4. Clustering

4.1. Clustering

4.2. A clustering objective

In Python, we can store the list of vectors in a `numpy` list of N vectors. If we call this list `data`, we can access the ith entry (which is a vector) using `data[0]`. To specify the clusters or group membership, we can use a list of assignments called `grouping`, where `grouping[i]` is the number of group that vector `data[i]` is assigned to. (This is an integer between 1 and k.) (In VMLS, chapter 4, we describe the assignments using a vector c or the subsets G_j.) We can store k cluster representatives as a Python list called `centroids`, with `centroids[j]` the jth cluster representative. (In VMLS we describe the representatives as the vectors $z_1, \ldots, z_k$.)

Group assignment. We define a function to perform group assignment. With given initial value of centorids, we compute the distance between each centroid with each vector and assign the grouping according to the smallest distance. The function then returns a vector of groupings.

In [ ]:
```
def group_assignment(data,centroids):
    grouping_vec_c = np.zeros(len(data))
    for i in range(len(data)):
        dist = np.zeros(len(centroids))
        for j in range(len(centroids)):
            dist[j] = np.linalg.norm(data[i] - centroids[j])
        min_dist = min(dist)
        for j in range(len(centroids)):
            if min_dist == dist[j]:
                grouping_vec_c[i] = j+1
    return grouping_vec_c
```

Update centroid. We define a function to update the centroid after the group assignment, returning a new list of group centroids.

In [ ]:
```
def update_centroid(data, grouping, centroids):
    new_centroids = [];
    for i in range(len(centroids)):
        cent = np.zeros(len(data[0]))
        count = 0
        for j in range(len(data)):
            if grouping[j] == (i+1):
                cent = cent+data[j]
                count += 1
        group_average = cent/count
        new_centroids.append(group_average)
    return new_centroids
```

Clustering objective. Given the group assignment and the centroids with the data, we can compute the clustering objective as the square of the RMS value of the vector of distances.

In [ ]:
```
def clustering_objective(data, grouping, centroids):
    J_obj = 0
    for i in range(len(data)):
        for j in range(len(centroids)):
            if grouping[i] == (j+1):
                J_obj += np.linalg.norm(data[i] - centroids[j])**2
    J_obj = J_obj/len(data)
    return J_obj
```

4.3. The k-means algorithm

We can define another function `Kmeans_alg` that uses the three functions defined in the above subsection iteratively.

In [ ]:
```
def Kmeans_alg(data, centroids):
    iteration = 0
    J_obj_vector = []
    Stop = False
    while Stop == False:
        grouping = group_assignment(data, centroids)
```