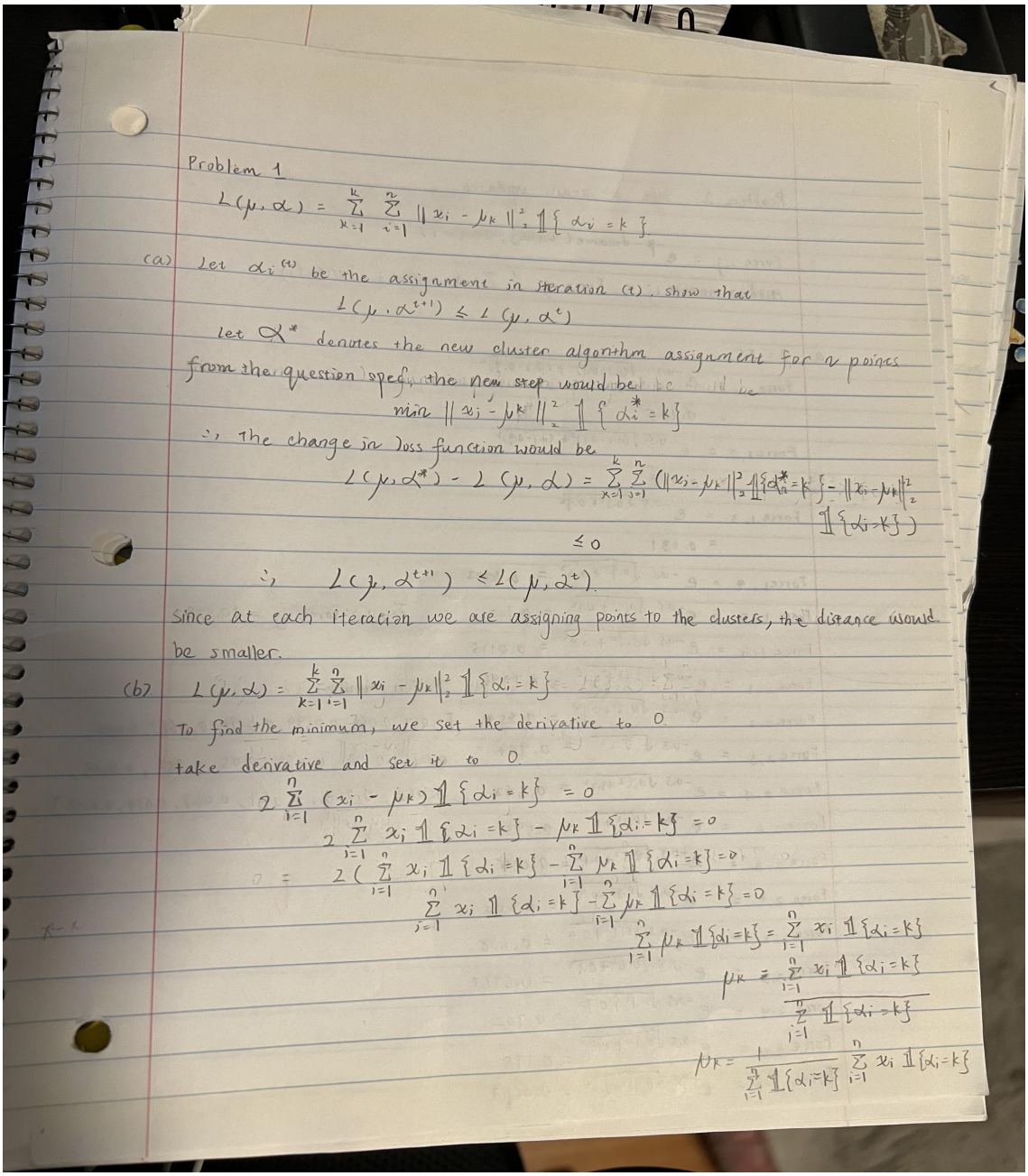
Problem 1. K-means convergence



Problem 1

$$L(\mu, \alpha) = \sum_{k=1}^{k} \sum_{i=1}^{n} \|x_i - \mu_k\|_2^2 \mathbb{1}\{\alpha_i = k\}$$

(a) Let $\alpha_i^{(t)}$ be the assignment in iteration (t). show that

$$L(\mu, \alpha^{t+1}) \leq L(\mu, \alpha^t)$$

Let α^* denotes the new cluster algorithm assignment for n points from the question spec, the new step would be

$$\min \|x_i - \mu_k\|_2^2 \mathbb{1}\{\alpha_i^* = k\}$$

∴ The change in loss function would be

$$L(\mu, \alpha^*) - L(\mu, \alpha) = \sum_{k=1}^{k} \sum_{j=1}^{n} (\|x_i - \mu_k\|_2^2 \mathbb{1}\{\alpha_i^* = k\} - \|x_i - \mu_k\|_2^2 \mathbb{1}\{\alpha_i = k\})$$

$$\leq 0$$

$$\therefore \quad L(\mu, \alpha^{t+1}) \leq L(\mu, \alpha^t)$$

since at each iteration we are assigning points to the clusters, the distance would be smaller.

(b) $$L(\mu, \alpha) = \sum_{k=1}^{k} \sum_{i=1}^{n} \|x_i - \mu_k\|_2^2 \mathbb{1}\{\alpha_i = k\}$$

To find the minimum, we set the derivative to 0

take derivative and set it to 0

$$2 \sum_{i=1}^{n} (x_i - \mu_k) \mathbb{1}\{\alpha_i = k\} = 0$$

$$2 \sum_{i=1}^{n} x_i \mathbb{1}\{\alpha_i = k\} - \mu_k \mathbb{1}\{\alpha_i = k\} = 0$$

$$= 2 \left(\sum_{i=1}^{n} x_i \mathbb{1}\{\alpha_i = k\} - \sum_{i=1}^{n} \mu_k \mathbb{1}\{\alpha_i = k\} \right) = 0$$

$$\sum_{j=1}^{n} x_i \mathbb{1}\{\alpha_i = k\} - \sum_{i=1}^{n} \mu_k \mathbb{1}\{\alpha_i = k\} = 0$$

$$\sum_{i=1}^{n} \mu_k \mathbb{1}\{\alpha_i = k\} = \sum_{i=1}^{n} x_i \mathbb{1}\{\alpha_i = k\}$$

$$\mu_k = \frac{\sum_{i=1}^{n} x_i \mathbb{1}\{\alpha_i = k\}}{\sum_{i=1}^{n} \mathbb{1}\{\alpha_i = k\}}$$

$$\mu_k = \frac{1}{\sum_{i=1}^{n} \mathbb{1}\{\alpha_i = k\}} \sum_{i=1}^{n} x_i \mathbb{1}\{\alpha_i = k\}$$

Problem 2. Soft k-means updates

(a). Using the partition function and β = 0.5 compute the E-step and show the Hidden Matrix.

```
data_j = [[0.1, 0.2],[0.2, 0.1],[0.3, 0],[1, 1.2],[0.8, 1],[9, 0.1]]
cluster_c = [[0.1, 0.9], [0.5, 0], [0.9, 0.5]]
beta = 0.5

def _partition_function(beta, data, center):
    return math.e**(-beta * _find_distance(data, center))

def _find_distance(point1, point2):
    return math.sqrt((point1[0]-point2[0])**2 + (point1[1] - point2[1])**2)


def _sum(data, center, beta):
    sum = 0
    for i in center:
        for j in data:
            sum += _partition_function(beta=beta, data=j, center=i)

    return sum

def _dot_product(a, b):
    dot_product=0
    for a,b in zip(a,b):
        dot_product += a*b

    return dot_product

def _E_step_a(data, center, beta):
    hidden_matrix = [[0]*6 for i in range(3)]
    sum = _sum(data, center, beta)

    for i in range(len(center)):
        for j in range(len(data)):
            hidden_matrix[i][j] = _partition_function(beta, data[j], center[i]) / sum


    return hidden_matrix


hidden_matrix = _E_step_a(data_j, cluster_c, beta)
hidden_matrix
```

```
[[0.0669909480006963,
  0.06352570444375792,
  0.05995416076179579,
  0.0591582381780423,
  0.06675332542917141,
  0.0010904780143607816],
 [0.07601652174534278,
  0.08116171312958592,
  0.08601807992090087,
  0.04962811489965731,
  0.05640411858252954,
  0.0013556260787073446],
 [0.06201382341688556,
  0.06352570444375794,
  0.06433121820160176,
  0.06675332542917141,
  0.07367080075466724,
  0.0016409856936781]]
```

As we can see from the hidden matrix, the first point [0.1, 0.2] would belong to cluster 2, the second point [0.2, 0.1] would belong to cluster 2, the third point [0.3, 0] would belong to cluster 2, the fourth point [1, 1.2] belongs to cluster 3, the fifth point [0.8, 1] belongs to cluster 3, the six point [9, 0.1] belongs to cluster 3.

(b). Using the assignments you made in (a), compute the M-step.

```
[ ] def _M_step_a(hidden_matrix, data):
        x_i1 = []
        x_i2 = []
        uni_vec = [1] * 6
        center = [[0] * 2 for i in range(3)]

        for i in data:
            x_i1.append(i[0])
            x_i2.append(i[1])

        for i in range(len(center)):
            center[i][0] = _dot_product(hidden_matrix[i], x_i1) / _dot_product(hidden_matrix[i], uni_vec)
            center[i][1] = _dot_product(hidden_matrix[i], x_i2) / _dot_product(hidden_matrix[i], uni_vec)

        return center

    _M_step_a(hidden_matrix=hidden_matrix, data=data_j)
```

```
[[0.5032420319987116, 0.49642990476853954],
 [0.4466586763024705, 0.3976588361063949],
 [0.5384320728620066, 0.5202548372394207]]
```

(c). Using the Newtonian inverse-square law of gravitation, compute the E-step.

```
def _partition_function_c(data, center):
    return 1/_find_distance(data, center)**2

def _sum_c(data, center):
    sum = 0
    for i in center:
        for j in data:
            sum += _partition_function_c(data=j, center=i)

    return sum

def _E_step_c(data, center):
    hidden_matrix = [[0]*6 for i in range(3)]
    sum = _sum(data, center, beta)

    for i in range(len(center)):
        for j in range(len(data)):
            hidden_matrix[i][j] = _partition_function_c(data[j], center[i]) / sum


    return hidden_matrix



hiddenMatrix = _E_step_c(data_j, cluster_c)
hiddenMatrix
```

```
[[0.19400955174956425,
  0.1462533543958253,
  0.11184080042033699,
  0.10562742261920717,
  0.19012936071457287,
  0.0011905407684068 43],
 [0.4753234017864322,
  0.9506468035728646,
  2.376617008932161,
  0.0562512901522405,
  0.08721530308007931,
  0.0013155920337294],
 [0.13022558953052937,
  0.14625335439582535,
  0.15584373829063353,
  0.19012936071457295,
  0.36563338598956324,
  0.0014454109830817 46]]
```

the first point [0.1, 0.2] would belong to cluster 2, the second point [0.2, 0.1] would belong to cluster 2, the third point [0.3, 0] would belong to cluster 2, the fourth point [1, 1.2] belongs to cluster 3, the fifth point [0.8, 1] belongs to cluster

3, the six point [9, 0.1] belongs to cluster 3.

▾ (d). Using the assignments you made in (c), compute the M-step.

```
def _M_step_a(hidden_matrix, data):
    x_i1 = []
    x_i2 = []
    uni_vec = [1] * 6
    center = [[0] * 2 for i in range(3)]

    for i in data:
        x_i1.append(i[0])
        x_i2.append(i[1])

    for i in range(len(center)):
        center[i][0] = _dot_product(hidden_matrix[i], x_i1) / _dot_product(hidden_matrix[i], uni_vec)
        center[i][1] = _dot_product(hidden_matrix[i], x_i2) / _dot_product(hidden_matrix[i], uni_vec)

    return center

_M_step_a(hidden_matrix=hiddenMatrix, data=data_j)
```

```
[[0.46812517429431016, 0.4945304826476962],
 [0.2757559666482577, 0.08739434703511283],
 [0.5908568951018927, 0.6413176706574215]]
```

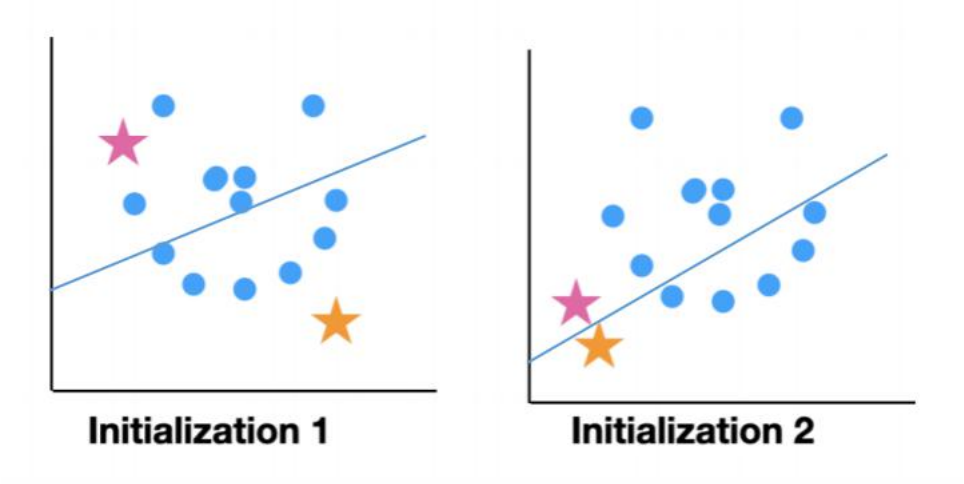
▾ (e) Any observations comparing the two different distance functions?

As I can observe from the updated centers for two different distance functions, the first center and the third center points stayed more or less the same for both method. The big difference is that for the second cluster center, it was (0.44, 0.39) with one method and now becomes (0.27, 0.09) with the other one.

To add to that, the two different distance also assigns the data points to the same clusters.

Problem 3. Decision boundaries in standard k-means

(a.) Given the current initialization, how are the points assigned using Initialization 1 and Initialization 2? In other words, how does the decision boundary look for each of these initialization in the current assignment step.



The first initialization has the decision boundary more tilted(the way we defined decision boundary would be any points lies on the decision boundary line would be equidistant from themselves to the centers in all the clusters.) The second initialization has its decision boundary line as a less tilted line.

(b). Is there a qualitative difference between the assignments in the current step given the different initialization?

The different initialization will result in different assignments of data points forming different clusters. For example, in this visualizations example, the data belonging to second cluster in initialization 1 could be belonging to the first cluster in initialization 2.

(c). If you were to run the Lloyd algorithm with both initialization, how would it behave in the long term? Do they converge to the same answer? Does one take longer to converge than the other?

So the Lloyd algorithm basically randomly selects k numbers of centers, in our case, k=2. Then it assigns each data points closer to a certain centroid and forms a cluster based on that centroid. After this step, each centroid is updated by taking the means of all the data points belonging to the cluster. And just repeat till convergence. Since our centroid is already defined as stars above, in the long run, the centroids would be sitting at the centers of each cluster(since we are repeatedly reassigning the position of each centroid based on the mean). However, just like I mentioned in part b, the algorithm does not necessarily converge to the same answer with different initializated centers. For visualization 1, the clusters could be formed differently as compared to visualization 2. Given the similar data points in each visualization, the convergence time would be similar to each other.

(d). Do you see a picture in the data?

A smiley face.

(e). Draw some clustered data in two dimensions that is trivially easy to cluster by eye, but impossible to cluster correctly using k-means.

Since k-means always has linear decision boundary, such as a line in 2d and a hyperplane in 3d, there is no way for k-means to separate the clusters below, as the decision boundary will not be linear.