
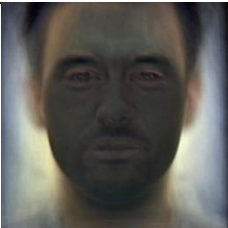
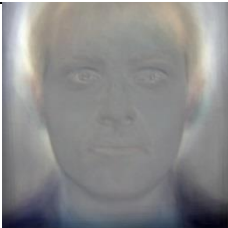
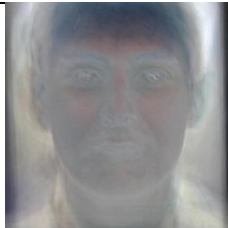



A. PCA of colored faces (Collaborators: None)

A.1. (.5%) 請畫出所有臉的平均。





Average Face Image		Process
		將所有的臉加總之後直接除以總臉數(415 張)再照助教要求的轉換法由 np.float 轉成 np.uint8(先減最小值，再除以最大值，最後*255 轉 np.uint8)

A.2. (.5%) 請畫出前四個 Eigenfaces，也就是對應到前四大 Eigenvalues 的 Eigenvectors。

Top 4 eigenfaces			
Eigenface 1	Eigenface 2	Eigenface 3	Eigenface 4
			

做法: 將所有臉 flatten 為(1,60*60*3)按照 RGB 方式排列的一維 vector，再將所有 415 個圖片的 vector stack 為一個大矩陣(415, 60*60*3)，做 numpy SVD 後得到 U, s, V 三個 array。則 V 的頭 4 個 vector (4, 60*60*3) restack 回來就是 eigenfaces 了

A.3. (.5%) 請從數據集中挑出任意四個圖片，並用前四大 Eigenfaces 進行 reconstruction，並畫出結果。

Results of reconstruction (左:重建結果/ 右:原圖)			
Img_10	Img_100	Img_250	Img_300
			

做法: 由上題的 U 中選出要選擇的圖片的編號對應到的 row(取前四個 column)*s 中相應的加權值，就是對這四個 eigenface 各自的加權值。另外如果是直接拿現成的 eigenfaces 做重建，則可以利用: $W = [u_i \cdot v]$ u_i =i-th eigenface vector, v =origin-img 來得到四個 weight 再做加權組合。

[A.4 at next page]

A.4. (.5%) 請寫出前四大 Eigenfaces 各自所佔的比重，請用百分比表示並四捨五入到小數點後一位。

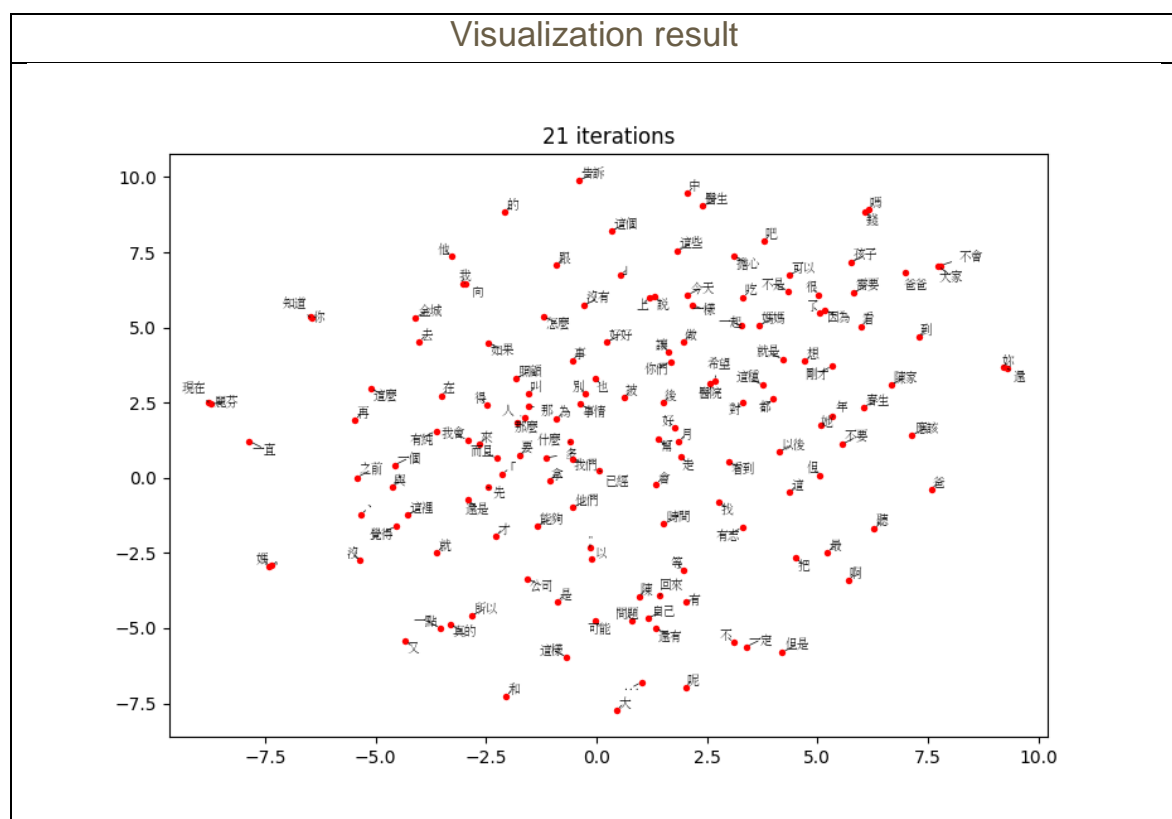
	Eigenface_1	Eigenface_2	Eigenface_3	Eigenface_4
Weights Ratio	4.1%	2.9%	2.4%	2.2%

B. Visualization of Chinese word embedding (Collaborators: None)

B.1. (.5%) 請說明你用哪一個 word2vec 套件，並針對你有調整的參數說明那個參數的意義。

- 使用的 word2vec 套件: genism
- 調整的參數: min_count=5000 / size=128 / iter=10

B.2. (.5%) 請在 Report 上放上你 visualization 的結果。



做法: 使用 TSMN(scikit 函式)將 128 維的 word vector 降維成 2 維，再用 matplotlib 將圖片呈現出來(搭配 adjustText 使用)。值得注意的是的要將中文顯示出來要另外設定 matplotlib 裡面的 font source，不然會顯示不出來。

B.3. (.5%) 請討論你從 visualization 的結果觀察到什麼。

降維之後詞的絕對分布看不出什麼特別的關連。然而一些字詞的性質可以由兩個詞的差向量來看出來，如：“爸”&”媽”的相對位置和”爸爸”跟”媽媽”的相對位置是相似的等等。另外，可以看到儘管將字的頻率調到至少要出現 5000 次，仍有”金城”、”麗芬”這類名字出現。

C. Image clustering (Collaborators: B04901003 許傑盛)

C.1. (.5%) 請比較至少兩種不同的 feature extraction 及其結果。(不同的降維方法或不同的 cluster 方法都可以算是不同的方法)

比較四種不同的方式:

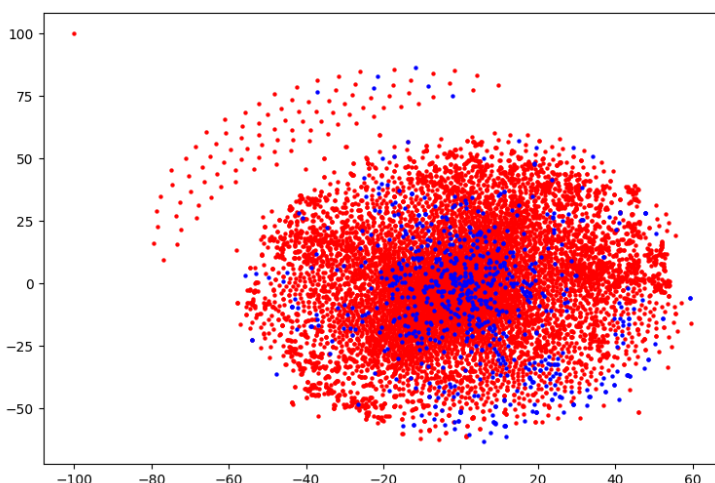
Method 1: Use Autoencoder

	feature extraction 1	feature extraction 2
Method	使用 5 層 deep autoencoder 將圖片壓縮為一個 400 維向量，再將這些向量用 scikit K-means clustering 分為兩個 class	使用 7 層 deep autoencoder 將圖片壓縮為一個 64 維向量，再將這些向量用 scikit K-means clustering 分為兩個 class
Model	<pre> Layer (type) Output Shape Param # ===== input_1 (InputLayer) (None, 784) 0 dense_1 (Dense) (None, 512) 401920 dense_2 (Dense) (None, 400) 205200 dense_3 (Dense) (None, 512) 205312 dense_4 (Dense) (None, 784) 402192 ===== Total params: 1,214,624 Trainable params: 1,214,624 Non-trainable params: 0 Train on 126000 samples, validate on 14000 samples Layers: 784/512/400/512/784 </pre>	<pre> Layer (type) Output Shape Param # ===== input_1 (InputLayer) (None, 784) 0 dense_1 (Dense) (None, 256) 200960 dense_2 (Dense) (None, 128) 32896 dense_3 (Dense) (None, 64) 8256 dense_4 (Dense) (None, 128) 8320 dense_5 (Dense) (None, 256) 33024 dense_6 (Dense) (None, 784) 201488 ===== Total params: 484,944 Trainable params: 484,944 Non-trainable params: 0 Layers: 784/256/128/64/128/256/784 </pre>
Result	Kaggle public score: 0.97806	Kaggle public score: 0.87965

Method 2: use PCA

	PCA (using scikit PCA)	PCA(using SVD)
Method	將圖片以 scikit PCA 壓縮成一個 400 維向量，再以 kmeans 分 2 類	將圖片以 SVD 做 PCA 壓縮成一個 600 維向量，再以 kmeans 分 2 類
Result	Kaggle public score: 0.02794	Kaggle public score: 1.0000

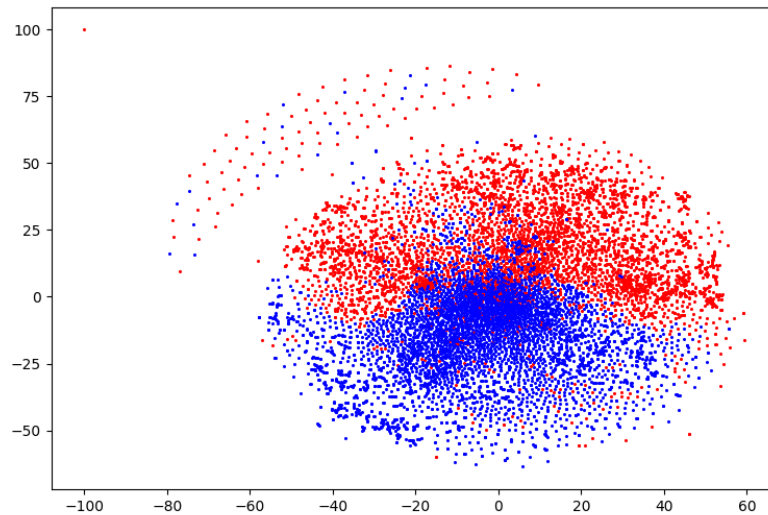
C.2. (.5%) 預測 visualization.npy 的 label，在二維平面上視覺化 label 分佈。



- PCA by SVD: 降至 600 維
- Kmeans clustering
- Visualize: 將 600 維的向量以 scikit TSNE 將其降為 2 維，按其 label 上色。
- 9279 v.s 721(elements)

[C.3 at next page]

C.3. (.5%) visualization.npy 中前 5000 個 images 跟後 5000 個 images 來自不同 dataset。請根據這個資訊，在二維平面上視覺化 label 的分佈，接著比較和自己預測的 label 之間有何不同。



與 C.2 的圖比較，發現實際上以 PCA 取出的 feature 可以大致分出兩個 class 的界線，上半部為 class A，下半部為 class B。比對上圖，可以發現藍點多很多，而且兩個 class 各自很集中不像 C.2 的雜合在一起。因此可以推測 feature 可能是取的相對正確，但是 clustering 並沒有做得很好。