# ML Hw8

‣Try to modify the code a little bit and make it back to symmetric SNE

首先要先來看 sne 跟 t-sne 的差別

**1. t-sne 使用的是對稱性 sne，也就是 p(ii),q(ii)定義 0**

**2. t-sne 使用 t-distribution 取代 normal-distribution**

sne:

$$q_{ij} = \frac{\exp(-\| y_i - y_j \|^2)}{\sum_{k \neq l} \exp(-\| y_k - y_l \|^2)}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

t-sne:

$$q_{ij} = \frac{(1+\| y_i - y_j \|^2)^{-1}}{\sum_{k \neq l} (1+\| y_i - y_j \|^2)^{-1}}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+\| y_i - y_j \|^2)^{-1}$$

本題是要修改成 symmetric SNE，可以忽略第一點的差別，所以實際上我們要修改的部分只有 q 跟 gradient 的計算:

Code 的部分:

```
# Compute pairwise affinities
sum_Y = np.sum(np.square(Y), 1)
num = -2. * np.dot(Y, Y.T)
#num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
num = np.exp(-np.add(np.add(num, sum_Y).T, sum_Y))
num[range(n), range(n)] = 0.
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)

# Compute gradient
PQ = P - Q
for i in range(n):
    dY[i, :] = np.sum(np.tile(PQ[:, i] , (no_dims, 1)).T * (Y[i, :] - Y), 0)
    #dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

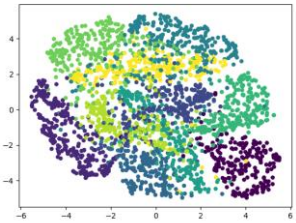
紅色部分是原本 t-sne 的 q 跟 gradient

藍色部分是修改後的 symmetric sne 的 q 跟 gradient

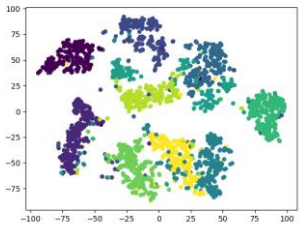
‣Try to visualize the embedding of both t-SNE and symmetric SNE and discuss their differences.

・perplexity = 20.0

symmetric-sne:



t-sne:

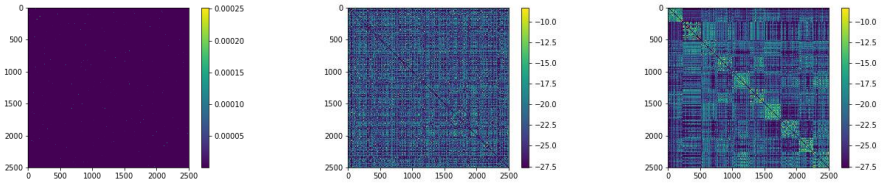


・sne 會有 crowding problem，也就是個個群集會聚集在一起，很難區分開來，並不是因為 sne 只關注局部構造的關係，因為就算使用 symmetric-sne 也是會發生一樣問題(如上圖左)，這是由於高維空間距離分布和低維空間距離分布的差異所造成，因此 t-sn 改用 t-分布即可改善這個問題。

‣Try to visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space, based on both t-SNE and symmetric SNE.

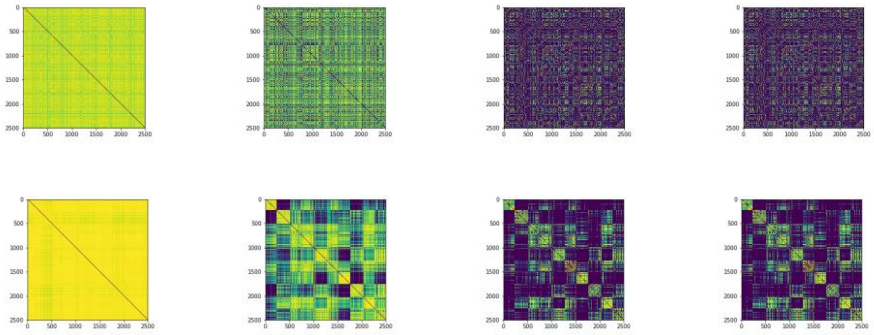
・我們將 symmetric-sne 和 t-sne 的高維 p 和低維的 q 做視覺化，因為這邊 symmetric-sne 和 t-sne 高維的 p 長的一樣，所以只秀一張視覺化的成果。

**high-dimensional**



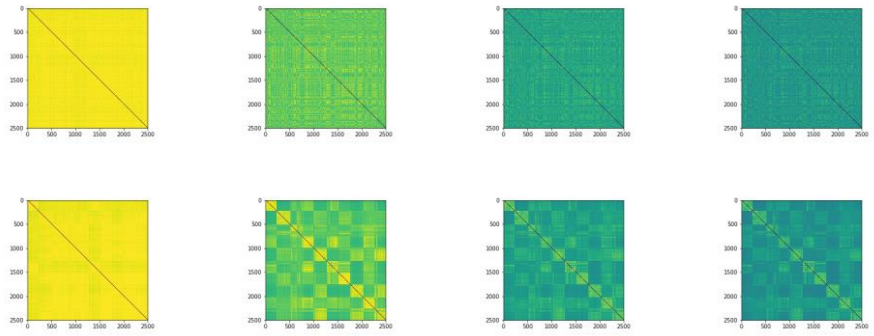
・我們如果直接將高維度的關聯度 p 視覺化，會得到最左邊那張圖，基本上就是一片同樣的顏色，除了少數幾個點顏色有差，因此我們取 log 做 rescale 後再次視覺化，會得到中間的圖，可以看到對角線都是紫色，也就是最小的值，因為我們設定 P(ii) 為 0。如果我們做點小 trick，把資料按照 labels 重新排過，就可以得到最右邊那張圖，可以明顯看到在對角線附近有一格一格的方塊，具有較高的值，也就是關係度較高，十分合理，因為相同 labels 的 data 本身就應該有比較高的關聯度。

**symmetric-sne low-dimensional**



iter=0　　iter=100　　iter=300　　iter=1000

**t-sne low-dimensional**



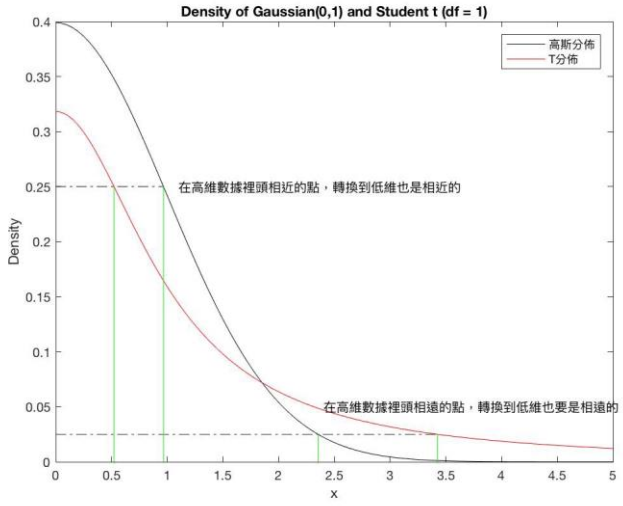
iter=0　　iter=100　　iter=300　　iter=1000

・我們將低維度的關聯 q 視覺化，第一行是未將資料先做排序的結果，第二排是先用 labels 將資料重新排序過，而從左到右是依序迭代後的 q 的變化(都取 log 做 rescale，顏色越暗代表關聯度越低;反之越亮關聯度越高)

・觀察圖表可以得知兩點:

1. 一開始每筆資的關聯度都很高(顏色較亮)，隨著迭代增加，除了同一群集的資料可以保持較高的關聯度，與其不同群集的資料顏色逐漸變深(逐漸分群)

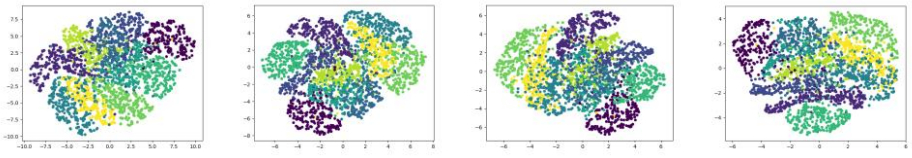
2. t-sne 的圖表顏色分布較為集中(不會太暗、也不會太亮)，推測原因是因為 t-sne 使用的是 t-分布。如下表所示，與高斯分布不同，t-分布分布較集中。



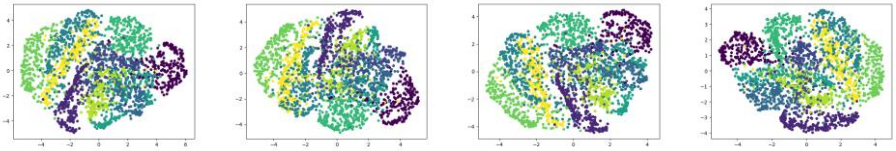

‣Try to play with different settings of perplexity, and see if there is any change in visualization.

・通常來說 perplexity 的值大概是 5~50 之間，因此我們就以每 5 作為一個區間，來試試看不同 perplexity 下對視覺化的影響，除此之外額外用一個 perplexity 100 來測試看看，當 perplexity 非常大的時候，會發生甚麼事情。

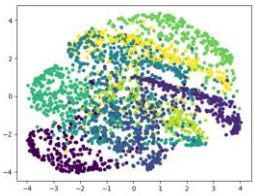
Symmetric-sne

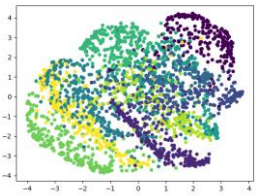


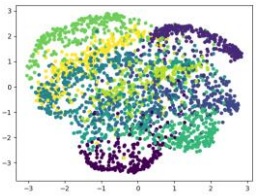
perplexity 5　perplexity 10　perplexity15　perplexity20



perplexity 25　perplexity 30　perplexity 35　perplexity 40

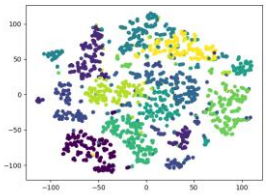

perplexity 45

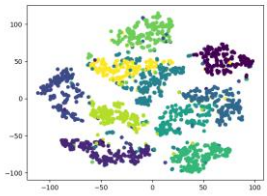

perplexity 50

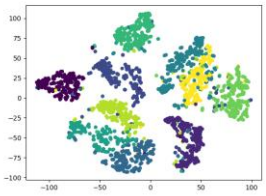

perplexity 100

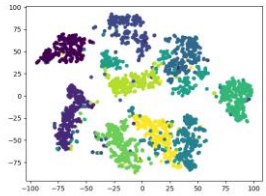
・在 symmetric-sne 改變不同的 perplexity，影響似乎不太大，視覺化後的圖片差異不大，接下來我們在 t-sne 下改變不同的 perplexity 看看。

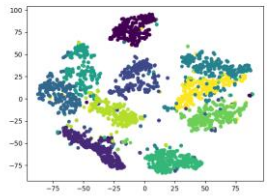
t-sne

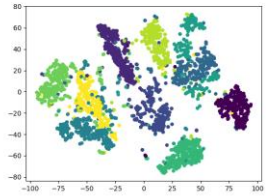

perplexity 5

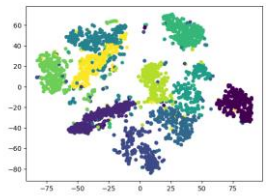

perplexity 10

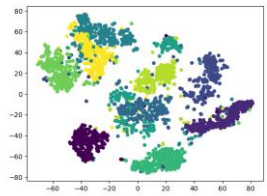

perplexity 15

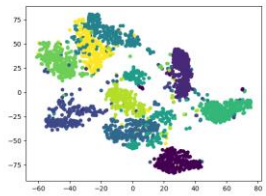

perplexity 20

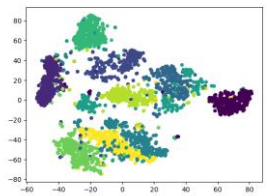

perplexity 25

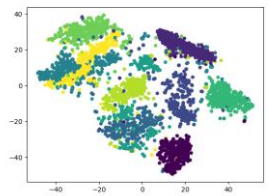

perplexity 30


perplexity 35


perplexity 40


perplexity 45


perplexity 50


perplexity 100

・在 t-sne 改變不同的 perplexity，可以看出在在 perplexity 比較小的時候，各群間各自的資料分布比較散，隨著 perplexity 的增加，群間資料會越來越靠攏，但大概到 25 左右的時候就靠攏了差不多了。隨著繼續增大 perplexity，各群似乎也會有點互相靠攏，有點像 sne 混在一起的感覺。

‣Submit a **report in pdf** format for showing your **code with detailed explanations**, giving **detailed discussion on experiments as well as your observations**.

・本次作業的 code 的是使用題目提供 https://lvdmaaten.github.io/tsne/ 的 python 版本的 code, 實際有做修改的地方只有題目 1 的那個部分。剩下做 visualization 的 code 就只有畫畫圖表，這邊就不特地貼上來解釋了。需要的話自行參考附件。使用的 IDE 是 jupyter。