309551101郭育麟

ML07

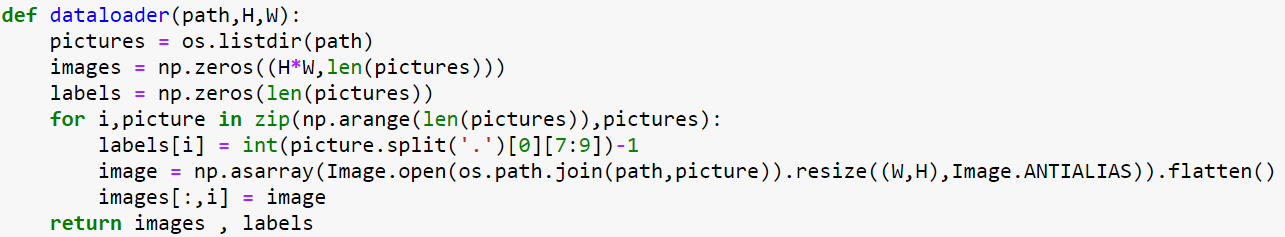
A. Code with detailed explanations

(1) Kernel Eigenfaces

Part1: Use PCA and LDA to show the first 25 eigenfaces and fisherfaces , and randomly pick 10 images to show their reconstruction .

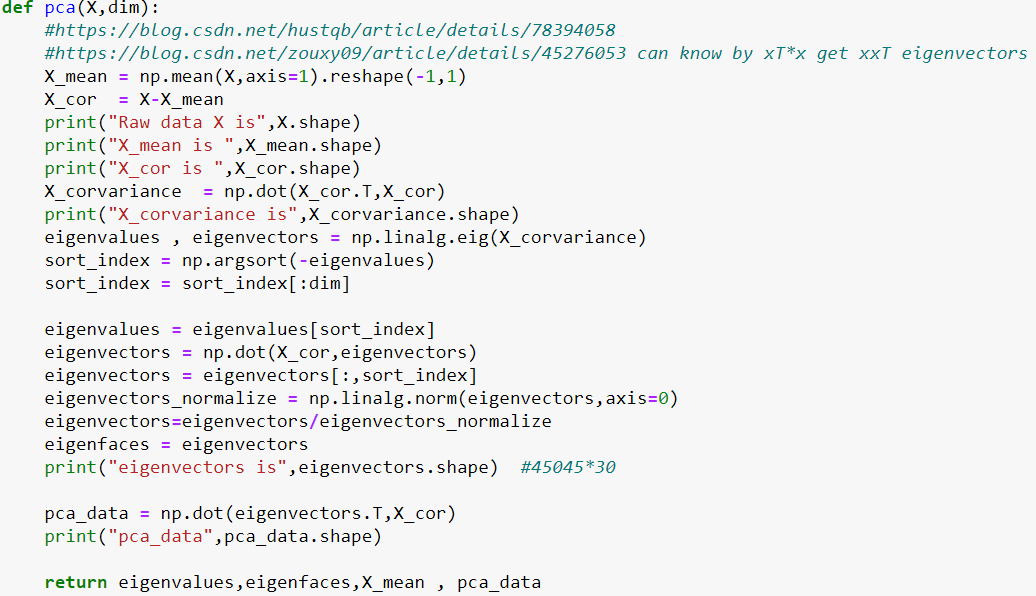
Eigenfaces : Use PCA to show the eigenfaces .

Step1: Dataloader



Given a path , and get every images which sizes is (H,W) and its corresponding label .

Step2 : Doing PCA



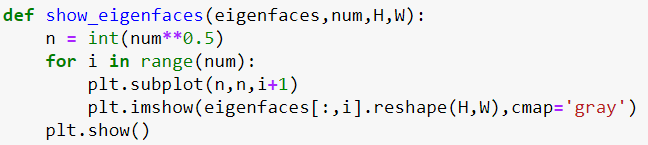
In this step , I use the PCA method , the PCA is the method to data dimensionality reduction and it is unsupervised . First , given a initial whole data “X” , and calculate every dimension mean “X\_mean” to get “X\_cor” , then calculate its covariance matrix to get “X\_corvariance” . We want the actual covariance matrix is “X\*X.T” not “X.T\*X” , but in practice , the actual covariance matrix is N\*N , N represents the data dimension , it could be very large , so when wants to gets it corresponding eigenvalue and eigenvectors are very waste time , so we calculate “X.T\*X” covariance matrix eigenvalue and eigenvectors , this matrix size is m\*m , m represent the number of data , it can significantly reduce the computational burden . Now I want to show the two matrix’s eigenvalue “X\*X.T” and “X.T\*X” are same , and the eigenvectors can calculate by a matrix product .

By the definition of eigenvalue and eigenvectors we can find the eigenvalue and its corresponding eigenvectors s.t

By this proof , we can get C’ and C has the same eigenvalue , and could get C eigenvectors by a matrix X \* C’ eigenvectors . So the “X\_covariance” is “X.T\*X” can help us reduce a lot of computational burden . After get its eigenvalue and eigenvectors , we want the top k eigenvalues and its corresponding eigenvectors , the reason why should choose top k eigenvalues is the covariance matrix can be diagonalize , it means that we could find eigenvalue and eigenvectors s.t

, e represent eigenvectors , D represents diagonal matrix and all the elements are eigenvalues . The eigenvalues can seen as variance , the more scattered the data in a certain feature dimension of the data, the more important the feature normally , so we choose top k eigenvalues and its corresponding eigenvectors to represent the new Principal Component . Finally , calculate the product of chosen eigenvectors and “X\_cor” can get the processed data from PCA method .

step 3 : show eigenfaces



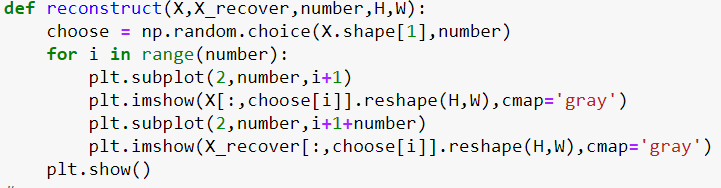
In this step , we want to show 25 eigenfaces , so we create 5\*5 figure and use plt.imshow to plot every eigenface from the previous step’s eigenvectors .

Step 4 : reconstruction

Recall that the data after PCA method are following equations

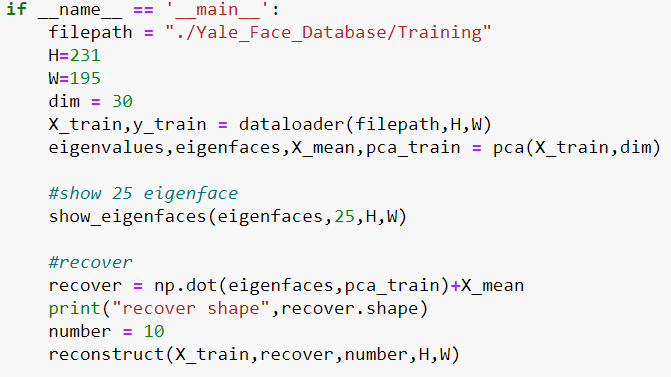
So

After this computation , we could reconstruct the data and to visualize .



There are two matrix , one is original matrix and the other is reconstruct matrix . Then random choose 10 pictures to plot the original picture and reconstruct picture .

Main:



Fisherfaces:

Step1: Dataloader (same as previous)

Step2: Do LDA

The LDA method purpose is similar as PCA , but it is supervised learning .

I will divide LDA into several parts . The main idea in LDA is wishing the data points which belong to same class are projected the closer the better , and the data which not belong to the same class are projected the far the better .

(1) compute the data every class mean vector

(2) By class mean vector , compute .

represent the degree of separation of different class projection data points

represent the degree of separation of same class projection data points

We want the bigger the better and want the smaller the better . So the objective function will be w represent the projection matrix . And by Rayleigh quotient and differential with respect to w

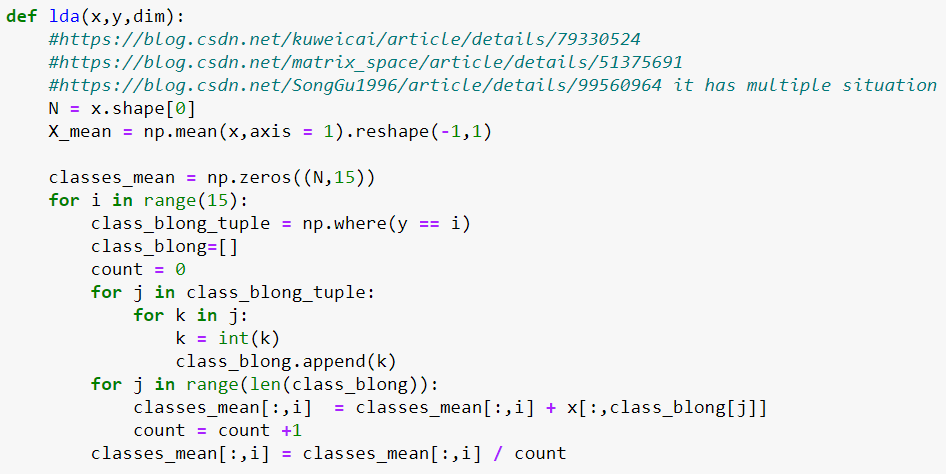
can seen as the definition of eigenvalue and eigenvectors .

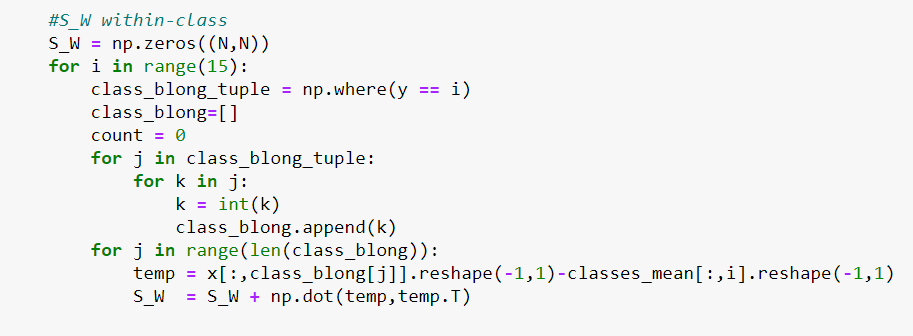
(3) The equation , compute the eigenvalue and eigenvectors

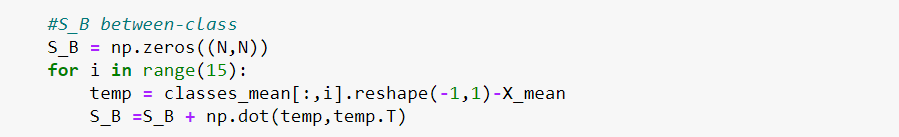
(4) Find top k eigenvalue and its corresponding eigenvectors to generate p projection matrix W

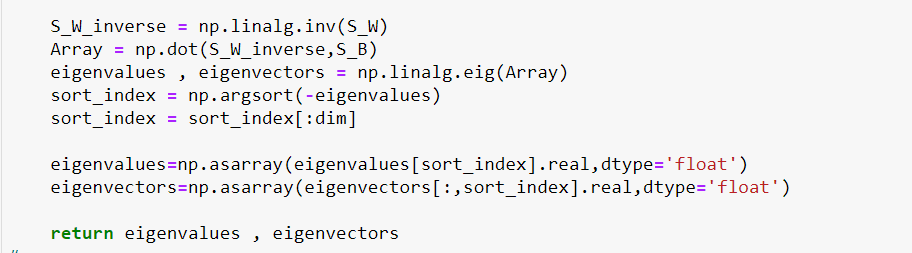
Because we want J(w) the bigger the better , so we wants the top k eigenvalue and its corresponding eigenvectors

(5) Y = WX





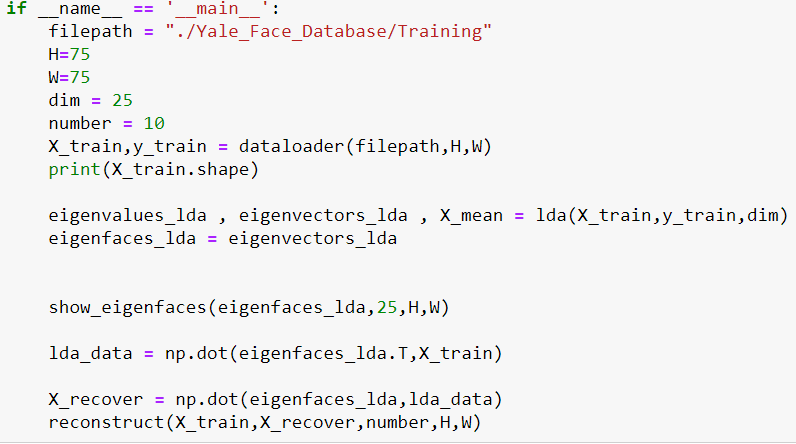




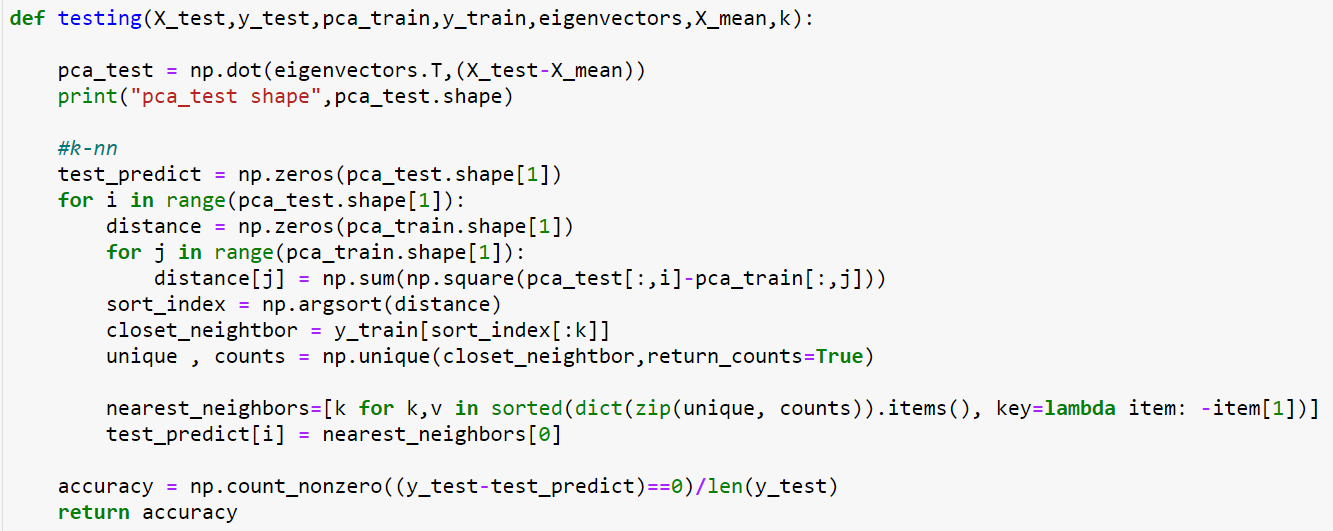
Step 3 show eigenfaces (same as PCA)

Step 4 reconstruction (same as PCA)

Main:

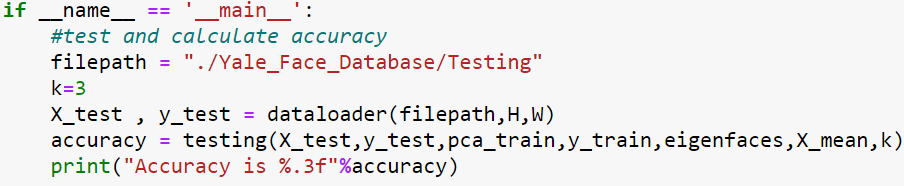


Part2 : Use PCA and LDA to do face recognition, and compute the performance.



In this function , first compute embedding data from pca or lda by using eigenvectors.T \* (data-data\_mean) and use k-nn algorithm to compute the distance between training data embedding feature and testing data embedding feature to find where testing data embedding feature belongs which class and find which occurs the most times in K times . So we can finally get predict result on every testing embedding feature to calculate the accuracy on testing data .

Main:



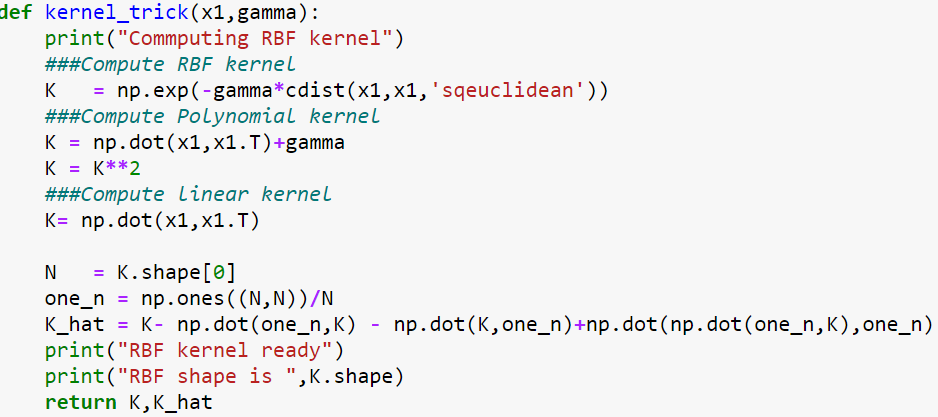
Part3: Use kernel PCA and kernel LDA to do face recognition, and compute the performance

In this part , we use kernel function to represent our covariance matrix before . By doing kernel trick , it can let datapoints projecting into infinite dimension , and use PCA or LDA method to classify in low dimension . The reason why use kernel function is that the kernel function could compute the similarity in high dimension between two vectors .

Eigenfaces : Following the steps

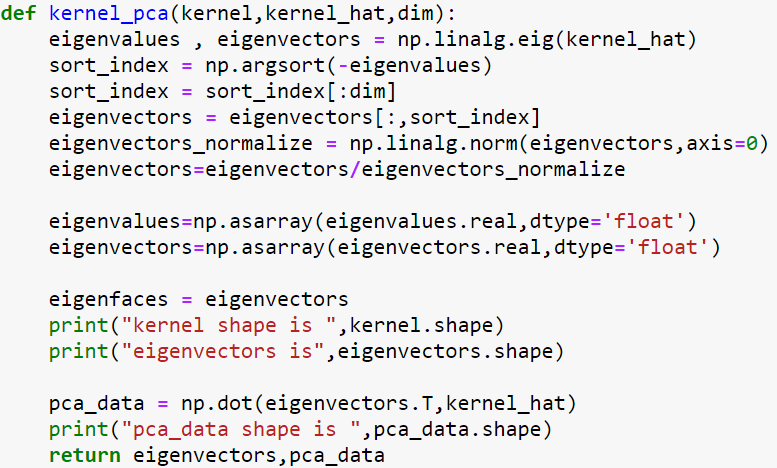
(1) Compute the similarities and centralization





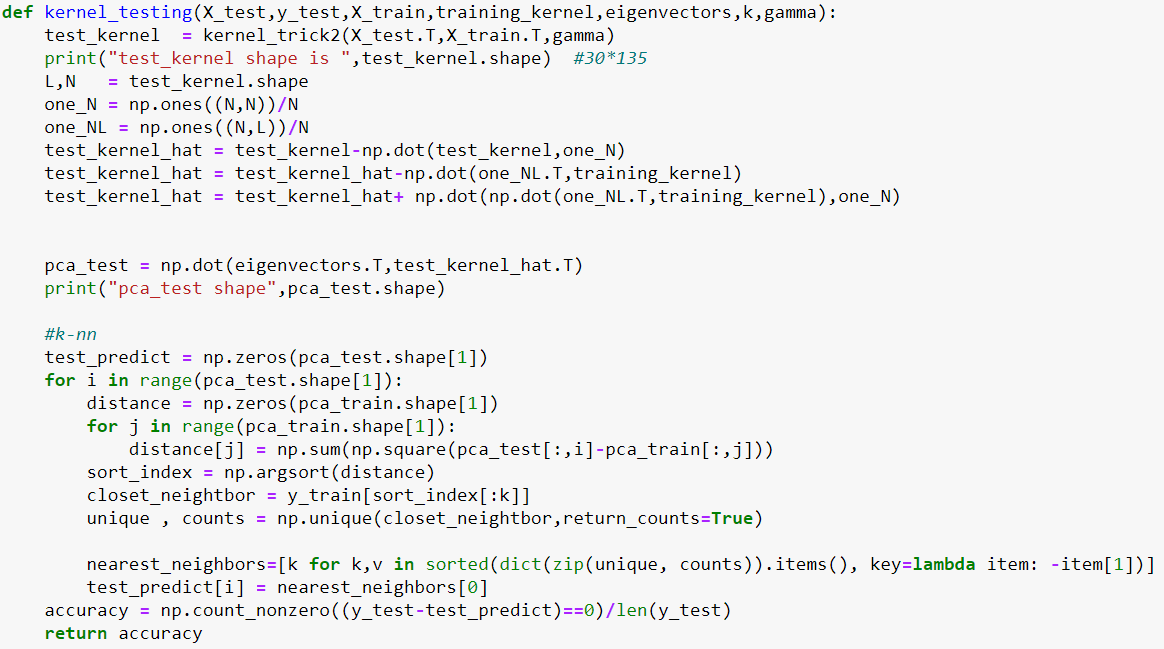
There are some kernel function to use and do the centralization .

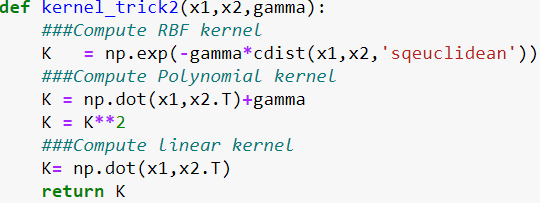
(2) doing PCA method

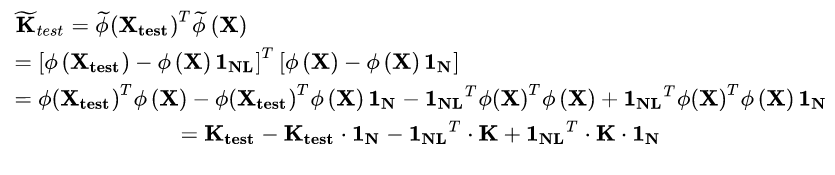


This steps is similar as previous

(3) Testing







Where

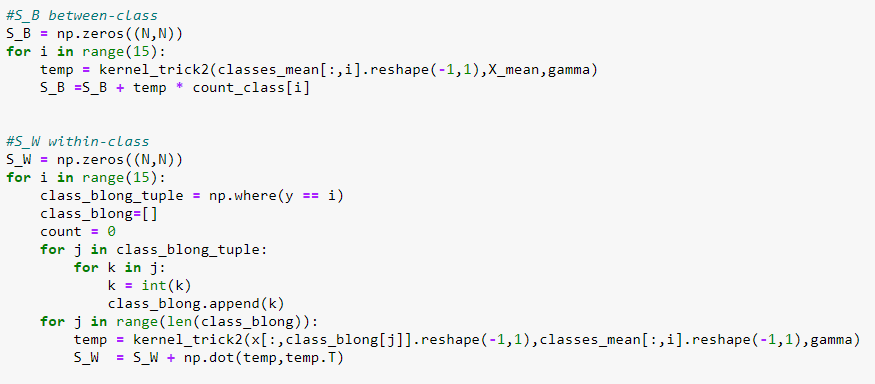
First compute the similarity between training data and testing data by using kernel function , and also do centralization . Finally do KNN algorithm as previous steps to get the predict result .

Main:



Kernel LDA:

The main difference between LDA and LDA is that SB(between class) and SW(within class) matrix are different . Kernel LDA use kernel trick to represent SB and SW matrix . Others idea are same as LDA . I use RBF kernel , linear kernel , Polynomial kernel to compare .

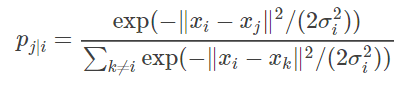


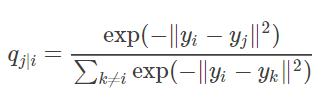
(2) t-SNE

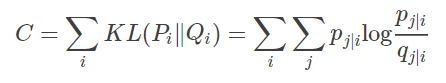
Part1: Try to modify the code a little bit and make it back to symmetric SNE

First I want to talk about SNE , different from PCA , SNE is a non-linear data dimensionality reduction method , it could images data from high dimension to low dimension efficiently , and keep data local structure in high dimension , t-SNE and symmetric SNE are extend of the SNE .

SNE:







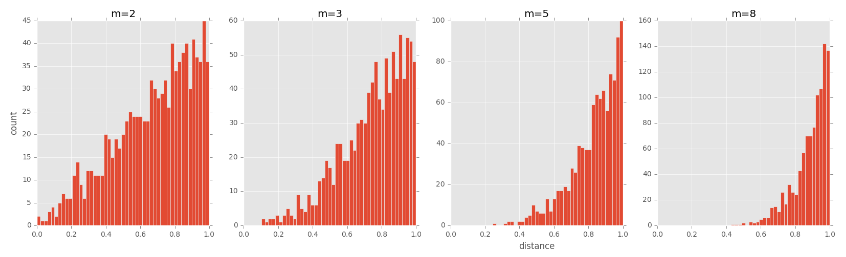
For setting variance , define Perplexity



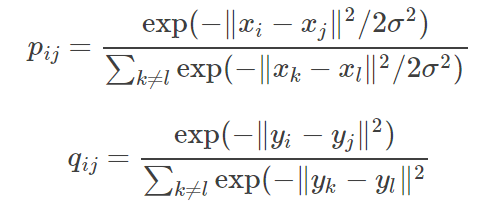


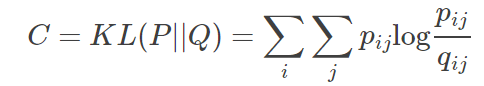
Symmetric SNE: use joint probability to replace conditional probability , it could simply the gradient formula . But it will suffer “crowding problem” .

“Crowding problem “ is indistinguishable when data gather together in low dimension . For example , when high dimension embedding into low dimension (under 10 ) , it has nice performance . However in 2 dimension , it could not get a credible embedding result . In the other words , suppose an m-dimension sphere with a data point as the center and a radius of r ( 3-dimension ) , its volume is increasing , assuming that the data points are uniformly distributed in the m-dimensional sphere , let’s look at other data points and the change of the distance as the dimension increases .

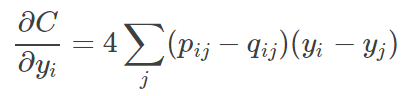


It can be seen from the figure above , as the dimensionality increases , most of the data points are clustered near the surface of the m-dimensional sphere , and the distance distribution from the points is extremely uneven . if this distance relationship is directly retained to a low dimension , there will be a “crowding problem”

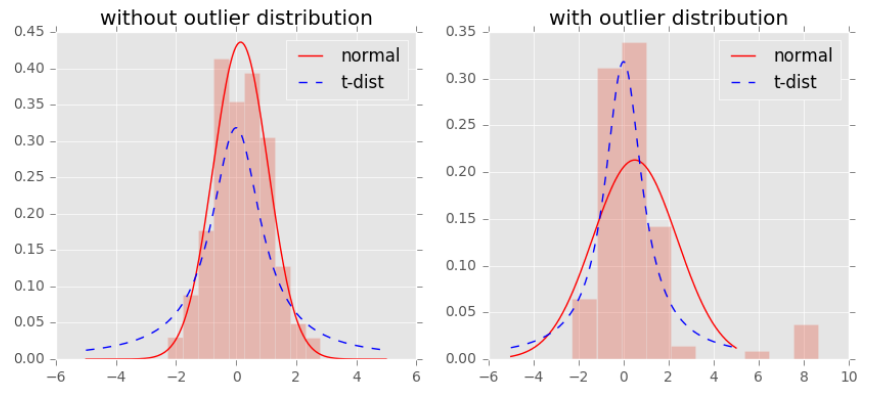




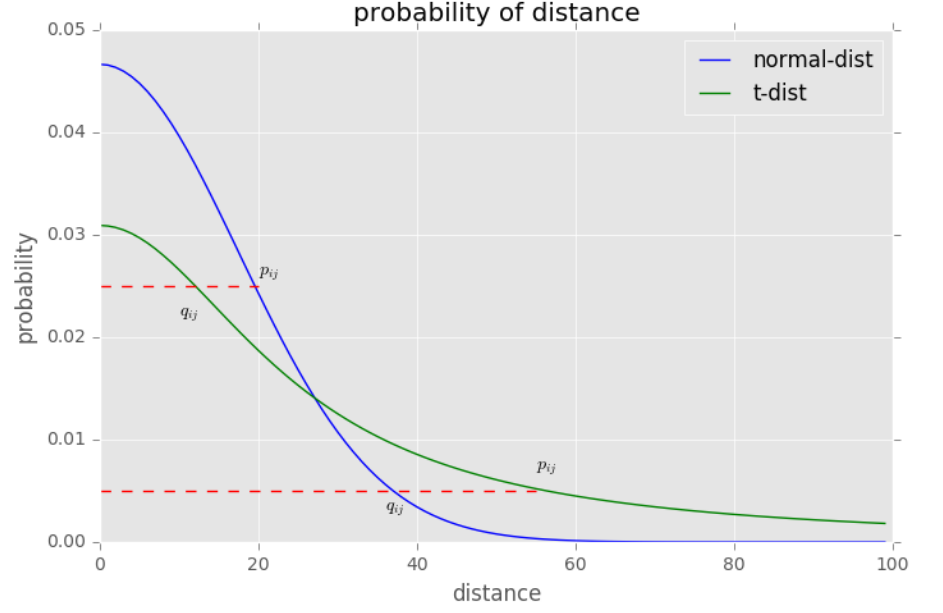
Objective function C



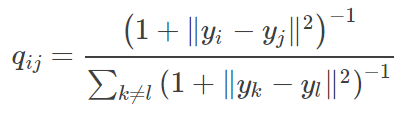
t-SNE use symmetric SNE to simply the gradient formula and use student-distribution to represent the similarity between any two datapoints , it can avoid crowding problem . Use student-distribution can more emphasis long tail distribution to transform distance into probability distribution , it can let small distance in high dimension could has a farther distance after images in low dimension .

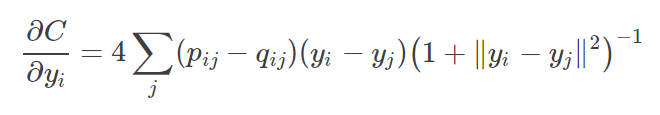


This figure shows that student-distribution is less affected by outliers and more reasonable fitting result .



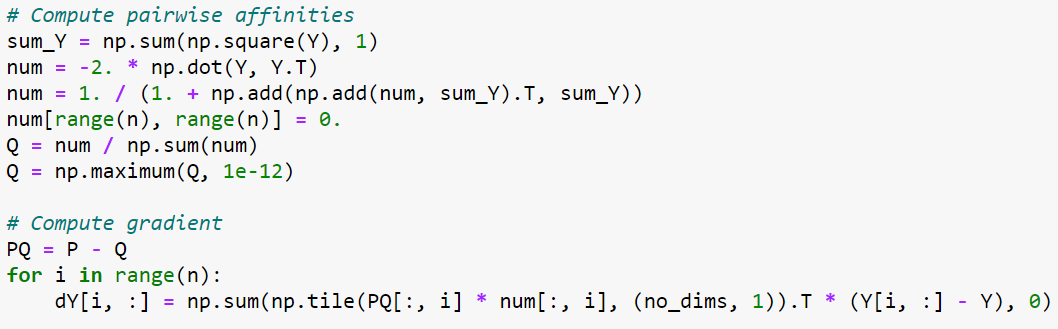
This figure shows that for high similarity data points , the distance will be little nearest , and for low similarity data points , the distance will be little farther than normal-distance . It means that the same class data points closer together and different class data points could be far away .



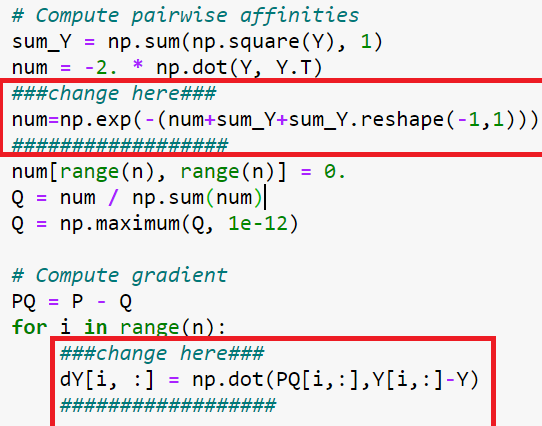


Use student-distribution to represent distance probability in low dimension

So the difference between symmetric SNE and -SNE are data distance probability representation in low dimension and the final gradient result .



t\_SNE



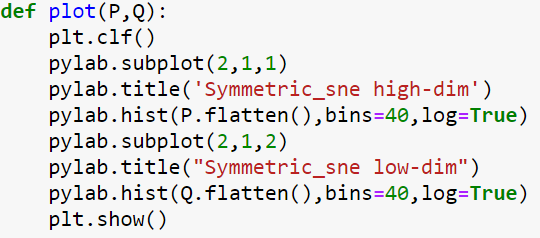
Symmetric SNE

Part2: Visualize the embedding of both t-SNE and symmetric SNE



Use pylab.scatter to plot the result

Part3: Visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space



P represent high dimension datapoints distance probability , Q represent low dimension datapoints distance probability .

Part4: Try to play with different perplexity values

Perplexity can seen as how many neighboring points are considered in the optimization process .

B. Experiments settings and results and discussions

(1) Kernel Eigenfaces

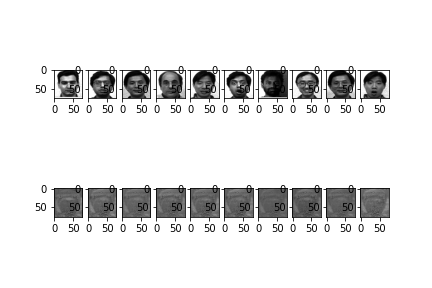
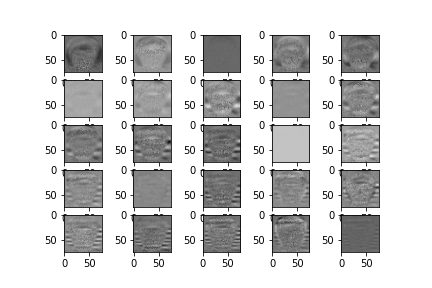
Part1:

Eigenfaces and reconstruction



The figure in the upper are original images , and pictures in the bottom are reconstruction pictures . It can conclude that the reconstruction are very well , it can directly see the face in the pictures .

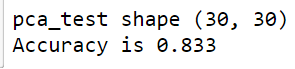
Fisherfaces and reconstruction



In fisherfaces , i think that there are some problem in my code , because the reconstruction pictures can not directly see a human face , it just a contour .

Part2:

Eigenfaces



Fisherfaces:



Although the fisherfaces’s reconstruction are not well . However , it could get higher performance in this dataset .

Part3:

Eigenface:

Linear kernel function

Polynomial kernel function

RBF kernel function

|  |  |  |  |
| --- | --- | --- | --- |
|  | Linear kernel | Polynomial | RBF |
| Parameters | None | C = 7e-10 | Gamma =7e-10 |
| Accuracy | 83.3% | 80% | 80% |

First , I thought use kernel in PCA will improve accuracy . However , my result could not get higher performance in this dataset . Maybe in other datasets , it could get a higher performance .

Fisherfaces

|  |  |  |  |
| --- | --- | --- | --- |
|  | Linear kernel | Polynomial | RBF |
| Parameters | None | C = 7e-10 | Gamma =7e-10 |
| Accuracy | 23.3% | 83.3% | 83.3% |

In this dataset , Linear kernel get a worst performance , it only has 23.3% accuracy .

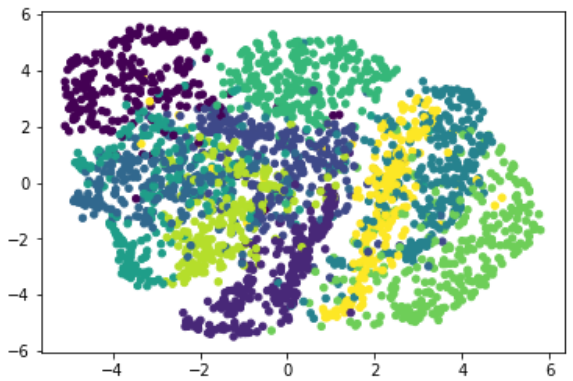
(2) t-SNE

Part1

Do not need to plot anything

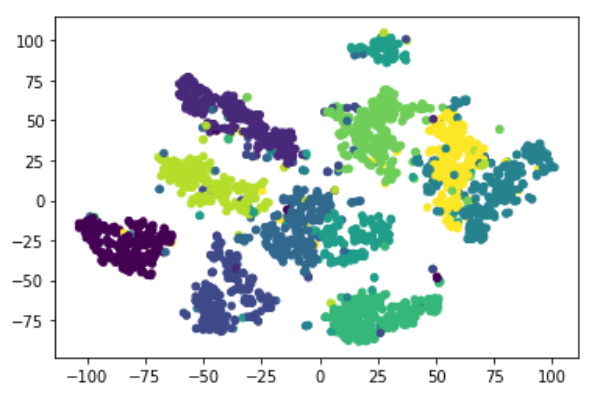
Part2

Symmetric-SNE final result



Perplexity=20

t-SNE final result

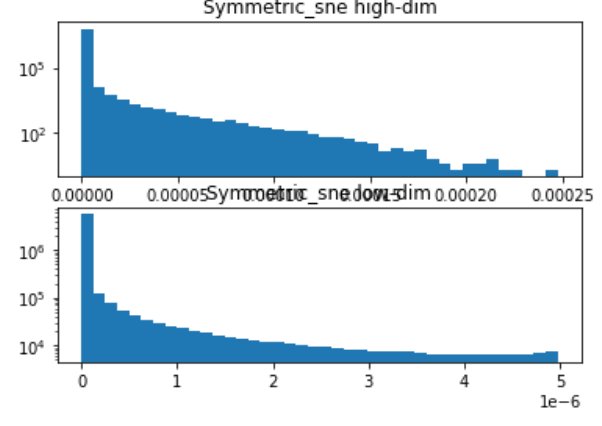


Perplexity = 20

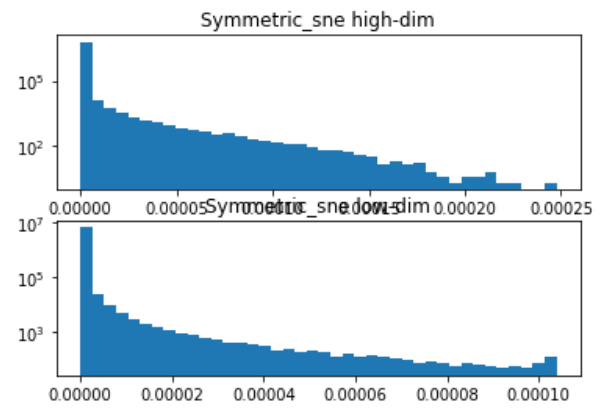
It can directly see that there has “crowding problem” in symmetric problem , the datapoints are get together , it shows that the datapoints are very crowded . However , the datapoints are separable in t-SNE , it shows that t-SNE can avoid “crowding problem “

Part3

Symmetric-SNE result



t-SNE result



It shows that t-SNE can keep initial data structure in the low dimension . However , symmetric SNE can not keep initial data structure in low dimension , it can observed it has “crowding problem “ by this graph too .

Part4:

Symmetric SNE

|  |  |  |
| --- | --- | --- |
| Perplexity =30 | Perplexity = 40 | Perplexity = 50 |
|  |  |  |

t-SNE

|  |  |  |
| --- | --- | --- |
| Perplexity =30 | Perplexity = 40 | Perplexity = 50 |
|  |  |  |

It can see that when perplexity = 40 in Symmetric SNE , it has the better performance . However , it still occur “crowding problem” , it can not beat the t-SNE.

And I think whether perplexity is , It still has the nice result in t-SNE .