ML2018 Spring HW4 TA Hours

MLTAs
ntumlta2018@gmail.com

Outline

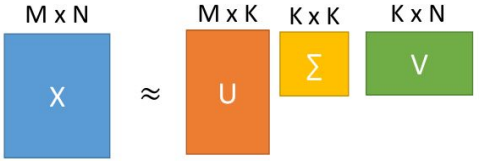
- Unsupervised Learning & Dimension Reduction
  - Principal Components Analysis (PCA) of colored faces
  - Image clustering
  - Ensemble Learning

PCA of colored faces - implementation 1/2

- 用 skimage 讀圖片：
  - from skimage import io
  - img = io.imread(file_name)
- 把一張圖片當作一個 vector：
  - 所以一張圖片相當於有 600 x 600 x 3 dimension
  - 可以用 numpy 裡面的 flatten 把一張圖片轉成一維
- eigenfaces 可能差一個正負號，但 reconstruction 時沒差
  - 所以 eigenfaces 可能看起來不一樣

PCA of colored faces - implementation 2/2

- 參考老師投影片第 24 頁(右圖):
  - M = 每張圖片的維度
  - N = 圖片數量
  - U, s, V = np.linalg.svd(X - X_mean, full_matrices=False)
  - U 的行就是 eigenfaces



K columns of U: a set of orthonormal eigen vectors corresponding to the K largest eigenvalues of XX^T

This is the solution of PCA

- 用 k 個 eigenfaces 重建 y (Mx1) 這張圖片:
  - 先減X_mean
  - 把 y 和 U 的前 k 個 column 做內積，得到 k 個 weight
  - 依據 weight 把 U 的前 k 個 column 加起來
  - 最後記得加回 X_mean
  - 再做下一頁投影片中的轉換

PCA of colored faces - tips

- 如果對原圖大小 (600x600) 下去做，full_matrices=False 時需要 10GB 的 RAM，可以先縮小做，visualize的結果不會差很多。
  - from skimage import transform
  - new_img = transform.resize(old_img, new_shape)
- 取 4 個 eigenfaces 其實很少，可能看不出原圖，因此可以自己試看看用越多 eigenfaces 是否還原的越像，以驗證程式的正確性。
- 最後 reconstruction 記得做轉換：
  - `M -= np.min(M)`
  - `M /= np.max(M)`
  - `M = (M * 255).astype(np.uint8)`

Image clustering - outline

- 用 autoencoder 降維
- 用 Kmeans 聚類

ttps://www.youtube.com/watch?v=Tk5B4seA-AU&list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49&index=25

Image clustering - load image data

```
import numpy as np
train_num = 130000
X = np.load('image.npy')
X = X.astype('float32') / 255.
X = np.reshape(X, (len(X), -1))
x_train = X[:train_num]
x_val = X[train_num:]
x_train.shape, x_val.shape
```

- 把 image pixel value 轉換到 [0, 1] 之間
- 切部分 data 當作 validation

Image clustering - build autoencoder

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.optimizers import Adam

input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)

# build encoder
encoder = Model(input=input_img, output=encoded)

# build autoencoder
adam = Adam(lr=5e-4)
autoencoder = Model(input=input_img, output=decoded)
autoencoder.compile(optimizer=adam, loss='mse')
autoencoder.summary()
```

- 使用 DNN autoencoder
- encoder 三層
- decoder 三層
- Adam optimizer
- mean square error

https://blog.keras.io/building-autoencoders-in-keras.html

Image clustering - train autoencoder

- autoencoder 的 input 跟 target 相同

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_val, x_val))
autoencoder.save('autoencoder.h5')
encoder.save('encoder.h5')
```

Image clustering - Kmeans

- 透過 encoder 把原始 image data 降維
- 用 Kmeans 聚類，使用 2 個 cluster

```
from sklearn.cluster import KMeans
encoded_imgs = encoder.predict(X)
encoded_imgs = encoded_imgs.reshape(encoded_imgs.shape[0], -1)
kmeans = KMeans(n_clusters=2, random_state=0).fit(encoded_imgs)
```

Image clustering - predict

- 根據 kmeans 給出的 cluster 來做預測

```
# get test cases
f = pd.read_csv('test_case.csv')
IDs, idx1, idx2 = np.array(f['ID']), np.array(f['image1_index']), np.array(f['image2_index'])

# predict
o = open('prediction.csv', 'w')
o.write("ID,Ans\n")
for idx, i1, i2 in zip(IDs, idx1, idx2):
    p1 = kmeans.labels_[i1]
    p2 = kmeans.labels_[i2]
    if p1 == p2:
        pred = 1  # two images in same cluster
    else:
        pred = 0  # two images not in same cluster
    o.write("{},{}\n".format(idx, pred))
o.close()
```

Image clustering - sample code for visualization

- a simple way to visualize
- 先用把原始資料降維，再用 tsne 投影到 2 維作圖

看有沒有辦法很清楚的分成兩團

```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# load images
imgs = np.load('visualization.npy')  # imgs is a 10000x784 matrix

# do something to reduce dimension
encoded_imgs = dimension_reduction(imgs)  # encoded_imgs should be a 10000xd matrix

# do TSNE
X_embedded = TSNE(n_components=2).fit_transform(encoded_imgs)  # X_embedded should be a 10000x2 matrix

plt.scatter(X_embedded[:5000, 0], X_embedded[:5000, 1], c='b', label='dataset A', s=0.2)
plt.scatter(X_embedded[5000:, 0], X_embedded[5000:, 1], c='r', label='dataset B', s=0.2)
plt.legend()
plt.savefig('tsne.png')
```