MIE 524 Data Mining
Assignment 3: Locality-Sensitive Hashing

This assignment allows students to get familiar with locality-sensitive hashing.

- Programming language: Python (Google Colab Environment)
- Due Date: Posted in Syllabus

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals, (2) for providing appropriate answers to the questions in a Python notebook (named `MIE524_A3.ipynb`) committed, together with any associated output files, to the student's assignment repository, and (3) attendance in the post-assignment lab quiz.

Please note the plagiarism policy in the syllabus. If you borrow or modify any multiline snippets of code from the web, you are required to cite the URL in a comment above the code that you used. You do not need to cite tutorials or reference content that demonstrate how to use packages – you should certainly be making use of such content.

What/how to submit your work:

1. All your code should be included in a notebook named `MIE524_A3.ipynb` that is provided in the cloned assignment repository.
2. Commit and push your work to your GitHub repository in order to submit it. Your last commit and push before the assignment deadline will be considered to be your submission. You can check your repository online to make sure that all required files have actually been committed and pushed to your repository.
3. Your committed notebook should include all the computed outputs by first running it from top to bottom on Google Colab and then exporting the notebook.
4. A link to create a personal repository for this assignment is posted on Quercus.

This assignment has 3 points in total and the point allocation is shown below:

- Q1a): 1.5 points
- Q1c): 0.5 points
- Q1d): 0.5 points
- Q1e): 0.5 points

Note: Partial points may be given for partial answers. Points will be deducted for missing or incomplete answers. Points could also be deducted for styling.

Attribution: Q1 is inspired by contents from Stanford's CS247.

Q1 - Locality-Sensitive Hashing for Approximate Nearest Neighbours

The Nearest neighbour Search

Assume we have a dataset $\mathcal{A}$ of n points in a metric space with distance metric $d(\cdot, \cdot)$. The Nearest neighbour Search (NNS) problem is defined as follows: Given a query point q, find point x^* in our dataset $\mathcal{A}$ with the smallest distance to q, i.e., with minimum $d(q, x^*)$.

Approximate Nearest neighbour Search

Given the difficulty of the NNS problem, we consider a relaxed version instead by asking for a *near* neighbour (i.e., a neighbour whose distance from the query is smaller than λ) rather than the *nearest* neighbour. The Approximate Nearest neighbour Search (ANNS) problem is defined as follows: Given a query point q, find point x' in our dataset $\mathcal{A}$ with $d(q, x') \leq \lambda$, assuming such point exists. The point x' is then called λ-near neighbour.

Finding λ-near neighbour point using LSH

In this problem, we study the application of LSH to the problem of finding approximate near neighbours. Let us consider a LSH family $\mathcal{H}$ of hash functions that is $(\lambda, c\lambda, p_1, p_2)$-sensitive for the distance measure $d(\cdot, \cdot)$. We amplify this family using the "bands and rows" technique, i.e., by using AND-construction followed by OR-construction. In particular, for b and r that will be specified later, we choose b "band" functions, each is a concatenation of r hash functions chosen uniformly at random from $\mathcal{H}$.

Figure 1 describes the pre-processing and query algorithms. The procedure is guaranteed to return a λ-near neighbour with $1-\delta$, where the failure probability δ depends on the choice of the parameters b and r (Conversely, for each δ, one can provide parameters b and r so that the failure probability is smaller than δ (**however this is outside the scope of this assignment that is not focused on the theoretical guarantees of the approach**).

Figure 1: Pre-processing and query algorithms.

Pre-processing:

1. Choose b functions $g_j, j = 1, ..., b$, each corresponding to a band of r concatenated rows: $g_j = (h_{1,j}, h_{2,j}, ..., h_{r,j})$ where $h_{1,j}, ..., h_{r,j}$ are chosen at random from the LSH family $\mathcal{H}$.
2. Create b hash tables (dictionaries), one for each band, such that the jth hash table contains the dataset points in $\mathcal{A}$ hashed using the band function g_j.

Query algorithm for a query point q:

```
1. For each $j = 1, 2, ..., b$:
    (a) Retrieve the points from the bucket of $g_j(q)$ in the $j$th hash table.
    (b) For each of the retrieved point, compute the distance from q to it, and report the point if it is a correct answer (i.e., if it is a $\lambda$-near neighbour).
```

Task

A dataset of images, `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 L1 distance metric to define similarity of images, and an LSH family that is $(\lambda, c\lambda, p1, p2) - \textit{sensitive}$ for this measure is provided. We would like to compare the performance of LSH-based approximate near neighbour search with that of linear search where we compare each query point q directly with every point x in our dataset.

Please use `MIE524_A3.ipynb` to complete this question. Make sure to **NOT** change defined class and variable names.

a) Complete the code to implement your own `my_LSH` class, the linear search method as well as any additional helper functions necessary. You may start by setting the number of bands, b, to be 10 and the number of rows in each band, b, to be 24.

b) Split the dataset such that 100 randomly selected data points will be the query points and the remaining points will serve as the dataset we are searching on ($\mathcal{A}$).

c) For each of the query images, find the top 3 near neighbours using both LSH and linear search. What is the average search time for LSH? What about for linear search?

d) Evaluate error values when changing b and r. Let $\{q_j | 1 \leq j \leq 100\}$ to be the set of 100 query images considered, $\{x_{ij}\}_{i=1}^{3}$ to be the approximate near neighbours of q_j found using LSH, and $\{x_{ij}^*\}_{i=1}^3$ to be the (true) top 3 near neighbours of q_j found using linear search, compute the following error measure:

$$\text{error} = \frac{1}{100} \sum_{j=1}^{100} \frac{\sum_{i=1}^3 d(x_{ij}, q_j)}{\sum_{i=1}^3 d(x_{ij}^*, q_j)} \tag{1}$$

Plot the error value as a function of b (for $b = 10, 12, 14, \cdots, 20$, with $r = 24$). Similarly, plot the error value as a function of k (for $r = 16, 18, 20, 22, 24$ with $b = 10$). Briefly comment on the two plots (one sentence per plot would be sufficient).

e) Plot the top 10 near neighbours found using the two methods (using the default b = 10, r = 24) for the image patch in row 100, together with the image patch itself. How do they compare visually?

f) **Optional bonus question (0.5pt)**: Implement an additional LSH family $\mathcal{H}$ (e.g., the LSH families discussed in class for Cosine/Euclidean distances). Repeat part d), what do you observe differently?