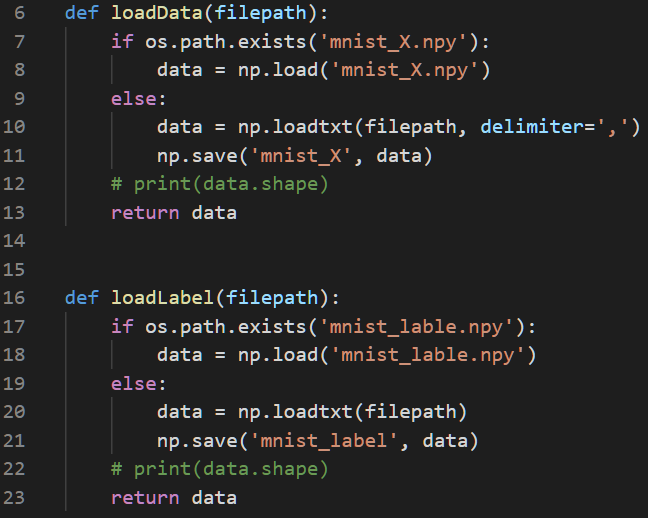
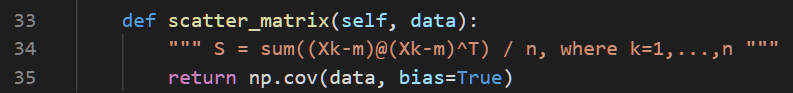
1. **PCA:**
2. Code:
3. Load data points and labels:

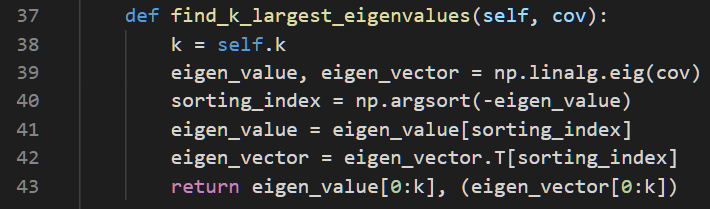


1. Declare a class **PCA** to implement PCA dimension reduction. The class has following methods:

* Use covariance to calculate scatter matrix.



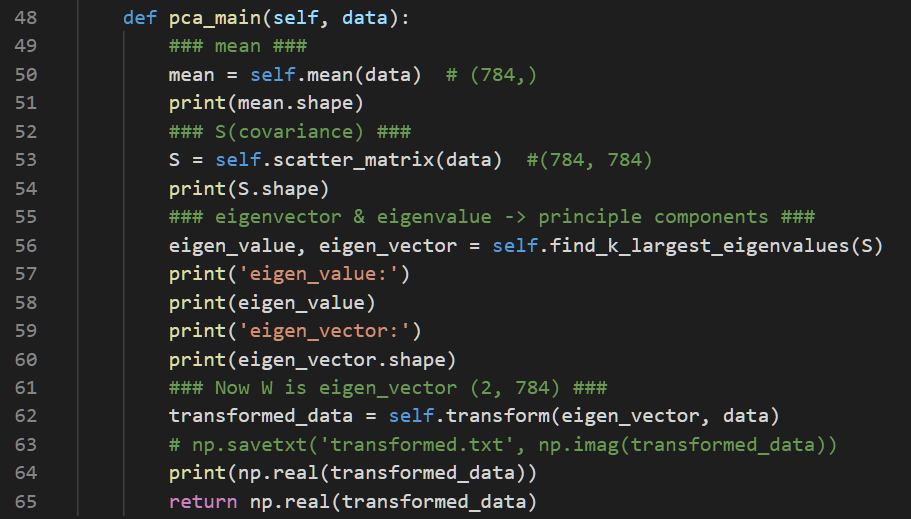
* Calculate eigenvalues and eigenvectors of scatter matrix and get the first k largest eigenvectors. (These k eigenvectors are principle components.)



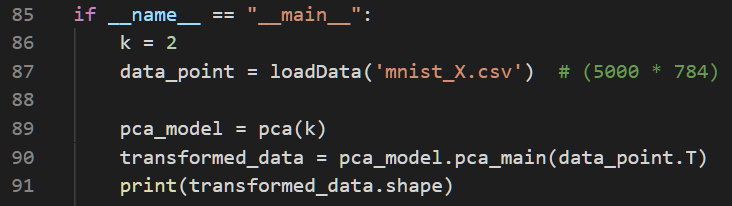
* Use the eigenvectors to be 2-to-784 dimension W matrix. Transform the samples onto the new subspace: y = WTX.



* The main PCA procedure is as following:

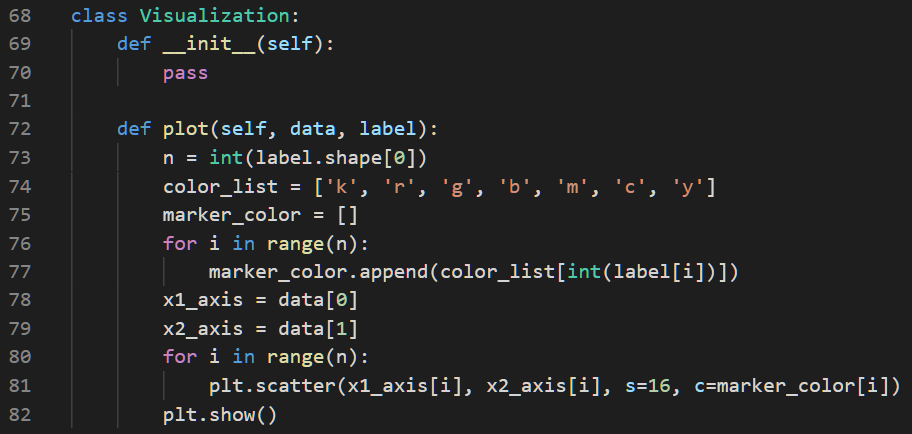


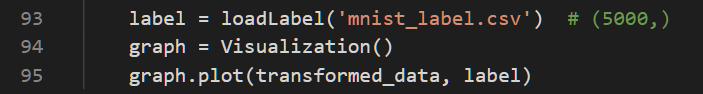
* In main function, use above class as PCA model to implement dimension reduction.



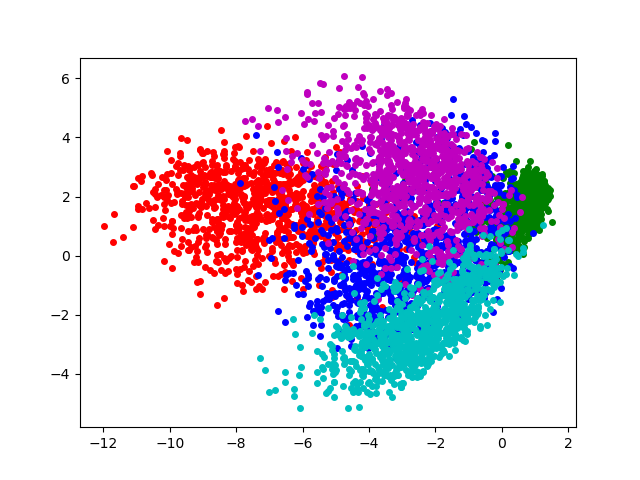
1. Visualization:

* Declare a class **Visualization** to visualize data points on low dimension. For different clusters, give them different colors.





1. Results:

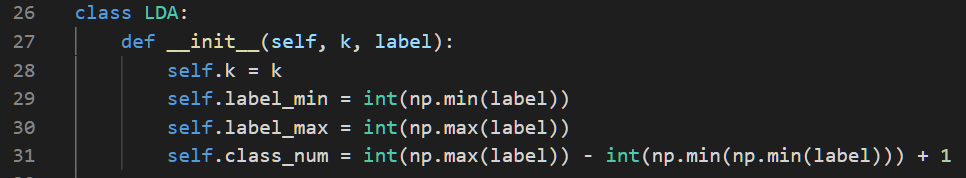


1. Discussion:

The result of PCA indicates those five different clusters in 2-dimension, but each cluster overlaps a lot and cannot be separate easily by linear decision boundary.

1. **LDA:**
2. Code:
3. Load data points and labels. (The code is same with PCA’s loading data.)
4. Declare a class **LDA** to implement LDA dimension reduction. The class has following methods and a main method.

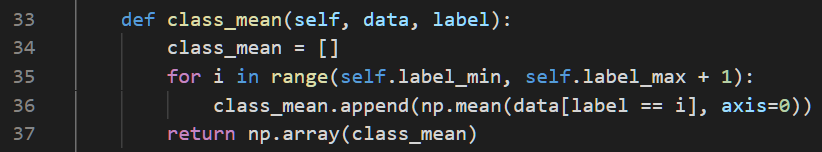
* Initial data: initialize class public variables.



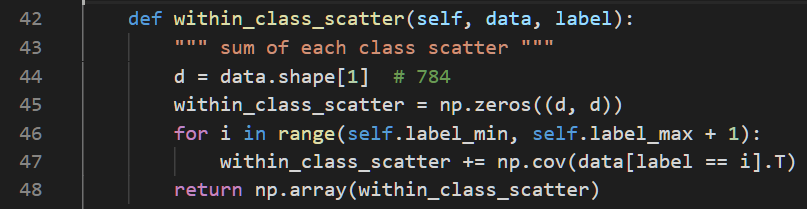
* Calculate overall mean:



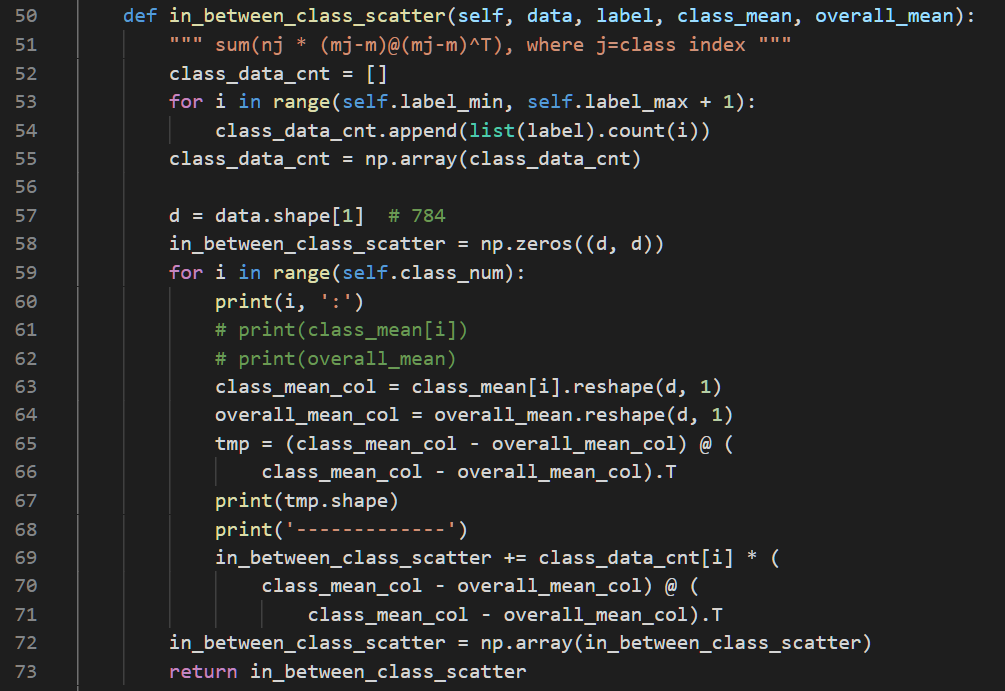
* Calculate class mean:



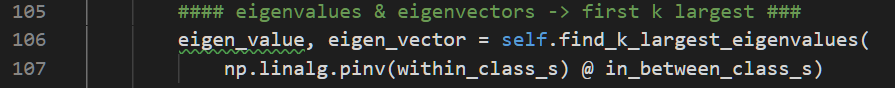
* Calculate within-class scatter matrix Sw:

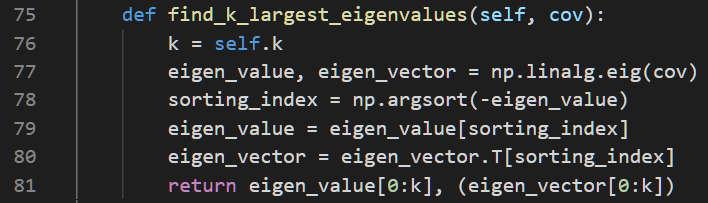


* Calculate in-between-class scatter matrix Sb:



* Calculate eigenvalues and eigenvectors of **Sw-1Sb** and get the first k largest eigenvectors. (These k eigenvectors are principle components.) Note here Sw will become invertible since n<D, so pseudo inverse need to be applied.





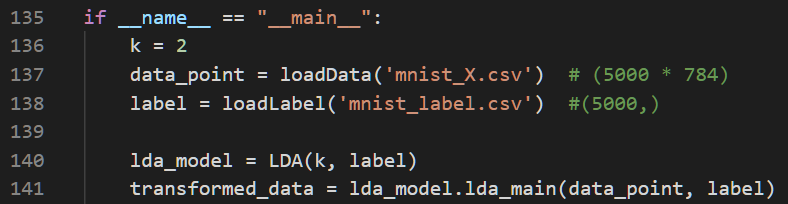
* Use the eigenvectors to be 2-to-784 dimension W matrix. Transform the samples onto the new subspace: y = WTX.



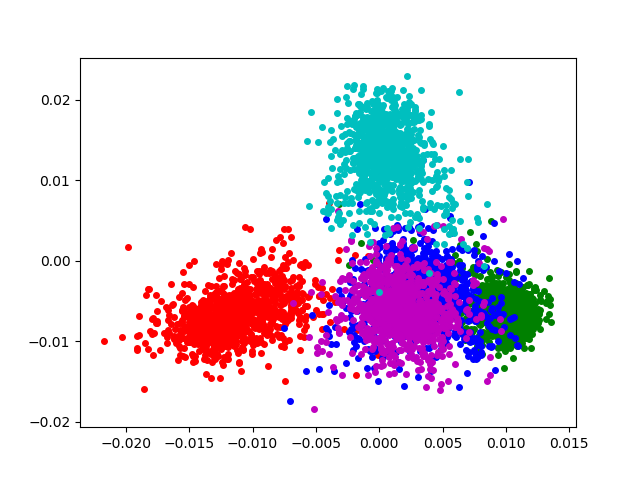
* The main LDA procedure is as following:



* In main function, use above class as LDA model to implement dimension reduction.



1. Visualization: the code is the same as PCA’s visualization code.
2. Results:



1. Discussion:

The main difference between PCA and LDA is that PCA is unsupervised, and LDA is supervised (i.e. PCA doesn’t use cluster labels when doing dimension reduction, but LDA does.)

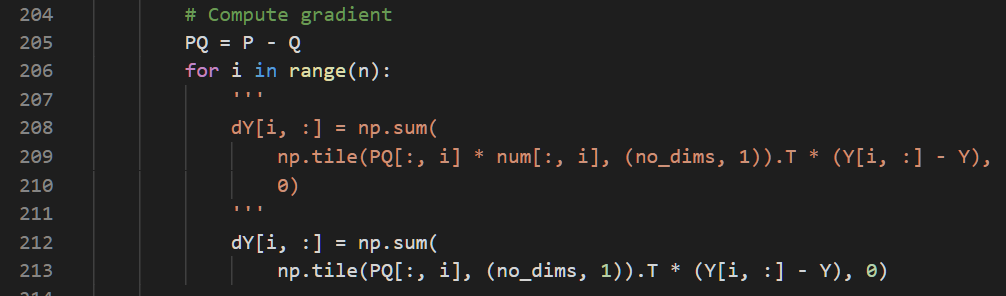
LDA makes data points in same cluster close to each other on low-dimension subspace, and those who are in different cluster are far from each other. In the picture of result, it is easy to find that data points in same cluster are closer than data points in PCA subspace. Also, although there are still some clusters overlapping with each other, but the situation is slighter than that in PCA.

1. **Symmetric SNE and T-SNE:**
2. Code: (iterate 400 times)
3. I have changed three things to implement S-SNE:

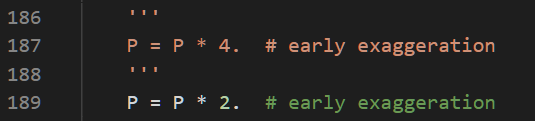
* Pairwise affinities:



* Gradient:

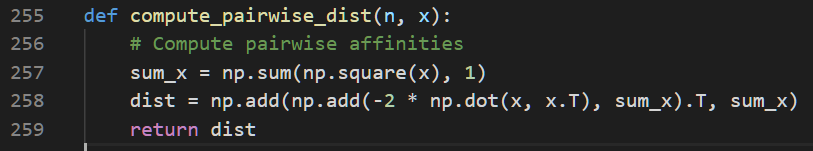


* Early exaggeration:

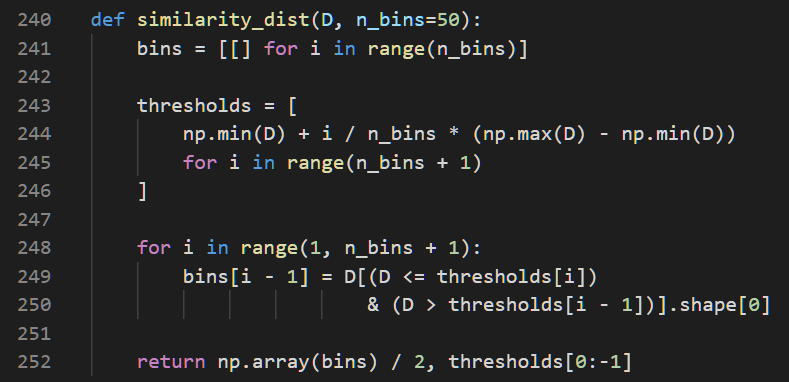


1. Visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space.

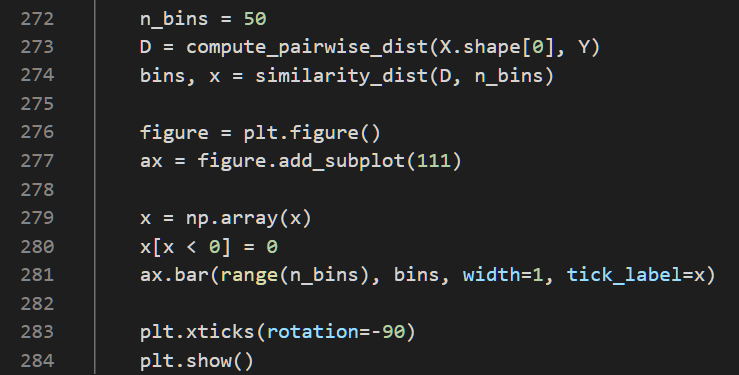
* First of all, compute pairwise similarities (distances) using the same equation for calculating Q.



* Then use these similarities (distances) to make a histogram. In this histogram, each bin counts data within a certain similarity.

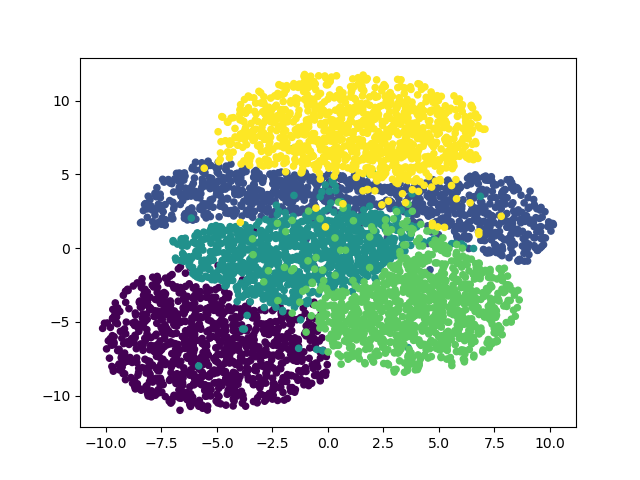


* Finally, visualize this histogram.

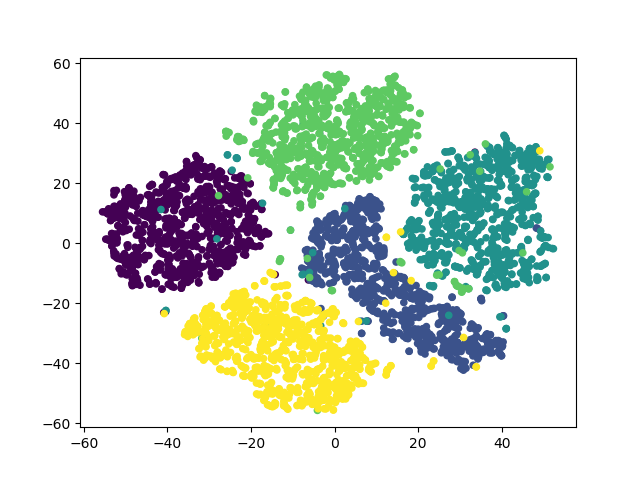


1. Results:

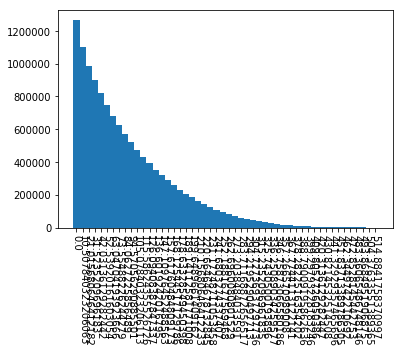
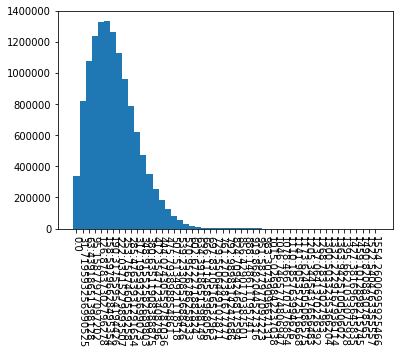
* S-SNE:

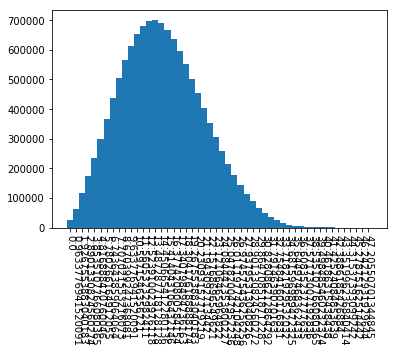
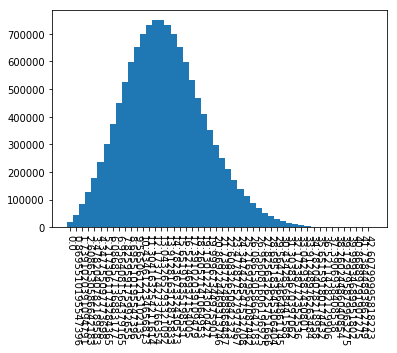


* T-SNE:

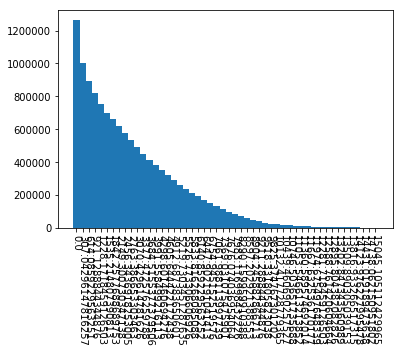
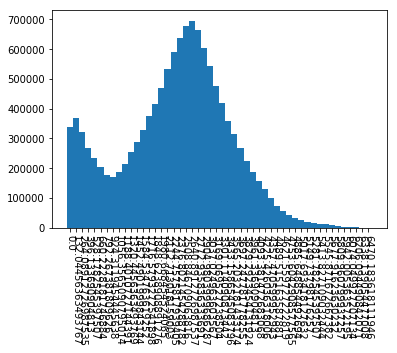


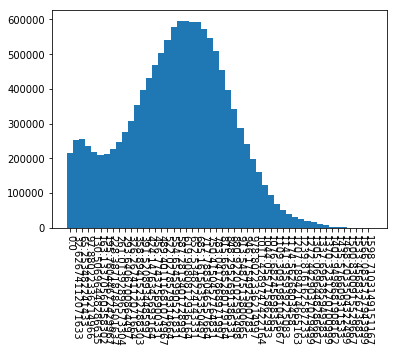
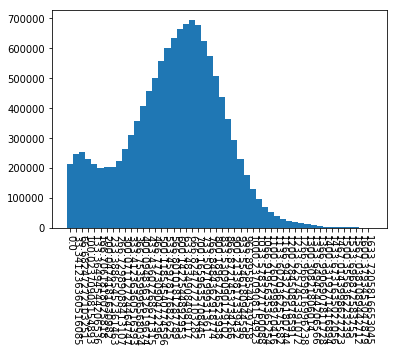
* Pairwise similarities distribution by S-SNE:

(k=2) (k=5)

(k=8) (k=10)

* Pairwise similarities distribution by T-SNE:

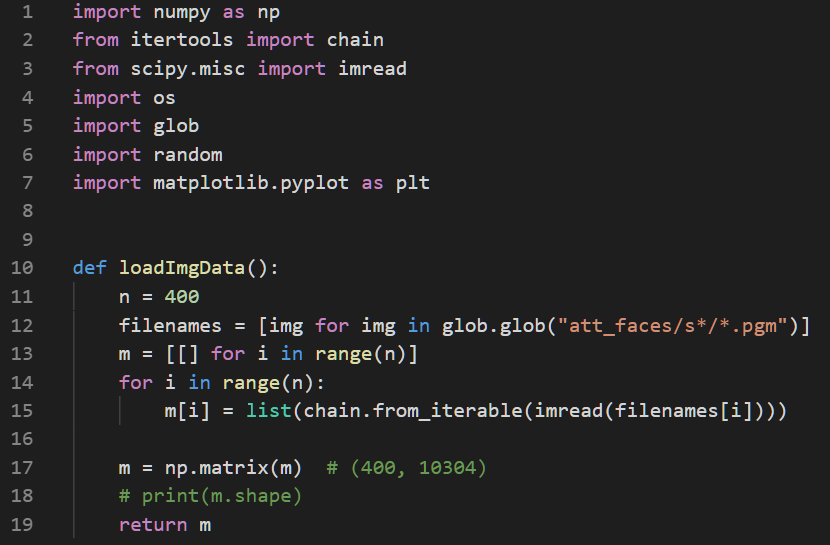
(k=2) (k=5)

(k=8) (k=10)

1. Discussion:

* According to the result figures of T-SNE and S-SNE, overlapping of each cluster data is solved. They preserve the pairwise similarities between hign-D and low-D.
* In S-SNE, the crowded problem can be found easily. In the result figure of S-SNE, the data points are close to each other. According to the figures of pairwise similarities distribution, with k decreasing, most data points concentrate within a certain distance.
* In T-SNE, the crowded problem can be solved.

1. **Eigenface:**
2. Code:
3. Load image data: use **glob** and **imread** (from **scipy.misc**) to read all images in the folders, and then store them in flattened shape into a face matrix.





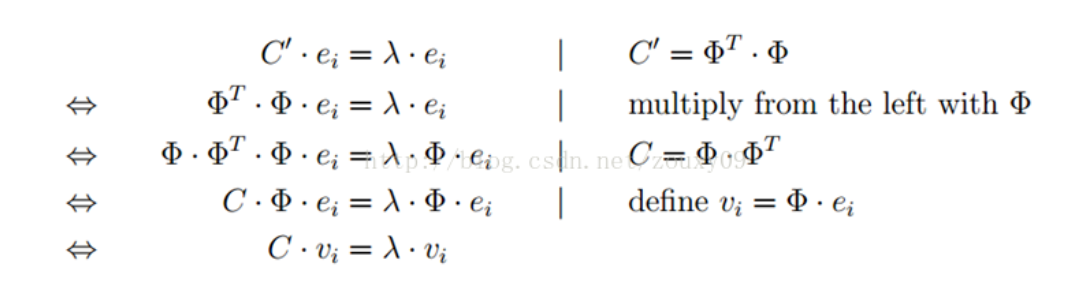
1. Use the PCA class to implement dimension reduction. Since the dimension d\*d(10304 \* 10304) is too large for eigenvector calculation and thus the whole training process is very slow, so here I use a speed-up method:

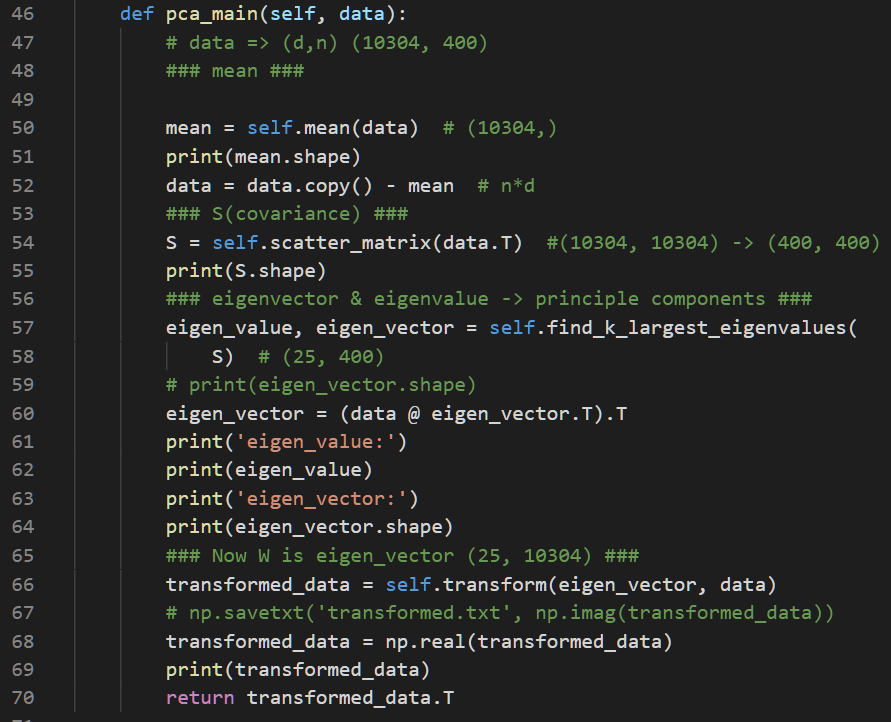
(Reference: [this page](https://blog.csdn.net/zouxy09/article/details/45276053?fbclid=IwAR2aEmp9-7W_8hCVNxzvilG0125W6HXAJH8EYjtukdLfrmxyXNZZ9_SKjrs))

* Calculate difference from data to its mean: data = data – mean.
* Calculate new scatter matrix: C = cov(data) -> C’ = cov(data.T)
* Use C’ to calculate k’s largest eigenvectors: Ei.
* Let original eigenvectors (k’s largest eigenvectors from C) be Vi.

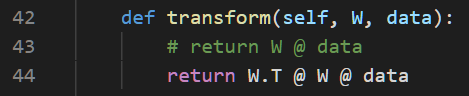
Vi = (data @ Ei).T

Proof is shown below:

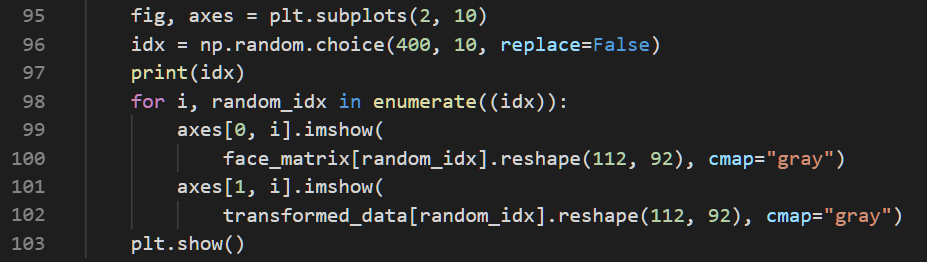




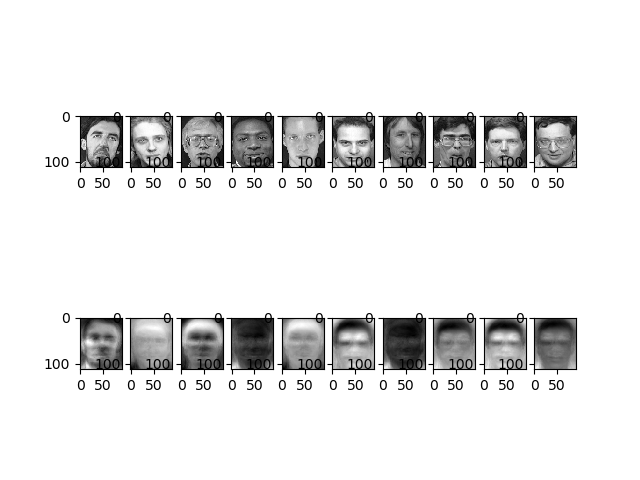
1. Transform data into new subspace: y = WTWx



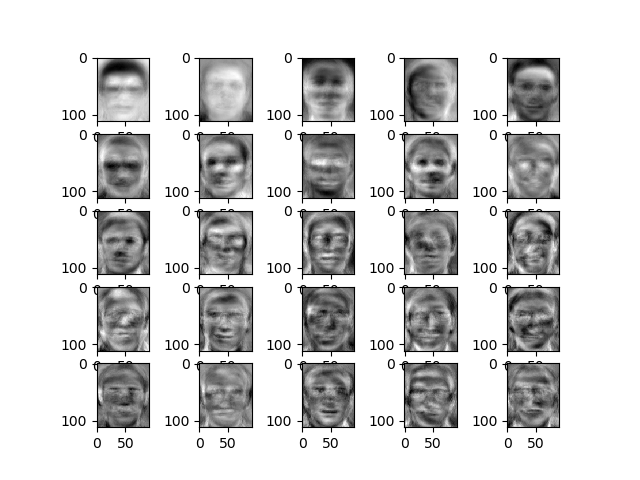
1. Visualization: Randomly choose 10 images. Rescale the transformed data into image shape (112\*92) and show it with the original image.



1. Results:
2. Randomly choose 10 images. The results of different trials are as following:



1. The first 25 eigenfaces:



1. Discussion:

When calculating eigenfaces, without the speed-up trick, the training time is really long (more than 10 minutes). After speeding-up, the pca model can thus calculating high-dimension data quickly, which makes it possible to deal with a big amount of faces.

Eigenfaces contains some “messages” of a certain person, which is useful for face recognition. The result of those 10 randomly chosen faces are mostly well-recognized.