This Problem Set consists of 4 questions, but you are free to skip *one* between {Problem 1, Problem 2}. The final deliverable should consist of solutions for 3 problems total. Submitting all 4 will not result in extra credit, though they make up nice exercises. :)

Problem 1: Distance & Metrics. Which of the following distance functions covered in the class are metrics: (a) Minkowski distance with respect to a convex set, (b) cosine distance, (c) edit distance, (d) dynamic time warping.

Justify your answer, namely, if it is a metric then argue that the three properties hold, and if it is not a metric then give a counter-example.

Solution.

(a) (James) **The Minkowski distance with respect to a convex set is completely unrelated to the ℓ^p norm $\|x\| = \sum_{i=1}^n |x_i|^p$. Yes, one needs the so-called Minkowski inequality to prove that the ℓ^p norms satisfy triangle inequality, but that is not part (a) asks.** Given a convex set C containing the origin, $d_C(p, q)$ is defined to be the smallest λ such that $q \in p + \lambda C$, i.e., the smallest scaled copy of C that contains p. Refer to Lecture 8.

In general, this is NOT a metric as it is not symmetric. Consider the 1-dimensional convex set $[-1.5, 1.5]$. Then $d_C(0, 0.5) = 1$ for the smallest λ satisfying $0.5 \in [-1.5\lambda, 0.5\lambda]$ is λ=1; on the other hand, $d_C(0.5, 0) = 1/3$ since the smallest λ satisfying $-0.5 \in [-1.5\lambda, 0.5\lambda]$ is $\lambda = 1/3$.

(b) (Advait) The consine distance is NOT a metric as it fails the identity of indiscernibles: if x and y point in the same direction but have different magnitudes, their cosine distance is still 0.

(c) (Aryaman & Divyansh) The edit distance IS a metric:

- The edit distance is clearly nonnegative and equals 0 iff two strings are identical, i.e., no edit needed.
- Symmetry is also obvious: whatever you do to transform from x to y, reverse them one by one to get from y to x. The total number of steps remains unchanged.
- Converting x to z can be achieved by first converting x to y, then y to z. The minimal cost of going directly from x to z cannot exceed the cost of the aforementioned two-step approach.

(d) (Nicolas) The DTW distance is NOT a metric. It fails the identity of indiscernibles: the DTW distance between $[1, 2, 3]$ and $[1, 2, 2, 3]$ is zero but clearly the two inputs are different.

Problem 2: L_p-norms in $\mathbb{R}^n$. In class we learned about the p-norm of $x \in \mathbb{R}^n$, defined by

$$\|x\|_p = \begin{cases} (\sum_{i=1}^n |x_i|^p)^{1/p} & p \geqslant 1 \\ \max_i |x_i| & p = \infty. \end{cases}$$

as well as the induced metric distance $d_p(x, y) = \|x - y\|_p$.

(1) An interesting fact about p-norms is what is known as the "L_p inclusion:" for $1 \leqslant p < q \leqslant \infty$ [note q can be ∞], we have $\|x\|_p \geqslant \|x\|_q$. Prove this claim using the following:

- First show that if $\|x\|_p = 1$, then $\|x\|_q \leqslant 1$. **Hint**: *raise to appropriate powers*.
- Complete the proof for general x by relating it to $x/\|x\|_p$.

(2) Given a norm $\|\cdot\|$, a unit circle is defined by $B = \{x \in \mathbb{R}^n : \|x\| = 1\}$. Sketch the unit balls in $\mathbb{R}^2$ under p-norm for each $p \in \{1, 2, 5, 10, \infty\}$. What do you observe? How do these observations connect to (1)'s results? (If you are using LaTeX, consider pasting a figure or using `tikz`.)

(3) For $p \in (0, 1)$, we also define $\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$. Sketch the "unit ball" under this "metric." You may notice that this shape looks weird compared to the ones in (2). This is because the "unit ball" for $p \in (0, 1)$ is not *convex*. Prove that $d(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p\right)^{1/p}$ for $p \in (0, 1)$ does not qualify as a metric, hence the quotation marks.

Solution.

(a) If $\|x\|_p = 1$, it follows that $|x_i| \leqslant 1$ for all i, and because $p < q$, $|x_i|^p \geqslant |x_i|^q$. Finally,

$$1 = \|x\|_p^p = \sum_{i=1}^n |x_i|^p \geqslant \sum_{i=1}^n |x_i|^q = \|x\|_q^q.$$

Taking the $(1/q)^{\text{th}}$ power completes the proof. Now for general x, we know the claim holds for the normalized $x/\|x\|_p$. But $c = \|x\|_p$ is a constant, so

$$\|x\|_p = c \cdot \|x/c\|_p \geqslant c \cdot \|x/c\|_q = \|x\|_q.$$

(b) The previous part states that if $p < q$ then $\|x\|_p \geqslant \|x\|_q$. For example, there are points x where $\|x\|_2 > 1$ but $\|x\|_5 = 1$, i.e., the unit ball grows as p increases.

(c) The "unit ball" for $p < 1$ shrinks into star-like shapes. They also fail to meet the triangle ienquality: consider $x = (0, 1), y = (1, 0)$ in $\mathbb{R}^2$. Then $\|x\|_p = \|y\|_p = (1^p + 0^p)^{1/p} = 1$, whereas $\|x + y\|_p = \|(1, 1)\|_p = (1^p + 1^p)^{1/p} = 2^{1/p}$. For $p \in (0, 1)$ we have $2^{1/p} > 2$, so $\|x + y\|_p > \|x\|_p + \|y\|_p$. Bad.

Problem 3: Polynomial-time 1D k-means. Let $X = \{x_1, \ldots, x_n\}$ be a set of real values (i.e., a set of n points in $\mathbb{R}^1$). Describe a polynomial-time algorithm for computing an optimal k-means clustering of X. (**Hint:** *Sort the input values and use dynamic programming*.)

Solution. In 1-dimensional k-means, the optimal clusters must occupy contiguous chunks of points and cannot interleave. To this end, we assume X is sorted (which takes $\mathcal{O}(n \log n)$ time to pre-process). The problem now reduces to partitioning X into k disjoint

Figure 1: Visualizations of different unit balls in $\mathbb{R}^2$. Left: from innermost to outermost: 1-, 2-, 5-, 10-, and ∞-norm. Right: outer diamond is 0.5-"norm" and inner is 0.25-"norm." Note every single unit ball goes through $(0, \pm 1)$ and $(\pm 1, 0)$.

segments. One way to define DP recurrence relation is by conditioning on the rightmost segment: in order to partition the first m elements (where $1 \leqslant m \leqslant n$ in X into j (where $1 \leqslant j \leqslant k$) clusters, we condition on where the $(j-1)^{\text{th}}$ cluster ends (if any).

Formally, define $D[m][j]$ to be the optimal cost of partitioning $\{x_1, \ldots, x_m\}$ into j clusters. The end goal is to compute $D[n][k]$. The recurrence is given by

$$D[m][j] = \min_{i \leqslant m} \Big[\underbrace{D[i][j-1]}_{\text{first } j-1 \text{ clusters}} + \underbrace{\text{Cost}(\{x_{i+1}, \ldots, x_j\})}_{j^{\text{th}} \text{ cluster}} \Big].$$

Note we assume this array is 1*-indexed.* This table is only valid when $m \geqslant j$. The base cases are simple: when $j = 1$, for any m, $D[m][j][= D[m][1] =$ the cost of cluster $\{x_1, \ldots, x_m\}$. For memoization, we will fill columns from left to right, i.e., $D[1 : m][1]$ first, then $D[2 : m][2]$, followed by $D[3 : m][3]$, and so on.

A naïve implementation of the above recurrence relation will take time $\mathcal{O}(kn^3)$, for the table size is $n \times k$, and in each iteration computing the cost takes $\mathcal{O}(n)$ (assuming $\mathcal{O}(1)$ elementary arithmetic operations).

Two questions remain. First, we want to know what the optimal clusters look like, in addition to the total cost. The fix is simple — for each $D[m][j]$ we also maintain a variable indicating the starting index of cluster j:

$$\text{last cluster of } D[m][j] \text{ starts at } \operatorname*{argmin}_{i \leqslant m}(D[i][j-1] + \text{Cost}(\{x_{i+1}, \ldots, x_j\}) + 1.$$

With these, after we compute $D[n][k]$, we can backtrack until we reach index 1. Doing so recovers the exact location of all k clusters. If your solution ends here, you get full credit since that's all the question asked for.

However, you may have noticed it is not very efficient to compute $C(i, j) = \text{Cost}(\{x_i, \ldots, x_j\})$ in each iteration. We want to find a way to reuse some of them. Let $\mu_{i,j} = 1/(j - i + 1) \sum_{\ell=i}^{j} x_\ell$ be the mean of $\{x_i, \ldots, x_j\}$. A common trick is to expand the terms and spot the

fundamental "building blocks:"

$$C(i,j) = \sum_{\ell=i}^{j}(x_\ell - \mu_{i,j})^2 = (j-i+1)\mu_{i,j}^2 + \sum_{\ell=i}^{j} x_\ell^2 - 2\mu_{i,j}\sum_{\ell=i}^{j} x_\ell.$$

The important observation here is that to compute $C(i,j)$, it suffices to maintain the prefix sums $S_m = \sum_{i=1}^{m} x_i$ as well as $T_m = \sum_{i=1}^{m} x_i^2$. Indeed:

- $\mu_{i,j} = 1/(j-i+1)\sum_{\ell=i}^{j} x_\ell = (S_j - S_{i-1})/(j-i+1)$,
- $\sum_{\ell=i}^{j} x_\ell = S_j - S_{i-1}$, and $\sum_{\ell=i}^{j} x_\ell^2 = T_j - T_{i-1}$.

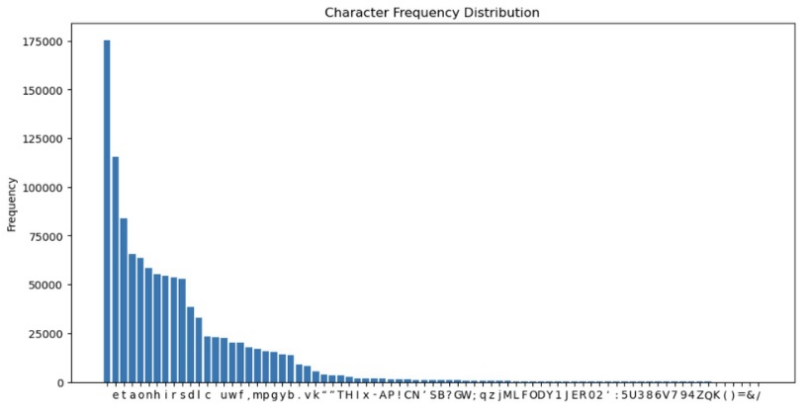
Therefore, if we have maintained S_m and T_m *before* filling out the DP table, computing *any* cluster cost just takes constant time, and so filling the entire table now only requires $\mathcal{O}(kn^2)$ time. The extra pre-processing of building S_m, T_m takes $\mathcal{O}(n)$. Therefore we have successfully derived a 1-D k-means algorithm that runs in $\mathcal{O}(kn^2)$ time.

Problem 4: Huffman Coding Implementation. This problem asks you to implement Huffman coding. Your final deliverable consists of a write-up (see below for details) as well as the source code. You may choose any programming language you prefer, as this problem will be graded mostly on the empirical results (though we will briefly check the code). Please implement the following four modules. You may choose any function signature and return type.

- **ComputeFrequency**. Takes a text corpus as input and computes the frequency distribution of each character.
- **BuildHuffmanTree**. Takes the character frequency distribution as input and converts it into a Huffman encoding tree. You may find it helpful to implement a Huffman class in order to organize your code.
- **Encode**. Takes a Huffman encoding tree and a string message consisting of existing characters in the Huffman encoding tree as inputs. Outputs a binary string that uniquely represents this message according to the Huffman encoding tree.
- **Decode**. Takes a Huffman encoding tree and a binary string as inputs. Outputs a message as a string of existing characters. This decoding should be unique.

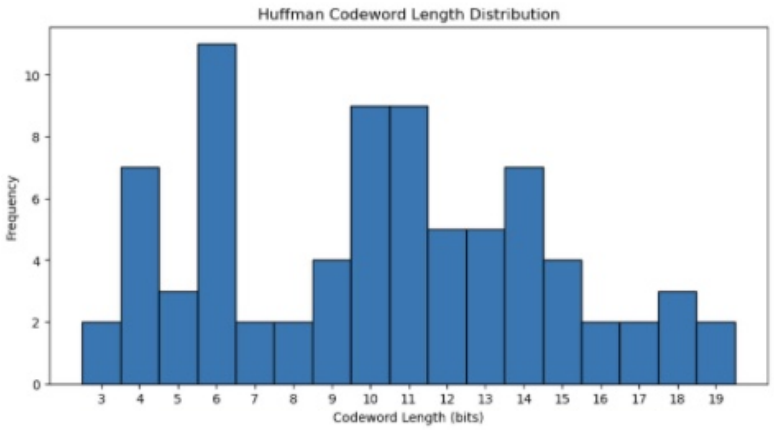
You are provided the short novel *The Strange Case Of Dr. Jekyll And Mr. Hyde* by Scottish author Robert Louis Stevenson as a data set. A plain text version is provided under `Modules/Assignments/CS390_HW2_JH.txt`. Your write-up should include the following results on the above data set.

(1) Compute the frequency distribution for all characters that appear in the novel and plot the distribution. Report the the number of distinct characters in the novel, and the number of nodes required by your Huffman tree.



Character distribution; credits to Nick Maroulis.

(2) Build the Huffman tree for the novel. Plot the distribution of codeword lengths. Report the number of nodes in the first three layers of the tree.



Codeword length distribution; credits to Nick Maroulis.

(3) Calculate the average number of bits per character in your Huffman encoding. Compare this against the theoretical entropy of the character distribution.

For our specific input, the average number of pits per character is around 4.476*, whereas the theoretical entropy is around* 4.444*.*

(4) Encode the entire novel, then decode it. Show a few example outputs and show that the decoded text matches the original input. For example, report a few sentences. For each sentence, print the original sentence, then print its Huffman code, and finally print the decoded output. (If done correctly, after decoding, you should re-obtain an exact copy of the original file.)

Omitted.

(5) Measure the runtime of each module. If you identify a bottleneck, what are some potential improvements?

One would expect the bottleneck to be decoding. One potential optimization is that instead of traversing through the Huffman tree to decode each individual character, once the tree is built, we can convert it into a lookup table.