Contents

Suppose we have

$$X_1 = \begin{pmatrix} x_{11}^{(1)} & x_{12}^{(1)} & \dots & x_{1p}^{(1)} \\ x_{21}^{(1)} & x_{22}^{(1)} & \dots & x_{2p}^{(1)} \\ \dots & \dots & \ddots & \dots \\ x_{n_11}^{(1)} & x_{n_12}^{(1)} & \dots & x_{n_1p}^{(1)} \end{pmatrix} \qquad X_2 = \begin{pmatrix} x_{11}^{(2)} & x_{12}^{(2)} & \dots & x_{1p}^{(2)} \\ x_{21}^{(2)} & x_{22}^{(2)} & \dots & x_{2p}^{(2)} \\ \dots & \dots & \ddots & \dots \\ x_{n_21}^{(2)} & x_{n_22}^{(2)} & \dots & x_{n_2p}^{(2)} \end{pmatrix}$$

0.1 `[dist_struct] = get_pairwise_squared_euclidean_distance(X1,X2)`

We want some matrix $D_{n_1\times n_2}^{(\text{squared euclidean distance})}$ such that

$$\begin{aligned} d_{ij} &:= \left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^2 + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^2 \\ &= \left({x_{i1}^{(1)}}^2 - 2x_{i1}^{(1)}x_{j1}^{(2)} + {x_{j1}^{(2)}}^2\right) + \ldots + \left({x_{ip}^{(1)}}^2 - 2x_{ip}^{(1)}x_{jp}^{(2)} + {x_{jp}^{(2)}}^2\right) \\ &= \left(\sum_{s=1}^{p} {x_{is}^{(1)}}^2\right) + \left(\sum_{s=1}^{p} {x_{js}^{(2)}}^2\right) - \left(2\sum_{s=1}^{p} x_{is}^{(1)}x_{js}^{(2)}\right) \\ &= \|\vec{x}_i^{(1)}\|^2 + \|\vec{x}_j^{(2)}\|^2 - 2\langle \vec{x}_i^{(1)}, \vec{x}_j^{(2)}\rangle \end{aligned}$$

where $\langle \vec{x}, \vec{y}\rangle$ is the ordinary dot product. We can therefore write $D_{n_1\times n_2}^{(\text{squared euclidean distance})}$ as the sum of three matrices $M_1 + M_2 + M_3$, where we have

$$M_1 := \begin{pmatrix} \|\vec{x}_1^{(1)}\|^2 & \|\vec{x}_1^{(1)}\|^2 & \dots & \|\vec{x}_1^{(1)}\|^2 \\ \|\vec{x}_2^{(1)}\|^2 & \|\vec{x}_2^{(1)}\|^2 & \dots & \|\vec{x}_2^{(1)}\|^2 \\ \dots & \dots & \ddots & \dots \\ \|\vec{x}_{n_1}^{(1)}\|^2 & \|\vec{x}_{n_1}^{(1)}\|^2 & \dots & \|\vec{x}_{n_1}^{(1)}\|^2 \end{pmatrix} \quad M_2 := \begin{pmatrix} \|\vec{x}_1^{(2)}\|^2 & \|\vec{x}_2^{(2)}\|^2 & \dots & \|\vec{x}_{n_2}^{(2)}\|^2 \\ \|\vec{x}_1^{(2)}\|^2 & \|\vec{x}_2^{(2)}\|^2 & \dots & \|\vec{x}_{n_2}^{(2)}\|^2 \\ \dots & \dots & \ddots & \dots \\ \|\vec{x}_1^{(2)}\|^2 & \|\vec{x}_2^{(2)}\|^2 & \dots & \|\vec{x}_{n_2}^{(1)}\|^2 \end{pmatrix}$$

and $M_3 = -2X_1X_2^T$.

0.2 `[dist_struct] = get_pairwise_euclidean_distance(X1,X2)`

We want some matrix $D_{n_1\times n_2}^{(\text{euclidean distance})}$ such that

$$d_{ij} := \sqrt{\left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^2 + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^2}$$

Our matrix $D_{n_1\times n_2}^{(\text{euclidean distance})}$ is just the matrix $D_{n_1\times n_2}^{(\text{squared euclidean distance})}$ where we take the square root of each entry.

0.3 `[dist_struct] = get_pairwise_dot_product(X1,X2)`

We want some matrix $D^{(\text{dot product})}_{n_1 \times n_2}$ such that

$$d_{ij} := \sum_{s=1}^{p} x^{(1)}_{is} x^{(2)}_{js}$$

and set $D = X1X_2^T$.

0.4 `[dist_struct] = get_pairwise_angular_distance(X1,X2)`

We want some matrix $D^{(\text{angular distance})}_{n_1 \times n_2}$ such that

$$d_{ij} := \theta$$

where θ is the angular distance between $\vec{x}^{(1)}_i$ and $\vec{x}^{(2)}_j$. Since we know that for any two vectors, we have

$$\cos\theta = \frac{\langle \vec{x}^{(1)}_i, \vec{x}^{(2)}_j \rangle}{\|\vec{x}^{(1)}_i\| \|\vec{x}^{(2)}_j\|}$$

Based on the matrices M_1, M_2 on Page 2, we set $\widetilde{M_1}, \widetilde{M_2}$ to be the matrices M_1, M_2 with the square root taken of each element, and $\widetilde{M_3} := X_1 X_2^T$. We therefore have that

$$d_{ij} := \arccos\left(\frac{\widetilde{m_{3ij}}}{\widetilde{m_{1ij}}\widetilde{m_{2ij}}}\right)$$

and construct D as such.

0.5 `[dist_struct] = get_pairwise_squared_lp_even_distance(X1,X2,p)`

We want some matrix $D_{n_1 \times n_2}^{(\text{squared } l_p \text{ distance})}$ such that

$$d_{ij} := \left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^p + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^p$$

The trick here is to look at the binomial expansion of $(x_i^{(1)} - x_j^{(2)})^p$ when p is even. Using the binomial expansion, we have

$$\begin{aligned}
d_{ij} &= (x_i^{(1)} - x_j^{(2)})^p \\
&= \begin{pmatrix} p \\ 0 \end{pmatrix} x_j^{(2)p} - \begin{pmatrix} p \\ 1 \end{pmatrix} {x_i^{(1)}}^1 x_j^{(2)p-1} + \begin{pmatrix} p \\ 1 \end{pmatrix} {x_i^{(1)}}^2 x_j^{(2)p-2} - \ldots + \begin{pmatrix} p \\ p \end{pmatrix} x_i^{(1)p} \\
&= \sum_{t=0}^{p} (-1)^t \begin{pmatrix} p \\ t \end{pmatrix} {x_i^{(1)}}^t x_j^{(2)p-t}
\end{aligned}$$

We are therefore going to let D be the sum of $p+1$ matrices $M_0 + M_1 + M_2 + \ldots + M_p$. Analogous to the Euclidean distance, we have M_0 to be the matrix where each i^{th} row of the matrix is the l_p norm of $\vec{x}_i^{(1)}$ to the p^{th} power, and M_p to be the matrix where each j^{th} column is the l_p norm of $\vec{x}_j^{(2)}$ to the p^{th} power.

However, each matrix $M_t, t = 1, \ldots, p-1$ will be the product of two other matrices $X_{1t} X_{2t}^T$, where X_{1t} will be the matrix X_1 with all elements raised to the t^{th} power, and X_{2t} will be the matrix X_2 with all elements raised to the $(p-t)^{\text{th}}$ power.

0.6 `[dist_struct] = get_pairwise_lp_even_distance(X1,X2,p)`

We want some matrix $D_{n_1 \times n_2}^{(l_p \text{ distance})}$ such that

$$d_{ij} := \left(\left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^p + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^p\right)^{\frac{1}{p}}$$

Our matrix $D_{n_1 \times n_2}^{(l_p \text{ distance})}$ is just the matrix $D_{n_1 \times n_2}^{(\text{squared } l_p \text{ distance})}$ where we take the p^{th} root of each entry.

0.7 [dist_struct] = get_pairwise_resemblance(X1,X2)

Here, $\vec{x}_i^{(1)}$s, $\vec{x}_j^{(2)}$s are binary.

We want some matrix $D_{n_1\times n_2}^{(\text{resemblance})}$ such that

$$d_{ij} := \frac{\text{sum of logical array in } (\vec{x}_i^{(1)} \texttt{ or } \vec{x}_j^{(2)})}{\text{sum of logical array in } (\vec{x}_i^{(1)} \texttt{ and } \vec{x}_j^{(2)})}$$

We can construct this matrix $D^{(\text{resemblance})}$ by considering two matrices M_1, M_2, and set D to be the elementwise division of elements in M_1 by M_2.

M_1 is set to be the matrix where the $(i,j)^{\text{th}}$ element is the sum of logical array in $(\vec{x}_i^{(1)} \texttt{ or } \vec{x}_j^{(2)})$, and M_2 to be the matrix where the $(i,j)^{\text{th}}$ element is the sum of logical array in $(\vec{x}_i^{(1)} \texttt{ and } \vec{x}_j^{(2)})$.

M_2 can be seen as the matrix product $X_1X_2^T$.

However, constructing M_1 isn't as easy. A loop is used to construct M_1 row by row (or column by column), depending on whether $n_1 \leq n_2$, or $n_2 > n_1$, where the loop with the shortest number of iterations is chosen.

For $n_1 \leq n_2$, the following snippet of code

```
sum(bsxfun(@or, X1(i,:),X2),2)'
```

is used to compute each row of M_1, as `bsxfun` is used to quickly compute the logical `or` for one $\vec{x}_i^{(1)}$ to the whole of X_2. Equivalently, we use

```
sum(bsxfun(@or, X2(j,:),X1),2)
```

to compute each column of M_1.

0.8 [dist_struct] = get_pairwise_squared_lp_odd_distance(X1,X2,p)

We want some matrix $D_{n_1 \times n_2}^{(\text{squared } l_p \text{ distance})}$ such that

$$d_{ij} := \left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^p + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^p$$

Unfortunately, unlike Page 4, we cannot repeat the trick of adding up matrices. Similar to Page 5, we can use the `bsxfun` function to construct D row by row, or column by column. This again depends on whether $n_1 \leq n_2$, or $n_2 > n_1$ where the loop with the shortest number of iterations is chosen.

For $n_1 \leq n_2$, the following snippet of code

```
sum((bsxfun(@minus, X1(i,:), X2)).^p,2)'
```

is used to compute each row of M_1, as `bsxfun` is used to quickly compute $\left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^p + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^p$ for a fixed $x_i^{(1)}$ and the rest of X_2. Equivalently, we use

```
sum((bsxfun(@minus, X2(j,:), X1)).^p,1)
```

to compute each column of D.

0.9 [dist_struct] = get_pairwise_lp_odd_distance(X1,X2,p)

We want some matrix $D_{n_1 \times n_2}^{(l_p \text{ distance})}$ such that

$$d_{ij} := \left(\left(x_{i1}^{(1)} - x_{j1}^{(2)}\right)^p + \ldots + \left(x_{ip}^{(1)} - x_{jp}^{(2)}\right)^p\right)^{\frac{1}{p}}$$

Our matrix $D_{n_1 \times n_2}^{(l_p \text{ distance})}$ is just the matrix $D_{n_1 \times n_2}^{(\text{squared } l_p \text{ distance})}$ where we take the p^{th} root of each entry.

0.10 [dist_struct] = get_pairwise_l1_distance(X1,X2,p)

We want some matrix $D^{(l_1 \text{ distance})}_{n_1 \times n_2}$ such that

$$d_{ij} := \left| x^{(1)}_{i1} - x^{(2)}_{j1} \right| + \ldots + \left| x^{(1)}_{ip} - x^{(2)}_{jp} \right|$$

Unfortunately, unlike Page 4, we cannot repeat the trick of adding up matrices. Similar to Page 5, we can use the `bsxfun` function to construct D row by row, or column by column. This again depends on whether $n_1 \leq n_2$, or $n_2 > n_1$ where the loop with the shortest number of iterations is chosen.

For $n_1 \leq n_2$, the following snippet of code

```
sum(abs(bsxfun(@minus, X1(i,:), X2)),2)'
```

is used to compute each row of M_1, as `bsxfun` is used to quickly compute $\left(x^{(1)}_{i1} - x^{(2)}_{j1}\right)^p + \ldots + \left(x^{(1)}_{ip} - x^{(2)}_{jp}\right)^p$ for a fixed $x^{(1)}_i$ and the rest of X_2. Equivalently, we use

```
sum(abs(bsxfun(@minus, X2(j,:), X1)),1)
```

to compute each column of D.

0.11 [dist_struct] = get_pairwise_l_infinity_distance(X1,X2,p)

We want some matrix $D^{(l_\infty \text{ distance})}_{n_1 \times n_2}$ such that

$$d_{ij} := \max \left\{ \left| x^{(1)}_{i1} - x^{(2)}_{j1} \right| + \ldots + \left| x^{(1)}_{ip} - x^{(2)}_{jp} \right| \right\}$$

We use `bsxfun` again, with the relevant snippet of code for row / column being

```
max(abs(bsxfun(@minus, X1(i,:), X2)),[],2)'
```

and

```
max(abs(bsxfun(@minus, X2(j,:), X1)),[],1)
```

0.12 [dist_struct] = get_pairwise_squared_lp_distance(X1,X2,p)

This function combines the previous functions

- get_pairwise_l1_distance(X1,X2)
- get_pairwise_squared_lp_even_distance(X1,X2,p)
- get_pairwise_squared_lp_odd_distance(X1,X2,p)
- get_pairwise_l_infinity_distance(X1,X2)

0.13 [dist_struct] = get_pairwise_lp_distance(X1,X2,p)

This function combines the previous functions

- get_pairwise_l1_distance(X1,X2)
- get_pairwise_lp_even_distance(X1,X2,p)
- get_pairwise_lp_odd_distance(X1,X2,p)
- get_pairwise_l_infinity_distance(X1,X2)

0.14 [dist_struct] = get_pairwise_jaccard_similarity(X1,X2)

We want some matrix $D_{n_1 \times n_2}^{(l_\infty \text{ distance})}$ such that

$$d_{ij} := \frac{\sum_{s=1}^{p} \min\left\{x_{is}^{(1)}, x_{js}^{(2)}\right\}}{\sum_{s=1}^{p} \max\left\{x_{is}^{(1)}, x_{js}^{(2)}\right\}}$$

We use `bsxfun` again, with the relevant snippet of code for row / column being

```
sum(bsxfun(@min, X1(i,:), X2),2)' ./ sum(bsxfun(@max, X1(i,:), X2),2)'
```

and

```
sum(bsxfun(@min, X2(j,:), X1),1) ./ sum(bsxfun(@max, X2(j,:), X1),1)
```

0.15 [dist_struct] = get_pairwise_hamming_distance(X1,X2)

We want some matrix $D_{n_1 \times n_2}^{(\text{hamming distance})}$ where with binary $\vec{x}_i^{(1)}, \vec{x}_j^{(2)}$, we have

$$d_{ij} := 1_{\{\vec{x}_{i1}^{(1)} \neq \vec{x}_{j1}^{(2)}\}} + \ldots + 1_{\{\vec{x}_{ip}^{(1)} \neq \vec{x}_{jp}^{(2)}\}}$$

We use `bsxfun` yet again, with the relevant snippet of code for row / column being

```
sum(bsxfun(@xor, X1(i,:), X2),2)'
```

and

```
sum(bsxfun(@xor, X2(j,:), X1),1)
```

0.16 [dist_struct] = get_pairwise_distances(X1,X2,dist_type,p)

This function combines all previous functions together, and outputs a distance matrix of the `dist_type` chosen, and p (only for l_p distances). Current choices of distance include

1. `angular_distance`
2. `dot_product`
3. `euclidean_distance`
4. `jacaard_similarity`
5. `lp_distance`
6. `resemblance`
7. `squared_euclidean_distance`
8. `squared_lp_distance`

0.17 [dist_struct] = get_pairwise_distances_big(X1,X2,dist_type,p)

This function constructs the distance matrix D_{ij} by computing the distance of "blocks" of size 250 by 250 (or less) at a time.

For example, if we had $n_1 = 510$ and $n_2 = 375$, then we would construct $D_{510\times 375}$ by computing the distance matrices of six blocks $D^{(1)}, \ldots, D^{(6)}$, as shown below

$$D_{510\times 375} = \left(\begin{array}{c|c} D^{(1)}_{250\times 250} & D^{(2)}_{250\times 125} \\ \hline D^{(3)}_{250\times 250} & D^{(4)}_{250\times 125} \\ \hline D^{(5)}_{10\times 250} & D^{(6)}_{10\times 125} \end{array}\right)$$

and concatenating them to form the matrix D.