Worksheet 05

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

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]`

Assign each point in the dataset to its closest center

(0, 0.5) for 0

(1.5, 2, 6, 6.5, 7) for 2

Compute the new centers as the means of each cluster

(0, 0.5) --> 0.25

(1.5, 2, 6, 6.5, 7) --> 4.6

Assign each point in the dataset to its closest center

(0, 0.5, 1.5, 2) for 0.25

(6, 6.5, 7) for 4.6

Compute the new centers as the means of each cluster

(0, 0.5, 1.5, 2) --> 1

(6, 6.5, 7) --> 6.5

Assign each point in the dataset to its closest center (0, 0.5, 1.5, 2) for 1

(6, 6.5, 7) for 6.5

Compute the new centers as the means of each cluster

(0, 0.5, 1.5, 2) --> 1

(6, 6.5, 7) --> 6.5

The data points are assigned to the same cluster after reiteration and their distance calculated from the center is the same, so the algorithm has converged. The final clusters are (0, 0.5, 1.5, 2) and (6, 6.5, 7).

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

The cost function is the variance within every clusters. It measures how spread the data points are in each cluster from its centroid.

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

There could be very different solutions to the K means algorithm on a given dataset due to the first step in the Lloyd's Algorithm, where we pick k centers randomly. This is because the data points are assigned to a cluster based on the initial random centroid and if the initial random center points are not well chosen, it might converge to a different state (local minimum instead of global minimum).

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

Lloyd's algorithm always converge because there are a finite number of data points and the cost function always decreases or remains the same. The cost function measures how spread the data points are within a cluster from the centroid and since the centroid gets updated for every iteration based on the current cluster's center, the cost function always decreases. With k also previously chosen, the algorithm should eventually reach to a state where reassigning the data points and the calculation of centroids remains the same, which means that Lloyd's algorithm always converge. However, as stated in the previous question part c, the initial random centroids chosen might affect the data points to be clustered in a way that it converges to a local minimum instead of the most optimal solution.

e) Follow along in class the implementation of Kmeans

```python

import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

centers = [[0, 0], [2, 2], [-3, 2], [2, -4]]
X, _ = datasets.make_blobs(n_samples=300, centers=centers, cluster_std=1, random_state=0)

class KMeans():

    def __init__(self, data, k):
        self.data = data
        self.k = k
        self.assignment = [-1 for _ in range(len(data))]
        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 is_unassigned(self, i):
        return self.assignment[i] == -1

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

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

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

    def assign(self, centers):
        for i in range(len(self.data)):
            self.assignment[i] = 0
            temp_assign = 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 temp_dist > new_dist:
                    self.assignment[i] = j
                    temp_dist = new_dist

    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.assignment[i] == j])
            ]
            centers.append(np.mean(cluster_j,axis=0))

        return np.array(centers)

    def dist(self, x, y):
        return sum((x - y) ** 2) ** (1/2)

    def lloyds(self):
        centers = self.initialize()
        self.assign(centers)
        self.snap(centers)
        new_centers = self.calculate_new_centers()
        while self.are_centers_diff(centers, new_centers):
            centers = new_centers
            self.snap(centers)
            self.unassign_all()
            self.assign(centers)
            new_centers = self.calculate_new_centers()
            print (new_centers)
        return

kmeans = KMeans(X, 4)
kmeans.lloyds()
images = kmeans.snaps

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

```
[[ 1.80375063 -4.10800509]
 [ 1.50017539  2.0040031 ]
 [-2.44237939  1.36463401]
 [ 1.56006775 -0.07495634]]
[[ 1.83329012 -4.16630999]
 [ 1.69300984  2.17984976]
 [-2.72190534  1.61469981]
 [ 0.81135838 -0.19543115]]
[[ 1.84012274 -4.13675116]
 [ 1.80042891  2.12251522]
 [-2.87651906  1.7164892 ]
 [ 0.44400863 -0.15210282]]
[[ 1.84012274 -4.13675116]
 [ 1.89583061  2.06047507]
 [-2.92468433  1.77583288]
 [ 0.2239685  -0.10571887]]
[[ 1.84012274 -4.13675116]
 [ 1.91735042  2.0452011 ]
 [-3.00866911  1.82497252]
 [ 0.11666371 -0.03777805]]
[[ 1.84012274 -4.13675116]
 [ 1.91535954  2.02383689]
 [-3.04862622  1.85340542]
 [ 0.04932005 -0.02271045]]
[[ 1.84012274e+00 -4.13675116e+00]
 [ 1.91535954e+00  2.02383689e+00]
 [-3.07703402e+00  1.85266958e+00]
 [ 3.66735856e-02  4.13365422e-03]]
[[ 1.84012274e+00 -4.13675116e+00]
 [ 1.91535954e+00  2.02383689e+00]
 [-3.07703402e+00  1.85266958e+00]
 [ 3.66735856e-02  4.13365422e-03]]
```