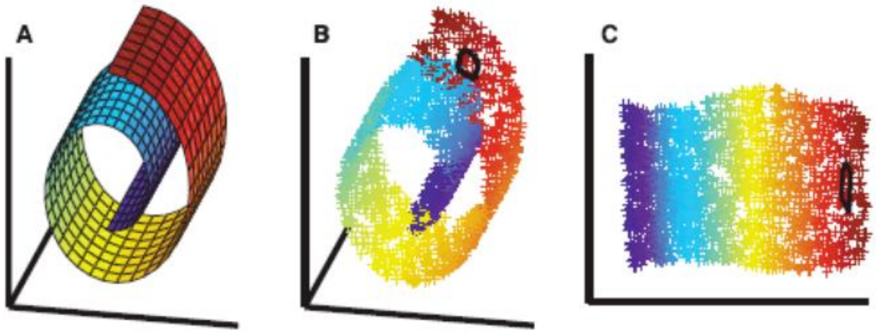
<LLE>

1. 개요

- 스위스롤과 같은 데이터셋을 푸는 매니폴드 방식의 차원축소
- 비지도 학습
- 인접한 데이터를 보존하면서 고차원에서 저차원을 바꾸는 방식



ex) 예를들어 이 3차원 데이터를 2차원으로 바꾸면 더 잘 분류할 수 있을 것 같다. 따라서 3차원에서 이웃했던 데이터들을 그대로 보존하면서 이를 2차원으로 바꾼다.

2.. LLE 알고리즘

Step1 : 이웃 데이터를 고른다.

Step2 : 이웃 데이터로 각 데이터를 가장 잘 표현하는 weight를 고른다.

Step3 : 새로운 축으로 바꿨을 때 가장 잘 매치되는 축으로 고른다.

2-1) Step1

- 각 데이터 포인트에 대해 그 포인트와 가장 가까운 k-개의 이웃점들을 선택, 이 때, k는 하이퍼 파라미터

2-2) Step2

- 각 데이터 포인트에 가장 가까운 k개의 이웃점들로부터 해당 데이터 포인트를 가장 잘 재구성하는 가중치를 찾음. -> reconstruction error을 **최소화**하는 weight

* reconstructino error : 원래 데이터 포인트와 이웃 데이터 포인트로 재구성된 데이터 포인트의 **유클리디언 거리**

$$\min \quad \varepsilon_i(\mathbf{w}) = \left\| \vec{x_i} - \sum_{\substack{j=1 \\ j\neq i}}^{k} w_{ij}\vec{x_j} \right\|^2$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j\neq i}}^{k} w_{ij} = 1$$

2-3) Step3

- Step2에서 찾은 관계를 최대한 보존시키는 고차원 -> 저차원 mapping을 찾음

- 변환된 이웃된 점들에 의해 재구성된 변환된 데이터 포인트와, 실제 데이터 포인트를 변환한 데이터 포인트 간의 차이를 최소화하는 변환을 찾음.

$$\min \quad \Phi(\mathbf{Y}) = \sum_{i=1}^{m} \left\| \vec{y}_i - \sum_{\substack{j=1 \\ j\neq i}}^{k} w_{ij}\vec{y}_j \right\|^2 \quad \text{s.t.} \quad \begin{cases} \frac{1}{m}\sum_{i=1}^{m}\vec{y}_i = 0 \\ \frac{1}{m}\mathbf{Y}^T\mathbf{Y} = \mathbf{I} \end{cases}$$

**제약조건 : Y의 각 열벡터는 선형독립.*

$$\begin{aligned}
\Phi(\mathbf{Y}) &= \sum_{i=1}^{m} \left\| \vec{y}_i - \sum_{\substack{j=1 \\ j\neq i}}^{k} w_{ij}\vec{y}_j \right\|^2 \\
&= \sum_{i=1}^{m} \left[\vec{y}_i^2 - \vec{y}_i \left(\sum_{j=1}^{k} w_{ij}\vec{y}_j \right) - \left(\sum_{j=1}^{k} w_{ij}\vec{y}_j \right) \vec{y}_i + \left(\sum_{j=1}^{k} w_{ij}\vec{y}_j \right)^2 \right] \\
&= \mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T(\mathbf{wY}) - (\mathbf{wY})^T\mathbf{Y} + (\mathbf{wY})^T(\mathbf{wY}) \\
&= \left(\mathbf{Y}^T - \mathbf{Y}^T\mathbf{w}^T\right)\mathbf{Y} - \left(\mathbf{Y}^T - \mathbf{Y}^T\mathbf{w}^T\right)\mathbf{wY} \\
&= \mathbf{Y}^T\left(\mathbf{I} - \mathbf{w}^T\right)\mathbf{Y} - \mathbf{Y}^T\left(\mathbf{I} - \mathbf{w}^T\right)\mathbf{wY} \\
&= \mathbf{Y}^T\left(\mathbf{I} - \mathbf{w}^T\right)(\mathbf{Y} - \mathbf{wY}) \\
&= \mathbf{Y}^T\left(\mathbf{I} - \mathbf{w}^T\right)(\mathbf{I} - \mathbf{w})\mathbf{Y} \\
&= \mathbf{Y}^T(\mathbf{I} - \mathbf{w})^T(\mathbf{I} - \mathbf{w})\mathbf{Y}, \quad \left(\mathbf{M} = (\mathbf{I} - \mathbf{w})^T(\mathbf{I} - \mathbf{w})\right) \\
&= \mathbf{Y}^T\mathbf{M}\mathbf{Y}
\end{aligned}$$

$$\min \quad \Phi(\mathbf{Y}) = \mathbf{Y}^T\mathbf{M}\mathbf{Y}$$

$$\text{s.t.} \begin{cases} \frac{1}{m}\sum_{i=1}^{m}\vec{y}_i = 0 \\ \frac{1}{m}\mathbf{Y}^T\mathbf{Y} = \mathbf{I} \end{cases}$$

Y : 각 데이터 포인터 벡터를 담은 열벡터(MX1)

W : 각각의 데이터에 대응하는 weight를 담은 행렬(MXM)

** 각 데이터 포인터들은 단위벡터*

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{Y}} &= \left(\mathbf{Y} + \mathbf{Y}^T\right)\mathbf{M} - \frac{2}{m}\alpha\mathbf{Y} \\
&= 2\mathbf{MY} - \frac{2}{m}\alpha\mathbf{Y} \\
&= 0
\end{aligned}$$

$$\therefore \mathbf{MY} = \frac{\alpha}{m}\mathbf{Y}$$

- Y는 M의 고유벡터들의 열집합, -> d로의 차원으로 축소하고 싶다면 가장 작은 고유값에 대응하는(오른쪽부터의) 고유 벡터를 선택하면 됨.

- 이 때의 목적식은 M의 가장 작은 고유값이 됨.

* Step2 에서는 각 데이터 포인트 마다의 선형식을 찾았다면, 모든 데이터의 변환은 같아야 하기 때문에

모든 데이터에 대해 최적화된 상을 찾아야 한다.

3) 계산 복잡도
- step1(k개의 가장 가까운 이웃 찾기) : $O(m\log(m)n\log(k))$
- step2(가중치 w의 최적화) : $O(mnk^3)$
- step3(저차원으로의 매핑) : $O(dm^2)$

-> step3에서의 m^2때문에 이 알고리즘을 대량의 데이터셋에 적용하기는 어려움

참고자료 : https://excelsior-cjh.tistory.com/168