Algorithms for Big Data Fall 2020

Homework 4 Solutions

Problem 1: Set Similarity

Consider a dataset X that consists of sets of integers in the universe $\{1, 2, \dots, n\}$, i.e., X is a set of sets. For example, there may be a set $A \in X$ which is $A = \{1, 4, 33\}$, and another set $B = \{2, 4, 33\}$. One way to measure the similarity of two non-empty sets is using Jaccard similarity:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

In the above example, $J(A, B) = \frac{|\{4,33\}|}{|\{1,2,4,33\}|} = 2/4 = 1/2$.

(a) Prove that the Jaccard distance $d(A, B) = 1 - J(A, B)$ is a valid distance metric for pairs of sets A, B (that is, show that it satisfies the three properties of a metric).

Solution: 1.

$$d(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} = 1 - \frac{|B \cap A|}{|B \cup A|} = d(A, B)$$

2.

$$d(A, B) = 0 \implies \frac{|A \cap B|}{|A \cup B|} = 1 \implies A = B$$

3. To prove: $d(X, Y) + d(Y, Z) \geq d(X, Z)$ i.e.

$$\frac{|X \cap Y|}{|X \cup Y|} + \frac{|Y \cap Z|}{|Y \cup Z|} - \frac{|X \cap Z|}{|X \cup Z|} \leq 1 \qquad (1)$$

We break up $X \cup Y \cup Z$ into seven disjoint subsets which we use to tackle inequality 1. Thus,

$$X \cup Y \cup Z = (A \cup B \cup C) \cup (D \cup E \cup F) \cup G$$

where, This is easily understood from Fig. 1. Let $a, b, c, \cdots$ denote the number of elements in the sets $A, B, C, \cdots$ respectively. Hence, the inequality 1 becomes,

$$\frac{d+g}{v-c} + \frac{e+g}{v-a} \leq \frac{f+g}{v-b} + 1$$

where $v = a + b + c + d + e + f + g$, the number of elements of $X \cup Y \cup Z$. Further it can be written as,

$$\frac{d'}{c'} + \frac{e'}{a'} \leq \frac{f'}{b'} + 1$$

or

$$a'b'd' + b'c'e' \leq a'c'f' + a'b'c'$$

$$(v^2 - av - bv + ab)d' + (v^2 - bv - cv + bc)e' \leq (v^2 - av - cv + ac)f' + (v^2 - av - bv + ab)c'$$

$$v^3 + v^2(-a - b - c - d - e + f - g) + v[ab + ac + bc + ad + bd + be + ce + 2bg - af - cf]$$
$$+\{-abc - abd - abg - bce - bcg + acg + acf\} \geq 0$$

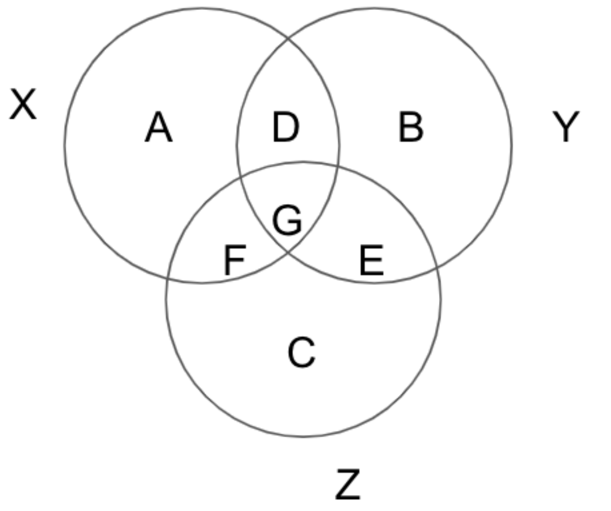

Figure 1: Set and subsets representation

The second term is equal to $v^2(2f - v)$ so that the sum of the first two terms is $2fv^2$. The inequality now, can be written symbollically as,

$$0 \leq v(2fv + [\ ]) + \{\ \}$$

Since $v = a + b + c + d + e + f + g$, the two negative terms of $[\ ]$ cancel with $2fv$. And the negative terms of $\{\ \}$ cancel with terms in $v[\ ]$. Because all the negative terms of the LHS cancel, the inequality is established.

Alternative Solution : Let $d'(X, Y) = |X \oplus Y|$ where $X \oplus Y = (X \cup Y) - (X \cap Y)$. Then, triangular inequality holds for metric $d'(.,.)$. This is easy to see using Figure 1.

$$a + d + c + e \leq (a + f + b + e) + (b + d + f + c)$$

$$d'(X, Z) \leq d'(X, Y) + d'(Y, Z). \tag{2}$$

Let $Y' = Y \cap (X \cup Z)$, then $(X \cup Y') = (X \cup Y) \cap (X \cap Z)$ and $(X \cap Y') = (X \cap Y)$. Similarly, $(Z \cup Y') = (Z \cup Y) \cap (X \cap Z)$ and $(Z \cap Y') = (Z \cap Y)$.
Observe that, if triangular inequality holds for Jaccard distance for Y', it will also hold for Y. Since,

$$\frac{|X \cap Y|}{|X \cup Y|} + \frac{|Y \cap Z|}{|Y \cup Z|} - \frac{|X \cap Z|}{|X \cup Z|} \leq \frac{|X \cap Y'|}{|X \cup Y'|} + \frac{|Y' \cap Z|}{|Y' \cup Z|} - \frac{|X \cap Z|}{|X \cup Z|} \leq 1 \text{ (from eqn 1)}$$

From Equation 2,

$$d'(X, Z) \leq d'(X, Y') + d'(Y', Z),$$

$$|X \oplus Z| \leq |X \oplus Y'| + |Y' \oplus Z|,$$

$$\frac{|X \oplus Z|}{|X \cup Z|} \leq \frac{|X \oplus Y'|}{|X \cup Z|} + \frac{|Y' \oplus Z|}{|X \cup Z|}$$

Since $|X \cup Z| \geq |X \cup Y'|$ and $|X \cup Z| \geq |Z \cup Y'|$, we can replace the two $|X \cup Z|$ in the above inequality with $|X \cup Y'|$ and $|Z \cup Y'|$ in RHS and the inequality will still hold. Therefore,

$$\frac{|X \oplus Z|}{|X \cup Z|} \leq \frac{|X \oplus Y'|}{|X \cup Y'|} + \frac{|Y' \oplus Z|}{|Y' \cup Z|}$$

. And $\frac{|X \oplus Z|}{|X \cup Z|}$ is nothing but the Jaccard distance $d(X, Z)$. Hence,

$$d(X, Z) \leq d(X, Y') + d(Y', Z)$$

Therefore, it also holds for Y.

(b) Consider the following LSH family for Jaccard distance. Each hash function will be based on a random permutation π of the universe $\{1, 2, \ldots, n\}$. Then, we let h_π operate on sets as follows:

$$h_\pi(A) = \operatorname*{argmin}_{a \in A} \pi(a),$$

so $h_\pi(A)$ is the "minimum valued" element in A according to π. Prove that

$$\Pr[h_\pi(A) = h_\pi(B)] = J(A, B),$$

where the probability is over a uniformly random permutation $\pi : [n] \to [n]$.

Solution: Consider an element $c \in A \cup B$.

$$\Pr[c = h_\pi(A \cup B)] = \frac{1}{|A \cup B|}$$

Consider, $h_\pi(A \cup B) \in A \setminus B$ then, h$_\pi(A) \in A \setminus B$ and thus $h_\pi(A) \neq h_\pi(B) \in B$.
Similarly, $h_\pi(A \cup B) \in B \setminus A$ then, $h_\pi(B) \in B \setminus A$ and thus $h_\pi(A) \neq h_\pi(B) \in A$.
If, $h_\pi(A \cup B) \in A \cap B$ then $h_\pi(A \cup B) = h_\pi(B) = h_\pi(A)$.

$$\Pr[_\pi(A) = h_\pi(B)] = \Pr[h_\pi(A \cup B) \in A \cap B)] = \frac{|A \cap B|}{|A \cup B|} = J(A, B)$$

Interestingly, Jaccard distance $d(A, b) = 1 - J(A, B)$ is just $\Pr[h_\pi(A) \neq h_\pi(B)]$ and this can be used to get a concise proof for the triangular inequality proof in part a.

$$\Pr[h_\pi(A) \neq h_\pi(B)] \leq \Pr[(h_\pi(A) \neq h_\pi(C)) \cup (h_\pi(B) \neq h_\pi(C))]$$

Using union bound,

$$\begin{aligned}\Pr[h_\pi(A) \neq h_\pi(B)] &\leq \Pr[(h_\pi(A) \neq h_\pi(C)) \cup (h_\pi(B) \neq h_\pi(C))] \\ &\leq \Pr[h_\pi(A) \neq h_\pi(C)] + \Pr[h_\pi(B) \neq h_\pi(C)]\end{aligned}$$

Therefore,

$$d(A, B) \leq d(A, C) + d(B, C)$$

(c) Explain how you might use the hash family h_π as an LSH family for ANNS under Jaccard distance. What values of r, c make sense? What values of p_1 and p_2 can you achieve? You do not need to analyze the formal near neighbor algorithm (a high-level description suffices).

Solution: Based on the idea of LSH family for Hamming distance that maps each vector to k bits. Choose k random permutations

$$\pi_1, \pi_2, \ldots \pi_k$$

from $[n]$. Then define

$$h^k(x) = [h_{\pi 1}(A), h_{\pi 2}(A), h_{\pi 3}(A) \ldots h_{\pi k}(A)]$$

Then, when $d(A, B) = 1 - J(A, B) = r$ and $\Pr[h(A) = h(B)] = J(A, B) = 1 - r$, we have that p_1 is $1 - r$. And when $d(A, B) = 1 - J(A, B) = cr$ and $\Pr[h(A) = h(B)] = J(A, B) = 1 - cr$, then p_2 becomes $1 - cr$. The above inference should hold good for r, c between 0 and 1. (Given that $J(A, B)$ is between 0 and 1)

Problem 2: 1D is Easy

Provide an algorithm for exact nearest neighbor search on the real line $\mathbb{R}$. For simplicity, assume you have a dataset of n vectors $X \subseteq \mathbb{R}$ such that each vector can be represented using $O(\log n)$ bits (for example X may consist of integers between $-n^2$ and $+n^2$).

The overall space of the algorithm should be $O(n \log n)$. Given a query $q \in \mathbb{R}$ you want to find the closest vector to q in X, that is, output

$$\operatorname*{argmin}_{x \in X} \; |x - q|.$$

A nearest neighbor query with your data structure should use $O(\log n)$ comparisons; so, the total query time should be $O(\log^2 n)$.

Hint: Consider a binary search tree over the vectors of X, and break up the real line into intervals. For each interval, store the largest and smallest vectors from X in the interval.

Solution: Consider $X = \{x_1, x_2, ...x_n\}$. Assume that the elements of X are in sorted order. If not, invoke a sort function to sort X. Build a Binary Search Tree(BST) over X where the leaves of the tree correspond to the elements of X in sorted order from left to right, and every intermediate node in the tree has 2 children - left child and right child. The values of left and right child represents the interval [left child, right child] which corresponds to the respective branch. This can also be viewed as a Hierarchical Clustering design where the inner most clusters are formed by each single element of X. The clusters are built by grouping the pair of nearest points to form one cluster recursively, till one giant cluster is formed which contains all the points in X.
Step 1: For a new query q check if q falls in the interval [min(X), max(X)]. If TRUE then move to step 2 else return $min(|min(X) - q|, |max(X) - q|)$.
Step 2: Check if q falls in the left branch of the tree or right branch of the tree by checking if q is in the interval defined by the left sub-tree or right sub-tree. If q falls in either of the branches then traverse to the respective branch. Else, return the minimum of $|LeftChildMaxValue - q|$ and $|RightChildMinValue - q|$
Step 3: Repeat Step 2 recursively until the closest point is found.
Space and Time: Storage of n elements is $O(n \log n)$ as desired. BST reduces the number of comparisons to $O(\log n)$ comparisons. Thus the total query time is $O(\log^2 n)$

Problem 3: Small Hamming Distance

Let $X \subseteq \{0,1\}^d$ be a dataset of n vectors consisting of d bits each. This problem will show how to solve exact nearest neighbor for Hamming distance one and two. For each subproblem, design a deterministic data structure, prove that it works as desired, and analyze the time/space.

(a) Given a query $q \in \{0,1\}^d$ output a vector $x \in X$ with $d_{\mathrm{H}}(q, x) = 1$ if there exists such a vector. Query time $O(d^2 \log n)$. Space $O(nd)$.

Solution: Sort the elements in X in increasing order and build a Binary Search Tree (BST). During sorting, for $x_i, x_j \in X$, to determine whether or not $x_i < x_j$, it is sufficient to find the smallest index $t \in [d]$ where $x_i^t \neq x_j^t$ such that $x_i < x_j$ if $x_i^t < x_j^t$.

Build the BST data structure with $\leq 2n$ intervals with the end points being elements in this sorted vector sequence. Each interval is represented by 2 vectors thus, occupies a space of $O(d)$. The total space requirement for all the intervals is $O(nd)$.

When a new query q is encountered, all possible vectors with a hamming distance of 1 with q are built. Since q is a d-dimensional vector, only d unique vectors of q are possible which has a hamming distance of 1. In a BST, a vector(if exists) can be found in $O(d \log n)$ time. So for d queries, the total time is $O(d^2 \log n)$.

(b) Given a query $q \in \{0,1\}^d$ output a vector $x \in X$ with $d_{\mathrm{H}}(q,x) \leq 2$ if there exists such a vector. Query time $O(d^3 \log n)$. Space $O(nd)$.

Solution: For a d-dimensional query q, d^2 vectors are possible which have a hamming distance of 2. Therefore, the exact algorithm discussed in $4(a)$ can be used here as well. The total time complexity now becomes $O(d^3 \log n)$.

(c) Given a query $q \in \{0,1\}^d$ output a vector $x \in X$ with $d_{\mathrm{H}}(q,x) \leq 2$ if there exists such a vector. Query time $O(d^2 \log(nd))$. Space $O(nd^2)$.

Hint: Consider the nd possible vectors that have Hamming distance one from vectors in X.

Solution: For each $x \in X$, build all possible vectors of x which has a hamming distance of 1. Now, build a BST on this dn vectors. So the search time in a BST is $O(d \log dn)$. When a new query q comes in, build all possible d vectors of q which has a hamming distance of 1. For each of the d vectors invoke the BST search algorithm. The total run time of the algorithm is $O(d^2 \log(nd))$ and Space is $O(nd^2)$.

Problem 4: Implementing Dimensionality Reduction

Implement and test the Johnson-Lindenstrauss dimensionality reduction method from Lecture 10. The goal here is for you to explore how well the dimensionality reduction works as you change the matrix and the dimensionality of the embedded data. Discuss your findings in addition to providing the experimental results. You do not need to provide code, and you can use whatever programming language you are comfortable with.

(a) Find *two* datasets of $n \geq 200$ points, either randomly generated or from a public repository (e.g., UCI, ScikitLearn, etc). The dimension of the dataset should be at least $d \geq 100$.

(b) Provide results (in a table or plot, clearly labeled) for the distortion of the projected points versus the original points, as you increase the dimensionality of the embedded points (e.g., compare the distortion as you scale from a small number of dimensions to the true dimension of the dataset). Is the behavior the same or different for the two datasets?

(c) Replace the normal distribution with ± 1 random variables. How does the embedding change (better, worse, different, ...)?

(d) **Extra Credit.** Consider the sparse JL transform, where the matrix has entries in $\{-1,0,1\}$ where ± 1 occurs with equal probability for the nonzero entries, but some fraction of each row is fixed to be 0 (the 0 entries chosen randomly in each row). How many non-zero entries do you need to achieve comparable distortion to the case where all entries are normal or ± 1?