AMDM HW-1

Abhilash Jain

Problem 1

Given: similarity function $cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$ where $x, y \in \mathbb{R}^d$ a and distance is defined as s

$$d_{cos} = 1 - \frac{cos(x, y) + 1}{2}$$

To Prove: Prove or disprove d_{cos} is a metric

Solution: Let us consider the triangle inequality property of a metric that do not hold for the given distance function

Let us consider 3 vectors, $\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$ such that $\boldsymbol{x}$ and $\boldsymbol{z}$ are orthogonal to each other (i.e $cos(\boldsymbol{x}, \boldsymbol{z}) = 0$) and $\boldsymbol{y}$ is the angle bisector of the angle between $\boldsymbol{x}$ and $\boldsymbol{z}$ (i.e $cos(\boldsymbol{x}, \boldsymbol{y}) = cos(\boldsymbol{y}, \boldsymbol{z}) = cos(45) = \frac{1}{\sqrt{2}}$.

Now let's check triangle inequality, $d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z})$

$$\begin{aligned}
d(\boldsymbol{x}, \boldsymbol{z}) &= 1 - \frac{cos(\boldsymbol{x}, \boldsymbol{z}) + 1}{2} \\
&= 1 - \frac{0 + 1}{2} \\
&= \frac{1}{2} = 0.5 \\
d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z}) &= 1 - \frac{cos(\boldsymbol{x}, \boldsymbol{y}) + 1}{2} + 1 - \frac{cos(\boldsymbol{y}, \boldsymbol{z}) + 1}{2} \\
&= 2 - \frac{\frac{1}{\sqrt{2}} + 1}{2} - \frac{\frac{1}{\sqrt{2}} + 1}{2} \\
&= 1 - \frac{1}{\sqrt{2}} \approx 0.293
\end{aligned}$$

$$\therefore d(\boldsymbol{x}, \boldsymbol{z}) \nleq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z})$$

Hence triangle inequality doesn't hold, therefore d_{cos} is not a metric.

Problem 2

Given: $d : X \times X \to \mathbb{R}_+$ be a metric on X

2.1

To Prove: Show that $D(x, y) = \frac{d(x,y)}{d(x,y)+1}$ is also a metric on X and is it still a metric if we change 1 to $k > 0$?
Solution:

1. Non Negativity
   We know that $d(x, y)$ is a metric on X, which implies $d(x, y) \geq 0, \forall x, y \in X$. Hence $D(x, y) = \frac{d(x,y)}{d(x,y)+1} \geq 0, \forall x, y \in X$.
   Therefore non negativity holds

2. Isolation
   We know that $d(x, y) = 0 \iff x = y$. Now $D(x, y) = \frac{d(x,y)}{d(x,y)+1} = 0$ only when $d(x, y) = 0$. Hence $D(x, y) = 0 \iff x = y$. Therefore isolation holds.

3. Symmetry
   We know that $d(x, y) = d(y, x)$. Now $D(x, y) = \frac{d(x,y)}{d(x,y)+1} = \frac{d(y,x)}{d(y,x)+1} = D(y, x)$. Hence symmetry holds.

4. Triangle Inequality We know that $d(x, z) <= d(x, y) + d(y, z)$. Now

$$\begin{aligned} D(x, z) &= \frac{d(x, z)}{d(x, z) + 1} \\ &\leq \frac{d(x, y) + d(y, z)}{d(x, y) + d(y, z) + 1} \\ &= \frac{d(x, y)}{d(x, y) + d(y, z) + 1} + \frac{d(y, z)}{d(x, y) + d(y, z) + 1} \\ &\leq \frac{d(x, y)}{d(x, y) + 1} + \frac{d(y, z)}{d(y, z) + 1} \\ &\leq D(x, y) + D(y, z) \end{aligned}$$

   Hence triangle inequality holds.
   When we replace 1 with a $k > 0$ then all the above proofs have a positive variable, therefore making no difference in the proving of the same and hence $D(x, y) = \frac{d(x,y)}{d(x,y)+k}$ is also a metric on X.

2.2

In the distance metric $D(x, y), k$ are inversely proportional to one and other, which implies as the value of k increases the values of $D(x, y)$ reaches 1 much slower. The choice of k determines the limit after which the values of $d(x, y)$ are

nearly 1 and hence help normalize $d(x, y)$ in a range of $[0, 1]$ and $D(x, y) \to 1$ as $d(x, y) \to \infty$.

2.3

The new metric $D(x, y)$ can be used in cases when we need to normalize the distances given by $d(x, y)$ in X to a range of $[0, 1]$. A possible application for the new metric would be to use the distance metric to create some similarity function eg: $sim(x, y) = 1 - D(x, y)$ which will yield a value also in the same range with 1 meaning highly similar and 0 meaning dissimilar.

Problem 3

Given: X, d is a metric space. Given 2 subsets $A, b \subseteq X$ then *Hausdorff* distance between A and B is given by

$$d_H(A, B) = \max_{x \in A} \min_{y \in B} d(x, y)$$

Symmetric Hausdorff distance is

$$D_H(A, B) = \max\{d_H(A, B), d_H(B, A)\}$$

To Prove: Prove or disprove D_H is a metric
Solution:
Let us define $d(a, B) = \min_{b \in B} d(a, b)$ similarly $d(b, A) = \min_{a \in A} d(b, a)$. Then we can define $d_H(A, B) = \max_{a \in A} d(a, B)$, $d_H(B, A) = \max_{a \in A} d(b, A)$ and $D_H(A, B) = \max\{d_H(A, B), d_H(B, A)\}$ Now we can move to prove that $D_H(A, B)$ is a metric.

1. Non Negativity
   We are given that d is a metric on X hence $d(x, y) \geq 0, \forall x, y \in X$. Hence $d(a, B) \geq 0$ & $d(b, A) \geq 0$, then it also follows that $d_H(A, B) \geq 0$ & $d_H(B, A) \geq 0$ so $D_H(A, B) \geq 0$. Therefore non-negativity holds.

2. Isolation
   If $D_H(A, B) = 0$, then $d_H(A, B) = d_H(B, A) = 0$. This implies that $\max_{a \in A} d(a, B) = 0$, this means that for the max to be 0 all the values of $a \in A$ have $d_H(a, B) = 0$. Then we have $\min_{b \in B} d(a, b)$ for any value $a \in A$. As B is a finite set, we can find a specific $b \in B$, such that $d(a, b) = 0$ which implies $b = a$.
   Hence we can say that $A \subset B$, similarly $B \subset A$. Therefore $A = B$. As $D_H(A, B) = 0 \iff A = B$, isolation holds.

3. Symmetry
   $D_H(A, B) = \max\{d_H(A, B), d_H(B, A)\} = \max\{d_H(B, A), d_H(A, B)\} = D_H(B, A)$. Therefore symmetry holds.

4. Triangle Inequality
   Take 3 finite point sets A, B, C, we need to prove that $D_H(A,B) \leq D_H(A,C) + D_H(C,A)$, that is :

   $\max\{d_H(A,B), d_H(B,A)\} \leq \max\{d_H(A,C), d_H(C,A)\} + \max\{d_H(C,B), d_H(B,C)\}$

   We start by trying to show

   $$d(a,B) \leq d(a,C) + d_H(C,B)$$

   Now for each $a \in A$ assume that

   $$d(a,C) = \min_{c \in C} d(a,c) = d(a,c_0), c_0 \in C$$
   $$d(c_0,B) = \min_{b \in B} d(c_0,b) = d(c_0,b_0), b_0 \in B$$

   Now because we know that $d(a,B) = \min_{a \in A} d(a,b)$, it means that $d(a,B)$ is the smallest distance from any $a \in A$ to some point $b \in B$. Also we know that $d_H(C,B) = \max_{c \in C} d(c,B)$, so it follows:

   $$\begin{aligned} d(a,B) &\leq d(a,b_0) \\ &\leq d(a,c_0) + d(c_0,b_0) \\ &= d(a,C) + d(c_0,B) \\ &\leq d(a,C) + d_H(C,B) \end{aligned}$$

   We have proved the initial inequality, now we can build from it as follows using the fact that $d_H(A,C) = \max_{a \in A} d(a,C)$ and that $d_H(A,C) \leq \max\{d_H(A,C), d_H(C,A)\}$

   $$\begin{aligned} d(a,B) &\leq d(a,C) + d_H(C,B) \\ &\leq d_H(A,B) + d_H(C,B) \\ &\leq \max\{d_H(A,C), d_H(C,A)\} + \max\{d_H(C,B), d_H(B,C)\} \\ &= D_H(A,C) + D_H(C,B) \end{aligned}$$

   Hence we have proved $d(a,B) \leq D_H(A,C) + D_H(C,B)$ for each $a \in A$. Now we know that $d_H(A,B) = \max_{a \in A}(a,B)$ is just the maximum value of $d(a,B)$. Therefore:

   $$d_H(A,B) = \max_{a \in A} d(a,B) \leq D_H(A,C) + D_H(C,B)$$

   Similarly

   $$d_H(B,A) \leq D_H(A,C) + D_H(C,B)$$

   Hence

   $$D_H(A,B) = \max\{d_H(A,B), d_H(B,A)\} \leq D_H(A,C) + D_H(C,B)$$

   The triangle inequality holds

Hence it has been proved that $D_H(A,B)$ is a metric.

Problem 4

Given: A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. An edge $e = \{u, v\}$ is an unordered pair of vertices, and we say that e is incident to vertices u and v. A walk W on a graph $G = (V, E)$ is defined as a sequence of vertices $W = \{v_0, v_1, \ldots, v_k\}$ so that $\{v_{i-1}, v_i\} \in E$ for all $i = 1, \ldots, k$. The vertices v_0 and v_k are called *source* and *destination* of the walk, respectively.

4.1

To Prove: Given a graph $G = (V, E)$ and two walks W and Z on G, propose a distance function to compare the walks W and Z.
Solution:
The proposed distance function is to take the two different walks W and Z as strings w and z and then using the string edit distance function to define the distance between 2 walks.

4.2

I will address three points on my intuition as to why this is a good distance function :

- In a Walk the order of the vertices is paramount as that defines the path taken.Therefore, the distance function must preserve the order between the vertices or else $A = \{a, b, c, d\}$ would be the same as a walk $B = \{a, c, b, d\}$. Hence any set based approach would remove the order of traversal
- The distance function should be able to compare walks of different lengths as there are no restrictions on the length of the walks in the question. Hence any hamming function based distance would not be appropriate.
- The Walks can be cyclic hence the distance function should account for it and find how similar they are, for example $A = \{a, b, c, d, e, c, d, e, f\}$ has some similarity with the walk $B = \{c, d, e, b, c\}$ as they are overlapping a lot. Hence any distance function that is based on sets will not be appropriate as it will remove multiple similar vertices.

In Conclusion,after these considerations it becomes intuitive to represent vertices the walk as a string s and the distance between two would be just computing the string edit distance between them.

4.3

To prove that the String edit distance function is a metric. Let A and B be two strings that represent the walks W and Z in $G = (V, E)$ the the distance $d(W, Z) = d_s(A, B)$, were $d_s(A, B)$ is the string edit distance between A and B

Non Negativity Since string edit distance is measured in the number of operations required, it cannot be negative.

Isolation If $A = B$ then there is no need to perform any operations hence the distance $d_s(A, B) = 0$. On the other hand if $A \neq B$ then we need at least one operation to make them equal, hence $d_s(A, B) = 0 \iff A = B$.

Symmetry Let $O = \{o_1, o_2, \ldots, o_n\}$ be the operations required to transform the string A to B, then $d_s(A, B) = |O|$. Then we can use $O' = \{o_n, o_{n-1}, \ldots, o_1\}$ to transform B to A and $|O| = |O'|$

Triangle Inequality Let A, B, and C be three strings corresponding to the walks W, X, and Z in a graph. Let $O_{A,B}, O_{B,C}$, and $O_{A,C}$ be the minimum set of operations to get from A to B, B to C, A to C respectively. Let us claim that $|O_{A,C}| > |O_{A,B}| + |O_{B,C}|$ then we could have replaced $O_{A,C}$ with $O'_{A,C} = |O_{A,B} \cup O_{B,C}|$ such that $|O'_{A,C}| < |O_{A,C}|$ which then contradicts our assumption that $O_{A,C}$ is minimal.Therefore triangle inequality holds
Citing: Exercise solution for this solution

4.4

The algorithm is highlighted by the pseudo-code in figure 1. The running time of this algorithm is algorithm is bounded by the time required to fill the dynamic programming matrix of dimension $m \times n$. Hence the complexity of the algorithm is $\mathcal{O}(mn)$.Do note this is not an optimal implementation of dynamic programming and there exist other better implementations.

```
function LevenshteinDistance(char s[1..m], char t[1..n]):
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t
  // note that d has (m+1)*(n+1) values
  declare int d[0..m, 0..n]

  set each element in d to zero

  // source prefixes can be transformed into empty string by
  // dropping all characters
  for i from 1 to m:
      d[i, 0] := i

  // target prefixes can be reached from empty source prefix
  // by inserting every character
  for j from 1 to n:
      d[0, j] := j

  for j from 1 to n:
      for i from 1 to m:
          if s[i] = t[j]:
            substitutionCost := 0
          else:
            substitutionCost := 1
          d[i, j] := minimum(d[i-1, j] + 1,                  // deletion
                             d[i, j-1] + 1,                  // insertion
                             d[i-1, j-1] + substitutionCost)  // substitution

  return d[m, n]
```

Figure 1: Pseudo-code of calculating the distance for two walks $s[1, \ldots, m]$ and $t[1, \ldots, n]$ represented as character arrays

4.5

For the case of computing similarity of the walks of 2 robots that navigate in $2-d$ space, the solution proposed earlier would not be sufficient. This is because it is a continuous space is $\mathbb{R}^2$. The problem is similar to a time series analysis of the two walks.

We can represent the walks through a $2-d$ space as 2 functions of time such as $f(t)$ and $g(t)$. Now that they are time series functions we can try and compare how similar they are and calculate the distance between them.

A good distance function would be dynamic time warping (DTW) to compute the distance of the two walks. DTW is not a metric but would be useful as it can compare the walks that have similar paths and different speeds, also it is helpful for walks that are just an offset of each other.

To find similarity between 2 walks we could just make $sim(X, Y) = 1 - DTW(X, Y)$.

Problem 5

Given: Consider a set $X = \{x_1, \ldots, x_n\}$of n vectors of dimension d, i.e., $xi \in \mathbb{R}^d$
To Prove: We want to compute the pair of furthest vectors in X according to the L_∞ distance. Provide such an algorithm that runs in time $\mathcal{O}(nd)$. Argue about the correctness of your algorithm.
Solution: The L_∞ distance is defined as $L_\infty(x, y) = \max_i|x_i - y_i|$ and the algorithm is shown in figure 2. This algorithm goes through all the vectors for

```
function maxChebyshevDistance(X[x1,x2, ... , xn]):
    #Input is n vecotrs
    #Each vector x_i has d dimensions
    #initiate global maximum dist global vector 1
    #and global vector 2
    global_max_dist = global_vec1 = global_vec2 = NULL

    for j from 1 to d:
        #Initialize temporay variables
        tmp_min = infinity
        tmp_max =  0
        tmp_vec1 = tmp_vec2 = NULL
        for i from 1 to n:
            if(X[i][j] < tmp_min):
                tmp_min  = X[i][j]
                tmp_vec1 = X[i]
            else if(X[i][j] > tmp_max):
                tmp_max  = X[i][j]
                tmp_vec2 = X[i]
        #If any two vectors have a larger distance in dimension j then update global
        if(global_max_dist < (tmp_max - tmp_min)):
            global_max_dist = (tmp_max-tmp_min)
            global_vec1     = tmp_vec1
            global_vec2     = tmp_vec2

    #return the maximum chebyshev distance and the vectors that correspond to it
    return global_max_dist,global_vec1,global_vec2
```

Figure 2: Algorithm for finding furthest vectors in Chebyshev distance for n, d-dimensional vectors

a given dimension and finds the two that are most distant in that dimension. Consequently updating the global variables with the maximum distance and vectors.The output will the furthest Vectors in any Dimension, with the distance between them. This output is the L_∞ distance. It can be seen that the running time of this algorithm is the time to loop through all the dimensions and for each looping over all vectors which is bounded by $\mathcal{O}(nd)$.

Consider that L is the set of all the L_∞ distances between x & y.

$$L = \{L_\infty(x, y) \mid x, y \in X\}$$

Then the problem of finding the furthest vectors along the L_∞ distance is formulated as

$$\max L$$
$$\max_{x,y} \{L_\infty(x, y) \mid x, y \in X\}$$
$$\max_{x,y} \left\{ \max_i |x_i - y_i| \middle| x, y \in X \right\}$$
$$\max_i \left\{ \max_{x,y} |x_i - y_i| \middle| x, y \in X \right\}$$

Hence this shows that finding the vectors that are maximally distant in any dimension will lead to finding the vectors that are the furthest in the L_∞ distance.

Problem 6

A lower bound function is defined as $d_l(x, y) \leq d(x, y), \forall x, y \in X$. This function can be imagined as a function which helps us to reduce our target sample space, in essence it helps us to define a threshold value which decides if or if not to call the costlier function $d(x, y)$. we can try to define it something in the lines of,

$$d_l(x, y) = \begin{cases} 0 & \text{if } y-\text{k} \leq x \leq y+\text{k and } x \neq y \\ |x - y| & \text{otherwise} \end{cases}$$

Defining the threshold(k) value requires paramount attention and care should be taken for the same, as to an inappropriate value, may not reduce the number of steps, but if the optimal threshold value is defined, it is obvious the lower bound function will indeed make less calls to the costlier function

6.2

The variant of the algorithm is as follows:

Algorithm NEAREST
Input: Dataset $D = \{x_1, ..., x_n\}$, distance function d and lower bounded distance function d_l, query point q
Output: object x^* that is the closest to q
1. $dmin \leftarrow d(x_1, q)$
2. $x^* \leftarrow x_1$
3. **for** $i = 2, \ldots, n$
4. $\quad dtmp \leftarrow d_l(x_i, q)$
5. $\quad$ **if** $(dtmp < dmin)$
6. $\quad\quad dtmp \leftarrow d(x_i, q)$

7. **if** $(dtmp < dmin)$
8. $dmin \leftarrow dtmp$
9. $x^* \leftarrow x_i$
10. return x^*

We have added the condition that the function $d(x, y)$ is called only if the $dtmp$ is smaller than the existing $dmin$.This reduces the calls made to the costly function $d(x, y)$.

6.3

A trivial lower bound function like $d_l(x, y) = 0, \forall x, y \in X$ in all practicality is pointless as it means we call the costly $d(x, y)$ for every element,hence not speeding up(or reducing the run-time) in anyway. Now if we have a trivial lower bound as small as possible, this signifies we make marginally less calls to the costly $d(x, y)$ function than when$d_l(x, y) = 0$, but our aim is to make as less calls as possible, and therefore when we have a bound 'as large as possible' we make less calls to the costly function and in turn reducing the run-time (or speeding it up) of the algorithm

6.4

A lower bound for the string edit distance function for two strings x and y of length m and n respectively can be given by

$$d_l(x, y) = |length(x) - length(y)| = |m - n|$$

Where $length(x)$ is a function that returns the length of the string x.

now to prove that $d_l(x, y)$ is actually a lower bound of the Levenshtein distance given by $d(x, y)$. Let us consider 3 cases :

1. The strings x and y are the same, i.e $x = y$

   In this case we know that $d(x, y)$ being a metric will give distance as 0, but the lower bound is also gives the value $d_l(x, x) = |m - m| = 0$. Hence $d_l(x, y) = d(x, y)$ when $x = y$.

   Example: $d(\text{"}hello\text{"}, \text{"}hello\text{"}) = d_l(\text{"}hello\text{"}, \text{"}hello\text{"}) = 0$

2. The strings are of equal length, i.e $length(x) = length(y) \implies m = n$

   We have already handled the case when $x = y$ above. Therefore, if two strings are of equal length and not the same then the string edit distance is **at least 1** operation, i.e substitution, $d(x, y) >= 1$, when $m = n$.

   Now the lower bound function return $d_l(x, y) = |m - m| = 0$ when $m = n$. Hence $d_l(x, y) < d(x, y)$ when $m = n$

   Example: $d(\text{"}HellO\text{"}, \text{"}hello\text{"}) = 2 > d_l(\text{"}HellO\text{"}, \text{"}hello\text{"}) = 0$

3. The strings are of unequal length, i.e $length(x) \neq length(y) \implies m \neq n$

Now when the strings are of unequal length then the string edit distance is at least the difference between the length of the two strings (deletion of $|m-n|$ from the larger string), $d(x,y) >= |m-n|$ when $m \neq n$.

Now the lower bound for two strings of different length is is $d_l(x,y) = |m-n|$ when $m \neq n$.

Hence $d_l(x,y) \leq d(x,y)$ when $m \neq n$

Example: $d(\text{"}seated\text{"}, \text{"}peat\text{"}) = 3 > d_l(\text{"}seated\text{"}, \text{"}peat\text{"}) = 2$

Hence we have shown that $d_l(x,y) \leq d(x,y) \forall x, y \in X$, where X is the set of all strings. Hence $d_l(x,y)$ is a lower bound distance function for $d(x,y)$ and is easy to compute.

Acknowledgments

1. Alexandru Dumitrescu
2. Ananth Mahadevan
3. http://journals.plos.org/plosone/article/file?type=supplementary&id=info:doi/10.1371/journal.pone.0136577.s001
4. Levenshtein Distance Algorithm (Wikipedia)
5. Chebyshev(Wikipedia)