CS 506 Spring 2020 - HW1

Clustering and Visualization

Due: October 19, 2020

1 Understanding K-means Clustering

In this exercise, you will implement the K-means clustering algorithm. You will start on an example 2D dataset that will help you gain an intuition of how the K-means algorithm works. You will be using *k_means_clustering.py* for this part of the exercise.

1.1 Implementing K-means

The K-means algorithm is a method to automatically cluster similar data examples together. Concretely, you are given a training set $\left\{x^1, \cdots, x^m\right\}$ where $x^i \in \mathbb{R}^n$, and want to group the data into a few cohesive clusters. The intuition behind K -means is an iterative procedure that starts by guessing the initial centroids, and then refines this guess by repeatedly assigning examples to their closest centroids and then recomputing the centroids based on the assignments.

The inner loop of the algorithm repeatedly carries out two steps:

- Assigning each training example x to its closest centroid
- Recomputing the mean of each centroid using the points assigned to it.

The K-means algorithm will always converge to some final set of means for the centroids. Note that the converged solution may not always be ideal and depends on the initial setting of the centroids. Therefore, in practice the K-means algorithm is usually run a few times with different random initializations. One way to choose between these different solutions from different random initializations is to choose the one with the lowest cost function value (distortion). You will implement the two phases of the K-means algorithm separately in the next sections.

1.1.1 Finding Closest Centroids

In the cluster assignment phase of the K-means algorithm, the algorithm assigns every training example x^i to its closest centroid, given the current positions of centroids. Specifically, for every example i we set

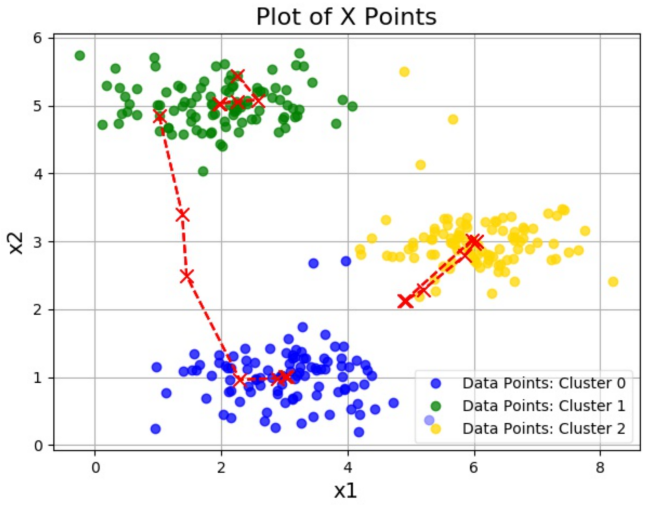
$$c^i := \arg\min_j \left\|x^i - \mu_j\right\|^2$$

where $c(i)$ is the index of the centroid that is closest to x^i, and j is the position (index) of the j-th centroid.

Your task is to complete the code in function *find_closest_centroids*. This function takes the data matrix samples and the locations of all centroids inside centroids and should output a one-dimensional array of clusters that holds the index (a value in $\{1, \cdots, K\}$, where K is total number of centroids) of the closest centroid to every training example. You can implement this using a loop over every training example and every centroid. Once you have completed the code in *find_closest_centroids*, you can run it and you should see the output $[0, 2, 1,]$ corresponding to the centroid assignments for the first 3 examples.

Please take a look at Figure 1 to gain an understanding of the distribution of the data. It is two dimentional, with x_1 and x_2.

Figure 1: Result 1



1.1.2 Computing Centroid Means

Given assignments of every point to a centroid, the second phase of the algorithm recomputes, for each centroid, the mean of the points that were assigned to it. Specifically, for every centroid k we set

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^i$$

where C_k is the set of examples that are assigned to centroid k. Concretely, if two examples say x^3 and x^5 are assigned to centroid $k = 2$, then you should update

$$\mu_2 = \frac{1}{2}(x(3) + x(5))$$

Once you have completed the code in *get_centroids*, the *k_means_clustering.py* will run your code and output the centroids after the first step of K-means. The final centroids should be [[1.95399466 5.02557006] [3.04367119 1.01541041] [6.03366736 3.00052511]].

When you run the next step, the K-means code will produce a visualization that steps you through the progress of the algorithm at each iteration. Close figure multiple times to see how each step of the K-means algorithm changes the centroids and cluster assignments. At the end, your figure should look as the one displayed in Figure1

1.2 Random Initialization

The initial assignments of centroids for the example dataset in previous section were designed so that you will see the same figure as Figure1. In practice, a good strategy for initializing the centroids is to select random examples from the training set. In this part of the exercise, you should complete the function *choose_random_centroids*. You should randomly permute the indices of the examples (using random seed 7). Then, it selects the first K examples based on the random permutation of the indices. This should allow the examples to be selected at random without the risk of selecting the same example twice. You will see how random initialization will affect the first few iterations of clustering, and also possibly, result in a different cluster assignment.

2 Working with the Algorithms

- In this assignment we will be working with the AirBnB dataset, that you can also find here. Our goal is to visualize areas of the NYC with respect to the price of the AirBnb listings in those areas.
  From the detailed *nyc_listings.csv* file, you will use **longitude** and **latitude** to cluster closeness and **price** to cluster for expensiveness.
  Note that spatial coordinates and price are in different units, so **you may need to consider scaling** in order to avoid arbitrary skewed results.

a) [8 pts.] **Find clusters using the 3 different techniques we discussed in class: k-means++, hierarchical, and GMM**. Explain your data representation and how you determined certain parameters (for example, the number of clusters in k-means++).

b) [3 pts.] List a few bullet points describing the pros and cons of the various clustering algorithms.

A few hints:
-Some listings contain missing values. Better strategy for this assignment is to completely ignore those listings.

-Pay attention to the data type of every column when you read a .csv file and convert them to the appropriate types (e.g. float or integer).

3 Data visualization

a) [1pt.] Start by producing a Heatmap using the Folium package (you can install it using pip). You can use the code below to help you (assumes the use of Pandas Dataframes):

```
def generateBaseMap(default_location=[40.693943,
    -73.985880]):
    base_map = folium.Map(location=default_location)
    return base_map

base_map = generateBaseMap()
HeatMap(data=df[['latitude', 'longitude', 'price']].
    groupby(['latitude', 'longitude']).mean().
    reset_index().values.tolist(), radius=8, max_zoom
    =13).add_to(base_map)
base_map.save('index.html')
```

Is this heatmap useful in order to draw conclusions about the expressiveness of areas within NYC? If not, why?

b) [2pts.] Visualize the clusters by plotting the longitude/latitude of every listing in a scatter plot.

c) [2pts.] For every cluster report the average price of the listings within this cluster.

d) Bonus points [1pt.] if you provide a plot on an actual NYC map! You may use Folium or any other package for this.

e) [1pt.] Are the findings in agreement with what you have in mind about the cost of living for neighborhoods in NYC? If you are unfamiliar with NYC, you can consult the web.

4 Image Manipulation

a) [8 pts.] Download the image found by clicking **here.** For this assignment, you will use the k-means algorithm in the CS506 python package that you built in class to manipulate this image. The goal is to give this image as input, and return the image with like pixels combined into 10 clusters.

A few hints:

-There are a number of useful packages for working with images; we recommend using cv2 (obtained by running pip install opencv). Using this

package, you can use the line **img = cv2.imread("file.jpg")** in order to load the image as a numpy array (note that this means you will also need to import numpy).

-If you follow the hint above, your data is no longer being opened from a file inside your k_mean() function so you may need to tweak it a bit.

-To display the image after you have run k-means, you can use the lines

cv2.imshow('Display Window', manipulated_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

-Each pixel is represented by three features: their red, green, and blue values. You only need to tweak your algorithm to find clusters and then replace the pixels with the value of their cluster center.

-The more clusters you work with, the slower this algorithm will run, so for testing it may be useful to work with only 2 clusters.

Here is the starting image:

And here is what your code should return: