Fair Clustering Under Misrepresented Data

Harikesh
2019CS10355
Dept. of Computer Science
Indian Institute of Technology, Delhi

Deepanshu Yadav
2019CS10343
Dept. of Computer Science
Indian Institute of Technology, Delhi

I. K-Means Clustering

A. Introduction

K-Means clustering is an iterative, unsupervised learning algorithm which is used to solve clustering problems in various computer science branches. This concept is used to divide an unlabeled data set into some clusters/groups such that each of the points in our initial dataset belongs to just one cluster which is having similar characteristics compared to this point. This means that all of the clusters we get after dividing the data set are disjoint i.e. no intersection between the two clusters. Now, the goal of this algorithm is to minimize the sum of square of distances between the datapoint and the cluster in which that particular datapoint belongs i.e. we will try to minimize the squared euclidean distance to the closest center. This algorithms works by taking the mean of the data points in a cluster. Now, clearly this is a NP-Hard problem and is thus it is very difficult to find the best possible centers we can get and therefore to compute it efficiently we can use some heuristic algorithms like Lloyd's algorithm [1]. On thing to note here is that although the heuristic algorithms don't give us the best possible answer but they are really close to the best possible answer and also take too much less time compared to the NP-Hard problems.

B. Working of the algorithm

Now, let's say that we want to divide our dataset into k clusters and assuming that the value of k is predetermined we can start by taking unlabeled dataset as the input and then try to find the k best clusters possible on any given dataset points. We will start by randomly allocating k-centers to the dataset. (One more thing is that it is not necessary to choose the points/centers from the set of points in the data set but the centers can be any point in the cartesian plane). Then, we will assign each point to it's nearest center among the k-chosen centers. Now, each data points belongs to the cluster which is closest to it by euclidean distance and therefore, we will be able to find all the points in a cluster (associated to a center). After that, we will reallocate the centers of our algorithm by taking the mean of the points belonging to that cluster. Again we will do the assignment and updating steps. This, will go on till convergence.Thus, the main steps of the algorithm are as follows:

Step 1: Use the given or determine the value of k, number of clusters.

Step 2: Select k random points on the cartesian plane.

Step 3: Assign each point in the data set to closest of the above defined random points.

Step 4: This lead to the formation of k clusters. Take the mean of the points of each cluster and make that the new center.

Step 5: Go back to step 3 if there are some changes/ different assignments otherwise the algorithm is converged terminate it.

Step 6: Thus, we have nearly the best possible k clusters using polynomial time.

C. Problem Description

Now, we will move towards our work which is closely related to the k-clustering problem as discussed by us earlier. The problem statement is as follows:

For any generic case assume that there are two types of populations/parameters n_1 and n_2 which are scattered uniformly in an area. Assume that area is a D×D square block which can be divided into small 1×1 squares. Each of these small squares is associated with one of the above parameters (n_1 or n_2). We can understand it by saying that each of the small square has either n_1 (Type1) or n_2 (Type2) population which can further be understand by saying that Type1 block has n_1 people living in it while Type2 block has n_2 people living in it.

Now, D, n_1, n_2, k are our parameters and we need to open/find k centers/facilities in the above area with the aim that the sum of cost paid by each point (the euclidean distances travelled by each of the person) in our dataset to the nearest facility is minimum. The solution of this is simply the algorithm discussed above where we will just simulate our algorithm by generating the points as discussed in the description and then simply running the k-mean clustering theorem on the generated input. In our case, k will be related to total population (total number of data points) as follows:

$$\boxed{k \propto \text{ total points } \propto D^2 \times (n_1 + n_2)}$$

Now, in the algorithm we are basically trying to get the best centroids as we move on in the algorithm. We are storing the centroids in the centroid array which will be updated as soon as we get better centroids in the new_centroids array. Also, change is the variable which will be crucial for the number of iterations we will do. Also, we assume that we have all the points generated in the points array by generating randomly in D×D square.

This is the pseudo code for our algorithm:

Algorithm 1 Finding the best k clusters

```
Require: D, $n_1$, $n_2$, k, n = $n_1$ + $n_2$, points[]
Ensure: Best possible centroids and cluster array
  centroids ← [[0,0]×k]
  change ← True
  cluster ← [0×n]
  sq_dist ← n
  sq_dist_cluster ← [0×k]
  while change do
    curr ← 0
    change ← False
    new_centroids ← [[0,0]×k]
    count_points ← [0×k]
    sq_dist_cluster ← [0×k]
    for j= 0 to n do
      min_dist_cluster ← 0
      min_dist ← 1
      point ← points[j]
      for i= 0 to k do
        centroid ← centroids[i]
        if distance(point, centroid) $\leq$ min_dist then
          min_dist ← distance(point, centroid)
          min_dist_cluster ← i
        end if
      end for
      curr ← curr + min_dist
      cluster[j] ← min_dist_cluster
      count_points[min_dist_cluster]+=1
      new_centroids[min_dist_cluster][0]+=point[0]
      new_centroids[min_dist_cluster][1]+=point[1]
      sq_dist_cluster[min_dist_cluster]+=min_distance
    end for
    for i= 0 to k do
      new_centroids[i][0] / = count_points[i]
      new_centroids[i][1] / = count_points[i]
    end for
    if curr < sq_dist then
      change ← True
      sq_dist ← curr
      centroids ← new_centroids
    end if
    return cluster, centroids
  end while
```

The results obtained by the algorithm are shown below pictorically:

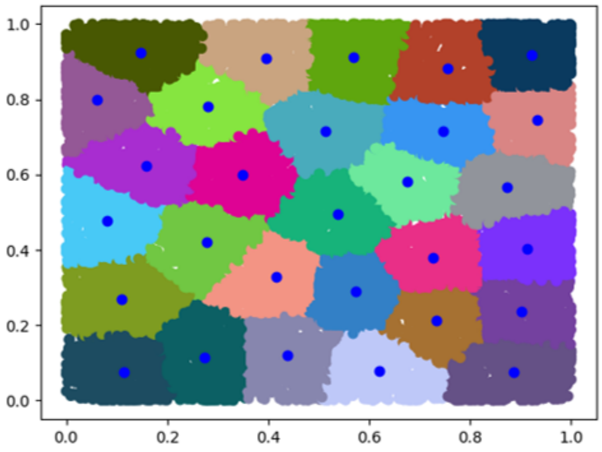

Fig. 1. Final centroids(in blue) in a very dense area

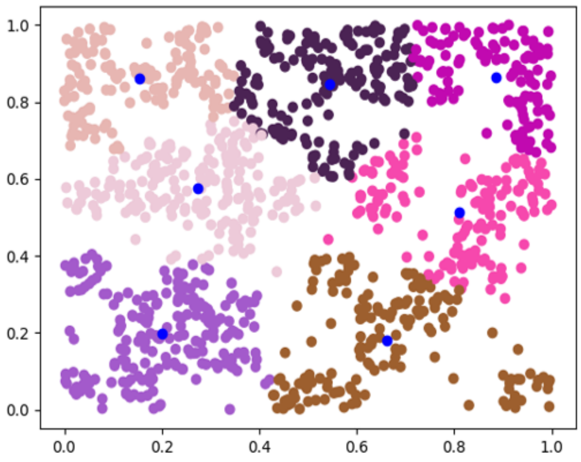

Fig. 2. Final centroids(in blue) in a very sparse area

D. Practical examples and Foul Play

Our problem can be formulated in a lot of practical examples some of which include formation of booth-polls in an area where two type of population lives, Black and White populations living somewhere like America, building hospitals so that they are equally accessible by two populations.

Now, there may be the case where one of the populations under-reports (or under-reported by another population). Although it is quite obvious that none of the population want to under-report themselves because in case they under-report, they will be the one suffering (as the booth/hospital will be farther from them). Thus, it can be interpreted as suppose the surveyor is from one of the populations and is therefore biased towards them and then the foul play on data is possible. One more interesting example of this can be formed by the timely census of a country, let's say that it occurs after every 10 years in a country (assume India), and someone want to compare the population of the two close cities (A and B) 5 years after the last census. One is also provided with the average percent increase of the population of cities of that country per year (α). Now, if population of one of the city (A) grows near to this rate i.e. α and the other one (B) grows at a rate of β where $\beta > \alpha$ then in our analysis of assuming that the increase in population of both of cities is α, in a way we have under-reported the population of city B. Thus, it can also be treated as a practical example to our problem.

E. Analysis of Under-reporting

As a general case, assume that Type1 population under-reports it's parameter given by some percentage (α). This will have some serious effects on both the populations (Type1 and Type2). As Type1 population has less number of people now (under-reported population) then their influence (sum of squares distance) will have less hold now (because of less number of people) and the facilities are shifted towards Type2 population. Hence, in a way this type of population (Type1) has to travel more now to avail the facility.
Based on the above observation we are coining two specific terms related to the two types of populations based on the distinctions of their privilege to the facilities. In the above example Type1 population is **unprivileged population** while Type2 population is **privileged population**.

Consider the underreporting to be α. Let the expected total number of points be n and the expected number of reported points be m. It is clear that $m \leq n$.

Then n = $\frac{D^2 \times (n_1 + n_2)}{2}$

and m = $\frac{D^2 \times (n_1 (1 - \frac{\alpha}{100}) + n_2)}{2}$

The number of non reported points, p = n-m
$\implies$ p = $\frac{D^2 n_1 \alpha}{200}$
$\implies$ p $\propto \alpha$

We can assume that the expectation cost for any point in the reported point is same. Let this value be c_1. In our algorithm, after calculating the centroids and partition for the reported points, we later come to know about non reported points. Hence we try to assign them to the nearest centroid. Let the expectation cost for non reported points be c_2. Thus it is clear that $c_2 > c_1$ as the point which is used in the algorithm has better chance of modifying the centroid location to be near to it rather than the non reported point which is never used in the algorithm.

Thus the total cost = m.c_1 + p.c_2
$\implies$ total cost = (n-p).c_1 + p.c_2
total cost = n.c_1 + p.$(c_2 - c_1)$

n.c_1 is fixed and doesn't depend of α and p $\propto \alpha$. Hence we can see that total cost increases almost linearly with α which explains the graph.

F. Results and Plots

Now, we will plot Cost (or distance travelled) as a function of α for different total number of points (population) in the dataset. We will try to obtain some trend for all the types of costs (Privileged cost, Total cost, Unprivileged cost) if we vary the percentage of under-reporting of one of the population.

Now, we can observe that unprivileged cost demonstrate linear increment. On the contrary, privileged cost demonstrates linear decrement. Talking about total cost we can say that total cost shows a linear increment with much less slope compared to unprivileged cost.

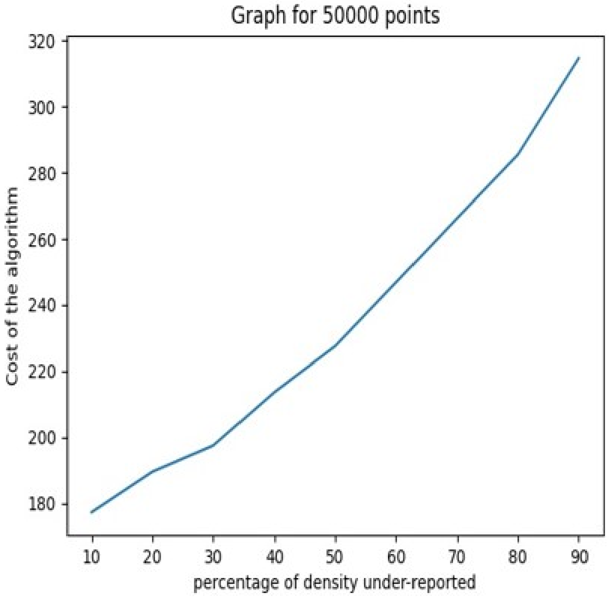

Fig. 3. Total cost v/s α

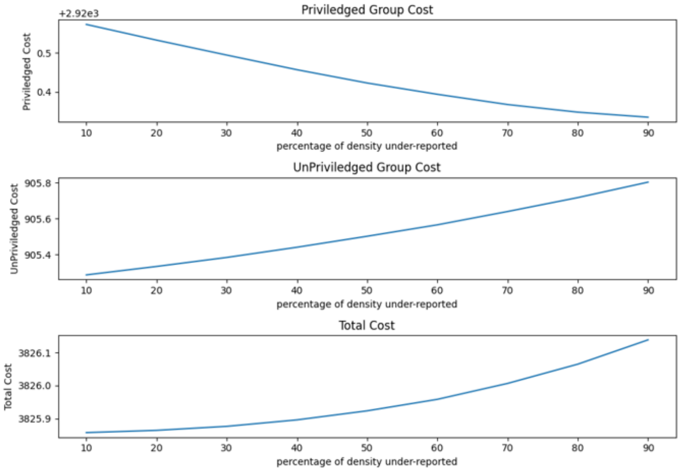

Fig. 4. All Costs vs α

G. Theoretical analysis of straight line

H. Similar profile problems (slight changes)

In this part, we will try to show that randomness in the generation of data points and some other factors like (randomness in the squares patterns) has no effect on our results we have obtained previously. For this hypothesis we will be testing our results on two other similar algorithms with slight differences:

1) Alternate squares algorithm (every square is adjacent to other type of square)
2) Implementation with n = n_1 = n_2, but squares of Type1 and Type2 differ in number.

Given below are the plots obtained for these two algorithms:

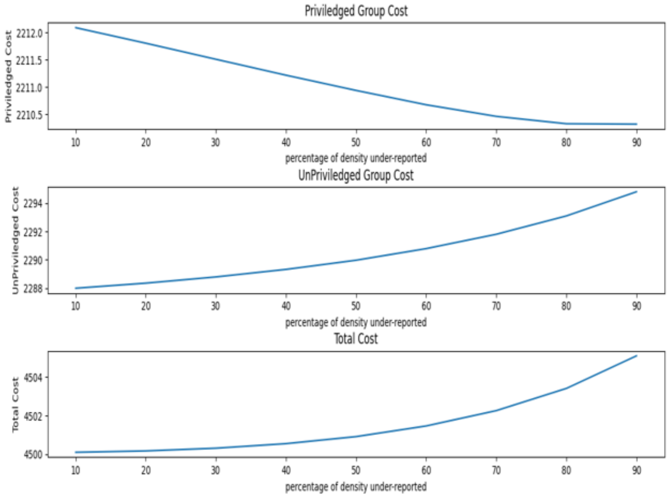

Fig. 5. Alternate Algorithm 1)

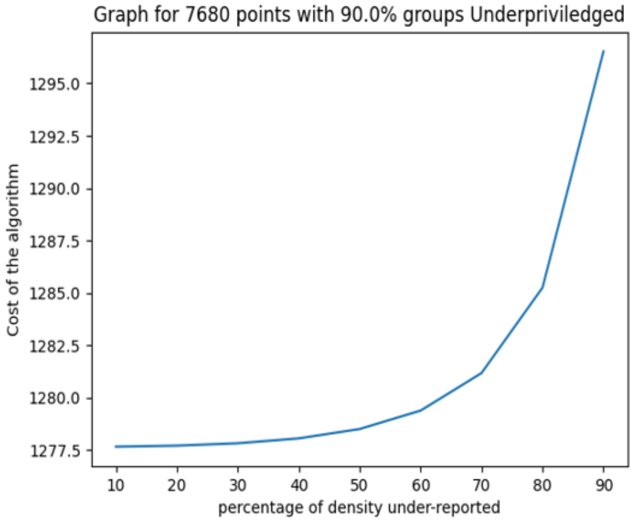

Fig. 6. Alternate Algorithm 2)

II. Tackling the problem

A. Solutions to Under Reporting Problem

Now, from the above results we can see that there is unfair distribution/disadvantage in case of unprivileged population as they have to pay more (cover more distance to reach the nearest center). This means the centers are over-crowded by privileged population. To tackle this problem we have to give a solution independent of α (percentage of underreporting) such that in that model/solution we can ensure the fairness in resource allocation. Also, while implementing this we have to make sure that we limit the percentage of both types of population (Type1 and Type2) at any center/facility.
The graphs that have been presented above give the impression that underreporting a group could be bad, as the underreported group would then have a greater distance to travel than it did before. Because of this, there may be a significant difference between the average cost for the two groups with the unpriviledged group at disadvantage. Thus, we would want to modify the algorithm so as to ensure similar average cost for both the groups. Also, we can see that the k-clustering step will not work here as in that step we take the mean of the points in a cluster while updating the new center of any particular cluster but here we have to also take care of the type of the points i.e. they belong to Type1 population or Type2. So, we have to come up with some new step in our fair algorithm.

B. Algorithm 1

For this algorithm, for each iteration, the step involving assigning each point to the nearest cluster remains the same. We would change the way how for a given cluster of points, the centroid point is defined.

1) Working [2]: Let $\mathcal{S}$ be the set of points which need to be divided into k clusters. Let $\mathcal{A}$ be the set of points which are from group 1 and $\mathcal{B}$ be the set of points which are from group 2.
Let $\delta(X, Y)$ denote the cost of the algorithm with given centroids as X = $\{x_1, x_2, \ldots, x_k\}$ and partition Y = $\{y_1, y_2, \ldots, y_k\}$
$\implies$ $\delta(X, Y) = \sum_{i=1}^{k} \sum_{p \in y_i} D(p, x_i)$ where D denotes the Euclidean Distance

Let $\mathcal{S}_X$ denotes the partition for $\mathcal{S}$ on clusters X such that each point is assigned to its nearest cluster.

Thus in the previous section, we had worked on an algorithm to find centroids $\mathcal{C}$ so as to minimize $\delta(\mathcal{C}, \mathcal{S}_\mathcal{C})$.

In this section, we will focus on minimizing difference of $\frac{\delta(\mathcal{C},\mathcal{A}_\mathcal{C})}{|\mathcal{A}|}$ and $\frac{\delta(\mathcal{C},\mathcal{B}_\mathcal{C})}{|\mathcal{B}|}$ in addition to minimizing $\delta(\mathcal{C}, \mathcal{S}_\mathcal{C})$.

Let a = $\frac{\delta(\mathcal{C},\mathcal{A}_\mathcal{C})}{|\mathcal{A}|}$, b = $\frac{\delta(\mathcal{C},\mathcal{B}_\mathcal{C})}{|\mathcal{B}|}$.
Our algorithm tries to minimize max(a,b). Thus if a>b, it implies that the cluster is closer to group 2 center and hence we move it closer to group 1 center. Let u = $\{u_1, u_2 \dots u_k\}$ and v = $\{v_1, v_2, \dots v_k\}$ denote the mean points for each of the k partitions of $\mathcal{A}_\mathcal{C}$ and $\mathcal{B}_\mathcal{C}$ respectively.

Now when we update the cluster point c_i to new point for a given partition, it must lie along the line joining u_i and v_i so as to minimize the total cost (as the projection component from c_i to that line would now be 0 hence minimizing the cost). Thus, now we want to find the point c_i along the line $\mathcal{L}$ (the line joining u_i and v_i) so as to minimize the difference of $\frac{\delta(c_i,\mathcal{A}_{\mathcal{C}_i})}{|\mathcal{A}|}$ and $\frac{\delta(c_i,\mathcal{B}_{\mathcal{C}_i})}{|\mathcal{B}|}$.

To find such a centroid $\mathcal{C}$ for a given partition, we have used line search algorithm. It find the centroid point based on the previous average costs for both the groups.

For each i = 1,2, ... k, we can find new c_i using Line search Algorithm [3].

Algorithm 2 Finding out new c_i using Line Search

```
$\alpha_i \leftarrow \frac{|\mathcal{A}|}{|\mathcal{S}|}$ $and$ $\beta_i \leftarrow \frac{|\mathcal{B}|}{|\mathcal{S}|}$
$l_i \leftarrow distance\ b/w\ u_i\ and\ v_i$
$f_i \leftarrow \frac{\delta(u,\mathcal{A}_u)}{|\mathcal{A}|}$
$g_i \leftarrow \frac{\delta(v,\mathcal{B}_v)}{|\mathcal{B}|}$
$\gamma \leftarrow 0.5$
for t=2 to 100 do
    $y_i \leftarrow \frac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i+(1-\gamma)\beta_i}$
    $f_i \leftarrow \frac{\delta(u,\mathcal{A}_u)}{|\mathcal{A}|} + \sum_{i=1}^{k} \alpha_i y_i^2$
    $g_i \leftarrow \frac{\delta(v,\mathcal{B}_v)}{|\mathcal{B}|} + \sum_{i=1}^{k} \beta_i (l_i - y_i)^2$
    if $f_i = g_i$ then
        break
    else if $f_i > g_i$ then
        $\gamma \leftarrow \gamma + 2^{-t}$
    else
        $\gamma \leftarrow \gamma - 2^{-t}$
    end if
end for
$Update\ c_i \leftarrow \frac{y_i v_i + (l_i - y_i) u_i}{l_i}$
```

Here we are using an approach similar to binary search algorithm in order to find out the ratio in which the line from u_i and v_i must be divided to ensure almost equal average costs for both the groups. Whenever the cost of a group exceeds the other, we move the centroid away from the mean of such group on the line l by a factor of $l_i 2^{-t}$ where t is the number of iteration till now incremented by 1. Thus the difference between their distance converges after some iterations and we get almost similar average costs for both the groups. Thus we can see that this approach gives similar average costs for both the groups irrespective of the underreporting.

C. Plots and Results

Now, the next few graphs show the result of this algorithm which are basically depicting the average cost of datapoint in both the types of population (Type1 and Type2). From the graphs we can see that the average cost of the two types of datapoint is almost equal and this means on an average both the types of population have to almost pay equal (travel equal distance), which sign towards fair distribution.

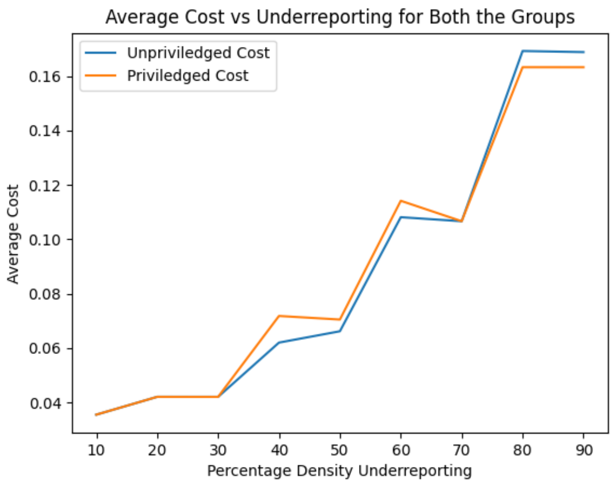

Fig. 7. Plot for nearly 10^4 points

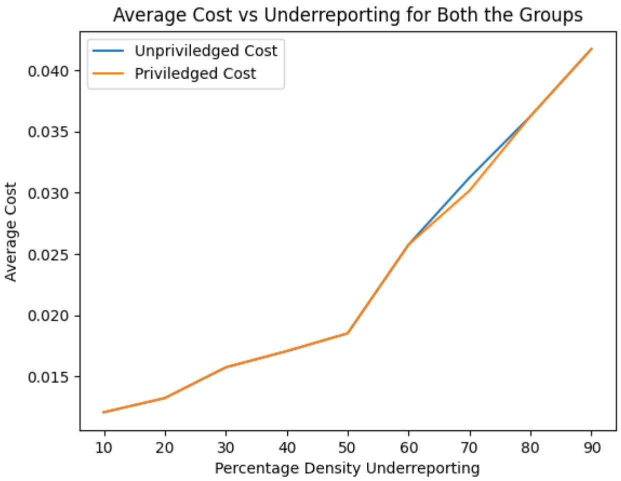

Fig. 8. Plot for nearly 2×10^5 points with more k

D. Algorithm II

For this algorithm, for each iteration, the step involving assigning each cluster centroid to the mean of all its including points remains the same. We would change the way how for a given cluster centroids, which point is assigned to which centroid.

1) Working: As we have seen in the past, one of the possible explanations for why we are experiencing bias against a particular group is that any given centroid is receiving an abnormally large number of points from that group in comparison to the points it receives from the other groups. It's possible that this will lead the centroid to move noticeably closer to the larger group, putting the other group at a disadvantage.

As a result, under this strategy, we have established a maximum limit for the percentage of a group's points that can be ascribed to a particular cluster for each of the k different clusters. The goal of this step is to guarantee that the centroid in the subsequent step will be located around the same distance from both of the group points.

Let $\mathcal{S}$ be the set of points which need to be divided into k clusters. Let $\mathcal{A}$ be the set of points which are from group 1 and $\mathcal{B}$ be the set of points which are from group 2.

Let $p_1 = \frac{|\mathcal{A}|}{|\mathcal{S}|}$, $p_2 = \frac{|\mathcal{B}|}{|\mathcal{S}|}$

Then in our algorithm, for cluster i, we are assigning the nearest points to cluster i with an upper bounds on the percentage of points for both the groups out of the total points assigned to cluster i.

Suppose t_i be the total number of points assigned to cluster i. Let a_i and b_i be the points assigned to cluster i from group 1 and group 2 respectively.

Then according to our algorithm, $\frac{a_i}{t_i} \leq p_1 + 0.1$ and $\frac{b_i}{t_i} \leq p_2 + 0.1$

For example, consider the case where the total number of points is same for both the groups. Then for each cluster, we would limit the percentage of any group points to atmost 60%.

The value of 0.1 (10%) is configurable and changed based on our preference of minimizing the total cost or the difference between the average cost for both the groups.

Another variation of this algorithm is in its fractional form, where each point may be sent in non-negative fractions to each of the clusters. The sum of all such fractions for a point is 1. In this algorithm, while updating the centroid, we would take the weighted mean.

For all i,

$$c_i = \frac{\sum_{j=1}^{n} f_{ji} c_j}{\sum_{j=1}^{n} f_{ji}}$$

Here c_j denotes the coordinates of jth point and f_{ji} denotes the fraction of jth point over ith centroid such that $f_{ji} \geq 0$ and $\sum_{i=1}^{k} f_{ji} = 1$ for all j.
Here also we are putting the same constraints as done in the above algorithm. Here a_i and b_i are a bit modified.

$$a_i = \sum_{point\ j \in \mathcal{A}} f_{ji}$$

$$b_i = \sum_{point\ j \in \mathcal{B}} f_{ji}$$

$$t_i = a_i + b_i$$

Then $\frac{a_i}{t_i} \leq p_1 + 0.1$ and $\frac{b_i}{t_i} \leq p_2 + 0.1$

Algorithm 3 Finding out fractions for given centroids

```
Require: centroids a 2D array, points, k
  m ← new Linear Programming Model
  fractions ← 2D array of size n × k with lower bound as 0 and upper bound as 100
  Add Constraint: to m that for all j = 1,2 ...k sum(fractions[i][j] for i in range(n)) = 100
  Add constraint that $\frac{a_i}{t_i} \leq p_1 + 0.1$ for each i
  Add constraint that $\frac{b_i}{t_i} \leq p_2 + 0.1$ for each i
  Add objective to minimize the weighted cost
  Run the linear program
  return fractions
```

Thus for given centroids, we have found out the fractions. Then we update the centroids to be the weighted means of the points with their fractions. We repeat the process for some number of iterations.

We have used linear programming to satisfy the above constraints and to minimize the total cost and the difference between the average costs of the two groups. The linear programming is modified in such a way so that each data point is given a weight (let's say 100) and then it is provided to all the clusters and it's fraction is also updated corresponding to it's weight. For more specific algorithmic details please have a look at MIPS library python [4] as well as our github repository [5].

E. Time Issue

Now, there has been an issue with the above approach and that is because of the fact that this algorithm (Algorithm 2) takes a lot of time to give us the result because we are using linear programming here and the number of variables in our model we are using is equal to total number of points which can go as high as 10^6, so to cut the time by a large margin we have implemented another similar approach which is basically assuming the fact that all of the n_1 (n_2 in case of Type2) data points in a small square lie at the center of the small square. Assuming this we are compromising with the result but are reducing the time complexity by a large margin. So, basically we are running our previous linear algorithm on a grid level instead of a point level. The total number of variables are also reduced by a large factor (now equal to the number of small squares).

F. Plots and Results

In this section we are basically plotting the similar curves as in Algorithm 1 and are trying to observe whether a fair resource allocation has been obtained or not. From the graphs below, we can see that the average cost of the two types of data point is almost equal and this means on an average both the types of population have to almost pay equal (travel equal distance), which sign towards nearly fair distribution.

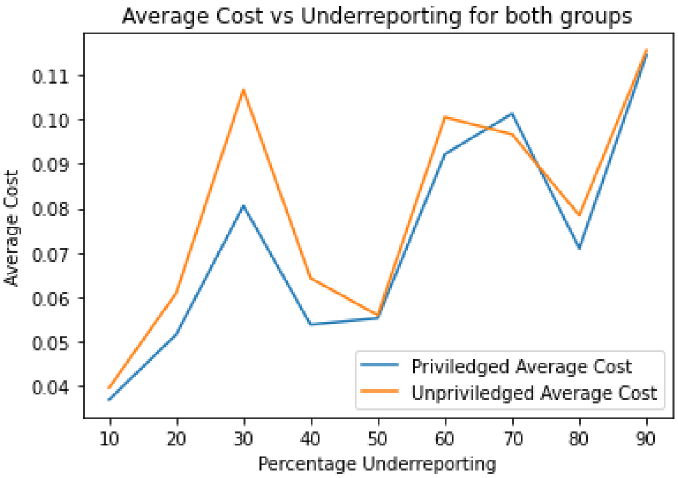

Fig. 9. Plot for nearly 10^3 points

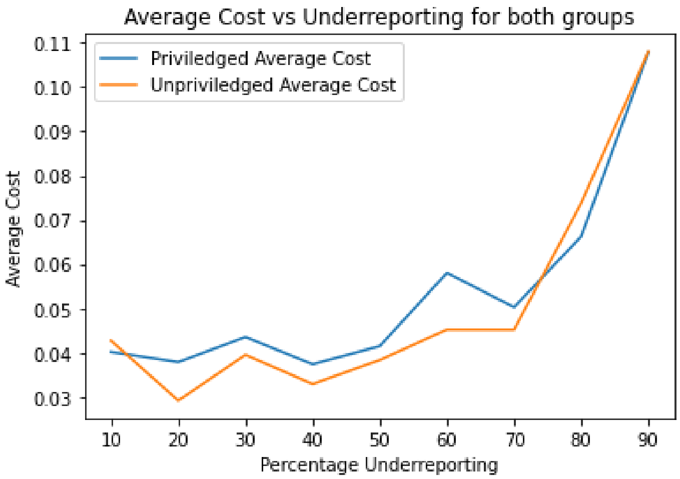

Fig. 10. Plot for nearly 2×10^3 points with more k

G. Worst possible distribution

Now, we have seen a very outlier kind of example to see whether we are actually getting a fair distribution in that case also or not. We are removing our randomness from the generation of the points and are generating the points (population) in such a way that there is only one box of Type1 population in the corner, all other boxes are of Type2. The below are the results we obtained in such a case for both the algorithms:

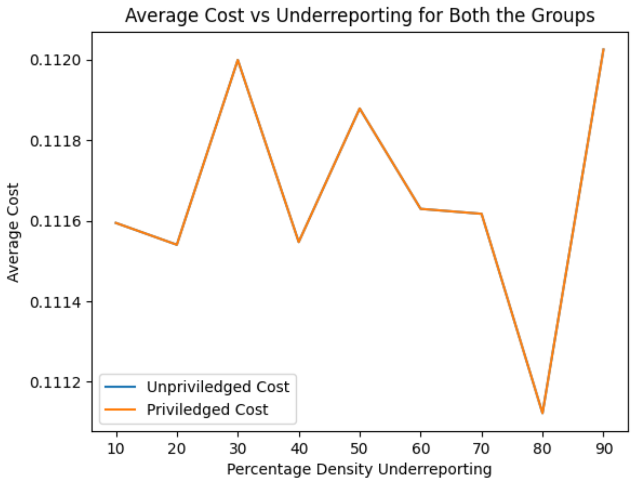

Fig. 11. Algorithm 1

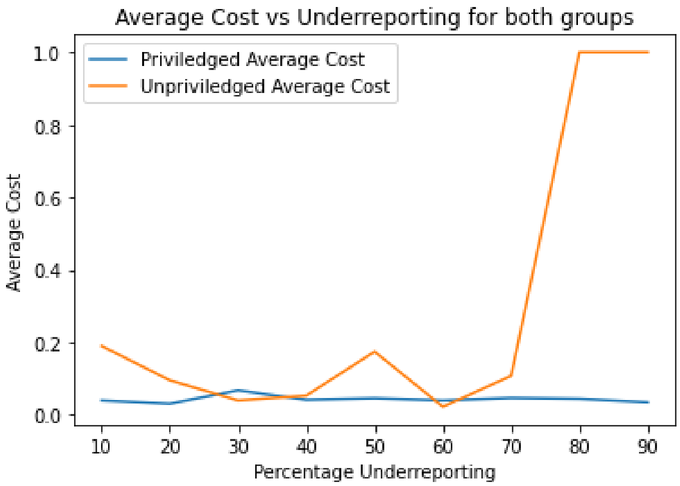

Fig. 12. Algorithm 2

H. Comparison

Now, we have to compare the two algorithms in both the cases and we can see that in case of Algorithm 1, the average cost of two type of populations is nearly equal while in Algorithm 2, although the average cost of both the populations tend to equal but is not as marginal as Algorithm 1. Now, we will try to plot the graph for Algorithm 1 on small values to compare it to the Algorithm2 result. The graph is shown below:

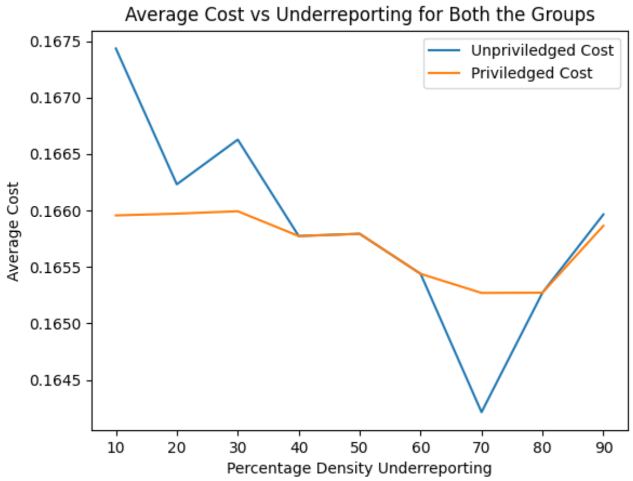

Fig. 13. Plot for nearly 10^3 points

Now, for the case of small values (nearly 10^3) we can see that algorithm 1 is better than algorithm 2 and we can see that thing by plotting the difference of the average cost of the privileged and unprivileged population. We can see that in almost all the values of under reporting the difference is less in Algorithm 1 and is thus a better algorithm for less number of points. This can be because of the fact that in case of Algorithm 2 we are taking the ratio to be the ratio of number privileged and unprivileged points in these two populations respectively and thus this can result of that.

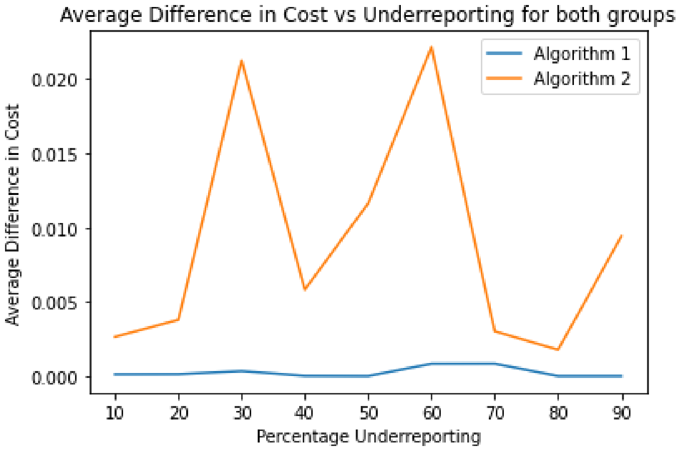

Fig. 14. Average difference for both populations 10^3 points

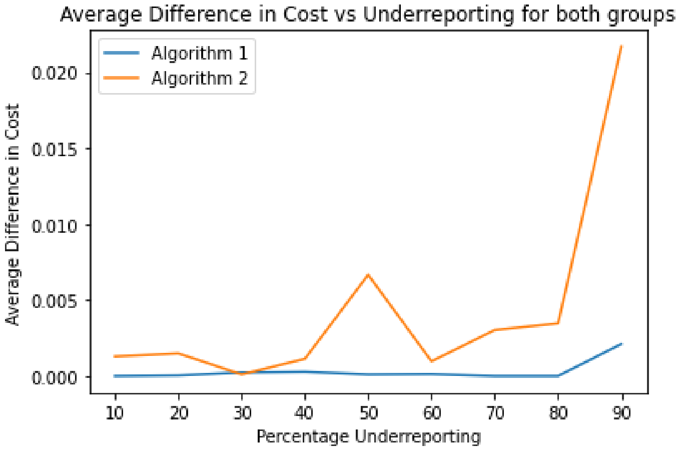

Fig. 15. Average difference for both populations 10^4 points

Thus, we can finally conclude that although both the algorithms try to be fair but algorithm 1 gives us better results compared to algorithm 2. This concludes the motive of our project and the analysis.

REFERENCES

[1] "Kmeans-clustering." https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning.

[2] "Fair-algorithm." https://dl.acm.org/doi/10.1145/3442188.3445906.

[3] "Fair-algorithm2." https://arxiv.org/abs/2006.10085: :text=Mehrdad

[4] "Mips." https://python-mip.readthedocs.io/en/latest/intro.html.

[5] "Repository." https://github.com/Deepanshu-Yadav-2001/K-Clustering.