Worksheet 05

Name: Lili Zhao UID: U18256657

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]`

1. The points assigned to the in intial centroids: [0, .5], [1.5, 2, 6, 6.5, 7].
2. Compute new centroids as means of each cluster: [0.25, 4.6] Repeat:
3. [0, 0.5, 1.5, 2], [6, 6.5, 7];
4. [1, 6.5]

We would try to reassign the centroids and realize they do not change; i.e. we converged.

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

- defined as distance from the mean (a spread/ variance)
- the sum of the square of (Euclidian) distances from each data point to the mean of its assigned cluster
- the cost function for kmeans essentially measures how well the data points are grouped together. It aims to minimize the distance or the "spread" of data around the cluster centroids, meaning that closer-together clusters will have a lower cost.

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

Yes, since the algorithm begins with the random initialization of centroids. Different initializations influence the final clustering results leading the kmeans algorithm to find different local optimum.

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

Yes, Lloyd's algorithm always converges. Eventually, all data points will remain at the same closest centroid and upon attempting to reassign them, they won't change (reaching convergence!). However, the convergence is not guaranteed to always be to the global optimum; it might converge to a local optimum. This depends on the random initialization, the presence of outliers or anomalies, a bad value of k given the data points, among other readons.

e) Follow along in class the implementation of Kmeans

In [ ]:

```
# import numpy as np
# from PIL import Image as im
# import matplotlib.pyplot as plt
# %pip install scikit-learn
# 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, r

# 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 == -1

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

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

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

#     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(len(centers)):
#                 #which center are you closest too
#                 new_dist = self.dist(self.data[i], centers[j])
#                 if new_dist < temp_dist:
#                     self.assignment[i] = j #assign to cluster as better opt
#                     temp_dist = new_dist
#     def calc_new_centers(self):
#         centers = []
#         #take assignments and look at indicies where they are that specific
#         for j in range(self.k):
#             cluster_j = self.data[np.array([i for i in range(len(self.data)
#             centers.append(np.mean(cluster_j, axis=0))  #axis=0 computes me
#         return np.array(centers)

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

#     def lloyds(self):
#         #init: pick centers at random
#         centers = self.initialize()
#         self.snap(centers)
#         #assign points to centers created
#         self.assign(centers)
#         self.snap(centers)
#         #calculate new centers
#         new_centers = self.calc_new_centers()

#         #repeat  until convergence/centers are the same
#         while (self.are_centers_diff(centers, new_centers)):
#             #assign points to centers from new_centers
#             center = new_centers
#             self.snap(centers)
#             #unassign
#             self.unassign_all()
#             #reassign
#             self.assign(centers)
#             new_centers = self.calc_new_centers()
#             print(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
# )

%pip install scikit-learn
```

```
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, ran

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,

    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
            ]
            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
)
```

```
Requirement already satisfied: scikit-learn in /Library/Frameworks/Python.fram
ework/Versions/3.10/lib/python3.10/site-packages (1.4.0)
Requirement already satisfied: joblib>=1.2.0 in /Library/Frameworks/Python.fra
mework/Versions/3.10/lib/python3.10/site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: numpy<2.0,>=1.19.5 in /Library/Frameworks/Pytho
n.framework/Versions/3.10/lib/python3.10/site-packages (from scikit-learn) (1.
26.3)
Requirement already satisfied: scipy>=1.6.0 in /Library/Frameworks/Python.fram
ework/Versions/3.10/lib/python3.10/site-packages (from scikit-learn) (1.12.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Library/Frameworks/Pyt
hon.framework/Versions/3.10/lib/python3.10/site-packages (from scikit-learn) (
3.2.0)
WARNING: You are using pip version 21.2.4; however, version 24.0 is available.
You should consider upgrading via the '/usr/local/bin/python3 -m pip install -
-upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
[[ 1.51982807  1.40986601]
 [-3.66002117  2.13691125]
 [-1.65587866  0.83058812]
 [ 1.82416454 -4.00058334]]
[[ 1.58872473  1.47002243]
 [-3.56443781  2.00117668]
 [-1.34819638  0.71708251]
 [ 1.82416454 -4.00058334]]
[[ 1.67966234  1.59483769]
 [-3.35776002  1.94982227]
 [-0.96244538  0.41161285]
 [ 1.82416454 -4.00058334]]
[[ 1.77617021  1.71681455]
 [-3.18218195  1.90395221]
 [-0.53741418  0.14125309]
 [ 1.84180743 -4.03258775]]
[[ 1.88587494  1.88900153]
 [-3.12164037  1.85868514]
 [-0.22866335  0.10753508]
 [ 1.84180743 -4.03258775]]
[[ 1.91058901  1.9500435 ]
 [-3.07703402  1.85266958]
 [-0.07343665  0.02329337]
 [ 1.84585117 -4.10098602]]
[[ 1.91248297  2.00204407]
 [-3.07703402  1.85266958]
 [-0.01886062  0.02039989]
 [ 1.84585117 -4.10098602]]
[[ 1.91248297  2.00204407]
 [-3.07703402  1.85266958]
 [-0.01886062  0.02039989]
 [ 1.84585117 -4.10098602]]
```