CS3943 Mining Massive Data Sets

HW1: LSH and k-means Clustering

Release Date: 03/05/2021; *Due Date*: Midnight, 03/22/2021

Academic Honesty

Aside from the narrow exception for
collaboration on homework, all work submitted in this course must be your own.
Cheating and plagiarism will not be tolerated. If you have any questions about a specific case, please ask me. We will be checking for this!
NYU Tandon's Poly's Policy on Academic Misconduct:
http://engineering.nyu.edu/academics/code-of-conduct/academic-misconduct

1 *k*-means Clustering (50 points)

Note: This problem should be implemented in a colab notebook. You should **not** use the Spark MLlib clustering library for this problem, or the sklearn library – we want you to learn how to implement the algorithms and compute the distances!

This problem will help you understand the nitty gritty details of implementing clustering algorithms. Let's say we have a set X of *n* data points in the *d*-dimensional space R_d. Given the number of clusters *k* and the set of *k* centroids C, we now proceed to define various distance metrics and the corresponding cost functions that they minimize.

Euclidean distance Given two points *A* and *B* in *d* dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Euclidean distance between *A* and *B* is defined as:

$$||a - b|| = \sqrt{\sum_{i=1}^{d}(a_i - b_i)^2} \qquad (1)$$

The corresponding cost function *φ* that is minimized when we assign points to clusters using the Euclidean distance metric is given by

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2 \qquad (2)$$

Note, that in the cost function the distance value is squared. This is intentional, as it is the squared Euclidean distance the algorithm is guaranteed to minimize.

Manhattan distance Given two random points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Manhattan distance between A and B is defined as:

$$|a - b| = \sum_{i=1}^{d} |a_i - b_i| \tag{3}$$

The corresponding cost function ψ that is minimized when we assign points to clusters using the Manhattan distance metric is given by:

$$\psi = \sum_{x \in \mathcal{X}} \min_{c \in C} |x - c| \tag{4}$$

Iterative *k*-Means Algorithm: We learned the basic *k*-Means algorithm in class which is as follows: *k* centroids are initialized, each point is assigned to the nearest centroid and the centroids are recomputed based on the assignments of points to clusters. In practice, the above steps are run for several iterations. We present the resulting iterative version of *k*-Means in Algorithm 1.

Iterative *k*-Means clustering: Implement iterative *k*-means. Please use the dataset from **data/q1** within the bundle for this problem.

The folder has 2 files:

1. **data.txt** contains the dataset which has 4601 rows and 58 columns. Each row is a document represented as a 58 dimensional vector of features. Each component in the vector represents the importance of a word in the document.

2. **random_centroids.tx**t contains k initial cluster centroids. These centroids were chosen by selecting k = 10 random points from the input data.

Algorithm 1 Iterative *k*-Means Algorithm:

```
1: procedure Iterative $k$-Means
2:     Select $k$ points as initial centroids of the $k$ clusters.
3:     for iterations := 1 to MAX ITER do
4:             for each point $p$ in the dataset do
5:             Assign point $p$ to the cluster with the closest centroid
6:             end for
7:             Calculate the cost for this iteration.
8:             for each cluster $c$ do
9:             Recompute the centroid of $c$ as the mean of all the data points assigned to $c$
10:            end for
11:    end for
12: end procedure
```

Set number of iterations (MAX_ITER) to 20 and number of clusters *k* to 10 for this question. Your program should ensure that the correct amount of iterations are run.

(a) Exploring clustering with Euclidean distance [20 pts]

Using the Euclidean distance (refer to Equation 1) as the distance measure, use the Algorithm to compute the cost function $\varphi(i)$ (refer to Equation 2) at every iteration *i*. Run the *k*-means on data.txt using random_clusters.txt as the initial clusters (so in your first iteration i=1, you'll be computing the cost function using these initial centroids). Generate a graph where you plot the cost function $\varphi(i)$ as a function of the number of iterations *i*=1..20

(Hint: Look at step 7 of Algorithm 1 - you should be able to calculate costs here, right after partitioning points into clusters.)

(b) Exploring clustering with Manhattan distance [20 pts]

Using the Manhattan distance metric (refer to Equation 3) as the distance measure, compute the cost function $\psi(i)$ (refer to Equation 4) for every iteration *i*. Run the *k*-means on data.txt using random_clusters.txt as the initial clusters (so in your first iteration i=1, you'll be computing the cost function using these initial centroids). Generate a graph where you plot the cost function $\varphi(i)$ as a function of the number of iterations *i*=1..20

(Hint: This problem can be solved in a similar manner to that of part (a). Also note that It's possible that for Manhattan distance, the cost do not always decrease. K- means only ensures monotonic decrease of cost for squared Euclidean distance. Look up K-medians to learn more.)

(c) Reasoning Percentage Improvements [10 pts]

What is the percentage change in cost after 10 iterations of the K-Means algorithm when the distance metric being used is Manhattan distance? What about when it is Euclidean distance? Is random initialization of clusters for *k*-means better coupled with Euclidean or Manhattan distance? Explain your reasoning.

What to submit:

(i) code for 2(a) and 2(b)

(ii) A plot of cost vs. iteration for both distance strategies

(iii) Percentage improvement values and your explanation

2 LSH evaluation (10 points)

Evaluate the S curve, which you might recall we discussed in class is as follows:
$1-(1-s^r)^b \; for\ s = 0.1, 0.2 \ldots 0.9$ and for the following values of r and b: (show all your work):

- r =2 and b = 5
- r =6 and b = 15
- r =4 and b = 30

3 LSH for approximate near neighbor search(40 points)

A dataset of images, **patches.csv**, and a starter code, **lsh.py**, are provided in **data/q3**.

Each row in the .csv is a 20 × 20 image patch represented as a 400-dimensional vector. We would like to evaluate the performance of LSH-based approximate near neighbor search. You should use the code provided with the dataset for this task.

The included starter code marks all locations where you need to contribute code with TODOs. In particular, you will need to use the functions lsh_setup() and lsh_search(). The default parameters L = 10*,* k = 24 to lsh_setup() work for this exercise, but feel free to use other parameter values as long as you explain the reason behind your parameter choice.

We will first evaluate the questions below using the L1 distance metric to define the similarity of the images, and then do the same using the Jaccard similarity instead.

- For each of the image patches in rows 1000*,* 2000*,* 3000*, . . . ,* 10000, find the top 3 near neighbors[1] (excluding the original patch itself) using LSH. Print their indices. What is the average search time for LSH?

- Plot the top 10 nearest neighbors (using the default L = 10*,* k = 24 or your alternative choice of parameter values for LSH) for the image patch in row 100, together with the image patch itself. You may find the function plot() useful. How do they compare visually?

How do the results for L1 and Jaccard similarity compare? Why could this be?

[1] Sometimes the function lsh_search may return less than 3 nearest neighbors. You can use a while loop to check that lsh_search returns enough results, or you can manually run the program multiple times until it returns the correct number of neighbors.

What to submit:

(i) 3 nearest neighbors for images 1000, 2000, …, 10000, and the average search time for LSH (both L1 and Jaccard)

(ii) Plot of 10 nearest neighbors of image 100 found by the two methods (also include the original image)

(iii) Brief comparison of the results obtained

(iv) Code for all the above.