

## Homework 4: Machine Learning

*Instructor: Siheng Chen***Zhou Haoquan**

## Q1 Principle Component Analysis

## Part A: Basic Calculation

1.

We can calculate the  $\vec{w}$  to be  $\begin{bmatrix} 0.89 \\ 0.45 \end{bmatrix}$  and  $w_0 = -4.329$ . The line  $\vec{w}^T \vec{x} + w_0 = 0$  is shown in figure.1.

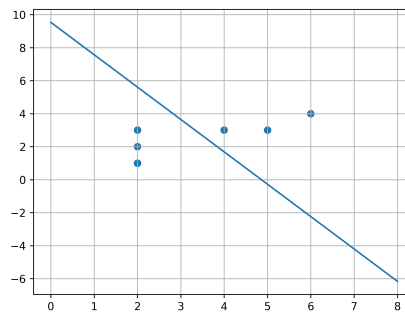


Figure 1

2.

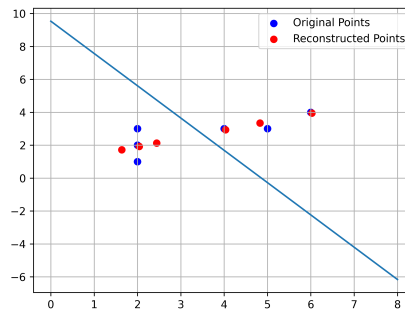


Figure 2

After projection and reconstruction, the result is shown in figure 2. We can see that the reconstructed points are close to the original points.

3.

The MSE is calculated to be 0.294.

4.

The Fisher Ratio is 14.67.

## Part B: MNIST Implementation

1.

The mean image for 1 is shown in figure.3 It is quite similar to 1. Five global eigen vectors are

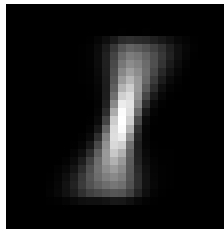


Figure 3

shown in the following figures for time needed for calculating the eigen vectors, method without

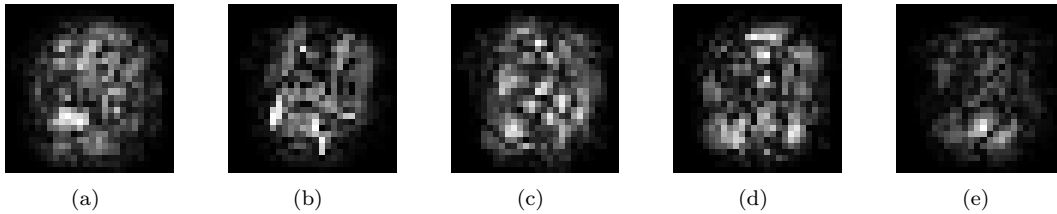


Figure 4

using Gram Matrix Trick only takes 1-2 seconds, while the method using Gram Matrix Trick uses 10 minutes, as shown in figure 5.

2.

In this problem, since there are 10000 pictures, the input size is really big. Using Gram Matrix trick can only deal with situations when input size is small but the resolution is high. So there is no need to apply Gram Matrix Trick here.

```

real    0m2.052s
user    0m4.832s
sys     0m4.966s
haoquan@haoquan-ThinkP
(1135, 784)

real    0m1.272s
user    0m1.733s
sys     0m1.534s

```

(a) Time used without Gram Matrix Trick

```

✓ 10m 38.6s

```

(b) Time used with Gram Matrix Trick

### 3.

The results are shown in following figures. It is strange to see more vectors causing larger MSE,

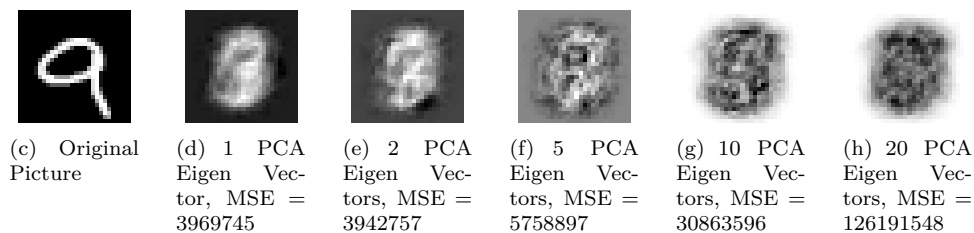


Figure 5

and I noticed that the **numpy.linalg.eig** function gives lots of one-hot eigen vectors, which makes it impossible to reconstruct the picture.

## Q2 K-means

### 1.

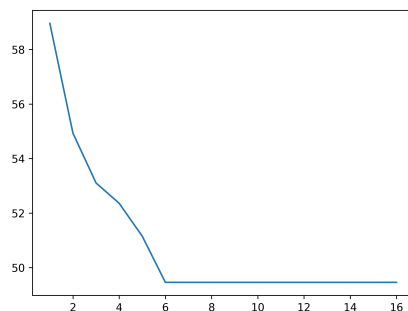


Figure 6

We choose  $K = 5$ . Figure 6 shows the objective function decreases as the iteration number increases. The corresponding clustering results are shown in figure 7.

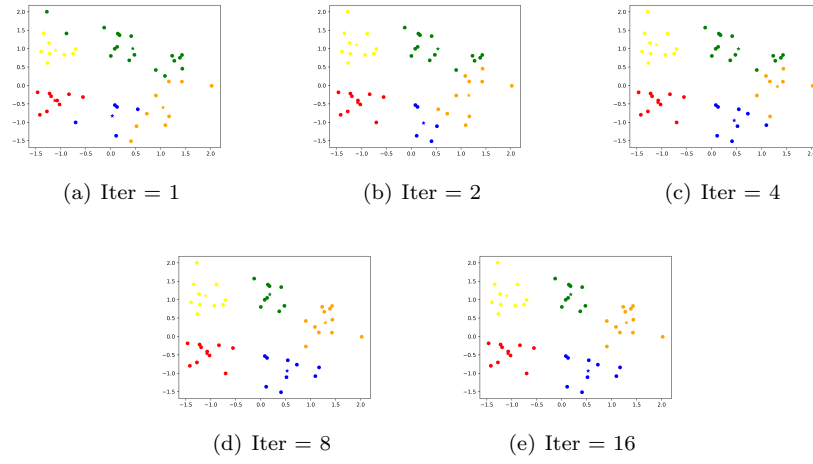


Figure 7

From the figure, we can see that the clustering becomes more appropriate after iterations and remains stable after iteration 6.

## 2.

We change  $K = 2, 3, 4, 5$ , the results are shown below We can observe that as  $K$  increases, more

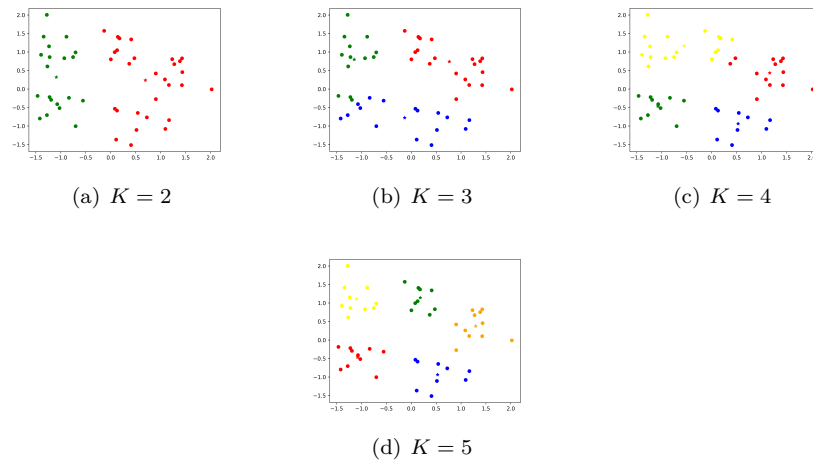


Figure 8

minor differences between clusters are found out. And the clusters are divided more appropriately.

3.

We first set  $K = 5$ , the results are shown below for different iterations. And the objective function is

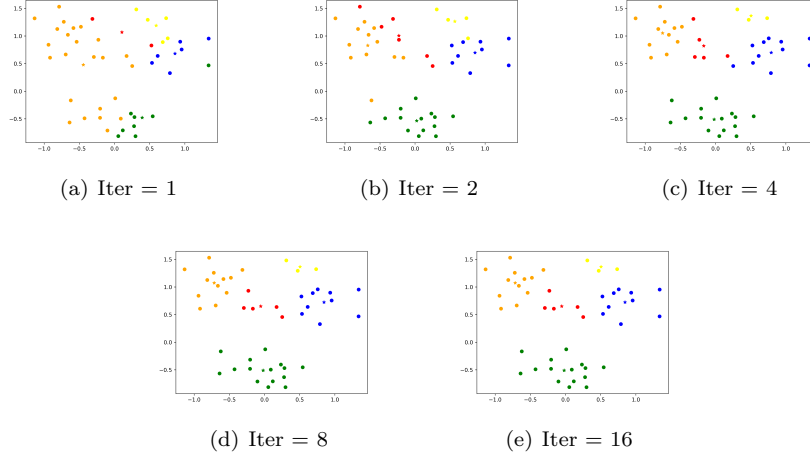


Figure 9

shown in figure 10. We can see that the clustering becomes better after iterations and the objective

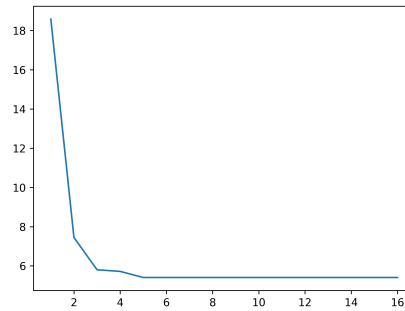


Figure 10

function converges after iteration 5.

Then we vary  $K$  from 2 to 5. We can observe that as  $K$  increases, more minor differences between clusters are found out. And the clusters are divided more appropriately.

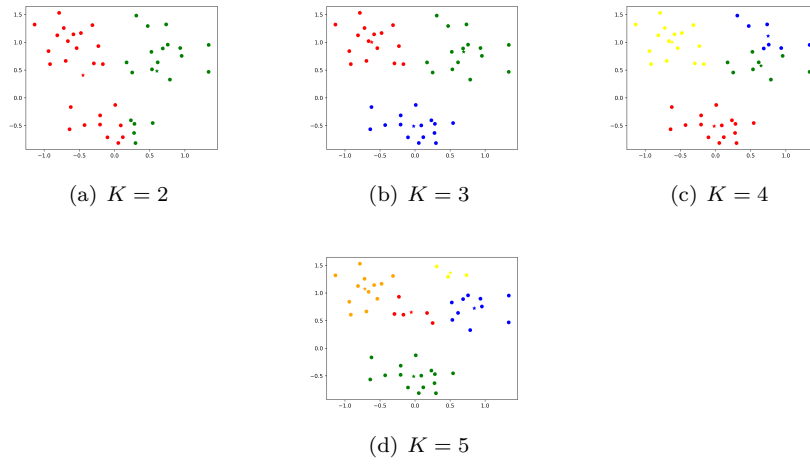


Figure 11

## Q3 Logistic Regression

### Task A: Visualization

1.

The functions are shown in figure 12

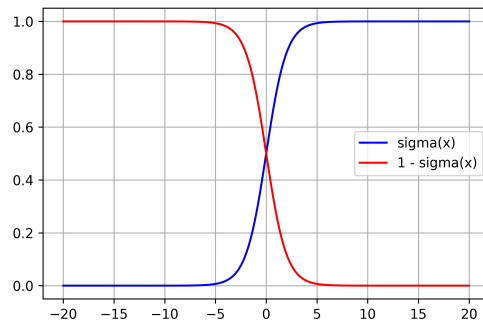


Figure 12

2.

$1 - \sigma(x)$  and  $\sigma(x)$  is symmetric about the y axis. Which means that  $\sigma(-x) = 1 - \sigma(x)$

3.

$\sigma(-x) = \frac{1}{1+e^x} = \frac{e^{-x}}{1+e^{-x}} = 1 - \sigma(x)$ . This indicates  $\sigma(-x) = 1 - \sigma(x)$

## Task B: Implementation

When epoch number equals 100, the result is shown in the following figures We then increase the

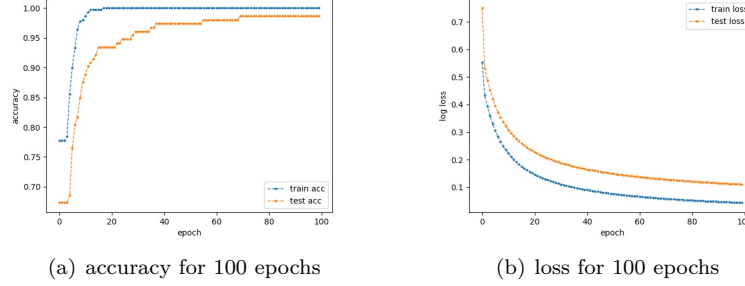


Figure 13

epoch number to 3000, the results are shown in the following figures We find that adding epochs

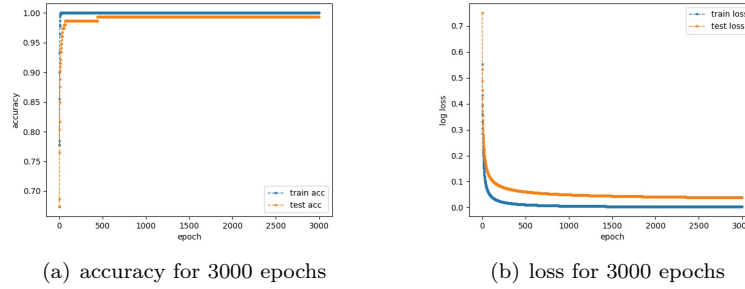


Figure 14

does not improve the accuracy significantly.

## Q4 Linear Regression

1.

$$\begin{aligned}
 f(w) &= \|Xw - y\|_2^2 + \lambda \|w\|_2^2 = \|y\|_2^2 - 2(Xw)^T y + \|Xw\|_2^2 + \lambda \|w\|_2^2 \\
 &\Rightarrow \nabla_w f(w) = -2\nabla_w (Xw)^T y + \nabla_w \|Xw\|_2^2 + \nabla_w \lambda \|w\|_2^2
 \end{aligned}$$

Note that

$$\nabla_w (Xw)^T y = \nabla_w w^T (X^T y) = X^T y$$

and

$$\nabla_w ||Xw||_2^2 = \nabla_w w^T (X^T X w) = 2X^T X w$$

So the equation becomes

$$(2X^T X + 2\lambda \mathbb{I})w = 2X^T y$$

$$\Rightarrow w = (X^T X + \lambda \mathbb{I})^{-1} X^T y$$

**2.**

We can directly calculate  $w = \begin{bmatrix} -4.44 \times 10^{-16} \\ 1 \end{bmatrix}$ , which is shown in the following figure We know

```
[[ -4.4408921e-16]
 [ 1.0000000e+00]]
```

that the right answer is  $w = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . The result is almost right.

**3.**

For  $\lambda = 1$ ,  $w = \begin{bmatrix} 0.36 \\ 0.66 \end{bmatrix}$

For  $\lambda = 0.001$ ,  $w = \begin{bmatrix} 0.002 \\ 0.999 \end{bmatrix}$

For  $\lambda = 0.00001$ ,  $w = \begin{bmatrix} 1.6 \times 10^{-6} \\ 0.9999 \end{bmatrix}$  The results are shown in the following figure

```
haoquan@haoquan-Th
[[ 0.36363636]
 [ 0.65909091]]
haoquan@haoquan-Th
[[ 0.00159474]
 [ 0.99860451]]
haoquan@haoquan-Th
[[ 1.5999472e-05]
 [ 9.9998600e-01]]
```

**4.**

The gradient is calculated in 1. as

$$\nabla_w f(w) = -2X^T y + 2X^T X w + 2\lambda \mathbb{I} w$$



And we can derive the update procedure to be

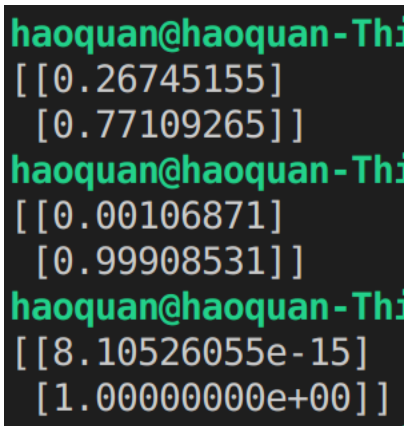
$$w = w - \nabla_w f(w)$$

## 5.

For 100 iteration,  $w = \begin{bmatrix} 0.27 \\ 0.77 \end{bmatrix}$

For 1000 iteration,  $w = \begin{bmatrix} 0.001 \\ 0.999 \end{bmatrix}$

For 10000 iteration,  $w = \begin{bmatrix} 8.1 \times 10^{-15} \\ 1 \end{bmatrix}$  The result is shown in the following figure



```
haoquan@haoquan-Th:~$ python3 main.py
[[0.26745155]
 [0.77109265]]
haoquan@haoquan-Th:~$ python3 main.py
[[0.00106871]
 [0.99908531]]
haoquan@haoquan-Th:~$ python3 main.py
[[8.10526055e-15]
 [1.00000000e+00]]
```

## 6.

Stronger  $l_2$  regulation is equivalent to fewer running iterations.