- **Individual Loss** The individuals' loss is a function of the distance between their location and the closest centroid: $l_{i,c} = f(distance)$ where $l_{i,c} \in [0, 1]$:

$$l_{i,c} = \frac{1}{1 + e^{d(i,c)}} \tag{1}$$

- **Subgroup Loss** l_s is the loss experienced by members of a subgroup s with probability $p \in [0, 1]$ of the cumulative distribution function.

- **Subgroup Divergence** The divergence between subgroups is defined as the maximum difference in loss between two subgroups' cumulative distribution function at a given probability of experiencing some level of loss, p [**?**]:

$$\text{div}(l_s, l_{s'}) = \max|l_{s,p} - l_{s',p}| \tag{2}$$

  where $l_{s,p}$ is the loss experienced by the subgroup with the maximum loss at p, $l_{s,p} = \arg\max_s l_p$, and $l_{s',p}$ is the loss experienced by the subgroup with the minimum loss at p, $l_{s',p} = \arg\min_s l_p$.

- **Objective Function** Identify the set of centroid locations, C, that minimize individual loss while also minimizing the divergence among subgroups.

$$\text{argmin} \left(\sum_{c \in C} \sum_{i \in I} \frac{l_{i,c}}{|I|} + \text{div}(l_s, l_{s'}) \right) \tag{3}$$

Algorithm 1: Pseudocode Update Function for Fair K-Means

Symbols	Description
$i \in I$, (x_i, y_i)	spatially distributed individuals, locations (x, y)
$c \in C$, $\{x_c, y_c\}$	spatially distributed centroids, locations (x, y)
$s \in S$	subgroups based on binary attributes
L, $l_{i,c}$, l_s	loss, individual loss, subgroup loss
div$(l_s, l_{s'})$	divergence in loss between maximum and minimum subgroups
$d(i, c)$	Euclidean distance between individual, centroid
top_K	Top-k centroids that minimize $l_{i,c}$
v	counts for individuals clustered around each centroid
$grad$	gradient with respect to $d(i, c) \rightarrow (\frac{c_x - i_x}{d(i,c)}, \frac{c_y - i_y}{d(i,c)})$
t	maximum number of iterations
lr	learning rate
b	mini-batch size

```
while iteration < t do
    v ← 0
    M ← b examples picked randomly from I
    for x in M do
        v[c] = v[c] + 1 //update per-centroid counts
        lr = 1/v[c] //adaptive per-center learning rate
        tempCentroids[c] ← c - grad*lr //optimizes centroid coordinates using gradient
        descent
    end
    //Test if new centroids improve L
    tempLoss ← L using tempCentroids
    if L != tempLoss and j < 3 then
        if L > tempLoss then
            L ← tempLoss
            C ← tempCentroids
        else
            j = j+1;
        end
        //Reassign all points to new centroid
        for i in I do
            i[c] ← argmin(top_K) //minimizes objective function
        end
        iteration++
    else
        return C
    end
end
```