Worksheet 05

Name: Rishab Sudhir

UID: U64819615

Topics

- Cost Functions
- Kmeans

Cost Function

Solving Data Science problems often starts by defining a metric with which to evaluate solutions were you able to find some. This metric is called a cost function. Data Science then backtracks and tries to find a process / algorithm to find solutions that can optimize for that cost function.

For example suppose you are asked to cluster three points A, B, C into two non-empty clusters. If someone gave you the solution `{A, B}, {C}` , how would you evaluate that this is a good solution?

Notice that because the clusters need to be non-empty and all points must be assigned to a cluster, it must be that two of the three points will be together in one cluster and the third will be alone in the other cluster.

In the above solution, if A and B are closer than A and C, and B and C, then this is a good solution. The smaller the distance between the two points in the same cluster (here A and B), the better the solution. So we can define our cost function to be that distance (between A and B here)!

The algorithm / process would involve clustering together the two closest points and put the third in its own cluster. This process optimizes for that cost function because no other pair of points could have a lower distance (although it could equal it).

K means

a) (1-dimensional clustering) Walk through Lloyd's algorithm step by step on the following dataset:

`[0, .5, 1.5, 2, 6, 6.5, 7]` (note: each of these are 1-dimensional data points)

Given the initial centroids:

`[0, 2]`

iter 1

cluster centroid 1 = [0],

Cluster 1 = [0,0.5]

cluster centroid 2 = [2],

Cluster 2 = [1.5,2,6.5,7]

iter 2

cluster centroid 1 = [0.25], mean of ([0,0.5])

Cluster 1 = [0,0.5,1.5,2]

cluster centroid 2 = [4.25], mean of ([1.5,2,6.5,7])

Cluster 2 = [6.5,7]

iter 3

cluster centroid 1 = [0.5], mean of ([0,0.5,1.5,2])

Cluster 1 = [0,0.5,1.5,2]

cluster centroid 2 = [6.75], mean of ([6.5,7])

Cluster 2 = [6.5,7]

iter 3

cluster centroid 1 = [0.5], mean of ([[0,0.5,1.5,2])

Cluster 1 = [0,0.5,1.5,2]

cluster centroid 2 = [6.75], mean of ([6.5,7])

Cluster 2 = [6.5,7]

CONVERGED

b) Describe in plain english what the cost function for k means is.

A cost function is a measure of how well a machine learning model performs by quantifying the difference between predicted and actual outputs.

For k means this is the sum of the euclidean distances (from a point x in a cluster to its cluster centroid(mean)) squared for each cluster (summed over all clusters).

c) For the same number of clusters K, why could there be very different solutions to the K means algorithm on a given dataset?

It depends on where the cluster centroids are placed in the beginning (usually randomly). Based on where they are originally placed the points near to each centroid will vary based on each placement.

d) Does Lloyd's Algorithm always converge? Why / why not?

Lloyd's algo will always converge to a local minimum as it is a greedy algorithm so it always reduces the objective function but may get stuck on a local minima based on the shape of the objective surface.

e) Follow along in class the implementation of Kmeans

In [25]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

#generating a dataset to use
centers = [[0, 0], [2, 2], [-3, 2], [2, -4]]
X, _ = datasets.make_blobs(n_samples=300, centers=centers, cluster_std=1

class KMeans():

    def __init__(self, data, k):
        self.data = data
        self.k = k
        self.assignment = [-1 for _ in range(len(data))] # -1 means unas
        self.snaps = []

    def snap(self, centers):
        TEMPFILE = "temp.png"

        fig, ax = plt.subplots()
        ax.scatter(X[:, 0], X[:, 1], c=self.assignment)
        ax.scatter(centers[:,0], centers[:, 1], c='r')
        fig.savefig(TEMPFILE)
        plt.close()
        self.snaps.append(im.fromarray(np.asarray(im.open(TEMPFILE))))


    def initialize(self):
        return self.data[np.random.choice(range(len(self.data)), size =

    def is_unassigned(self, i):
        return self.assignment[i] == -1

    def unassign_all(self):
        self.assignment = [-1 for _ in range(len(self.data))]

    def assign(self, centers):
        for i in range(len(self.data)):
            self.assignment[i] = 0
            temp_dist = self.dist(self.data[i], centers[0])
            for j in range(1, len(centers)):
                new_dist = self.dist(self.data[i], centers[j])
                if new_dist < temp_dist:
                    self.assignment[i] = j
                    temp_dist = new_dist

    def dist(self, x, y):
        return sum((x-y)**2) ** 0.5

    def are_centers_diff(self, c1, c2):
        for i in range(len(c1)):
            if c1[i] not in c2:
                return True
        return False

    def calculate_new_centers(self):
```

```
        centers = []
        for j in range(self.k):
            cluster_j = self.data[
                np.array([i for i in range(len(self.data)) if self.assig
            ]
            centers.append(np.mean(cluster_j, axis=0))
        return np.array(centers)


    def lloyds(self):
        centers = self.initialize()
        #self.snaps.append(self.snap(centers))
        self.assign(centers)
        self.snap(centers)
        new_centers = self.calculate_new_centers()
        while self.are_centers_diff(centers, new_centers):
            centers = new_centers
            self.unassign_all()
            self.assign(centers)
            self.snap(centers)
            new_centers = self.calculate_new_centers()

        return



kmeans = KMeans(X, 6)
kmeans.lloyds()
images = kmeans.snaps

images[0].save(
    'kmeans.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```

In [ ]:

In [ ]: