaning that $d_{i+1n_{i+1}}$ is closer to $c_i + 1$ than c_i, however if b_i crossed to the right of $d_{i+1n_{i+1}}$ that would imply it was closer to c_i than c_{i+1}. This also means that b_i can move at most $n_{i+1} - 1$ data points to the right.

Combining the fact that b_i can move at most n_{i-1} spots to the left, or move at most $n_{i+1} - 1$ spots to the right, the absolute maximum a cluster border can move in one iteration is $\max(n_{i-1}, n_{i+1} - 1)$.

Property 2.2: Let us start with an unadjusted cluster border b_i surrounded by adjusted cluster borders b_{i-1} and b_{i+1}. If b_i is moved to the left to get to its correct position, it cannot cross to the left of b_{i-1}, as that would imply that an element left of b_{i-1} was closer to c_{i+1} than c_{i-1}, when we already know that the element is closest to c_{i-1}.

If b_i is instead moved to the right to get to its correct position, it cannot cross to the right of b_{i+1}, as that would imply that an element to the right of b_{i+1} was closer to c_{i-1} than c_{i+1}, when we already know that the element is closest to c_{i+1}.

Conclusion: By properties 2.1 and 2.2, cluster borders cannot cross each other, even if the f-cluster algorithm is executed in parallel, and the maximum number of spots a cluster border b_i can be moved is $\max(n_{i-1}, n_{i+1} - 1)$

3.4 Empirical Runtime Analysis

Since each iteration of the f-cluster algorithm produces identical results to the normal k-means algorithm, it logically follows that it will take the exact same number of iterations to converge to its final result. The number of iterations is known to scale with the size of the dataset, but is still an area of active research and is highly variable depending on the initial conditions (Note: reference a paper that shows a bound on the number of iterations here), and so our analysis of the total runtime will simply include iterations as a variable i.

We also won't be giving any mathematical proofs of the run times quoted here as the proof would likely be non-trivial enough to be out of the scope of this project.

For each iteration and each cluster border, we must check if the cluster border needs to be moved, and then move it into the correct position if necessary. If we say that the total amount the cluster borders moved plus $k-1$ (to account for checking if each cluster border needs to be moved) is x, then the total number of operations done for a single iteration is simply $O(x)$.

For convenience, define $N_{\text{sum}} = n_1 + \cdots + n_k$, so that N_{sum} is the number of data points in the dataset

Our first observation is that $x < N_{\text{sum}}$: First, note that the maximum number of spots a cluster border b_i can move is $\max(n_{i-1}, n_{i+1} - 1)$ as was shown in part 2 of the proof of correctness. Assume the worst case scenario — that every border except the last moves as far to the left as it can go, and the last border goes to either the left or right. b_1 will move $n_1 - 1$ spots to the left, b_2 will move $n_2 - 1$ spots to the left, etc... and the last border will move $\max(n_{k-1} - 1, n_k - 1)$ spots, either to the left or right depending on which is larger. So we have

$$x = (n_1 - 1) + (n_2 - 1) + \cdots + (n_{k-2} - 1) + \max(n_{k-1} - 1, n_{k-1}) + (k-1)$$
$$= n_1 + n_2 + \cdots + n_{k-2} + \max(n_{k-1} - 1, n_k - 1) + 1 < n_1 + n_2 + \cdots + n_{k-2} + \max(n_{k-1}, n_k) < n_1 + n_2 + \cdots + n_{k-1} + n_k \tag{2}$$

thus proving the property that $x < N_{\text{sum}}$.

As we noted in the proof of correctness, there are at most 3 possibilities for which centroid a given data point is closest to. This implies a different optimized k-means algorithm in which you only check the nearest 2 or 3 centroids, instead of checking all k centroids as naive k-means does. The fact that $x < N_{\text{sum}}$ proves that our algorithm is faster than even this optimized version of k-means.

For the whole algorithm including all iterations, the total number of operations done is simply the sum of x values over all iterations, which we denote x_{sum}, making the total number of operations for the clustering portion of k-means $O(x_{\text{sum}})$. Below is a plot of how x_{sum} scales with the size of the dataset, for a few different distributions of datasets.

As we can clearly see, x_{sum} scales linearly with the size of the dataset, although the slope changes based on the distribution. This is simply the big O constant scaling factor, and we can safely ignore it. We can then conclude that the actual clustering part of K-means, including all iterations,takes a total of $O(n)$ operations. Of course, the base requirement to use f-cluster is that the dataset is sorted. However since real data generally doesn't come pre-sorted, we can consider the full runtime of clustering a dataset with a size of n using f-cluster to be O(n log n). This is in stark contrast to other optimized K-means algorithms which take $O(n^2)$ time. (Reference paper with $O(n^2)$ optimized k-means here).

As a side note, if one were to zoom in on the start of the graph posted above, one would notice that the slope isn't entirely constant. However this is okay because it quickly converges towards a constant slope as the size of the dataset increases. Below is a graph of $x_{\text{sum}}/N_{\text{sum}}$:

In addition, as with normal K-means, our algorithm is parallelizable. As we noted in the proof of correctness, the cluster dividers cannot cross each other and interfere with sum/count calculations. This holds true even when done in parallel, and as such parallelizing the algorithm is possible. If you parallelized both the sorting and the clustering steps (it would be pointless to parallelize just one), the total runtime to sort and cluster the data would be $O(\log n + n/(k-1)) = O(n/k)$ (Note: Reference a paper here that mentions the complexity of parallelized sorting algorithms).

4 F-clustering algorithm

5 Results

6 Visualization

7 Conclusion