1 降維 - PCA

1.1 降維的意義

降維是統計中『化繁為簡』最好的體現，當你的統計對象有過多的特徵的時候，例如：語言分析的時候我們的文章變成數萬個詞欄位，一般的演算法都是派不上用場的，因為你想想，以決策樹為例

1. 要怎麼創建才能包含萬個欄位
2. 難道第一層用『喜歡』這個詞分左右，『愛』就不用管了嗎?

再或者是第二個例子，一張圖片的欄位就是他的每一個『像素』

1. 難道我們的決策樹是利用一個一個像素分左右嗎?

上面兩個例子你想一想就知道不該是這樣進行的，那怎麼辦呢?

這問題我們稱之為『維度災難』，其實解決辦法就一個，想辦法真正找出『真正維度』，又或者也可以說，將相關的特徵化簡，讓他變成少數幾個獨立的特徵，這也就是『降維』!

1.2 降維的時機

我在通常在幾個時機會使用降維

1. 維度過高的時候
2. 畫圖 (只能 2D/3D)

1.3 降維的方式

1. 機器學習: PCA, tSNE…等等線性/非線性的降維
2. 深度學習: 深度學習的降維效果才是真正一等一的厲害，藉由非線性的組合特徵實現更準確的降維

1.4 PCA 演算法

1.4.1 共變異數 (Covariance)/相關係數 (Correlation)

在學習 PCA 之前，首先你要先了解共變異數和相關係數的關係

$$\rho = \frac{\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i-\mu_x)^2\sum_{i=1}^{n}(y_i-\mu_y)^2}}$$

$$\rho = \frac{x\text{和}y\text{的共變異數}}{x\text{的標準差}\times y\text{的標準差}}$$

$$\text{共變異數(covariance)}: \operatorname{cov}(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)$$

$$\text{變異數(variance)}: \quad \operatorname{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)^2$$

$$\text{標準差(standard deviation)}: \quad std(x) = \sqrt{\operatorname{var}(x)}$$

其實相關係數就只是把標準差去掉的共變異數，所以你在看待共變異數的時候可以先試著用相關係數想，會比較簡單

1.4.2 共變異數矩陣

[程式]:
```
from keras.datasets.mnist import load_data
# ((訓練圖片, 訓練答案), (驗證圖片, 驗證答案))
(x_train, y_train), (x_test, y_test) = load_data()
```

```
Using TensorFlow backend.


Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

[程式]:
```
print(x_train.shape)
print(x_test.shape)
```

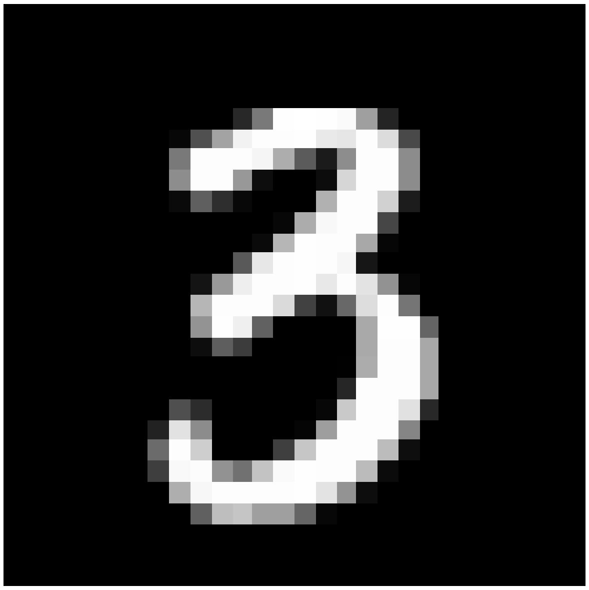
```
(60000, 28, 28)
(10000, 28, 28)
```

[程式]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import random
```

```
c = random.randint(0, 59999)
plt.axis("off")
print("答案:", y_train[c])
plt.imshow(x_train[c], cmap="gray")
```

答案: 3

[輸出]: <matplotlib.image.AxesImage at 0x7f699963e4e0>



[程式]:
```
import pandas as pd
pd.DataFrame(x_train[c])
```

[輸出]:

	0	1	2	3	4	5	6	7	8	...	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	7	...	73	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	121	...	140	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	141	...	140	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	10	...	28	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

11	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	…	6	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	…	117	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	…	254	100	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	…	253	168	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	…	253	168	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	…	253	168	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	79	…	225	40	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	34	226	…	133	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	107	253	…	14	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	62	249	…	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	174	…	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0

[28 rows x 28 columns]

[程式]:
```
x_shape = x_train.reshape((60000, 784))
pd.DataFrame(x_shape)
```

[輸出]:

	0	1	2	3	4	5	6	…	777	778	779	780	781	782	783
0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0

```
22       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
23       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
24       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
25       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
26       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
27       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
28       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
29       0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
...    ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
59970    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59971    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59972    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59973    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59974    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59975    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59976    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59977    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59978    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59979    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59980    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59981    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59982    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59983    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59984    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59985    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59986    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59987    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59988    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59989    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59990    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59991    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59992    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59993    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59994    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59995    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59996    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59997    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59998    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
59999    0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0

[60000 rows x 784 columns]
```

[程式]:

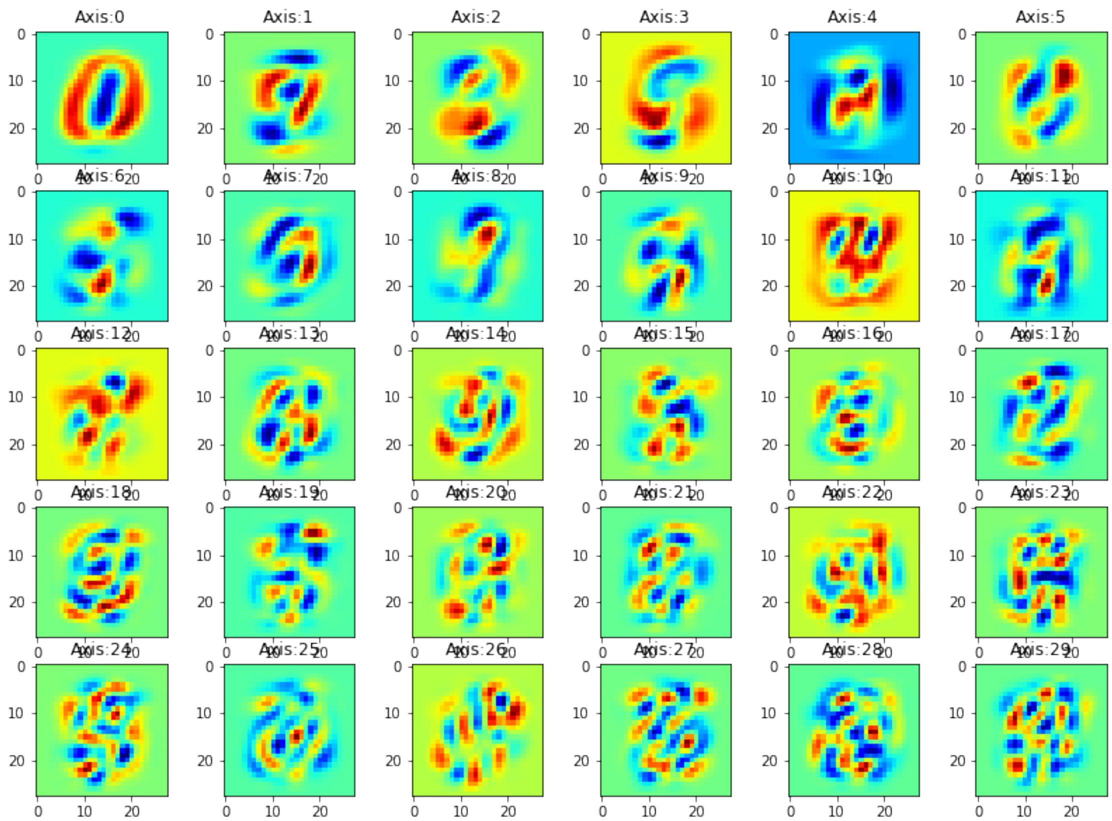
```python
from sklearn.decomposition import PCA
import numpy as np
embedding = PCA(n_components=30)
result = embedding.fit_transform(x_shape, y_train)
```

```
print("成分維度:", embedding.components_.shape)
print("成分維度:", embedding.components_)
print("常數:", embedding.singular_values_)
print("原本的多少:", embedding.explained_variance_ratio_)
print("原本的多少加總", np.sum(embedding.explained_variance_ratio_))
print("轉化過後的維度:", result.shape)
```

```
成分維度: (30, 784)
成分維度: [[ 6.66284109e-18 -7.56191198e-19 -5.27851191e-18 …  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [-7.75230751e-17  1.39214456e-17  4.89498525e-17 … -0.00000000e+00
  -0.00000000e+00 -0.00000000e+00]
 [-3.56254025e-17 -1.03055968e-17  1.10312874e-16 … -0.00000000e+00
  -0.00000000e+00 -0.00000000e+00]
 …
 [ 2.58003474e-17  8.97348483e-18 -3.16781952e-17 …  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [-8.12740288e-18  4.91209840e-18  2.12772883e-17 … -0.00000000e+00
  -0.00000000e+00 -0.00000000e+00]
 [-3.48322742e-17  2.24497786e-17 -1.97129456e-17 …  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]
常數: [141291.00226882 120817.18859621 112650.9232813  105291.96728785
 100077.18497144  94183.59653123  82040.11204589  77021.85990459
  75376.92118236  69631.23337981  65869.14984457  64509.13672405
  59410.04489727  58998.19503317  56985.71111788  55231.63218163
  52198.75702572  51250.98600462  49419.40913254  48694.393342
  46831.29446306  45506.87935159  44289.40499038  43326.09746335
  42628.79043338  41550.1563216   40883.63618253  40216.55567855
  39134.65687662  37658.95811246]
原本的多少: [0.09704664 0.07095924 0.06169089 0.05389419 0.04868797 0.04312231
 0.0327193  0.02883895 0.02762029 0.02357001 0.0210919  0.02022991
 0.01715818 0.01692111 0.01578641 0.01482953 0.01324561 0.01276897
 0.01187262 0.01152682 0.01066164 0.01006713 0.00953567 0.00912537
 0.008834   0.00839261 0.00812551 0.00786251 0.00744517 0.00689427]
原本的多少加總 0.7305247341029446
轉化過後的維度: (60000, 30)
```

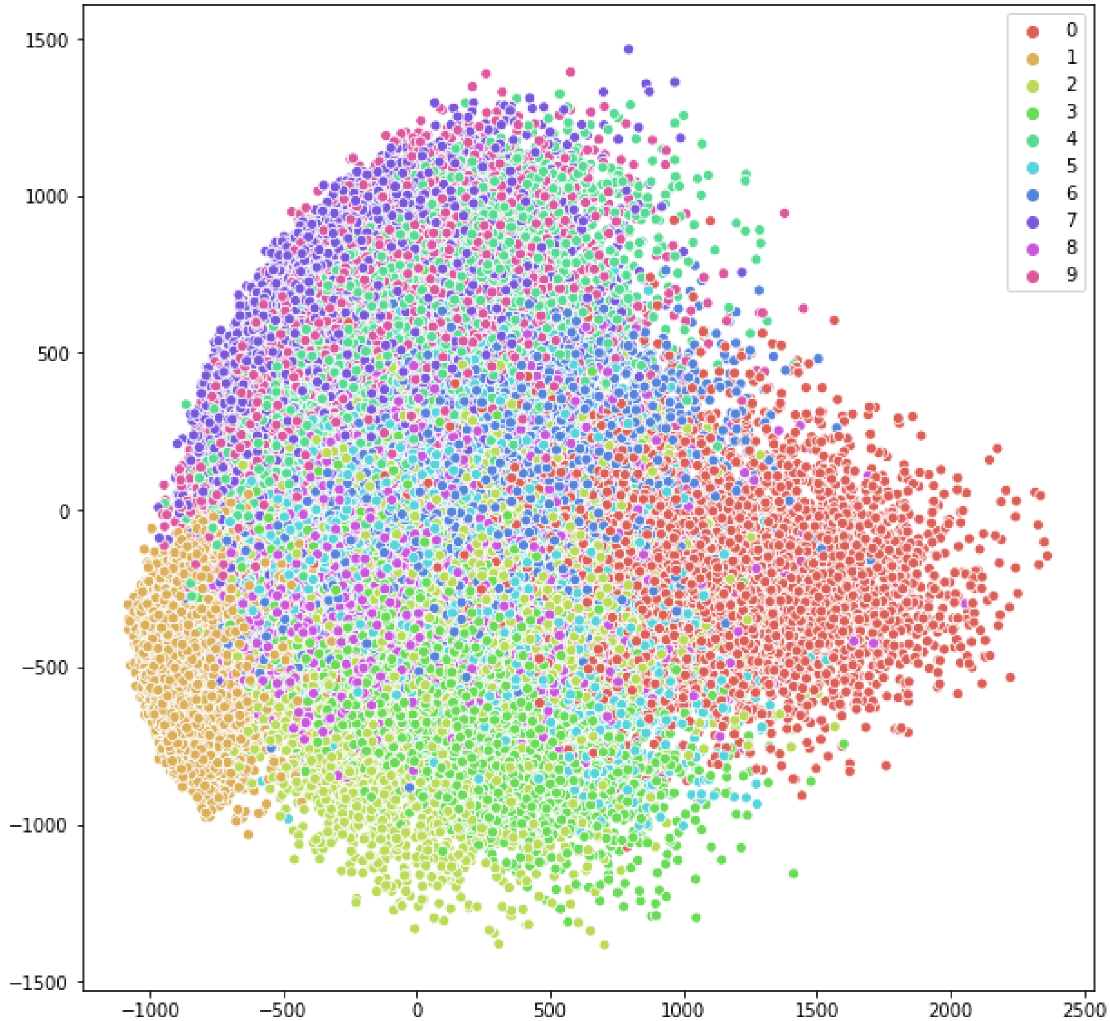
[程式]:
```
embedding = PCA(n_components=30)
embedding.fit_transform(x_shape, y_train)
com = embedding.components_
plt.figure(figsize=(14,10))

h = 5
w = 6
for (index, c) in enumerate(com):
```

```
    plt.subplot(h, w, index + 1)
    plt.imshow(c.reshape(28, 28), cmap="jet")
    plt.title("Axis:" + str(index))
```



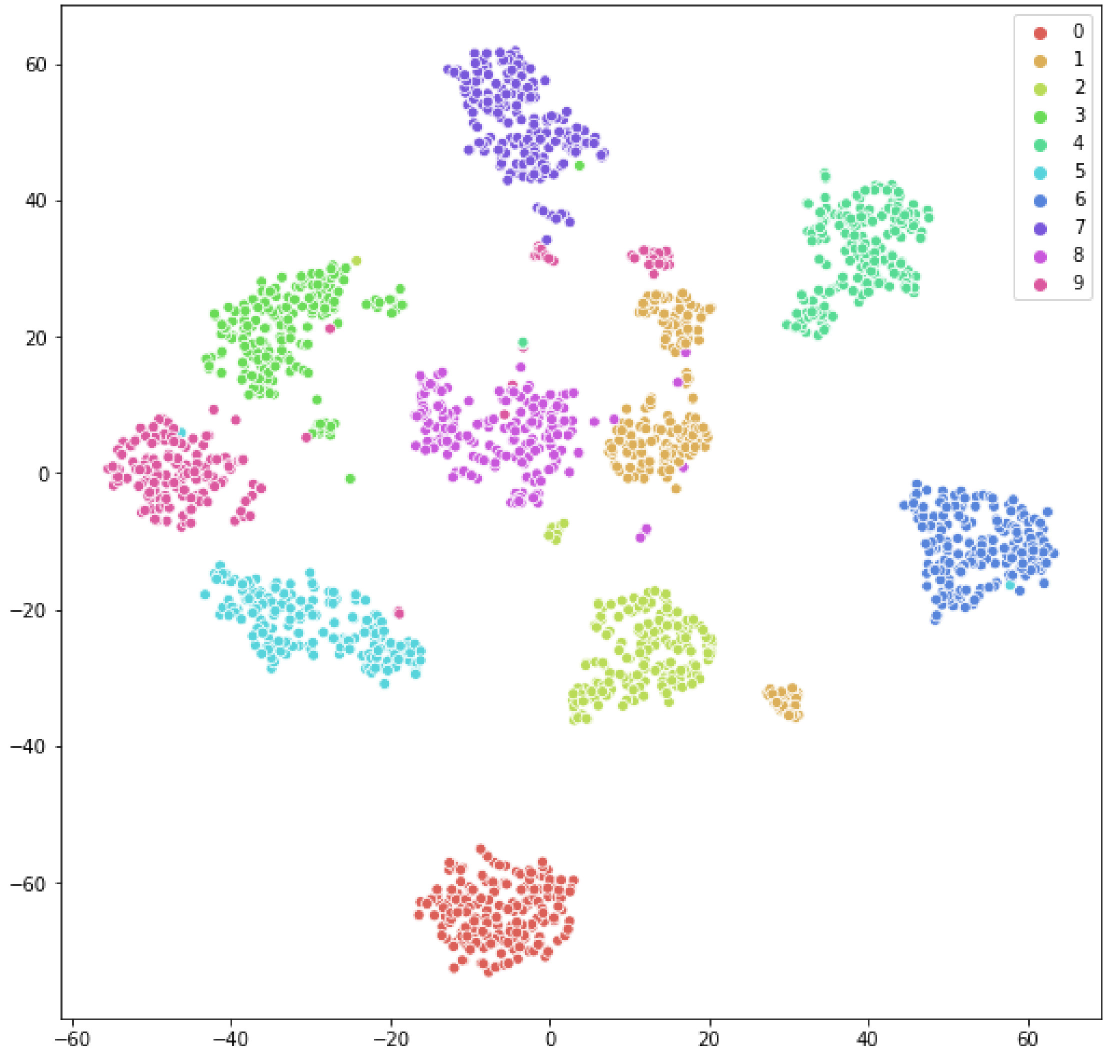
[程式]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

embedding = PCA(n_components=2)
result = embedding.fit_transform(x_shape, y_train)
plt.figure(figsize=(10, 10))
sns.scatterplot(result[:, 0], result[:, 1],
                hue=y_train, palette=sns.color_palette("hls", 10))
```

[輸出]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6991ee5470>
```



[程式]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
%matplotlib inline
import seaborn as sns
from sklearn.manifold import TSNE
digits = load_digits()
embedding = TSNE(n_components=2)
result = embedding.fit_transform(digits["data"], digits["target"])
plt.figure(figsize=(10, 10))
sns.scatterplot(result[:, 0], result[:, 1],
                hue=digits["target"], palette=sns.color_palette("hls", 10))
```

[輸出]: <matplotlib.axes._subplots.AxesSubplot at 0x7f699092d278>


0
1
2
3
4
5
6
7
8
9
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60