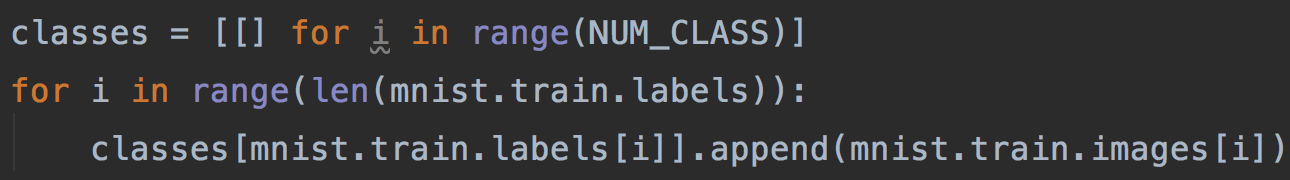
**Problem Set 1**

1. **Dataset**
   1. Extract feature values and labels from the data: Download the MNIST dataset from the Internet and process the original data file into a Numpy array that contains the feature vectors and a Numpy array that contains the labels.

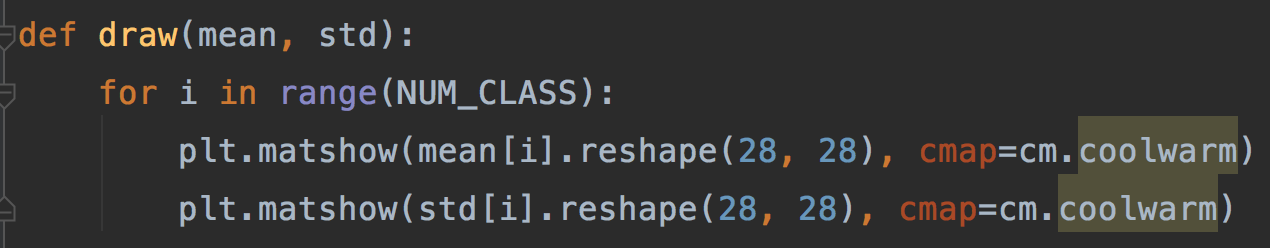


* 1. Data Partition: The MNIST dataset is originally partitioned into a training set(mnist.train, 55000), a testing set(mnist.test, 10000) and a validation set(mnist.validation, 5000).
  2. Classify data by label: Divide the dataset into 10 categories.



1. **Problem 1:** 
   1. Estimate mean and standard deviation of each category(MLE)

Given i.i.d. (i is the index of the feature vectors in one category. j is the index of the pixel in the image)



* 1. Draw mean and std images

1. **Classify the images**
   1. Apply naïve Bayes classifier with attribute conditional independence assumption.

Gaussian Distribution:

Bayes:

is the evidence. is prior. is likelihood.

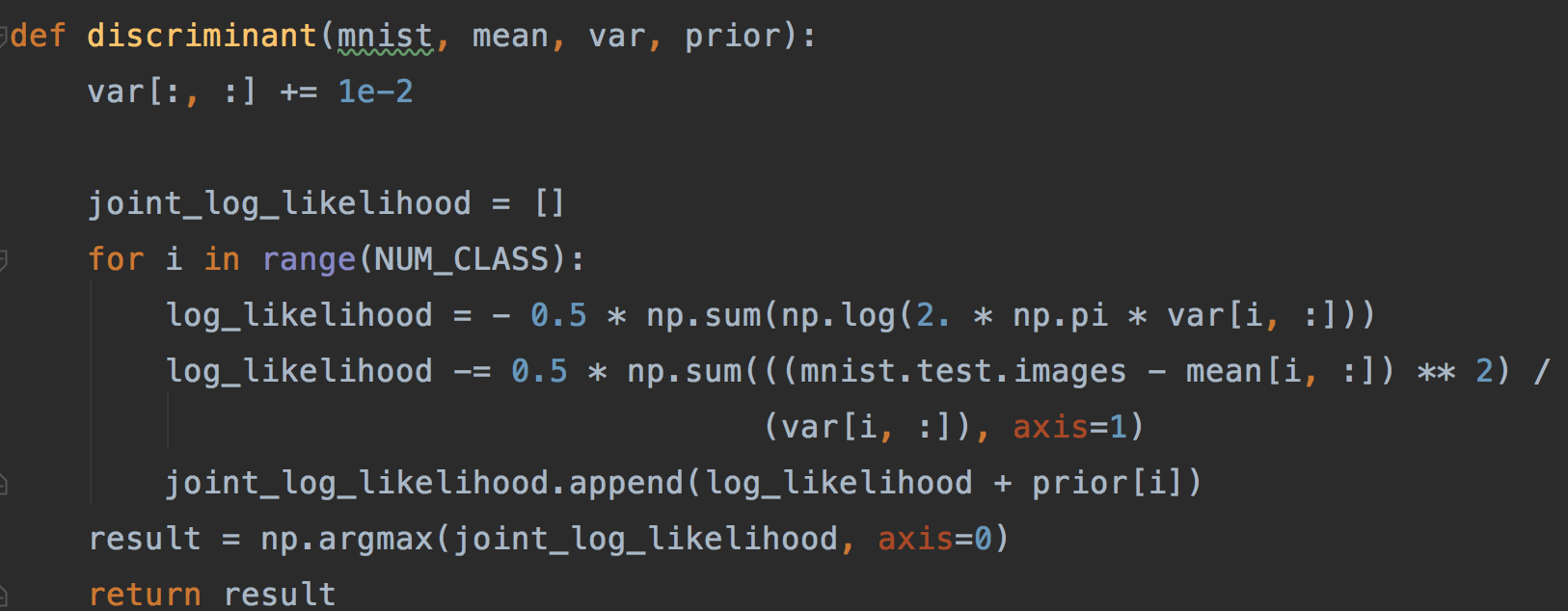
Discriminant Function:

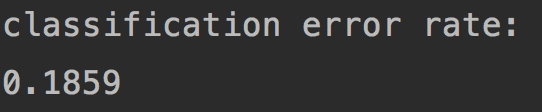
zero-one loss function:

Conditional risk function:

Bayesian decision rule:

So we select the class that maximizes the posterior probability

* 1. Code If the ratio of data variance between dimensions is too small, it will cause numerical errors. To address this, we artificially boost the variance by epsilon, a small fraction of the standard deviation of the largest dimension.
  2. Output

****

**Correct rate：81.41%**

* 1. Analysis

Why it doesn't perform as good as many other methods on LeCuns web page?

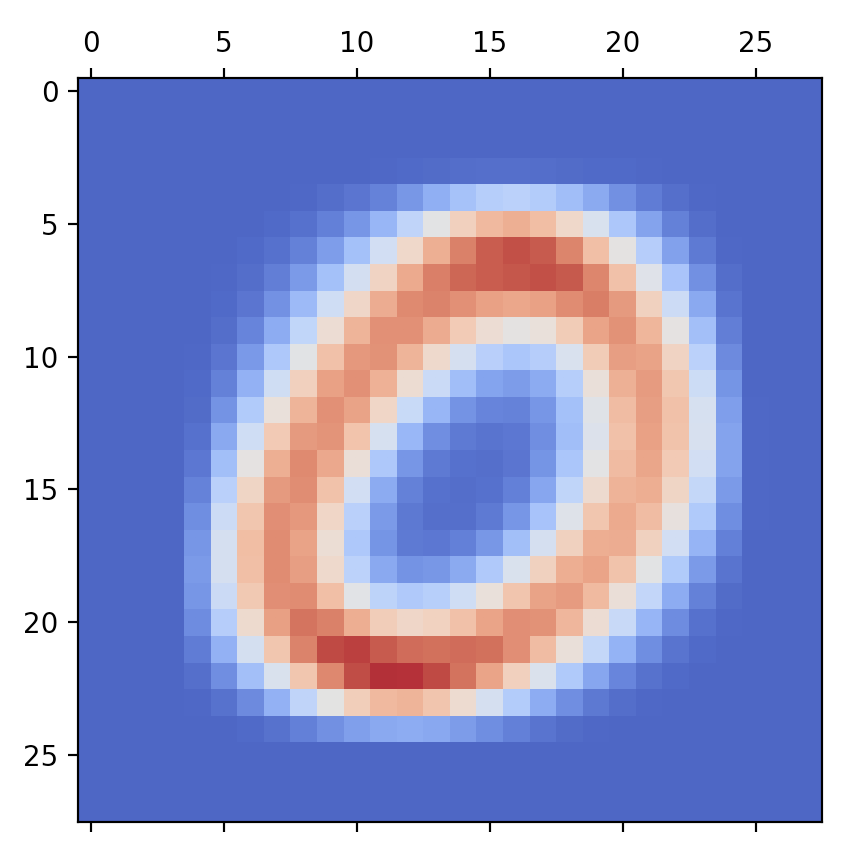
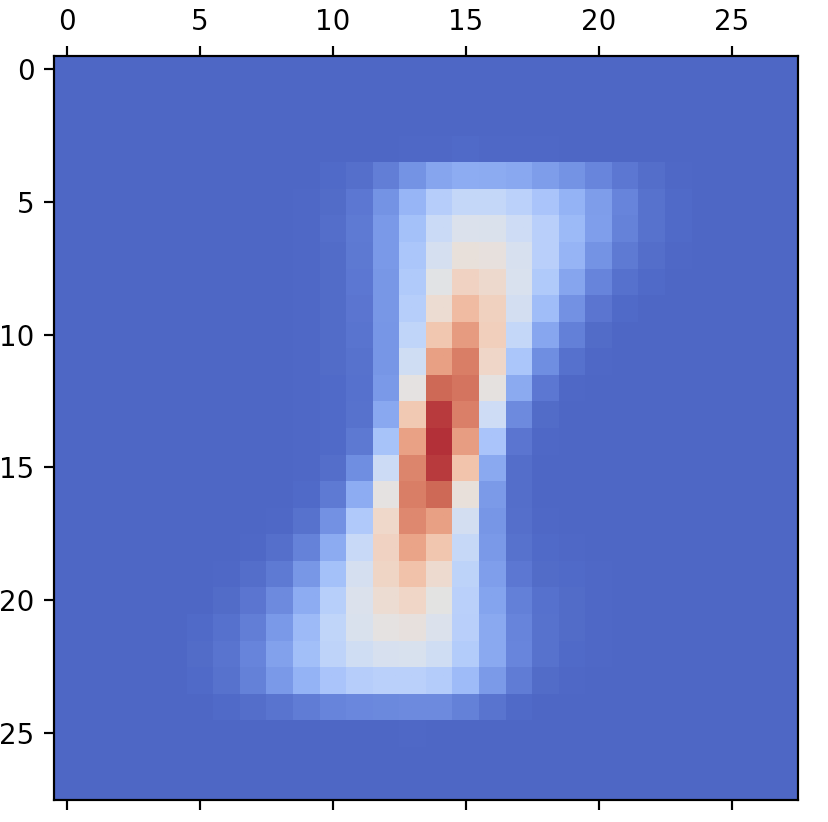
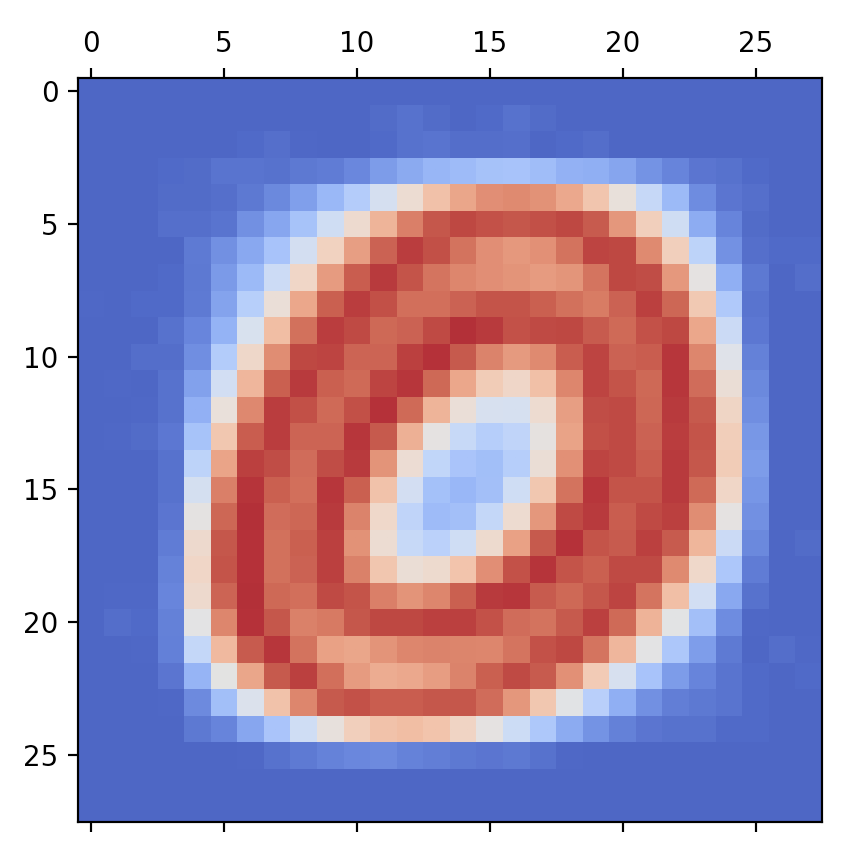
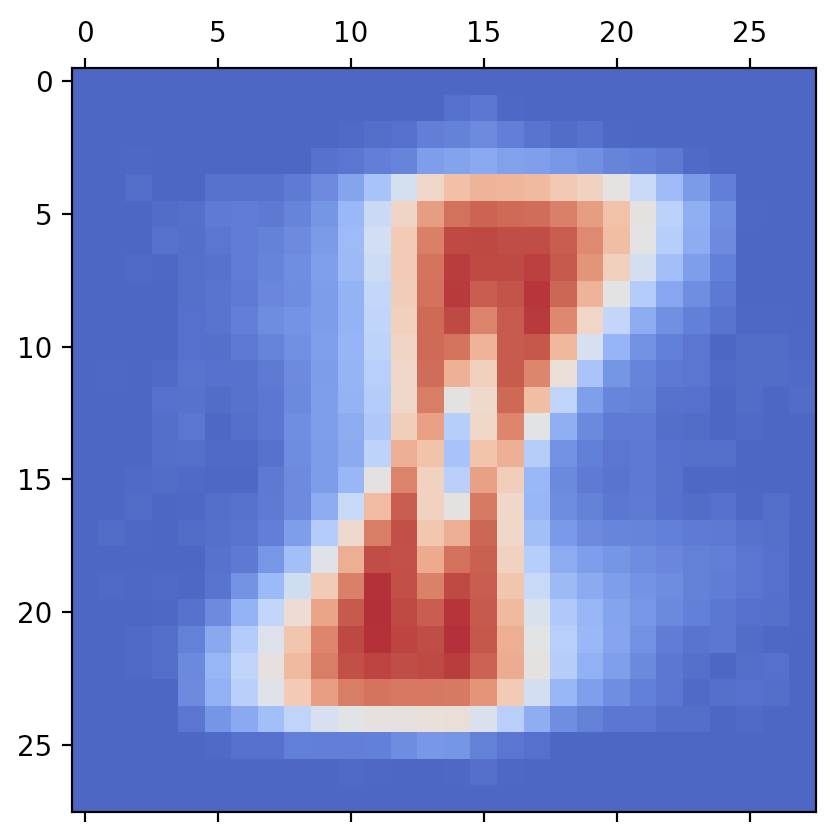
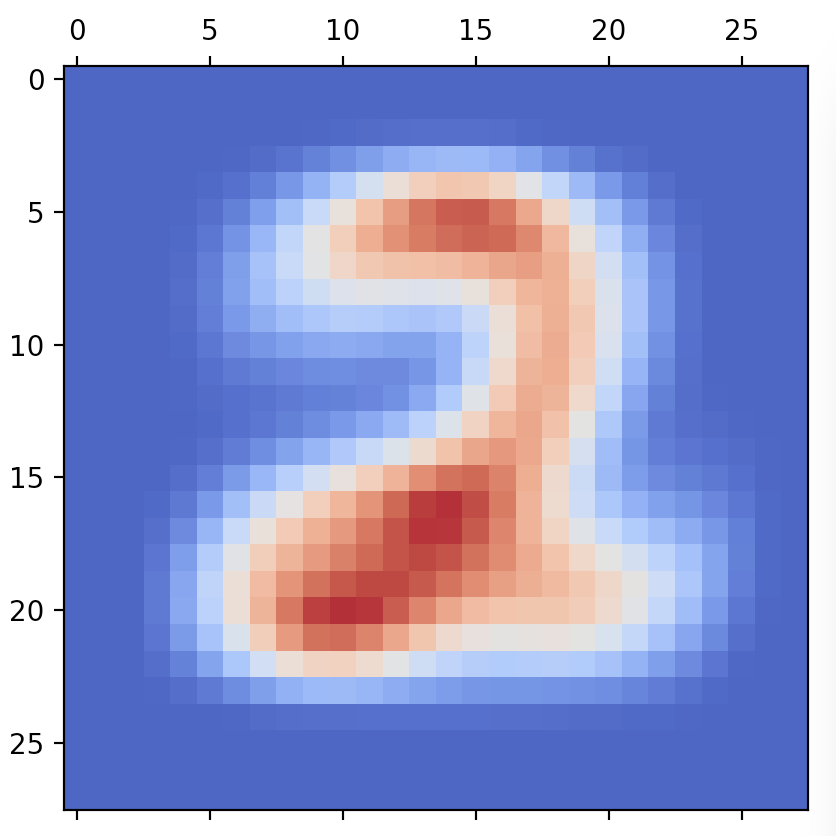
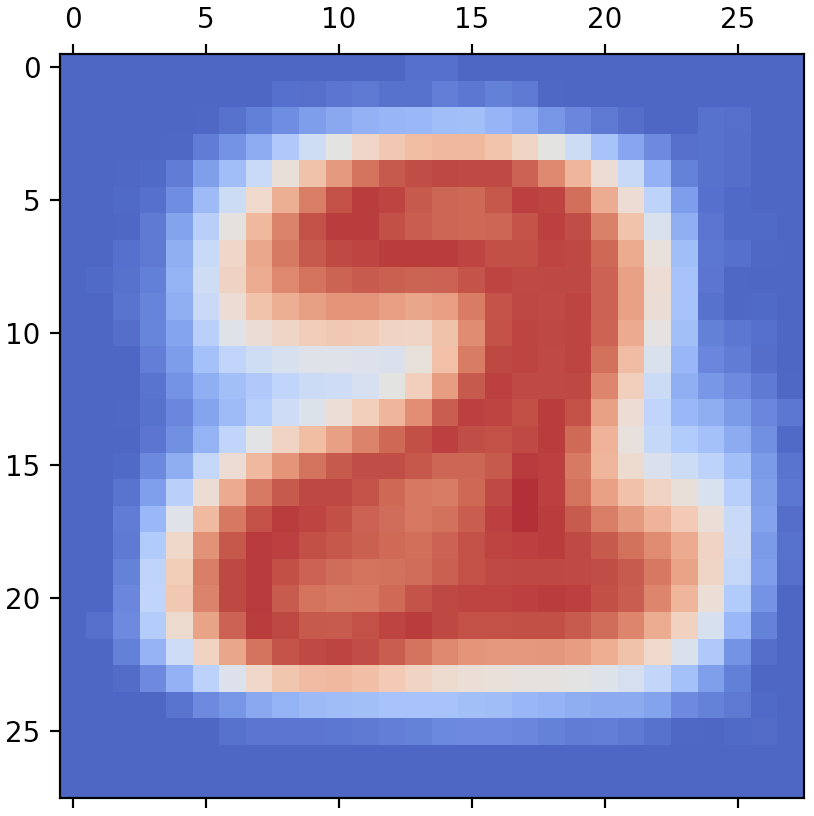
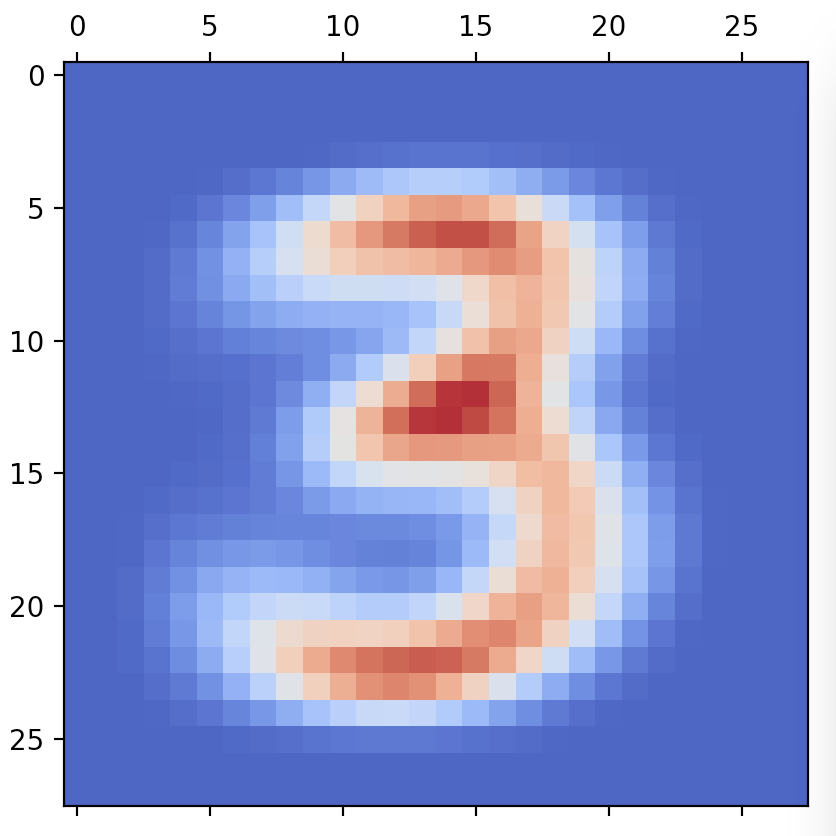
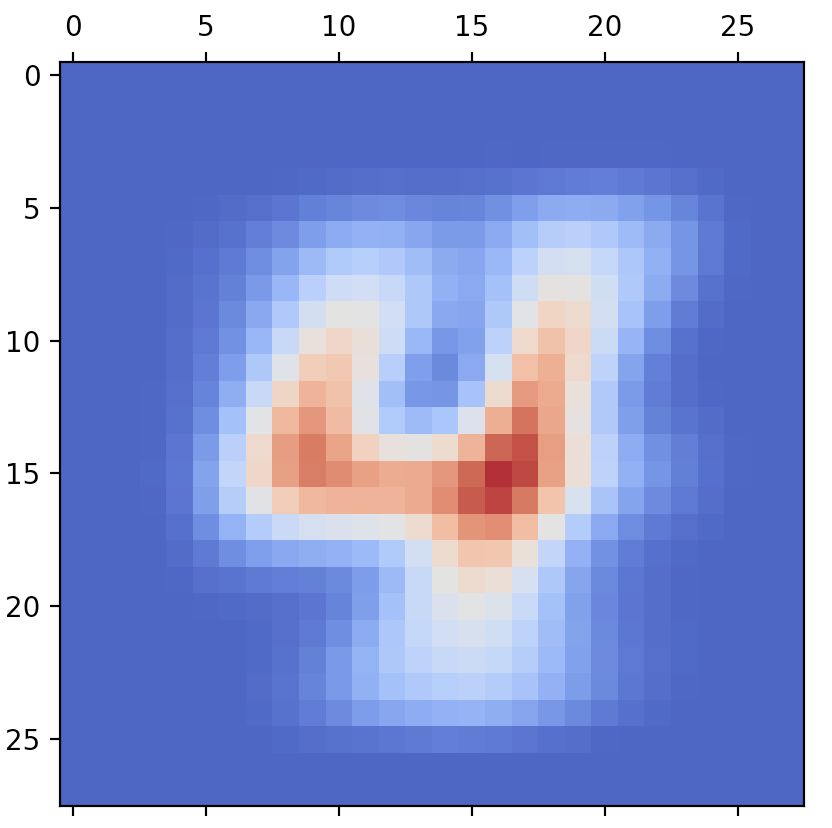
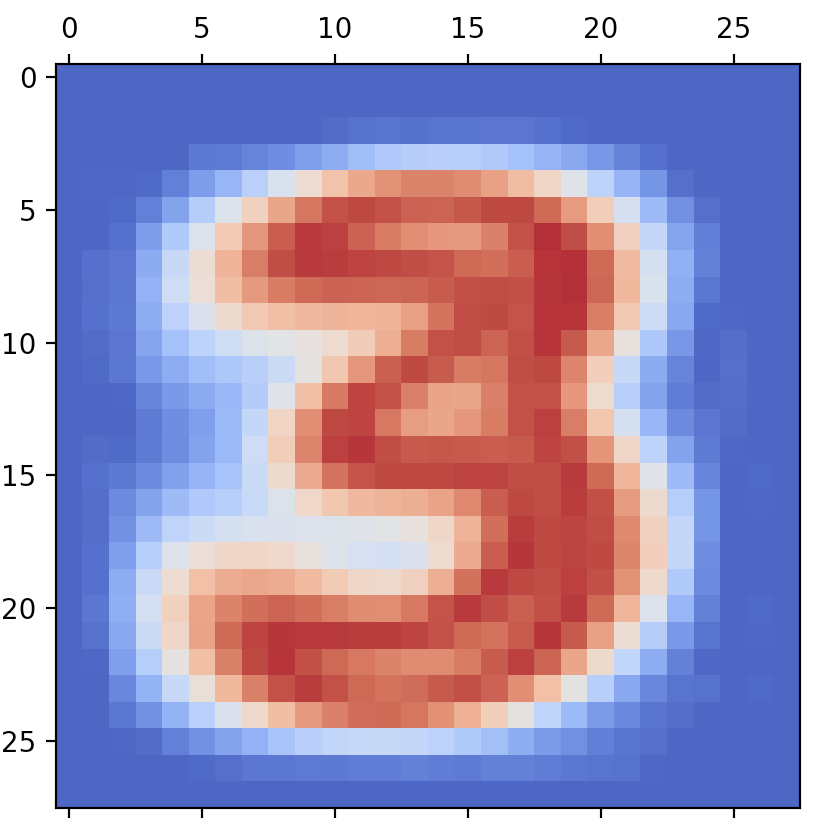
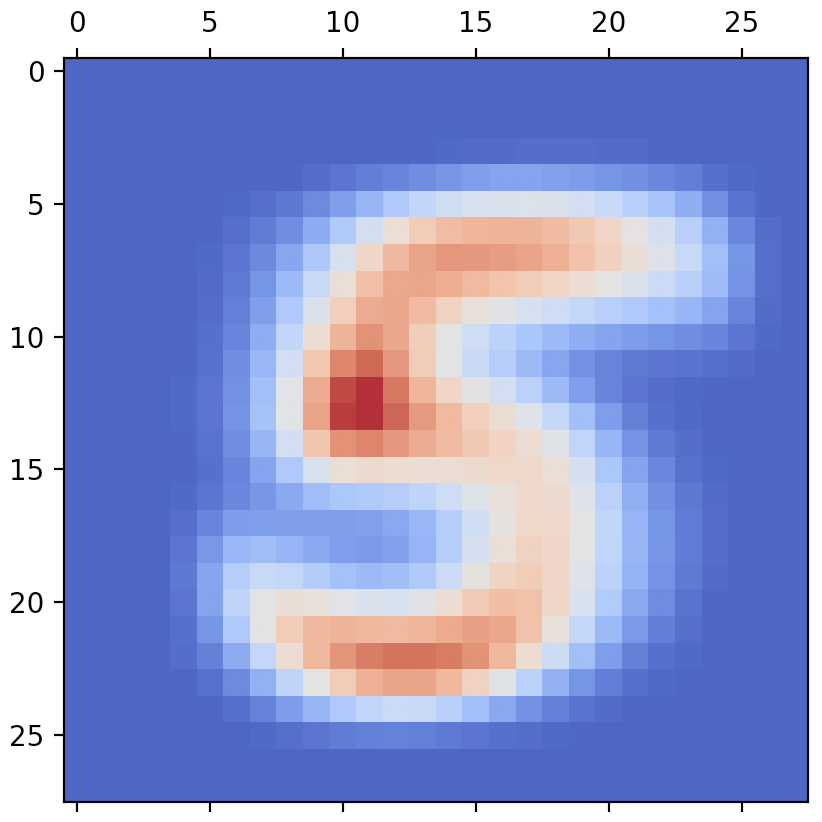
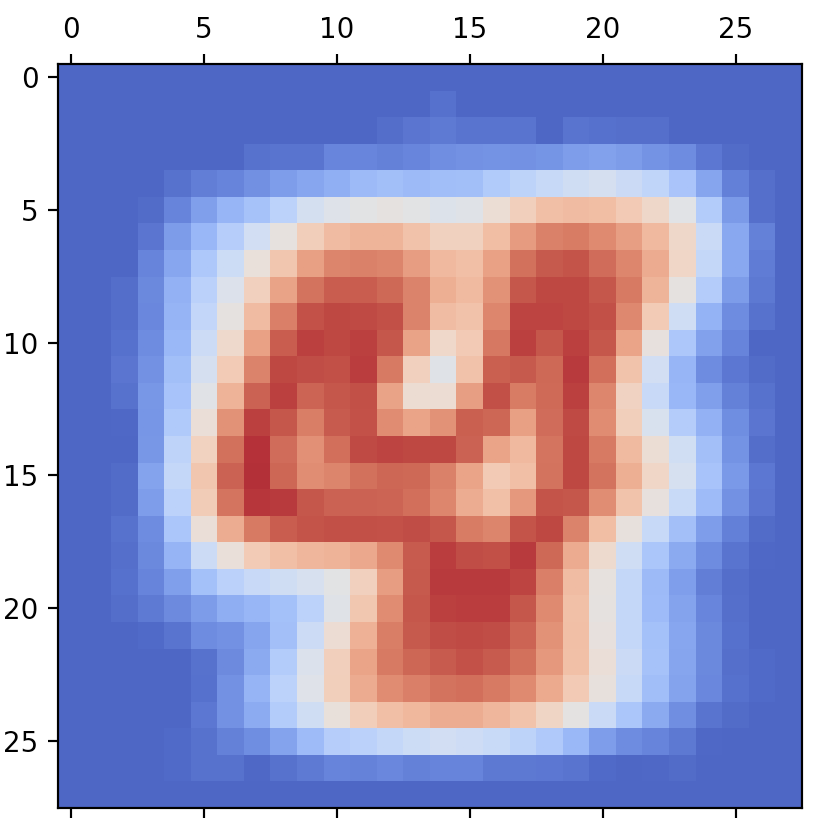
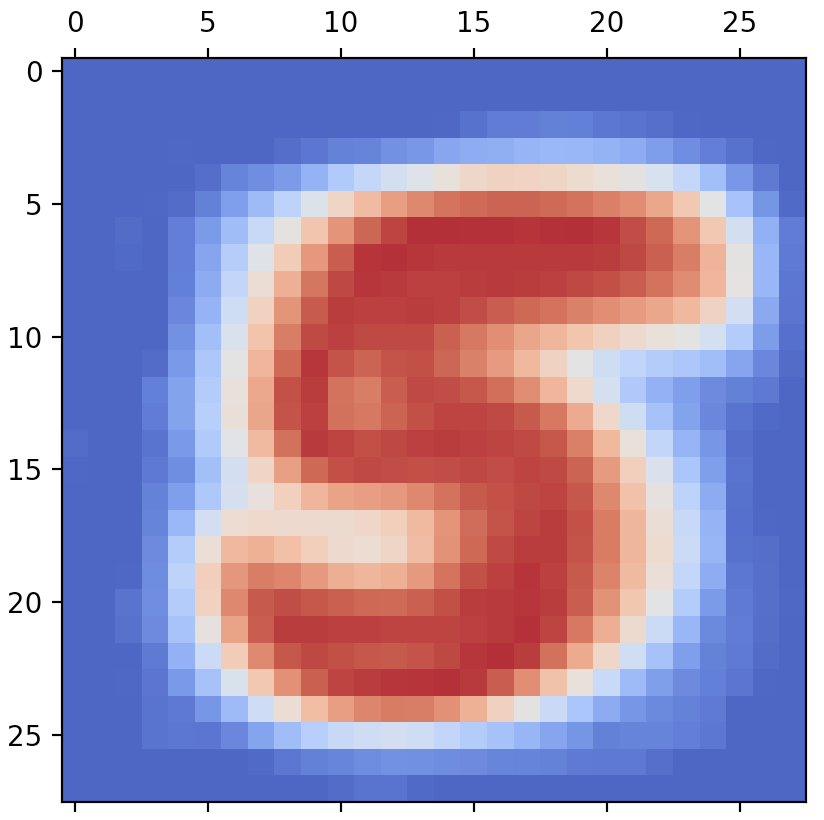
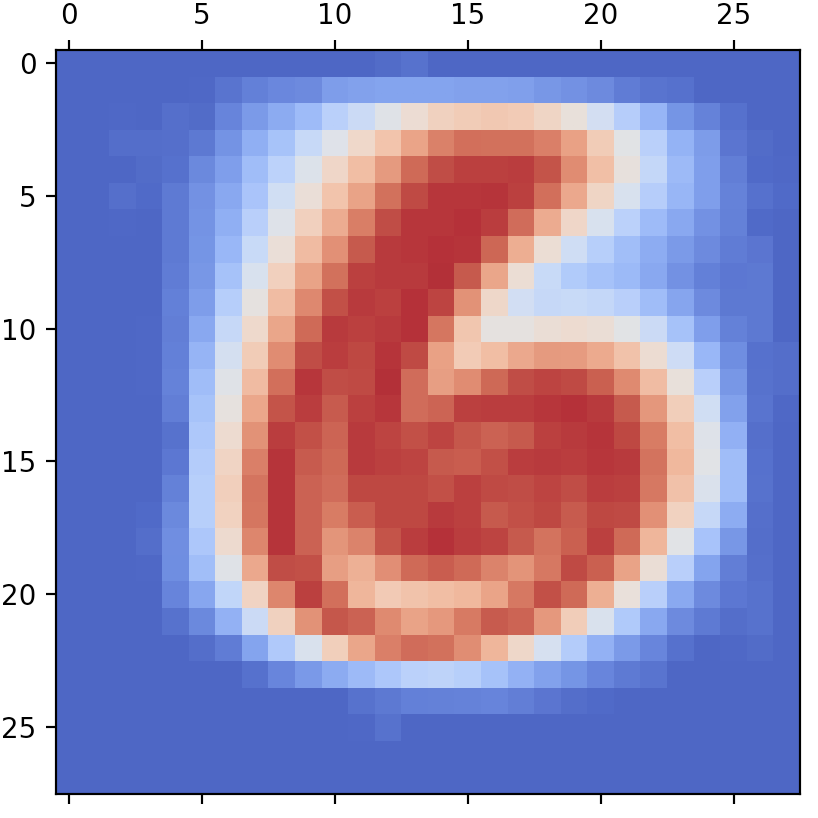
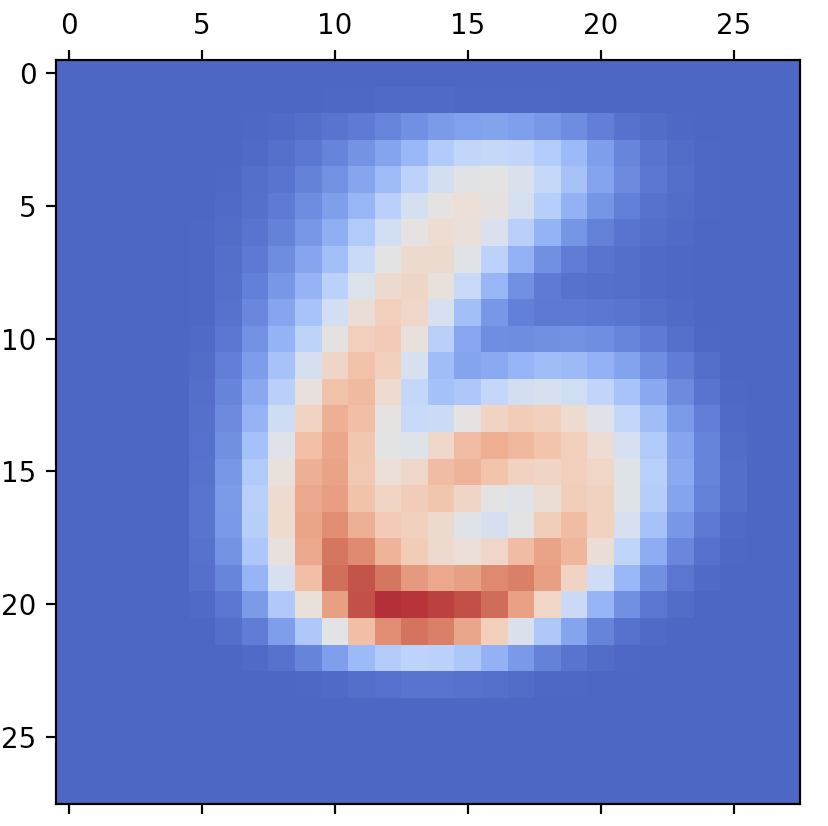
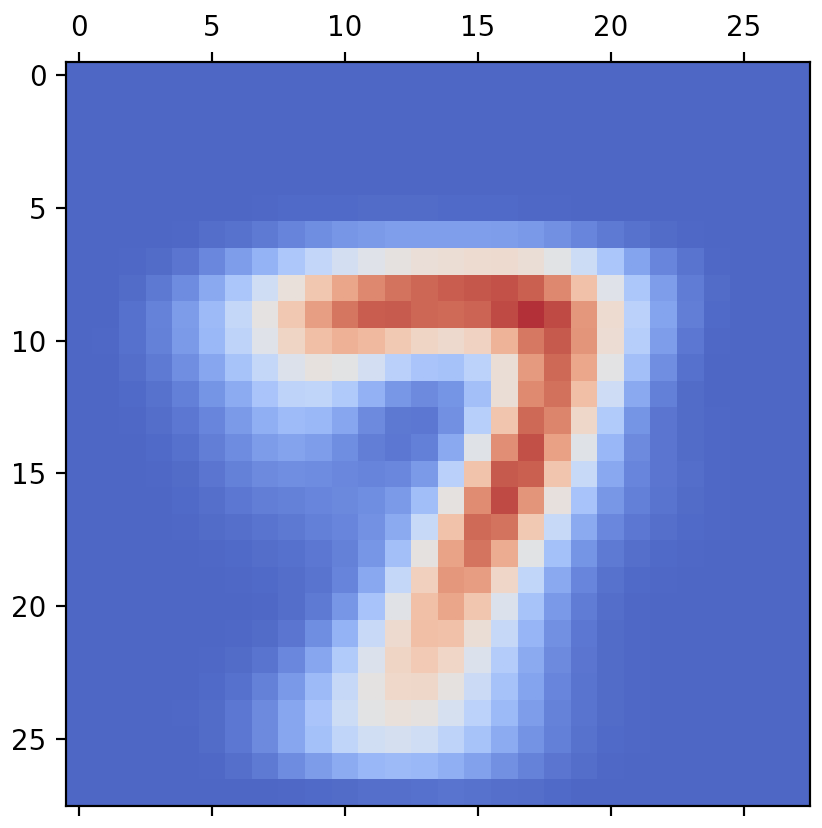
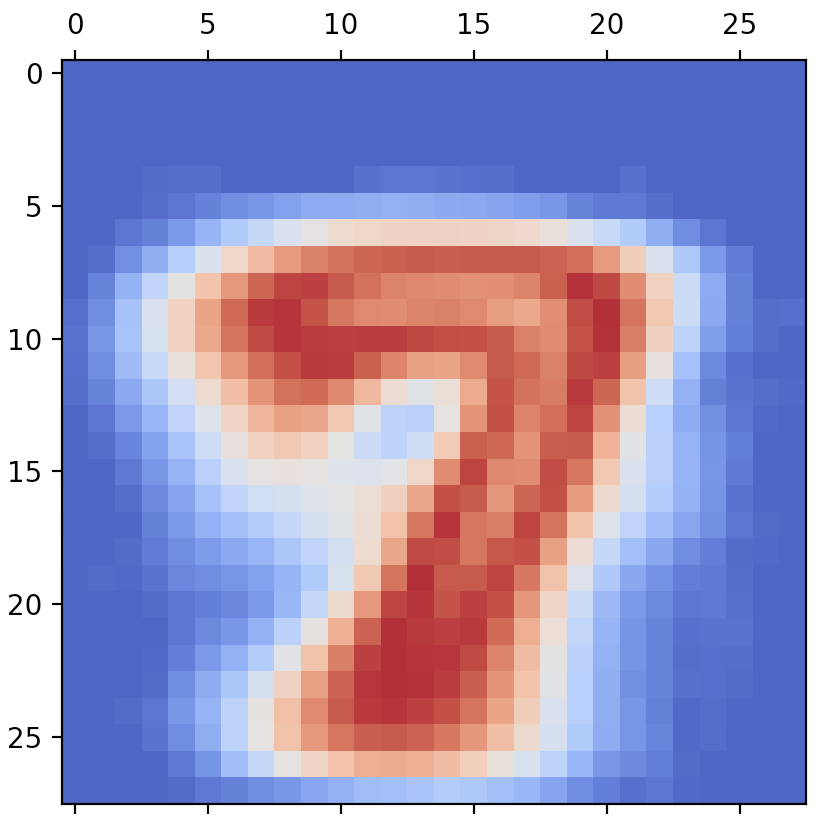
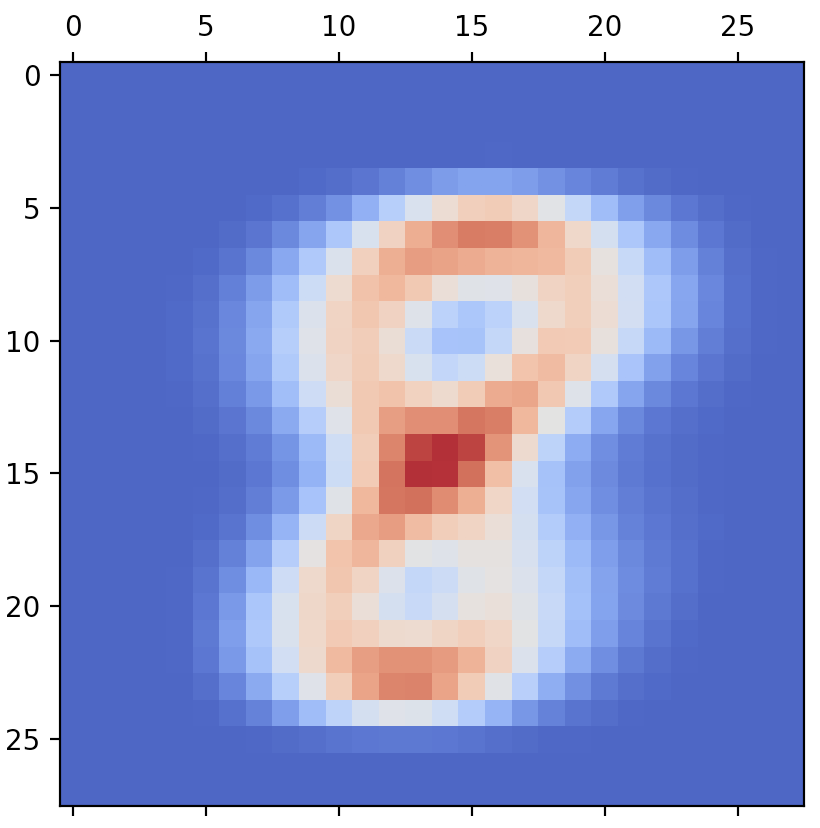
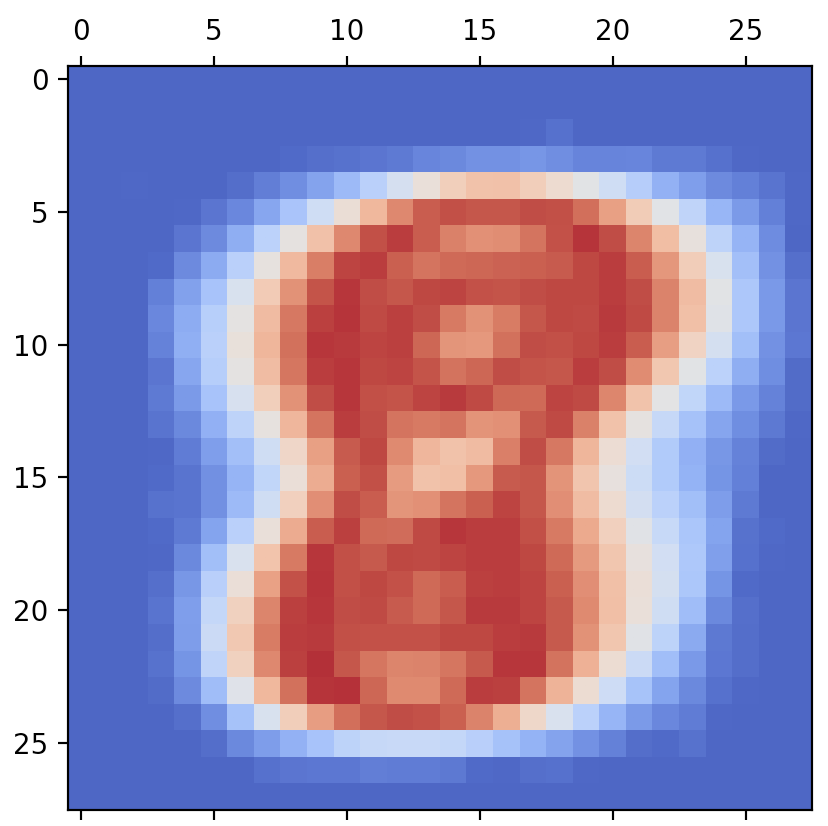
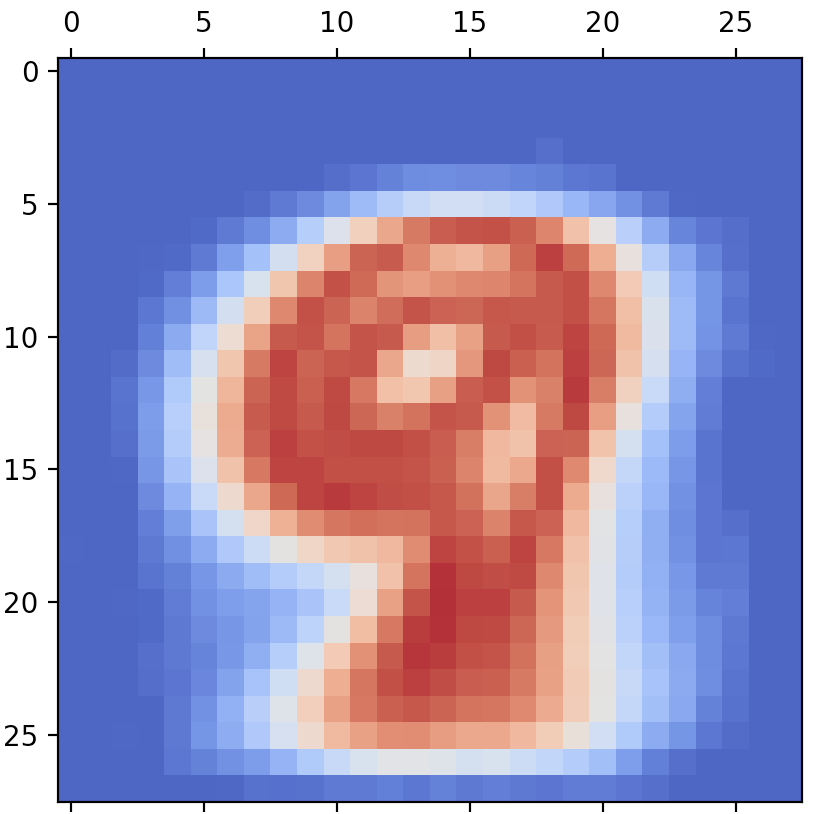
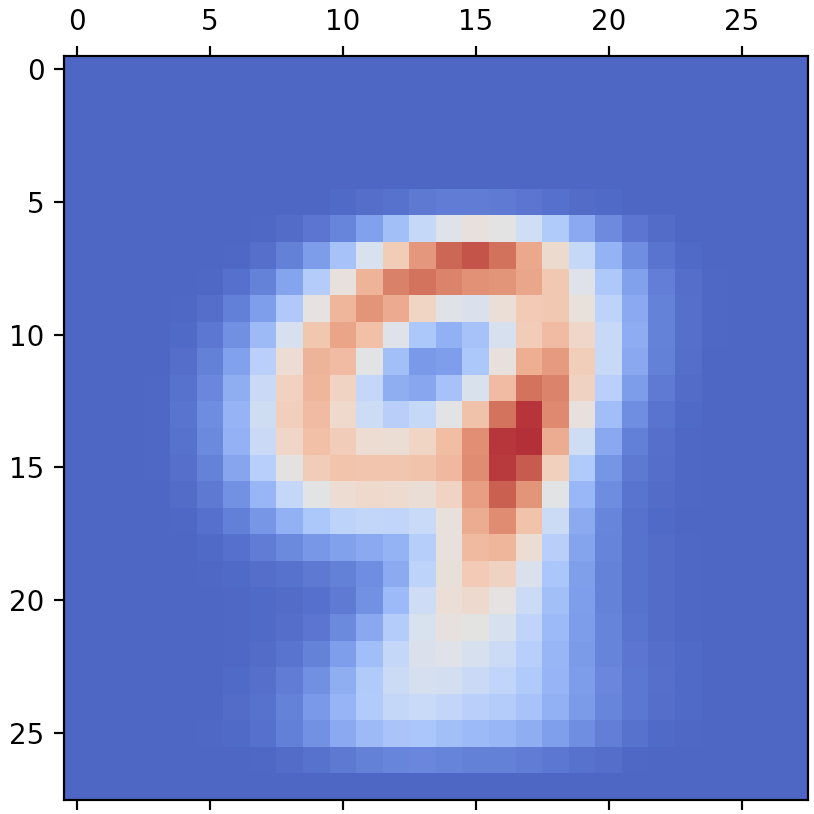
1. We assume that assume the feature vectors in each category in the training data have Gaussian distribution. But we cannot determine whether the distribution fits the data.
2. With naïve Bayes classifier, we assume strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence)  between the features.
3. We use Maximum likelihood to estimate sigma and mean. This means we choose to believe Frequentist’s view, the mean and sigma are constants. Perhaps the mean and sigma have their distribution. In this case, Bayesian parameter estimation can perform better.

**Reference:**

<http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

Zhou, Z.-H.: Machine Learning. Tsinghua University Press. ISBN 978-7-302-42328-7

**Appendix**

Code:

from tensorflow.examples.tutorials.mnist import input\_data  
import numpy as np  
from matplotlib import pyplot as plt  
from matplotlib import cm  
  
NUM\_CLASS = 10  
  
  
def test(set\_pred\_y, set\_y):  
 error = 0  
 for i in range(set\_y.shape[0]):  
 if set\_pred\_y[i] != set\_y[i]:  
 error += 1  
 return error / set\_y.shape[0]  
  
  
def separate(mnist):  
 classes = [[] for i in range(NUM\_CLASS)]  
 for i in range(mnist.train.labels.shape[0]):  
 classes[mnist.train.labels[i]].append(mnist.train.images[i])  
 mean = []  
 std = []  
 var = []  
 prior = []  
 for i in range(NUM\_CLASS):  
 classes[i] = np.array(classes[i])  
 mean.append(np.mean(classes[i], axis=0))  
 std.append(np.std(classes[i], axis=0))  
 var.append(np.var(classes[i], axis=0))  
 prior.append(classes[i].shape[0] / mnist.train.labels.shape[0])  
 return mean, std, var, prior  
  
  
def discriminant(mnist, mean, var, prior):  
 var[:, :] += 1e-10  
  
 joint\_log\_likelihood = []  
 for i in range(NUM\_CLASS):  
 log\_likelihood = - 0.5 \* np.sum(np.log(2. \* np.pi \* var[i, :]))  
 log\_likelihood -= 0.5 \* np.sum(((mnist.test.images - mean[i, :]) \*\* 2) /  
 (var[i, :]), axis=1)  
 joint\_log\_likelihood.append(log\_likelihood + prior[i])  
 result = np.argmax(joint\_log\_likelihood, axis=0)  
 return result  
  
  
def draw(mean, std):  
 for i in range(NUM\_CLASS):  
 mean[i] = mean[i].reshape(28, 28)  
 std[i] = std[i].reshape(28, 28)  
 plt.matshow(mean[i], cmap=cm.coolwarm)  
 plt.matshow(std[i], cmap=cm.coolwarm)  
  
  
def main():  
 mnist = input\_data.read\_data\_sets("MNIST\_data/")  
 mean, std, var, prior = separate(mnist)  
 # draw(mean,std)  
 pdf\_set = discriminant(mnist, np.array(mean), np.array(var), np.array(prior))  
 error = test(pdf\_set, mnist.test.labels)  
 print("classification error rate: ")  
 print(error)  
  
  
main()