CSCI 5702/7702 - Fall 2019 - Assignment 1
Locality Sensitive Hashing for Approximate Nearest Neighbor Search

Due Date: 09/25

Problem Description

In this assignment, we study the application of LSH to the problem of finding *approximate* near neighbors. You will be provided a template of partially completed PySpark code to fill in, then you will use your implementation to study the performance of LHS for approximate nearest neighbor search.

Assume we have a dataset A of n points in a metric space with distance metric $d(\cdot, \cdot)$. Let c be a constant greater than 1. Then, the (c, λ)-Approximate Near Neighbor (ANN) problem is defined as follows: Given a query point z, assuming that there is a point x in the dataset with $d(x,z) \leq \lambda$, return a point x' from the dataset with $d(x',z) \leq c\lambda$ (this point is called a (c,λ)-ANN). The parameter c therefore represents the maximum approximation factor allowed and is a user-defined parameter.

Let us consider an LSH family H of hash functions that is $(\lambda, c\lambda, p_1, p_2)$-sensitive[0] for the distance measure $d(\cdot,\cdot)$. Let G[1] $= H^k = \{g = (h_1,...,h_k) | h_i \in H, \forall\ 1 \leq i \leq k\}$, where $k = \log_{1/p_2}(n)$.

Let us consider the following procedure:

1. Select $L = n^\rho$ random members $g_1,...,g_L$ of G, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$
2. Hash all the data points as well as the query point using all g_i $(1 \leq i \leq L)$.

3. Retrieve at most[2] $3L$ data points (chosen uniformly at random) from the set of L buckets to which the query point hashes.

4. Among the points selected in Step 3 (above), report the one that is the closest to the query point as a (c, λ)-ANN.

The goal of the first part of this problem is to show that this procedure leads to a correct answer with constant probability.

[0] A family ***H*** of hash functions is said to be ***(d1, d2, p1, p2)-sensitive*** if for any ***x*** and ***y*** in ***S***: 1. If ***d(x, y) < d1***, then the probability over all ***hî H***, that ***h(x) = h(y)*** is at least ***p1*** 2. If ***d(x, y) > d2***, then the probability over all ***hî H***, that ***h(x) = h(y)*** is at most ***p2.***

[1] The equality $G = H^k$ is saying that every function of G is an AND-construction of k functions of H, so $g(x) = g(y)$ only if $h_i(x) = h_i(y)$ for every h_i underlying g.

[2] If there are fewer than $3L$ data points hashing to the same buckets as the query point, just take all of them.

A dataset of images[3], patches.csv, is provided.

Each row in this dataset is a 20 × 20 image patch represented as a 400-dimensional vector. We will use the L_1 distance metric (see https://en.wikipedia.org/wiki/Taxicab_geometry for a definition) on R^{400} to define similarity of images. We would like to compare the performance of LSH-based approximate near neighbor search with that of linear search.[4] You should use the code provided with the dataset for this task.

The included starter code in lsh.py 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 and implement your own linear 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.

- For each of the image patches in columns 100, 200, 300, …, 1000, find the top 3 nearest neighbors[5] (excluding the original patch itself) using both LSH and linear search. What is the average search time for LSH? What about for linear search?
- Assuming $\{z_j | \ 1 \le j \le 10\}$ to be the set of image patches considered (i.e., z_j is the image patch in column 100j), $\{x_{ij}\}_{i=1}^{3}$ to be the approximate near neighbors of z_j found using LSH, and $\{x^*_{ij}\}_{i=1}^{3}$ to be the (true) top 3 nearest neighbors of z_j found using linear search, compute the following error measure:

$$error = \frac{1}{10}\sum_{j=1}^{10}\frac{\sum_{i=1}^{3} d(x_{i,j}, z_j)}{\sum_{i=1}^{3} d(x^*_{i,j}, z_j)}$$

- Plot the error value as a function of L (for L = 10, 12, 14, …, 20, with k = 24). Similarly, plot the error value as a function of k (for k = 16, 18, 20, 22, 24 with L = 10). Briefly comment on the two plots (one sentence per plot would be sufficient).
- Finally, plot the top 10 nearest neighbors found[6] using the two methods (using the default L = 10, k = 24 or your alternative choice of parameter values for LSH) for the image patch in column 100, together with the image patch itself. You may find the function plot useful. How do they compare visually?

[3] Dataset and code adopted from Brown University's Greg Shakhnarovich

[4] By linear search we mean comparing the query point z directly with every database point x.

[5] 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.

[6] Same remark as [5], you may sometimes have less that 10 nearest neighbors in your results; you can use the same hacks to bypass this problem.

What to submit

Submit the following to Canvas under Assignment 1:

1. A write-up in PDF format including the following:

    - Average search time for LSH and linear search.
    - Plots for error value vs. L and error value vs. K, and brief comments for each plot
    - Plot of 10 nearest neighbors found by the two methods (also include the original image) and brief visual comparison

2. Your code

Some notes:

- Please download your assignment after submission and make sure it is not corrupted. We won't be responsible for corrupted submissions and will not be able to take a resubmission after the deadline.
- You are highly encouraged to ask your question on the designated channel for Assignment 1 on MS Teams. Please DO NOT include your solutions in the comments you share on the board. Feel free to help other students with general questions.
- If you need help from the TAs, please email them.

Hints

- Read the entire code template before you start implementing; the class methods for LSH are meant to build off of each other. Some methods will be pure Python, others will require PySpark, make sure the distinction is clear before you proceed.
- To access a class field from inside of the class in Python you need to use keyword `self` (e.g. to access the object's `functions` field you will need to call `self.functions`)
- Similarly to call a method from inside the class you will need to call `self` (e.g. to call class method `fn()` you need to call `self.fn()`)
- Class functions can be called from within a map function in PySpark using `sc.map(lambda x: self.fn(x), pySparkDF)`