DATA.STAT.770

Exercise set 7.

1.

Assume that we have n-datapoints $\{(w_i, h_i, l_i, 0, 0)\}_{i=1}^{n} \subset \mathbb{R}^5$ such that $0 \leq w_i \leq w$, $0 \leq h_i \leq h$, $0 \leq l_i \leq l$. We can cover all datapoints whitin a hypercuboid HC_5 with measures of $(w, h, l, 0, 0)$, where first three represent the widht, height and length in $\mathbb{R}^3$.

We can decompose HC_5 into smaller hypercuboids by multiplying each measure by $\epsilon < 1$. For example, we can define $\epsilon = \dfrac{1}{m}$, where $m \in \mathbb{Z}_+$. Now each hypercuboid has measures of $(\epsilon w, \epsilon h, \epsilon l, \epsilon 0, \epsilon 0) = (\dfrac{1}{m}w, \dfrac{1}{m}h, \dfrac{1}{m}l, 0, 0)$, and the number of smaller hypercuboids that cover the original hypercuboid is m^3, say $N(\epsilon) = N(1/m) = m^3$.

Now we can determine the box-counting dimension of dataset by calculating the limit

$$\lim_{\epsilon\to 0} \frac{log(N(\epsilon)}{log(1/\epsilon)} = \lim_{m\to\infty} \frac{log(N(1/m))}{log(1/(1/m))} = \lim_{m\to\infty} \frac{log(m^3)}{log(m)} = \lim_{m\to\infty} \frac{3log(m)}{log(m)} = 3$$

Therefore the intrinsic dimension of the dataset is three.

2.

Some output from the code:

```
ev_needed =  3
Cube location:
 X: 0 - 0.2
 Y: 0.2 - 0.4
 Z: 0 - 0.2
Eigenvalues needed: 3
povs: 0.00302544, 0.0738224, 1

ev_needed =  3
Cube location:
 X: 0 - 0.2
 Y: 0.2 - 0.4
 Z: 0.2 - 0.4
Eigenvalues needed: 3
povs: 0.0014183, 0.494569, 1

ev_needed =  3
Cube location:
 X: 0 - 0.2
 Y: 0.2 - 0.4
 Z: 0.4 - 0.6
Eigenvalues needed: 3
povs: 0.00214601, 0.187597, 1

.
.
.

ev_needed =  3
Cube location:
 X: 0.8 - 1
 Y: 0.6 - 0.8
 Z: 0.4 - 0.6
Eigenvalues needed: 3
povs: 0.00214476, 0.192936, 1

ev_needed =  3
Cube location:
 X: 0.8 - 1
 Y: 0.6 - 0.8
 Z: 0.6 - 0.8
Eigenvalues needed: 3
povs: 0.00253984, 0.238319, 1

ev_needed =  3
Cube location:
 X: 0.8 - 1
 Y: 0.6 - 0.8
 Z: 0.8 - 1
Eigenvalues needed: 3
povs: 0.00137533, 0.162759, 1
```

The mean of estimated dimensions is 3.

3.

Let

$\{x_i\}_{i=1}^{n} \subset \mathbb{R}^{d_h}$

$\{y_i\}_{i=1}^{n} \subset \mathbb{R}^{d_l}$

where $d_l < d_h$

Cost function 1: $Cost = \sum_i \sum_{j \neq i} |y_i - y_j|^2 - \sum_i \sum_{j \neq i} |x_i - x_j|^2$

Problem:

This cost function doesn't measure the similarity between x's and y's because all distances are calculated between samples in the same set. Therefore the cost function tends to get smaller values when the set of x's is as sparse as possible and when the set of y's is as dense as possible. We can't decide the sparsity of x's since they are fixed, but the sparsest set of y's has a closed form where $y_i = y_j$ for all $i, j \in [1, ..., n]$. This is completely useless result.

Cost function 2: $Cost = \sum_i \sum_{j \neq i} (|x_i - x_j|^2 - |y_i - y_j|^2)^2$

There is no problem in this cost function. The cost function gives us an objective to find the low-dimension representation of the data in a way that the distances between point pairs stays approximately same in both dimensions. In this sense, if we happen to obtain a small value of cost function, we can deduce that the same inner-data characteristics are present in both representations, regardless of the dimensionality difference.

Actually, this cost function is almost equivalent with the MDS stress function, where p_{ij} and d_{ij} are both euclidean distances and f is identity function $f(x) = x,$ presented in the lecture slides. The only difference is that each $(|x_i - x_j|^2 - |y_i - y_j|^2)^2$ is considered twice.

Cost function 3: $Cost = \sum_i |x_i - y_i|^2$

Problem:

Using this cost function, we are trying to minimize the sum of pairwise distances. However, because $d_l < d_h$, the vector subtraction is not defined and thus the cost function is mathematically incorrect.

Cost function 4: $Cost = \sum_i (|x_i - x_{Ref}|^2 - |y_i - y_{Ref}|^2)^2$

Problem:

If we are trying to minimize this cost function, there is no guarantee that the inner characteristics of high-dimensional data will be preserved in low dimension. This is because the similarities are measured implicitly, or globally, via reference point. Therefore local similarities of projected samples in some neighbourhood are not preserved. For example, if we obtain a small value for cost function, it is possible that the samples in a local point neighbourhood of $B_\epsilon(x_i)$ are different in respective neighbourhood of $B_\epsilon(y_i)$. That's because there are infinite number of positions where the euclidean distance between a sample point and reference point is the same. Therefore some local neighbourhood points in high dimension might be lost on lower dimension.

4.

$$Cost = \sum_{i,j} \left(|x_i - x_j| - \sqrt{(y_i - y_j)^T (y_i - y_j)} \right)^2$$

$$\nabla_{y_m} (y_i - y_m)^T (y_i - y_m) = 2(y_m - y_i)$$

$$\begin{aligned}\nabla_{y_m} \sqrt{(y_i - y_m)^T (y_i - y_m)} &= \frac{1}{2}((y_i - y_m)^T (y_i - y_m))^{-\frac{1}{2}} 2(y_m - y_i) \\ &= \frac{y_m - y_i}{\sqrt{(y_i - y_m)^T (y_i - y_m)}}\end{aligned}$$

$$\nabla_{y_m} \left(C - \sqrt{(y_i - y_m)^T (y_i - y_m)} \right)^2 = -2 \left(C - \sqrt{(y_i - y_m)^T (y_i - y_m)} \right) \frac{y_m - y_i}{\sqrt{(y_i - y_m)^T (y_i - y_m)}}$$

Because m can be involved as either index term i or j, the gradient equals to

$$\begin{aligned}\nabla_{y_m} Cost &= -2 * 2 \sum_i \frac{|x_i - x_m| - \sqrt{(y_i - y_m)^T (y_i - y_m)}}{\sqrt{(y_i - y_m)^T (y_i - y_m)}} (y_m - y_i) \\ &= -2 * 2 \sum_i \frac{p_{im} - d_{im}}{d_{im}} (y_m - y_i)\end{aligned}$$

We can interpret the negative gradient as a sum of forces pulling the reference point towards/away from other points if we inspect the terms inside the sum function more specifically

$$\frac{p_{im} - d_{im}}{d_{im}} (y_m - y_i) = (\frac{p_{ij}}{d_{ij}} - 1) y_m - (\frac{p_{ij}}{d_{ij}} - 1) y_i$$

When we treat p_{ij} as constant, we have two cases to consider: the situation where $d_{ij} < p_{ij}$ and situation $d_{ij} > p_{ij}$, alias the cases where the distance after projection is under and overestimated. In the case of underestimation, $(\frac{p_{ij}}{d_{ij}} - 1) < 0$ and therefore in the above equation the both points y_m and y_i are scaled in a way that they become closer of each other (force pulling towards). In the distance overestimation case, say $d_{ij} > p_{ij}$, $(\frac{p_{ij}}{d_{ij}} - 1) > 0$ and hence the both points y_m and y_i are scaled in a way that they become apart of each other (force puling away). In this sense, the gradient descent optimization is well behaving.

The global optimum of this cost function is obviously obtained in a case when $p_{im} = d_{im}$ for all i.