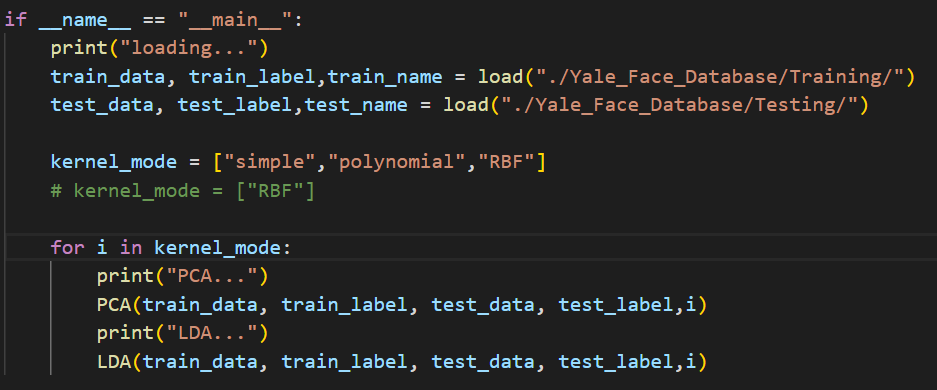
HW7 311552055 蕭育泓

**Kernel Eigenfaces**

Part a. code explanations

Main function



Load data

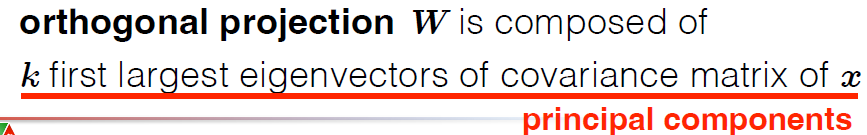
First, loading the data and resizing the picture(231 \* 195) to the picture(21 \* 15) by using the mean of the picture pixel. And then resize the picture again(resize 21 \* 15 to 315). Finally, we will get the all training data (135 \*315) and return it.



**Part 1:**

PCA function

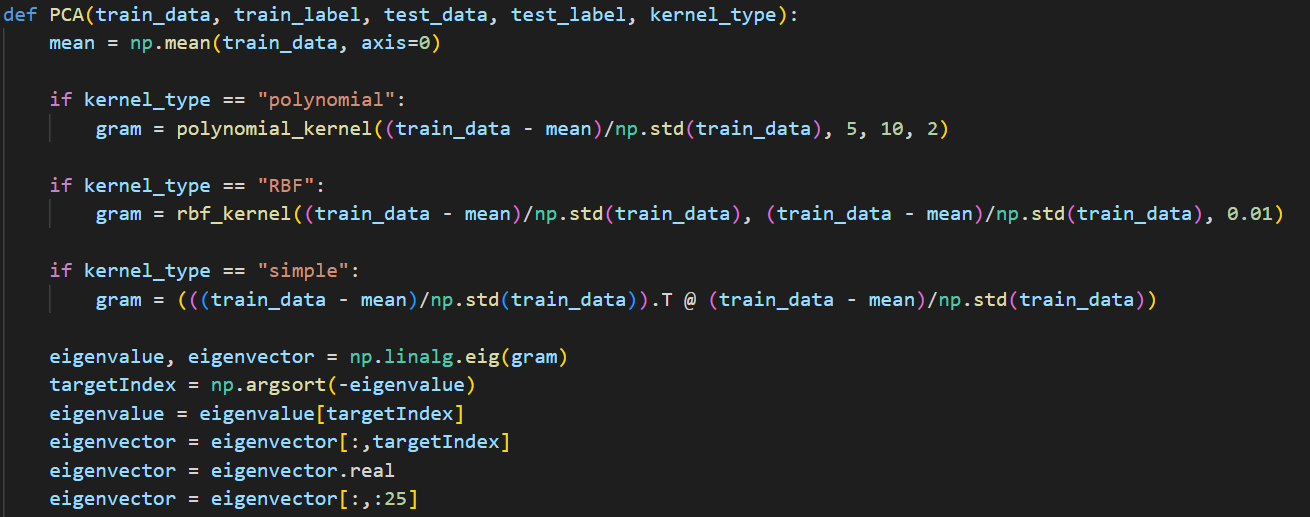
For part1, I used the kernel\_type = “simple” to implement the kernel linear. First, I standardize the training data and do the select kernel. Using the kernel to get the first 25 eigenfaces.



For the reconstruction, I using the formula below.



Because of the training data is standardization, so I need to adjust my training data for fit the original real type.

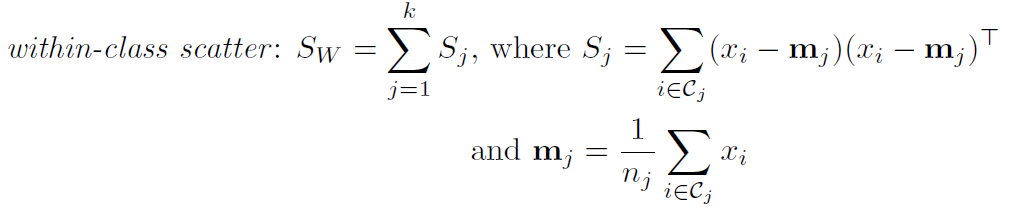
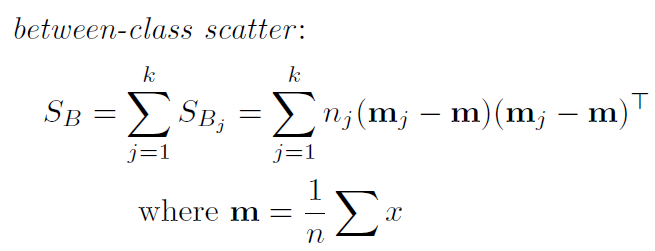




LDA function

For LDA, some concepts are as same as PCA function.

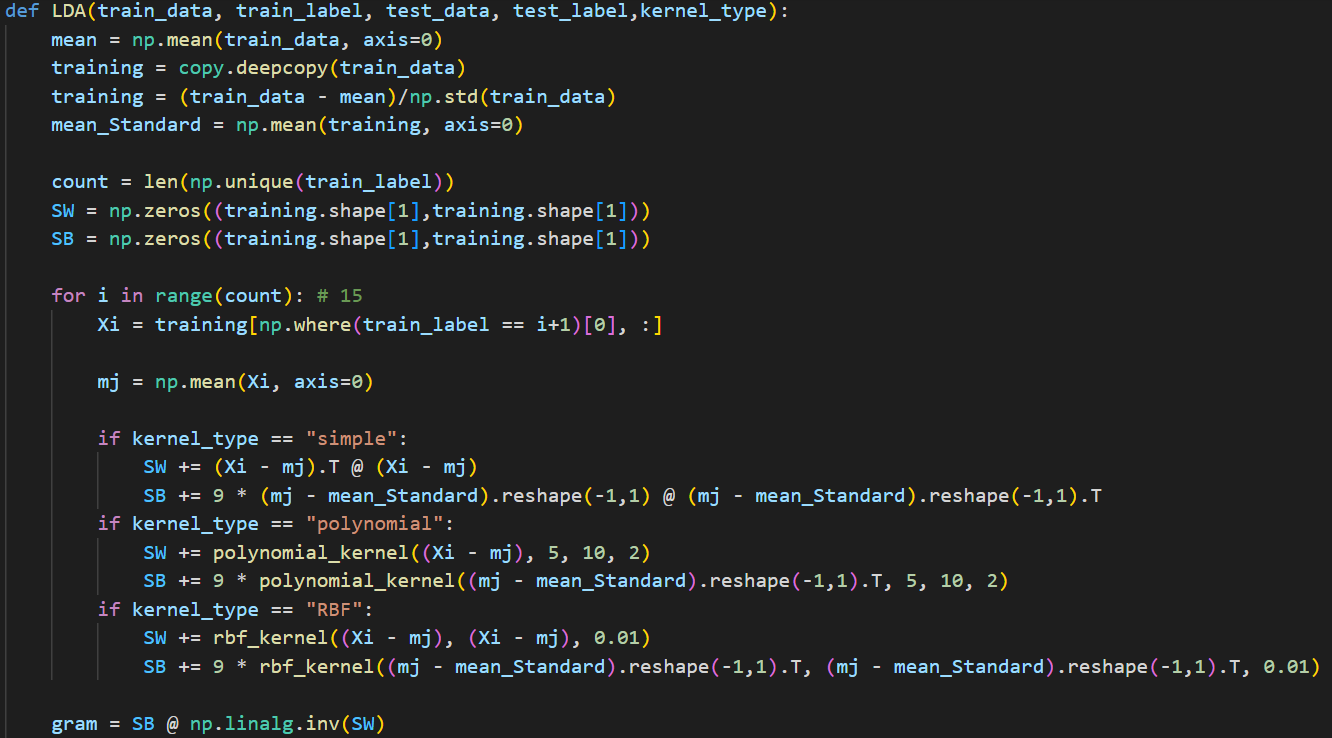
The different between two functions is only about the gram matrix generate. In the LDA, we need to using the standardize training data to do the formula below and find out the SW and SB.



And using the below formula to calculate the gram matrix.



But because my data resize type is different with the class sample, so I need to swap the function S-1W SB to SB S-1W.



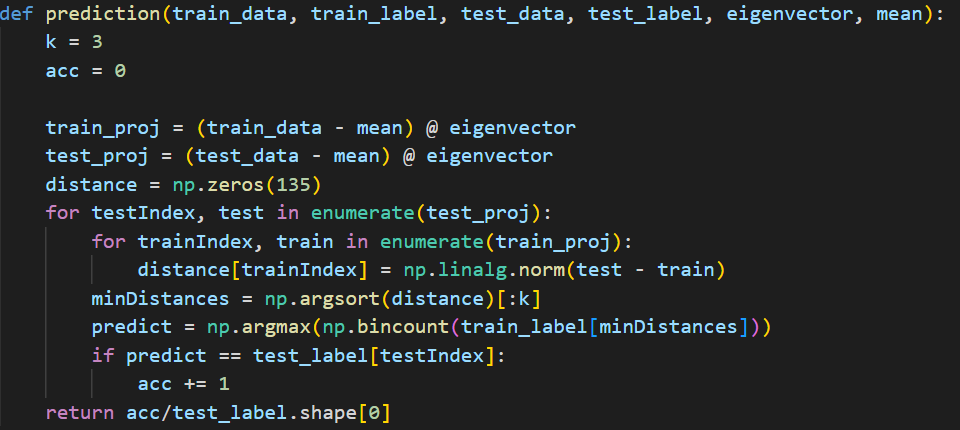
The remaining parts are as same as PCA function.

(include eigenvector generate and reconstruct part)

**Part 2:**

Function prediction

Using the k nearest neighbor to classify the testing image belongs to. I set the k = 3 to find the class. Find out the distance between the projection of training data and the projection of testing data. The k smallest distance will be select to be which cluster and predict.

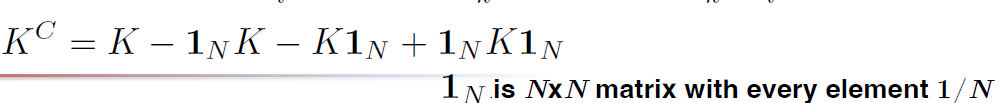


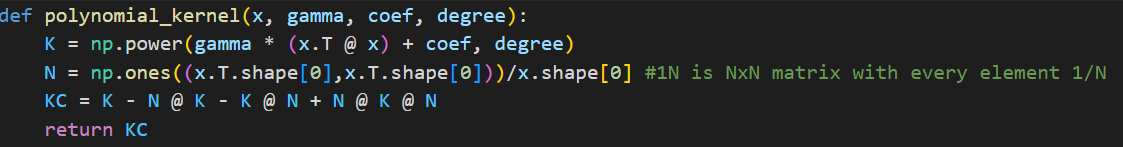
**Part 3:**

In the function PCA and the function LDA, both of them, I select the kernel RBF and kernel polynomial.

Function polynomial\_kernel

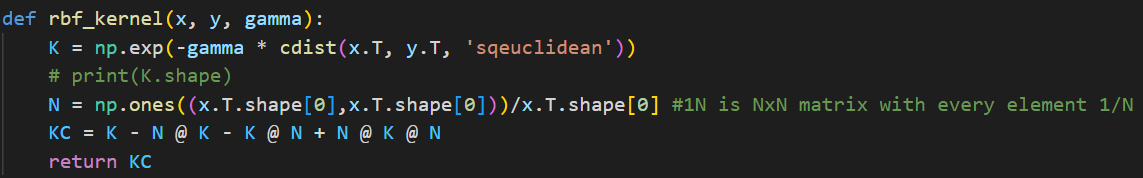
First, I will find out the kernel K by using the formula from previous homework. And using the formula below to get the Kernel PCA.





Function rbf\_kernel

The introduce is same as Function polynomial\_kernel.



In PCA, I will return the KC to the original function to be the gram matrix.

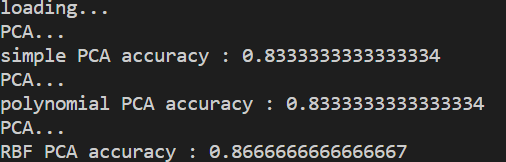
In LDA, I will use kernel to calculate the SW and SB.

Part b. Result

PCA

|  |  |  |
| --- | --- | --- |
| kernel | Eigenfaces | reconstruction. |
| simple |  |  |
| Poly-  nomial |  |  |
| RBF |  |  |

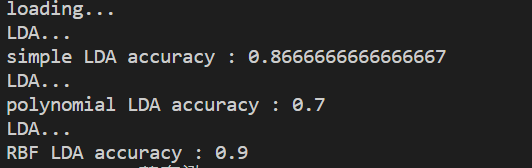
Performance:



LDA

|  |  |  |
| --- | --- | --- |
| kernel | Fisherfaces | reconstruction. |
| simple |  |  |
| Poly-  nomial |  |  |
| RBF |  |  |

Performance:



Observing in the part3:

1. The reconstruct images have the better resolution in PCA, which is more clear.
2. Both of them are having the better performance in RBF kernel.
3. The kernel polynomial in LDA, the reconstructions are similar.
4. The performance in polynomial kernel is the worst in both of them.
5. The simple performance is well in both of them.

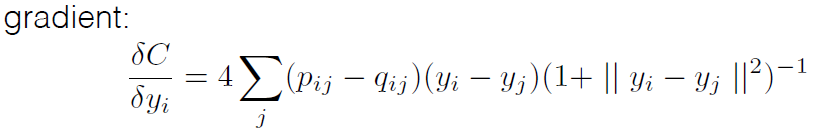
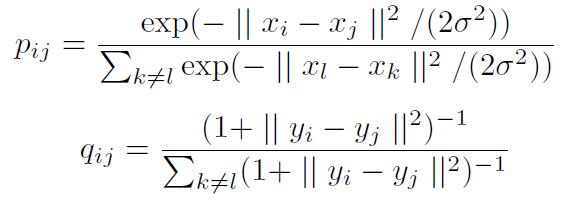
**t-SNE**

Part a. code explanations

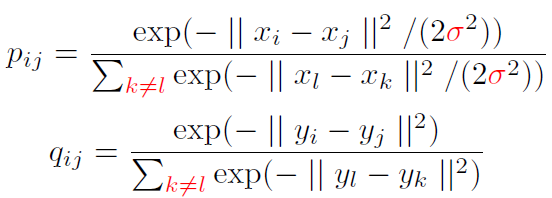
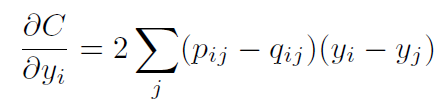
**Part 1:**

To modify the code a little bit and make it back to symmetric SNE. The difference of the s-SNE and t-SNE are about the q and gradient.

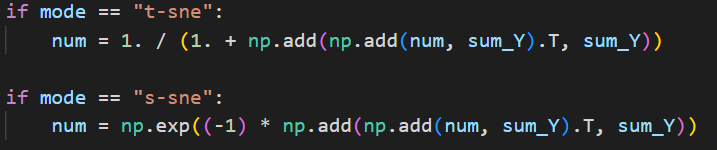
In the t-SNE, we calculate the q and gradient by the formula below.



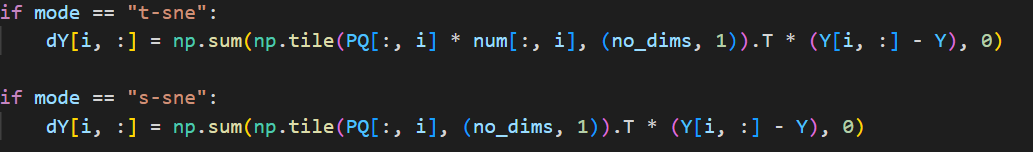
In the s-SNE, we calculate the q and gradient by the formula below.



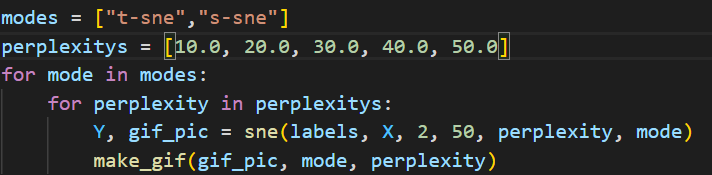
Calculate the different q :



Calculate the different gradient :



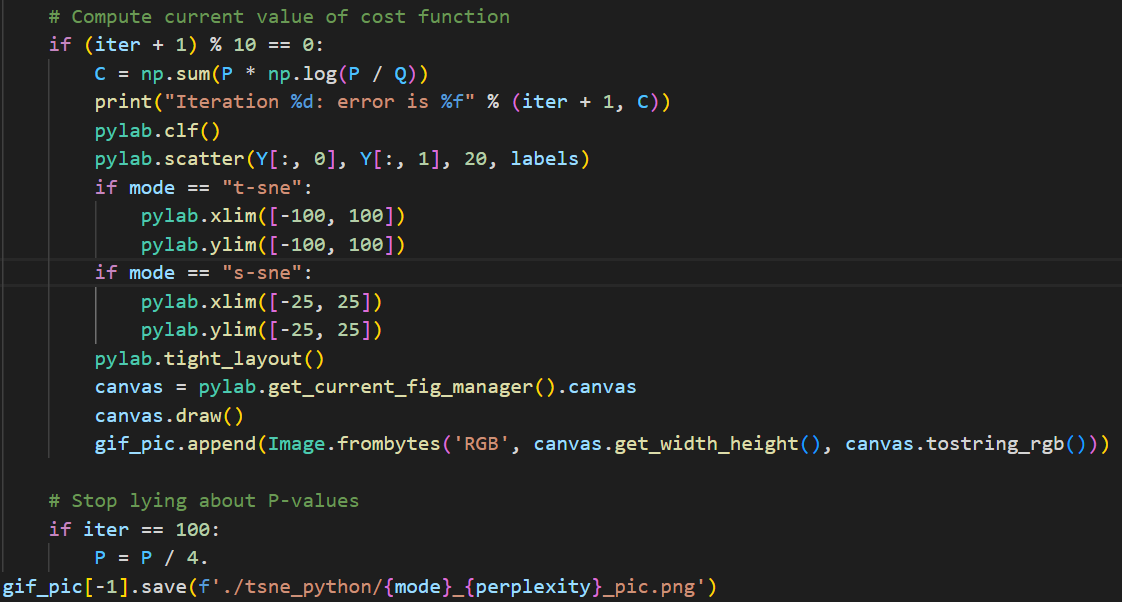
I select the mode to control, which SNE is doing.

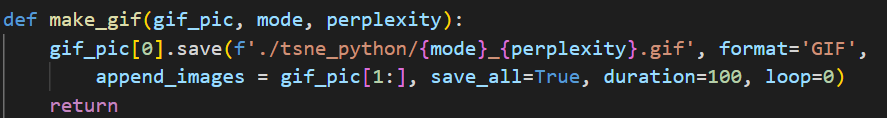


**Part 2:**

Visualize the embedding of both t-SNE and symmetric SNE.

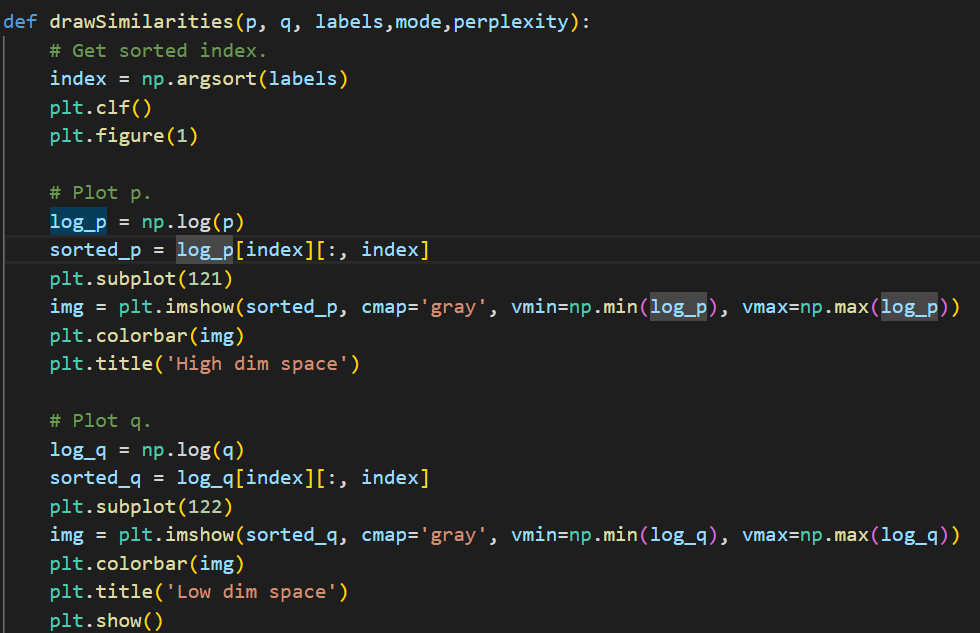
I will store the image of each 10 iterations and using the function make\_gif to make the .gif.





**Part 3:**

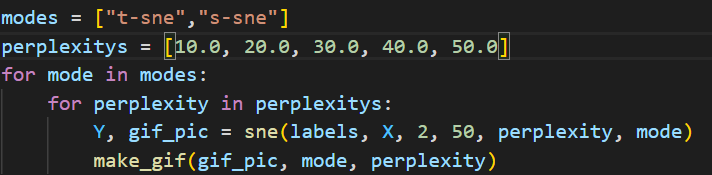
Using function drawSimilarities to show the high-dimensional space and low-dimensional space. The p and q are using the last iteration result.



**Part 4:**

I Try to play with different perplexity values with

[10 20 30 40 50] for the input.



Part b. Result

**Visualize the embedding of both t-SNE and symmetric SNE.**

|  |  |  |
| --- | --- | --- |
| perplexity | s- SNE | t-SNE |
| 10 |  |  |
| 20 |  |  |
| 30 |  |  |
| 40 |  |  |
| 50 |  |  |

**Visualize the distribution of pairwise similarities**

|  |  |  |
| --- | --- | --- |
|  | s- SNE | t-SNE |
| 10 |  |  |
| 20 |  |  |
| 30 |  |  |
| 40 |  |  |
| 50 |  |  |

Observing in the part1:

1. Both will have better performance after the 100 iteration and error will drop down from around 20 to about 2
2. Training speed in s-SNE is faster than t-SNE.

Observing in the part4:

1. Using the bigger perplexity will get the all data closer.
2. Using the bigger perplexity will get each clusters closer.
3. Using the bigger perplexity will get smaller group classify.

Part c. observations and discussion

1. Eigenfaces is the set of eigenvectors.
2. The meaning of eigenfaces, I think is going to get the weight to know which places are more important.
3. The eigenfaces is more clear than fisherfaces.
4. The reconstruct data will similar to the original image.
5. If we did not resize to the small size, the calculate time will be horrible.
6. The result of the s-SNE is more crowded than t-SNE and mix together.
7. t-SNE can classify better.
8. The range of pairwise similarities are smaller in t-SNE, so that s-SNE has more crowded problems.