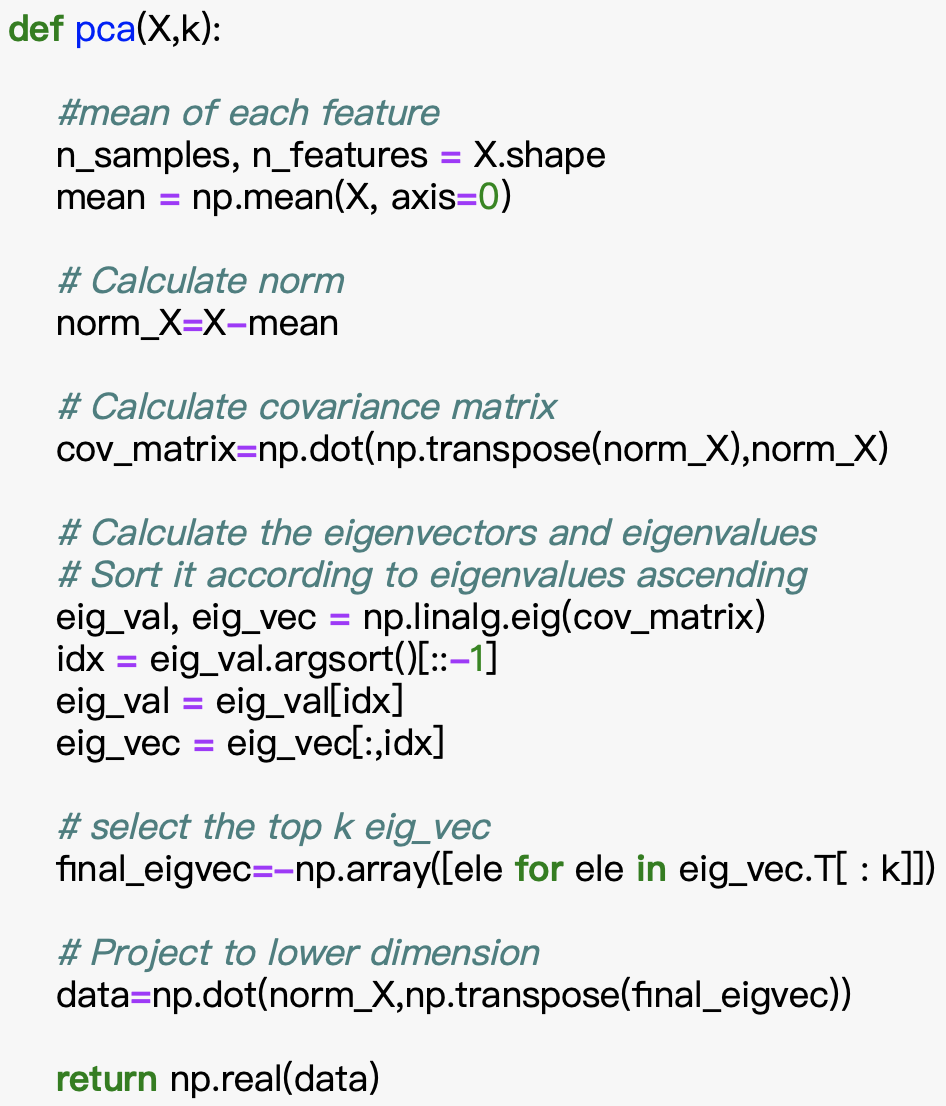
ML HW7 report

資科工碩一

0756138

黃伯凱

1. **PCA**



最後希望降到k dimension

投影到eigenvector

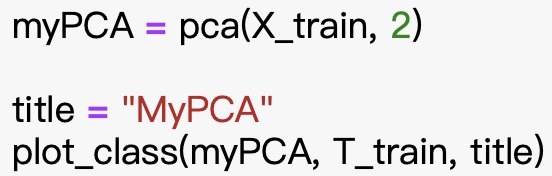
即為降維的動作

選前k大的eigenvector

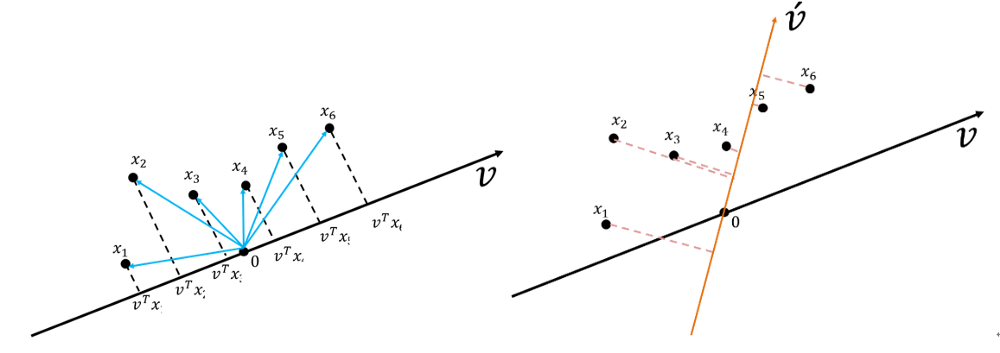
找出covariance matrix的eigenvalue, eigenvector

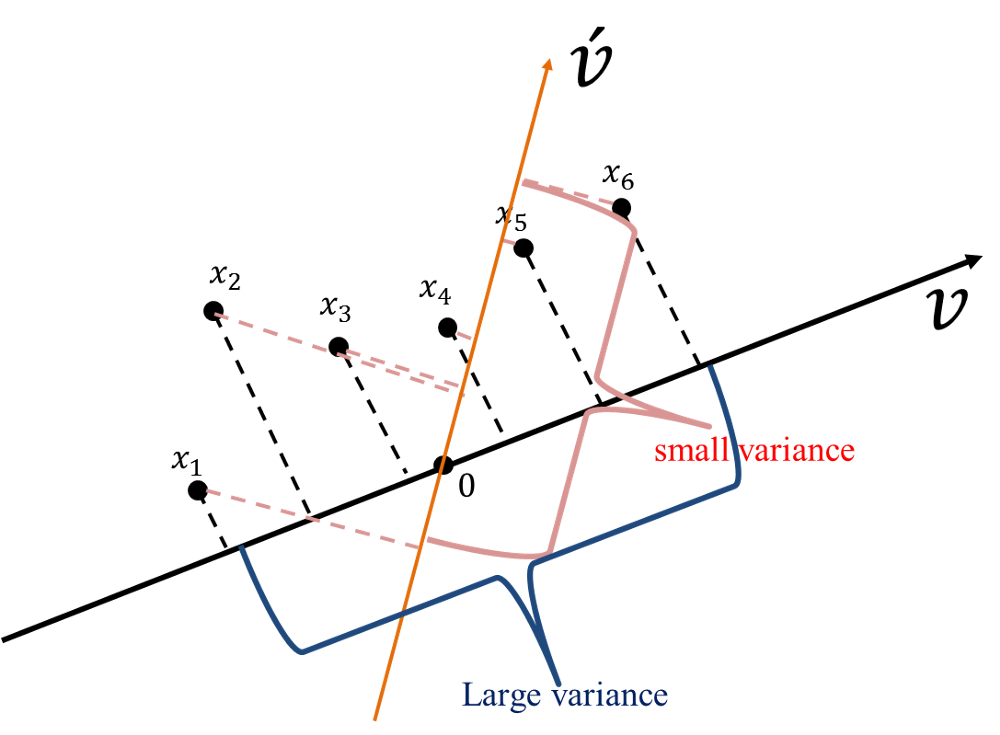
並由大到小排序

算出training data的平均值

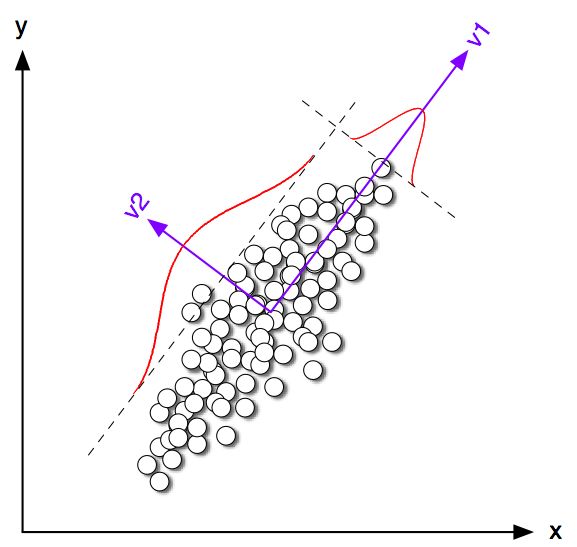


PCA是利用降維來達避免因為高維而無法確實分類的一種方法。他的方法是把data投影到某個vector上，以下有兩個不同的vector投影結果：



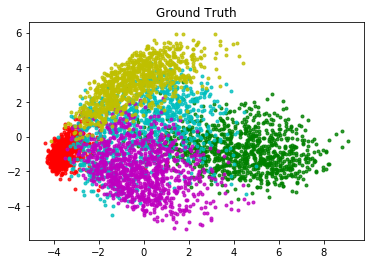
如果把data通通投影到左圖的vector v，可以清楚的分出每一個data之間的差異。但是如果把data都投影到右圖的vector v’，有些data反而會靠得很近，甚至難以辨認。所以希望找到不改變相對距離的向量，而這種向量其實就是data的eigenvector。

以二維的data為例，v1和v2都是這個data的eigenvector，而v1、v2分別是eigenvalue較大、較小的eigenvector。下圖則可以很清楚的看出來，如果把data投影到v1，可以把data分的較散，所以區分data的時候比較好分辨。但如果把data投影到v2，則data都會擠在一起，最後難以分辨。



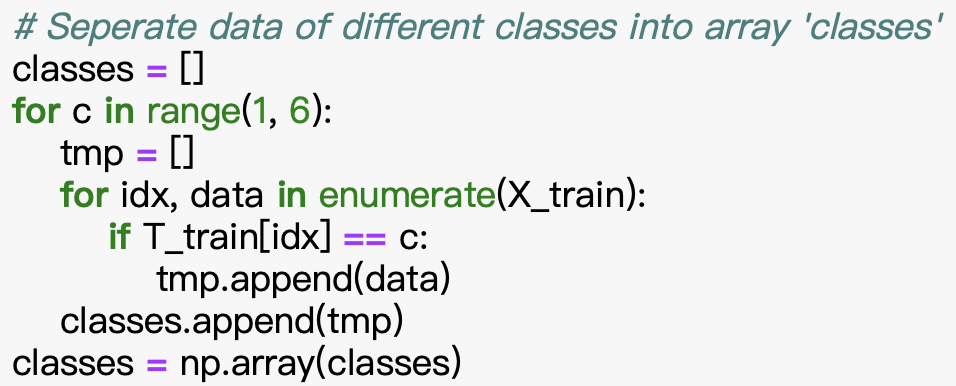
因此PCA要先算出data的covariance，利用covariance算出eigenvalue, eigenvector之後，再把data投影到eigenvector上。

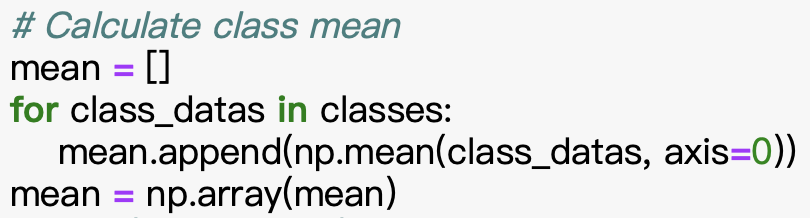
以下是PCA的結果：



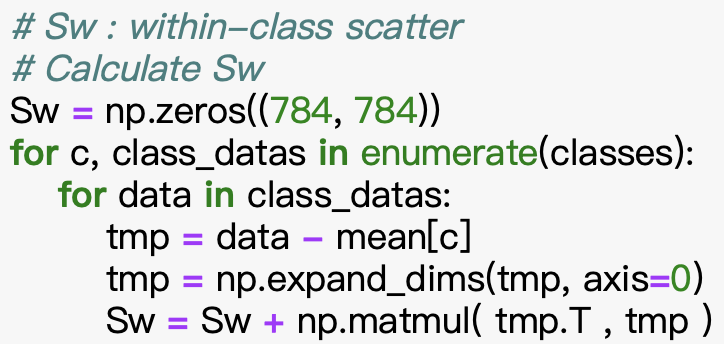
這邊的ground truth是用sklearn的function畫出來的，用來比對自己寫的PCA有沒有做錯。我自己寫的PCA，eigenvector最後有乘上-1才會跟sklearn算的一模一樣。不過其實這一步是沒有關係的，因為eigenvector乘上一個負號指示方向相反而已，對結果並不會有影響。

1. **LDA**

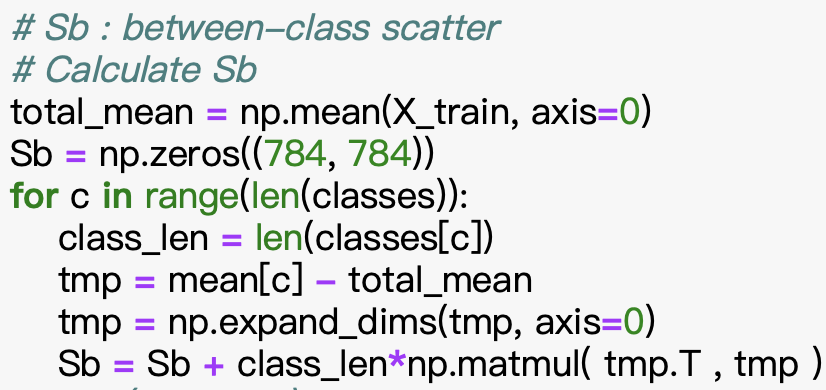
這邊initial了一個classes的list，相同label的training data都先分到同一個tmp list裡，這些tmp即為classes的每一個row。



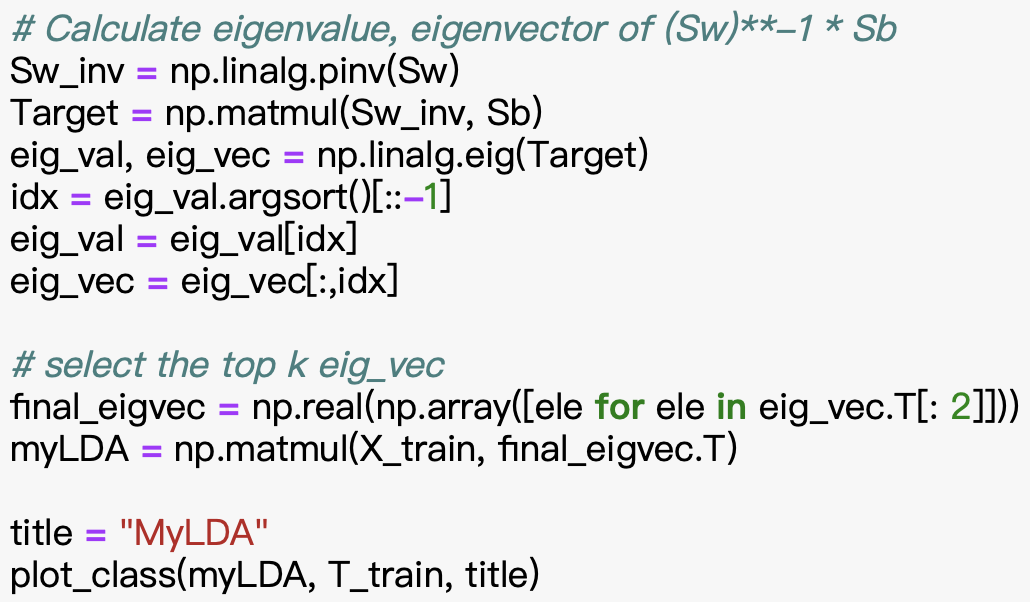
算出每一個class當中每一個pixel的mean



Sw是within-class scatter，每一張圖片要先減掉相對應class的pixel mean，減完之後要做自相關，再加到Sw裡。

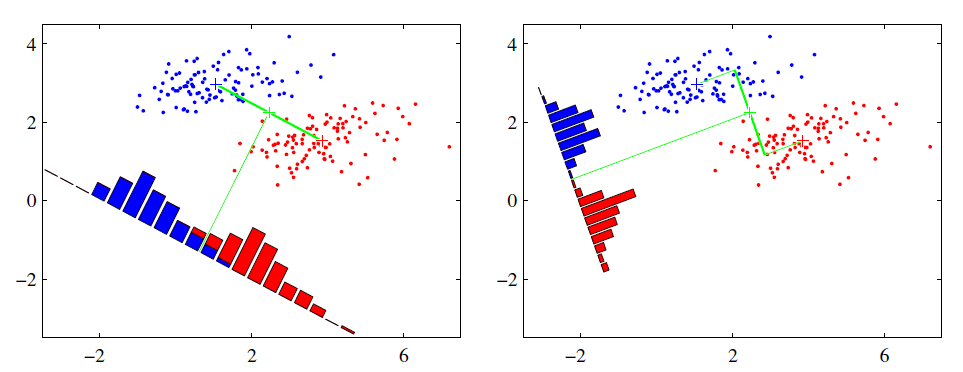


Sb是between-class scatter，在算Sb之前要先算出所有圖片每個pixel的mean。利用每個class的mean減去全部圖片的mean之後做自相關，再加到Sb。

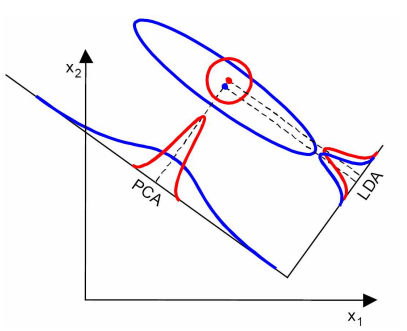


算出(Sw\*\*-1)\*Sb的covariance之後，找到covariance的eigenvalue, eigenvector，再挑出k個eigenvector。最後將data投影到這k個eigenvector上。

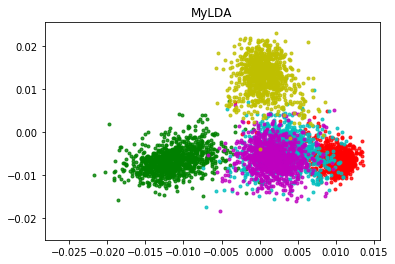
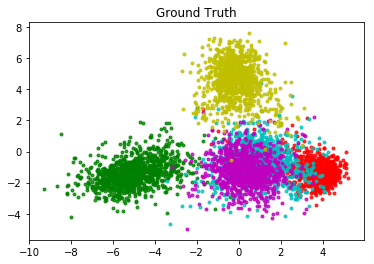
LDA和PCA一樣的地方是在：一樣是將data投影到eigenvector上。但他不一樣的地方是在：他希望**不同class的data**之間的**差異大**、**相同class的data**之間的**差異小**。如下圖所示，左圖不同class之間的差異小，因此在testing時容易分錯。希望可以做到和右圖一樣的結果，不同class才能被清楚分開。



但是當然還是會有一些例外，如下圖的data。如果用PCA會得到投影到左邊向量的結果，LDA則會得到右邊。LDA在這種情況下幾乎會把2的class都分為同一個class，所以其實哪一種model比較好其實還是要看data的分佈大概是長怎樣。

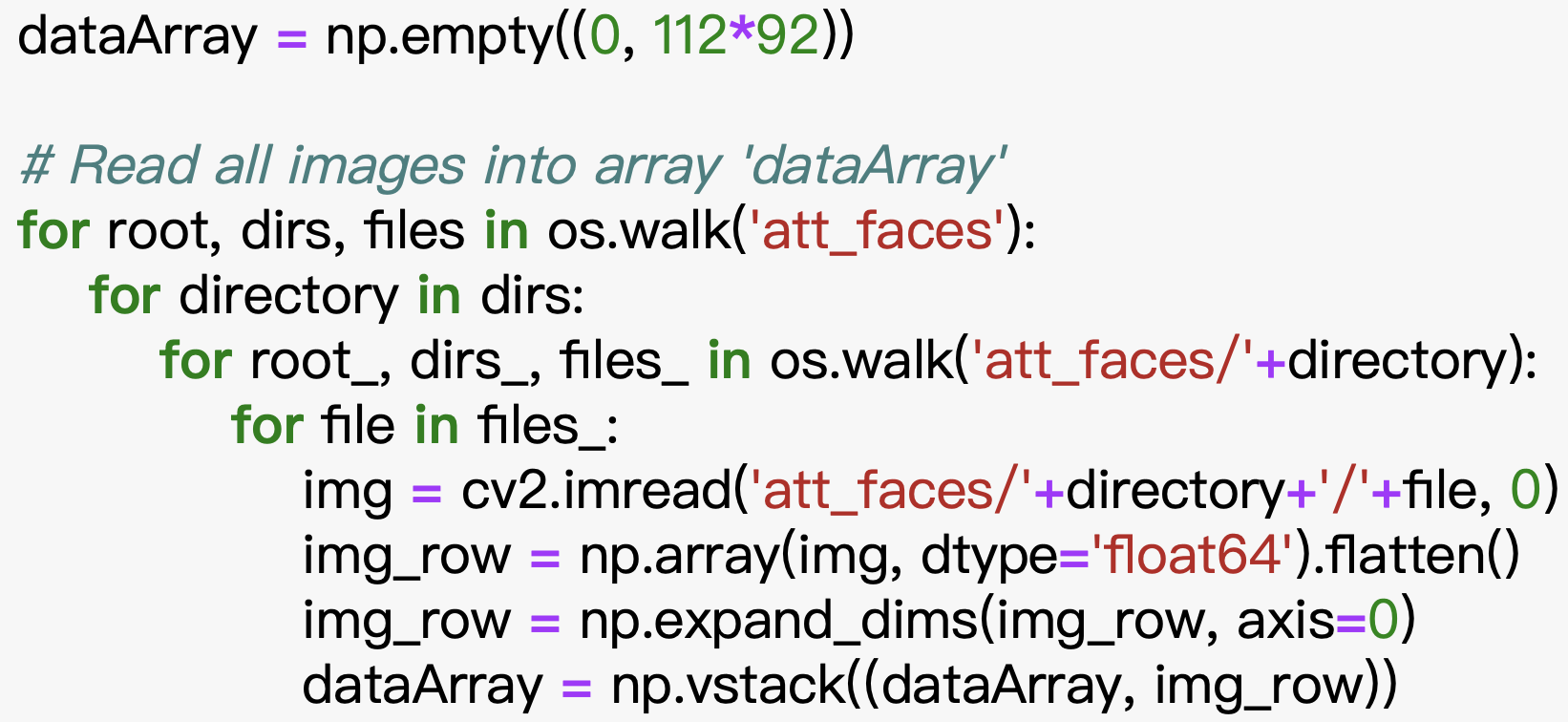


以下是LDA的結果：

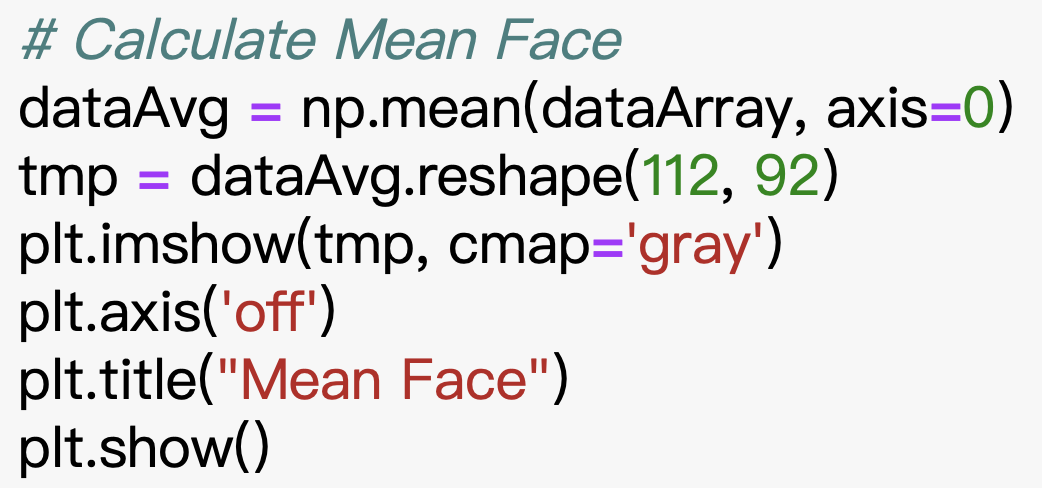


左圖的ground truth一樣是用sklearn算出來的，用來比對自己的答案是否正確。這兩張的分佈大致上一樣，只有scale和sklearn不一樣，不清楚是不是我eigenvector的處理方式和sklearn不一樣，初步猜測或許是sklearn在scale太小的時候會自動把間距拉大以免誤判。

1. **Eigenfaces**

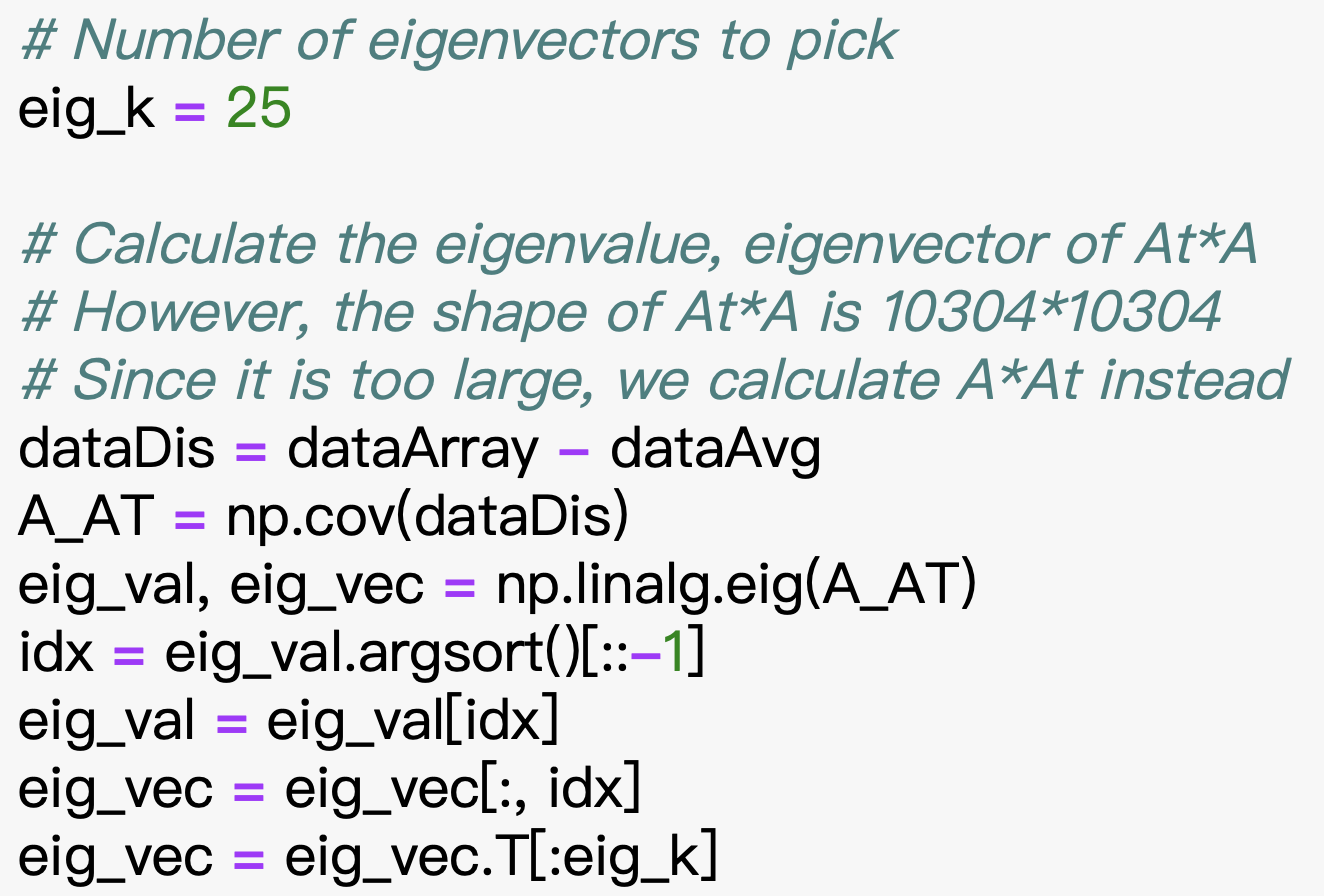


利用os.walk和cv2.imread來把所有資料夾裡的所有圖片都讀進dataArray裡，並將其存成grayscale。我的方法是把每一張照片用一列存在dataArray裡。

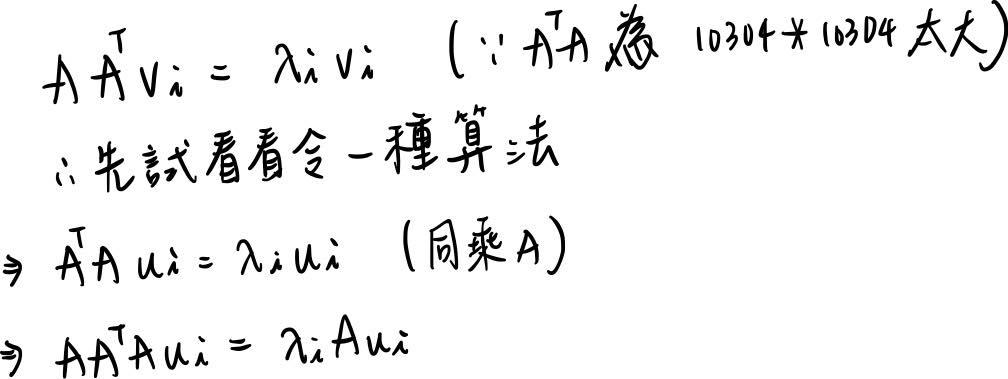




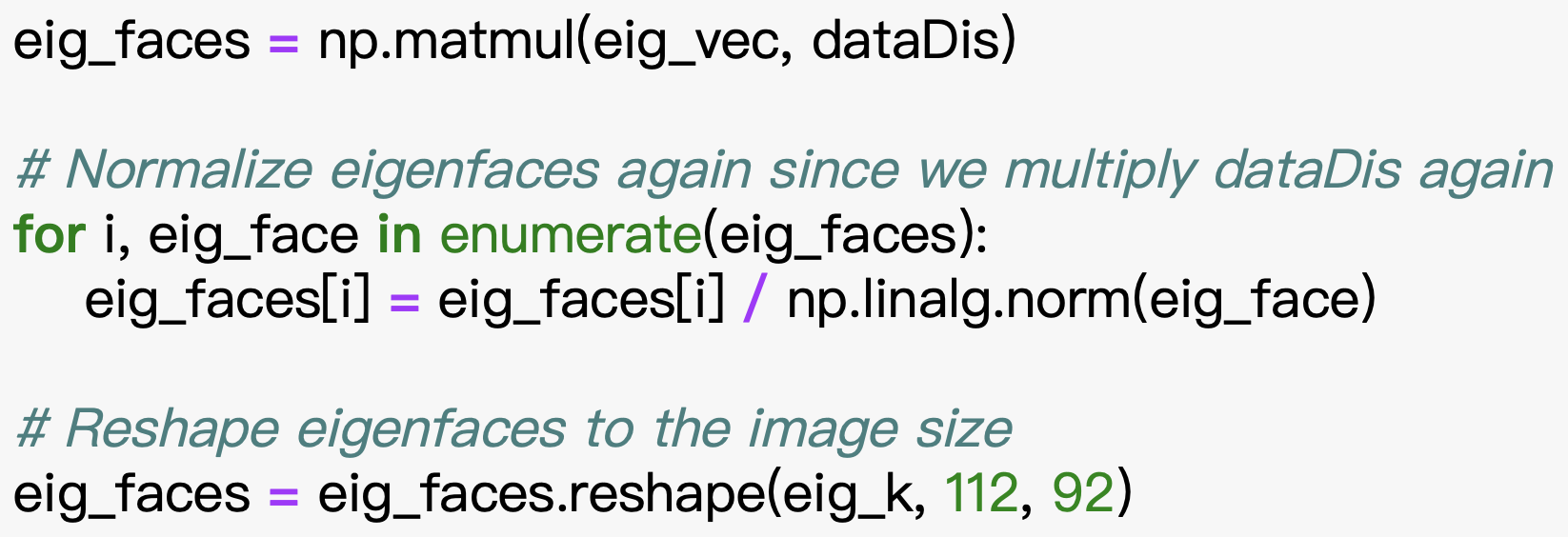
把所有圖片每個pixel的mean算出來，把它reshape成圖片的size。最後再畫出來，就是mean face。



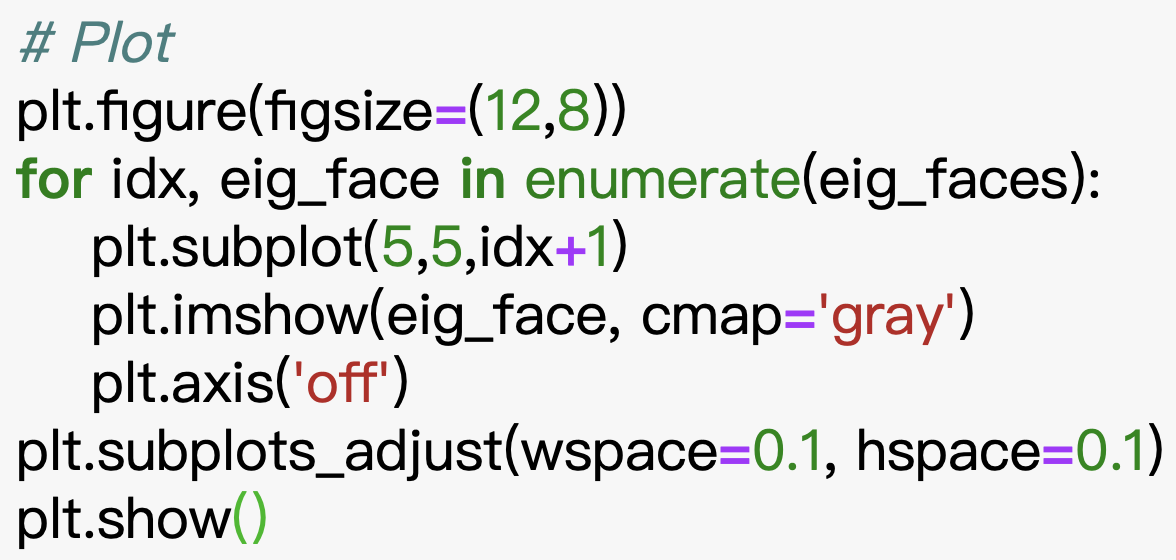
其實eigenface就是利用PCA達成的，所以接下來就是要算dataDis自相關的eigenvalue, eigenvector。但是由於現在dataDis是(400, 10304)，dataDis用A簡寫。如果要算AT\*A的covariance，就是要算(10304, 10304)極大矩陣的eigenvalue, eigenvector。所以有以下這樣的快速算法：

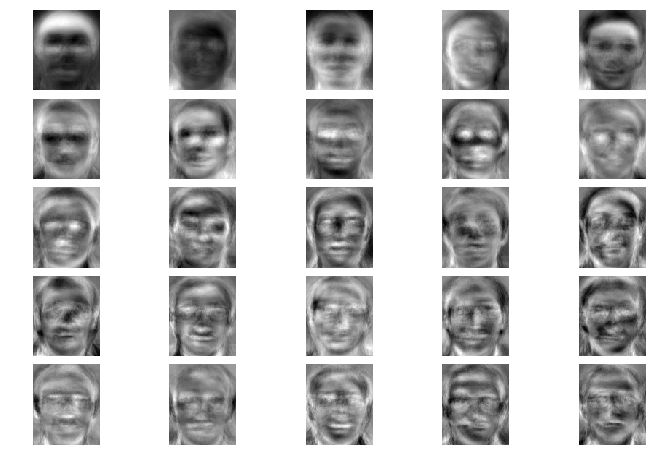


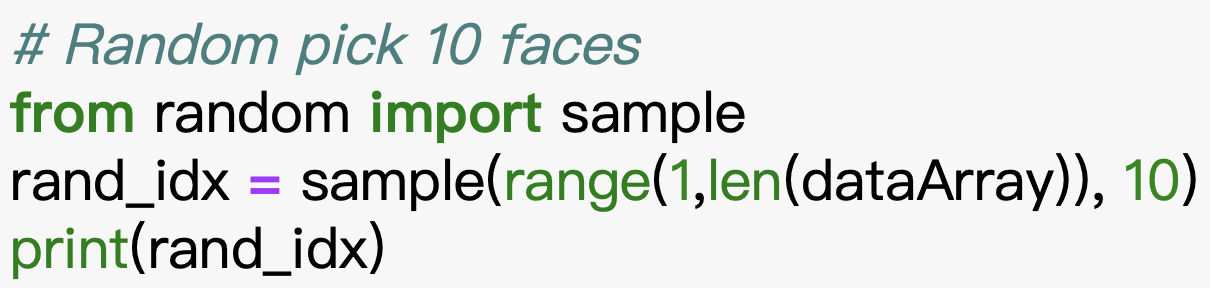
此時發現Au其實就是v，而我們的目的剛好就是v，所以我們只要先算出At\*A的eigenvector後，將其\*A並且別忘記normalize，就可以得到原本的v。



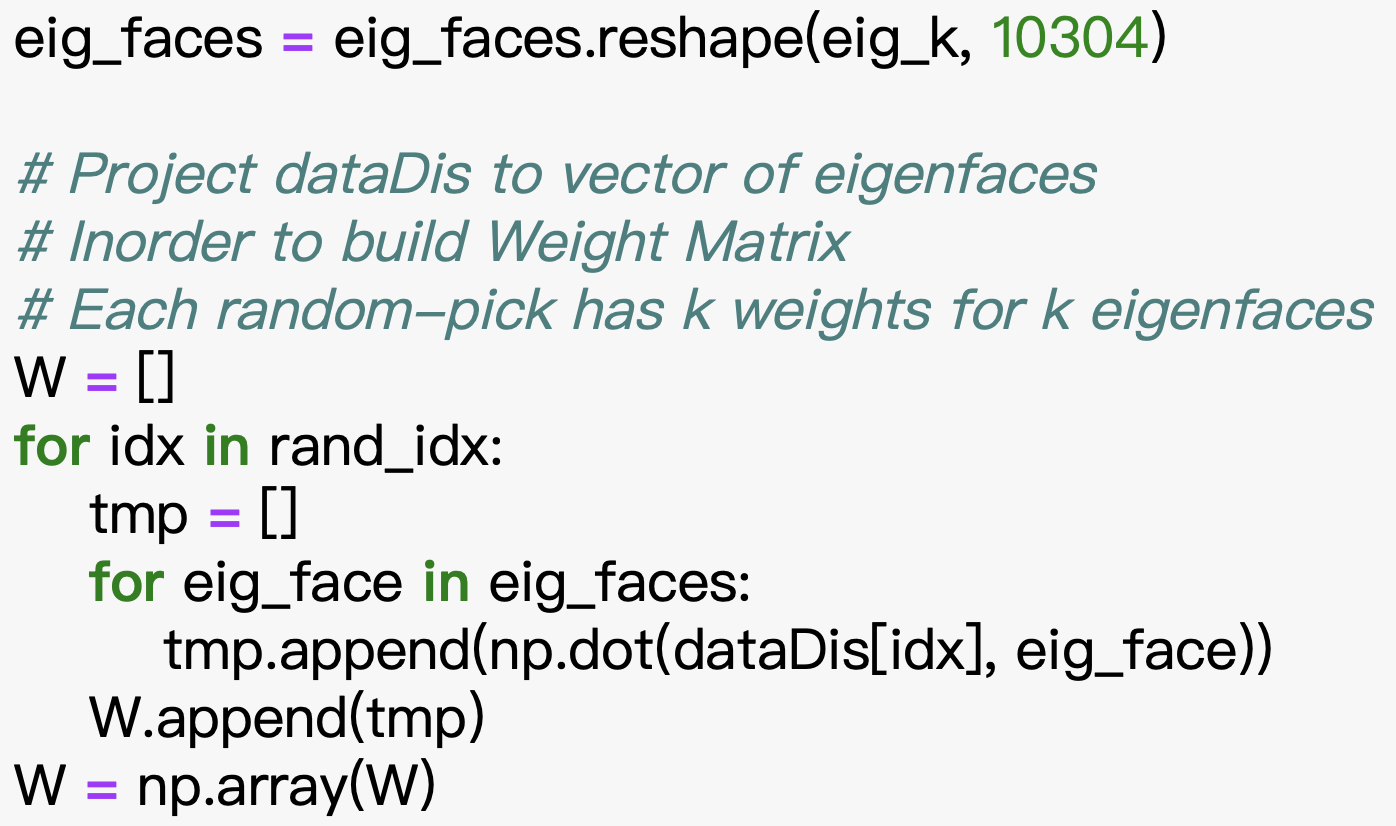
這邊就是上述乘回一個A的程式碼，但因為我是把圖片存成一列，異於我上面推導的式子，所以是u\*A而非A\*u。



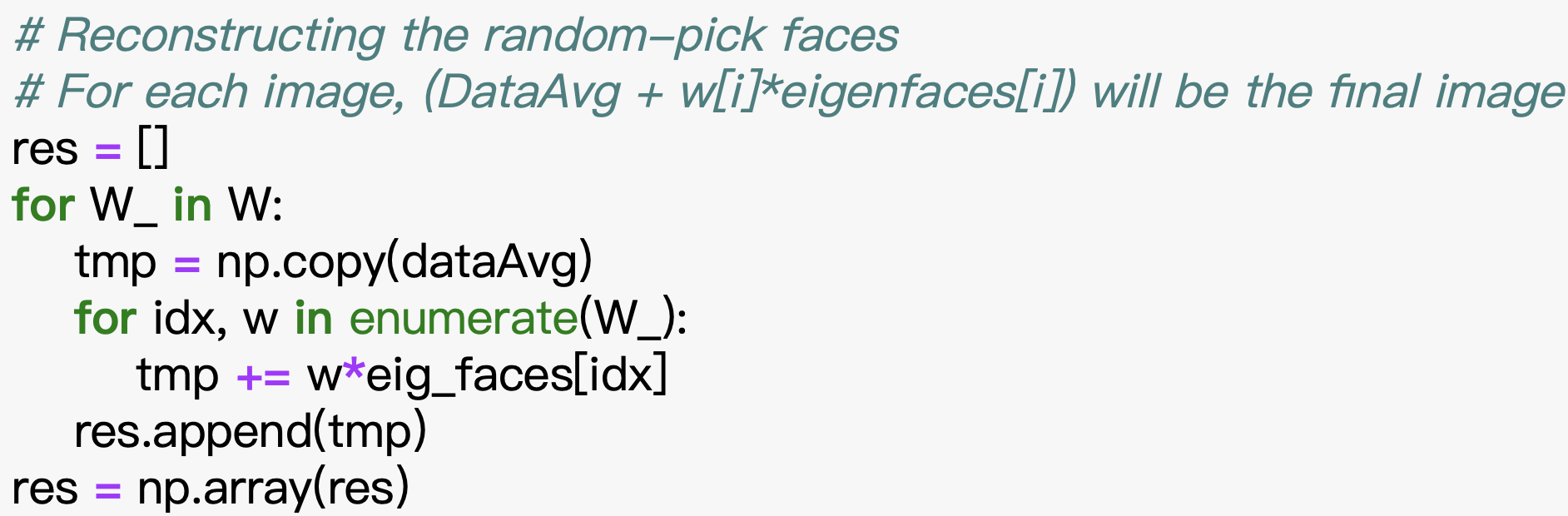
這邊就是把eigenfaces都show出來看看



隨機挑出10張face



把單張face的dataDis投影到eig\_face上（其實就是在做PCA的投影），這樣可以得到每一張圖對每一個eigenface的weight。



把這10張臉的w乘上所有eigen\_face，最後再加上dataAvg，即可得到reconstruct的結果



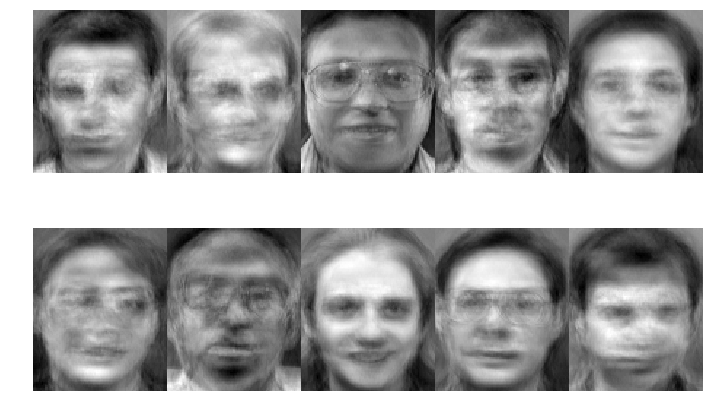
這是挑到的10張臉



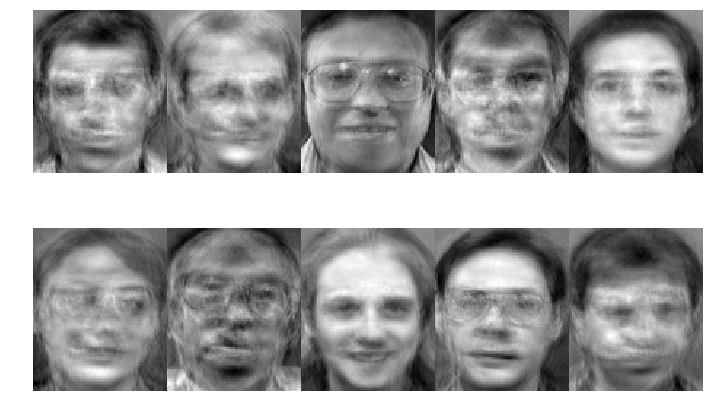
這是我reconstruct (k=25)的結果



這是我reconstruct (k=50)的結果



這是我reconstruct (k=70)的結果



這是我reconstruct (k=100)的結果

