# CS-E4600
# Algorithmic Methods of Data Mining
# Answers to Homework 1

Adam Ilyas 72581

September 28, 2018

Lukas Haug, Bianca Lachennmann , Francsico Caldas, Zeyneb Erdogan

## Problem 1

Consider the cosine similarity function cos(x,y) = $\frac{x \cdot y}{||x||||y||}$ defined for vectors x,y $\in \Re^d$, and the induced distance function

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

Prove or disprove: $d_{cos}$ is a metric.
The induced distance function $d_{cos}$ is not metric as it does not fulfil axiom 4.

**Axiom 4** $d(x, y) \leq d(x, z) + d(z, y)$ (The triangle inequality)
We first analyse the Left hand side

$$\begin{aligned} d_{cos}(x, z) + d_{cos}(z, y) &= 1 - \frac{cos(x, z) + 1}{2} + 1 - \frac{cos(x, y) + 1}{2} \\ &= 2 - \frac{cos(x, z) + cos(z, y) + 2}{2} \\ &= 1 - \frac{cos(x, z) + cos(z, y)}{2} \end{aligned} \tag{1}$$

One counter example is when the angle between x and y becomes large ($\pi$),

$$cos(x, y) = -1, \quad d_{cos}(x, y) = 1$$

Such as when $x = \begin{pmatrix} 1 & 0 \end{pmatrix}^\intercal, y = \begin{pmatrix} -1 & 0 \end{pmatrix}^\intercal$ This happens when $x$ and $y$ are in opposite directions. In Euclidean space, the sum of angles of a triangle is $\pi$, the right hand side of the axiom $d_{cos}(x,z) + d_{cos}(z,y) = 0$

As such, $d_{cos}(x,y) > d_{cos}(x,z) + d_{cos}(z,y)$ and axiom 4 is not fulfilled and induced distance function above $d_{cos}$ is not metric

# Problem 2

Let $d : X \times X \to \Re_+$ be a metric on $X$

**Question 2.1** Show that $D(x,y) = \frac{d(x,y)}{d(x,y)+1}$ is also a metric on X. Is D still a metric if we change 1 to an arbitrary value $k > 0$.

**Axiom 1** $d(x,y) \geq 0$ (no negative distances)
Since $d(x,y)$ is metric and is non-negative, $D(x,y)$ is also non-negative and fulfils axiom 1

**Axiom 2** $d(x,y) = 0$ iff x = y
$d(x,y) = 0$ if x=y. Thus $D(x,y) = 0$ when x=y.

**Axiom 3** $d(x,y) = d(y,x)$ Distance is symmetric.
Since $d(x,y)$ is metric, values do not change if we swap x with y. Since $D(x,y)$ is function of $d(x,y)$, $D(x,y)$ too will be symmetric

**Axiom 4** $d(x,y) \leq d(x,z) + d(z,y)$ (The triangle inequality)

Since $d(x,y)$ is metric, we know that $d(x,y) \leq d(x,z) + d(z,y)$ is true and the following:

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

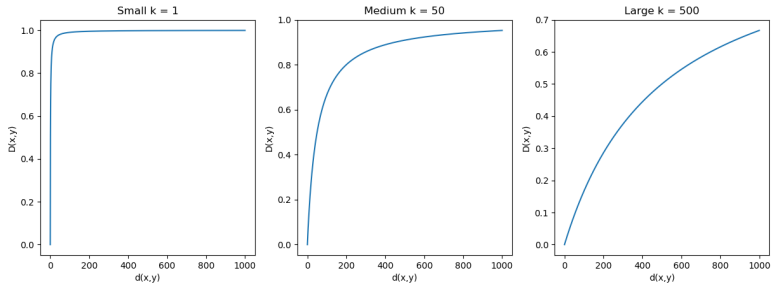
We can assume that right hand side is greater than the left hand side because the function $f(t) = \frac{1}{1+t}$ is a monotonically increasing function. This can be shown by differentiating $f(t)$ to $f'(t) = \frac{1}{(1+t)^2} > 0$ Hence, by replacing $d(x,y)$ with a larger $d(x,z) + d(z,y)$, the value of the output increases. We then

continue:

$$\begin{aligned}\frac{d(x,z)+d(z,y)}{d(x,z)+d(z,y)+1} &= \frac{d(x,z)}{d(x,z)+d(z,y)+1}+\frac{d(z,y)}{d(x,z)+d(z,y)+1}\\ &\leq \frac{d(x,z)}{d(x,z)+1}+\frac{d(z,y)}{d(z,y)+1}\\ &= D(x,z)+D(z,y)\end{aligned} \tag{2}$$

The triangle inequality (axiom 4) is fulfilled and $D(x,y)$ is metric.

**Question 2.2** What is the role of k? How can the choice of k affect the new metric?




$k$ allows us to determine range of values that has the most influence to the value of the $D(x,y)$

**Question 2.3** One possible application is to compare an object with different attributes with different units. An attribte with a smaller range requires a smaller $k$ while larger values require larger $k$.

# Problem 3

In class we defined the Hausdorff distance for comparing sets of points.
In more detail, let $(X,d)$ be a metric space. Given two subsets $A,B\in X$ we define the *Hausdorff* distance between $A$ and $B$ as

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

As already discussed in class, one potential problem with $d_H$ is that it is not symmetric. To overcome this problem we symmetrize Hausdorff by defining

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

Prove or disprove: $D_H$ is a metric.

**Solution**
**Axiom 1** $d(x, y) \geq 0$ (no negative distances)
Since $d(x, y)$ is metric and is non-negative, the max and min of these distance will be non-negative as well. Thus $d_H$ and $D_H$ are non negative and fulfil Axiom 1

**Axiom 2** $d(x, y) = 0$ iff x = y
If A and B are the same set, all distance $d(x, y) \forall x \in A, y \in B$ will be 0. Thus, the following max and min of 0 will be 0 and Axiom 2 is fulfilled. If A and B are different), it will no longer be zero due to choosing the max of the minimum distance and as such, dropping the 0.

**Axiom 3** $d(x, y) = d(y, x)$ Distance is symmetric.

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

Here, we are choose the max between the 2 permutations of A and B .

**Axiom 4** $d(x, y) \leq d(x, z) + d(z, y)$ (The triangle inequality)

Here we have to show that

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

Recall.

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

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

First, we define $d(a, B) := \min_{b \in B} d(a, b)$

We then prove that

$$\begin{aligned} d(a, B) &\leq d(a, C) + d_H(C, B) \\ \min_{b \in B} d(a, b) &\leq \min_{c \in C} d(a, c) + \max_{c \in C} \min_{b \in B} d(c, b) \end{aligned} \tag{3}$$

We assume that for a fixed $a$, $c_0$ is a point on c that is closest to a such that

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

$b_0$ is a point on $B$ that is closest to $c_0$ such that

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

As such

$$\begin{aligned} d(a, B) &\leq d(a, b_0) \\ &\leq d(a, c_0) + d(c_0, b_0) \\ &= d(a, C) + d(c_0, B) \\ &\leq d(a, C) + d(c, B) \end{aligned} \tag{4}$$

This shows that equation(3) holds true. Hence, for each fixed $a \in A$

$$\begin{aligned} d(a, B) &\leq d(a, C) + d_H(C, B) \\ &\leq d_H(A, C) + d_H(C, B) \\ &\leq \max\{d_H(A, C), d_H(C, A)\} + \max\{d_H(C, B), d_H(B, A)\} \\ &= D_H(A, C) + D_H(C, B) \end{aligned} \tag{5}$$

This is true for both:

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

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

As such, the triangle inequality holds.

# Problem 4

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 $W = \{v_0, v_1, ..., v_k\}$ so that $\{v_{i1}, v_i\} \in E$ for all i = 1, . . . , k. The vertices v 0 and v k are called source and destination of the walk, respectively.

**Question 4.1** 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. You should try to design your distance function so that (i) it is intuitive, and (ii) it safisfies the metric properties.

**Question 4.2** Discuss the intuition of the distance function you proposed.

**Solution** The two walks W and Z and a sequence of vertices can be viewed as a 'string', where each element corresponds to a vertices. As such, we can use a string edit distance as a distance function, where each the distance is the minimum number of operations (insertion, deletion and substution) to transform walk W to walk Z.

Since we are using the string edit distance, it is know to be metric.

**Question 4.3** Is your distance function a metric? Prove or disprove your claim.

**Solution** The prove that the *string edit distance* is metric has been shown in the exercise class. We take the 2 walks W and Z (strings in this case), and denote the string edit distance beween them to be $d(X, W)$

**Non-negativity** Since the distance is measured as the minimum of the edit distance, which is non-negative since the cost of an operation (insertion, deletion, substitution) is $+1$. The minimum of a sequence of non-negative numbers is also non-negative

**Isolation/ Coincidence** If $X = W$ that they are the same string then there is no transformation operation required and hence $d(X, W) = 0$. However, if they are both different strings such that $X \neq W$ then we need at least one edit operation to transform $X$ to $W$ and viceversa. Thus the string edit distance will more than 0 in this case.

**Symmetry** Let $T = (t_1, t_2, ..., t_n)$ ) be a minimum set of transformations from X to W. The string edit distance for this set of transformation is the number of elements denoted as $|T|$. The reverse transformation (W to X) denoted as $T' = (t_n, t_{n-1}, ..., t_1)$ is just the reversed of the sequence $T$. Thus, both set of transformations has the same number of elements and have the same string edit distance.

**Triangle Inequality**
For string edit distance $d(X, W)$, if there exist an intermediate string $Z$ such that

$$d(X, W) = d(X, Z) + d(Z, W)$$

then string $Z$ must lie between the transformations from W to Z.

However, if string $Z$ does not lie between these transformation, then $X \rightarrow$

$Z \to W$ is not the minimal edit distance and more transformation is required. For this case,

$$d(X, W) < d(X, Z) + d(Z, W)$$

Thus, from these 2 cases, we can be sure that

$$d(X, W) \leq d(X, Z) + d(Z, W)$$

which agrees with the triangle inequality

**Question 4.4** Provide an algorithm to compute the distance function you proposed. (i) Argue that your algorithm is correct. (ii) What is the complexity of your algorithm?

**Algorithm:**
**Input**: The 2 walks (X,W) $X = \{x_1, .., x_m\}$, $W = \{w_1, .., w_m\}$
**Output:** Minimum string edit distance
**Complexity:** $\mathcal{O}(mn)$

```
Initialize m * n array d

for i in [0..m]
    d[i,0]=i // Initialize

for j in [0..n]
    d[0,j]=j // Initialize

for j in [1..n]
    for i in [1..m]
        if x[i] == w[j]
            d[i, j] = d[i-1, j-1] // no operation required
        else
            d[i, j] = minimum of (
                d[i-1, j] + 1,  // a deletion
                d[i, j-1] + 1,  // an insertion
                d[i-1, j-1] + 1 // a substitution
            )
    return d[m,n] // minimum string edit distance
```

The algorithm here works by reserving a matrix $d$ that stores the edit distance between all prefixes of the first and prefixes of the second string. Wxe can compute the values in the matrix by flood filling the matrix, and thus find the distance between the two full strings as the last value computed.

**Question 4.5** We now want to compute the similarity of the walks of two robots that navigate one 2-d space. Propose an appropriate distance function. You may want to reuse some of the ideas developed in this problem, or you may want to propose a completely different distance function. In either case, provide a justification of your answer.

**Solution** We can implement the string edit distance above to this problem by 'discretizing' time and space.

- Discretize a 2-d space into 1 x 1 metre partitions, each will be represented as Node/ Vertice $V_{i,j}$ where $i, j \in \mathbb{Z}^2$. Let walk will be a sequence of $V_{i,j}$, which is the sequence of location that the robot has been.
- Discretize time to 1 second intervals. Hence every additional second spent on the node will be an additional instance of the same vertice in the walk.

We then can apply string edit distance betweent the 2 sequence of vertices of the 2 robots.

# Problem 5

Consider a set $X = \{x_1, .., x_n\}$ of n vectors in dimensions d, i.e. $x_i \in \Re^d$. We want to compute e the pair of furthest vectors in X according to the $L_\infty$ distance. Provide such an algorithm that runs in time $O(nd)$. Argue about the correctness of your algorithm.

let $x_i = (x_{i1} \quad x_{i2} \quad ... \quad x_{id})^\intercal$
**Algorithm:**
**Input:** dataset x
**Output:** Pair of vectors that has the furthest $L_\infty$ distance.

```
Initialize largestDistance = 0
Initialize furthestVector1 = 0
Initialize furthestVector2 = 0
for currentDim from 1 to d:
        set currentMax = (-inf)
        set currentMin = (+inf)
        set vector1 = 0
        set vector2 = 0
```

```
for i from 1 to n:
        set currentValue as x[i][currentDim]
        if currentValue > currentMax:
                currentMax = currentValue
                vector1 = i
        else if currentValue < currentMin:
                currentMin = currentValue
                vector2 = i
distance = currentMax - currentMin
if distance < 0:
        distance = distance * -1
if distance > largestDistance:
        largestDistance = distance
        furthestVector1 = vector1
        furthestVector2 = vector2

return x[furthestVector1], x[furthestVector2]
```

The suggested algorithm above iterate through all the dimensions (from 1 to d) once, and for each iteration of the dimensions, iterates through all the vectors (from 1 to n) once. As such, the time complexity of the suggested algorithm is $O(nd)$

The algorithm will find the min and max elements in each dimension, and stores which vectors the min and max elements are from. From this it can compute the largest distance for each dimension. It compares the largest of each dimension with each of the other dimension, and returns the vectors that results in the largest distance.

# Problem 6 (Distance lower bounding)

We are given a dataset $D = x_1, ..., x_n$ of n objects. Each object $x_i \in D$ belongs in a space X . We assume that distances of objects in $X$ are measured with a distance function $d : X \times X \to R$. Given a query object q we want to find the object $x^*$ in the dataset D that is the closest to q. In other words we want to find

$$x^* = \arg\min_{x \in D} d(x, q)$$

As a simple example, if D is a set of genomic sequences (strings), given a new sequence q we want to find the sequence in D that is the closest to q. A simple algorithm to find $x^*$ is the following.

Algorithm
**Input:** dataset $D = x_1, ..., x_n$, distance function $d$, query point $q$
**Output:** object $x^* \in X$ that is closest to $q$

1: $dmin \leftarrow d(x_1, q)$
2: $x^* \leftarrow x_1$
3: **for** $i = 2, ..., n$ **do**
4: $\quad dtmp \leftarrow d(x_i, q)$
5: $\quad$ **if** $dtmp < dmin$ **then**
6: $\quad\quad dmin \leftarrow dtmp$
7: $\quad\quad x^* \leftarrow x_i$
8: **return** x

Assume that computing one instance of a distance evaluation requires time T . Then the running time of the algorithm NEAREST is $\mathcal{O}(nT)$.

Issues arise when computing the distance function d is expensive. In this case, the algorithm NEAREST may become quite slow.

For instance, in the example of genomic sequences discussed above, if the distance function is the string edit distance, then one distance computation requires time that is proportional to the product of the length of the two sequences. For large sequences this is prohibitively expensive.

One way to address this computational challenge is to come up with an alternative distance function $d_l$ , which is much faster to compute than d, and it is a lower bound of d. In particular, we want to find a distance function $d_l$ so that

$$d_l(x, y) \leq d(x, y) \text{ for all } x, y \in X$$

**Question 6.1** Explain how a distance function $d_l$ that is fast to compute and is a lower bound of d can be used to speed up the task of finding the nearest object in X for a given query object q.

**Solution** By finding the lower bound distance $d_l$ of each points to q, we do not need to compute the expensive distance $d$ of every single point.

We only need to compute $d$ for points that we are confident of (i.e points with $d_l$ that is more than the minimum of the points before). By removing (pruning) some points, we will be able to save time in computing.

**Question 6.2** Provide pseudo code for a variant of algorithm NEAREST

that uses a lower bound distance $d_l$.

**Input:**
dataset $D = x_1, ..., x_n$
distance function $d$
lower-bound distance function $d_l$
query point $q$

**Output:** object $x^* \in X$ that is closest to $q$

```
1: L = {L_1, ..., L_n}                          ▷ Initialize an array of length n
2: for i = 1, ..., n do
3:     L_i ← d_l(x_i, q)                        ▷ Store the lower bound distance
4:
5: dmin ← d(x_1, q)
6: x* ← x_1
7: for i = 2, ..., n do
8:     if L_i < d_min then                      ▷ Only consider points with larger lower bound
9:         dtmp ← d(x_i, q)
10:        if dtmp < dmin then
11:            dmin ← dtmp
12:            x* ← x_i
13: return x
```

**Question 6.3** While it is easy to come up with trivial lower bound distance functions that are very fast to compute (e.g. $d_l(x, y) = 0$ for all $x, y \in X$ ) we want to find lower bound distance functions that are as large as possible. Explain why

We want a lower bound that is as large as possible (as close to the real value) so that we have more information on the real value, while a small value of lower bound gives use little information

Consider that our lower bound is 0, (or a very small number). Most of the time, this value will be lesser than the current calculated minimum, and we would still have to calculate the expensive $d$. With a larger lower bound, more points can be pruned and as such, we calculate the expensive distance d for lesser points.

**Question 6.4** Propose lower-bound distance functions for the string edit distance.

Lower bound that are easy to compute can be the following

1. hamming distance
2. string edit distance the prefix of the strings with a shorter length (e.g instead of taking the whole word we only take the first 4 letters)
3. Shingling of words where we transform words to a vector of sequence of word (kitten $\rightarrow \{kit, itt, tte, ten\}$)