Data handling: image data

ECE30007 Intro to AI Project

한동대학교
HANDONG GLOBAL UNIVERSITY

Contents

- Goal
- Intro to image data
    - What is image data
- Simple image processing techniques
- Image processing with MNIST data

Goal of image classification

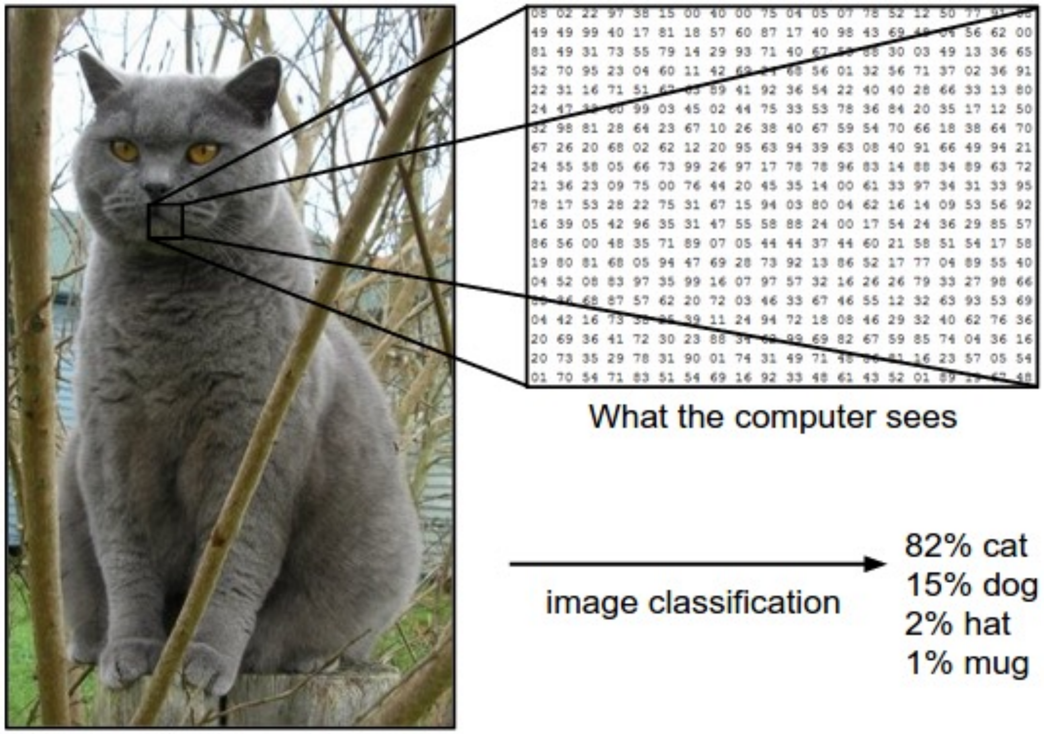
image

class

5

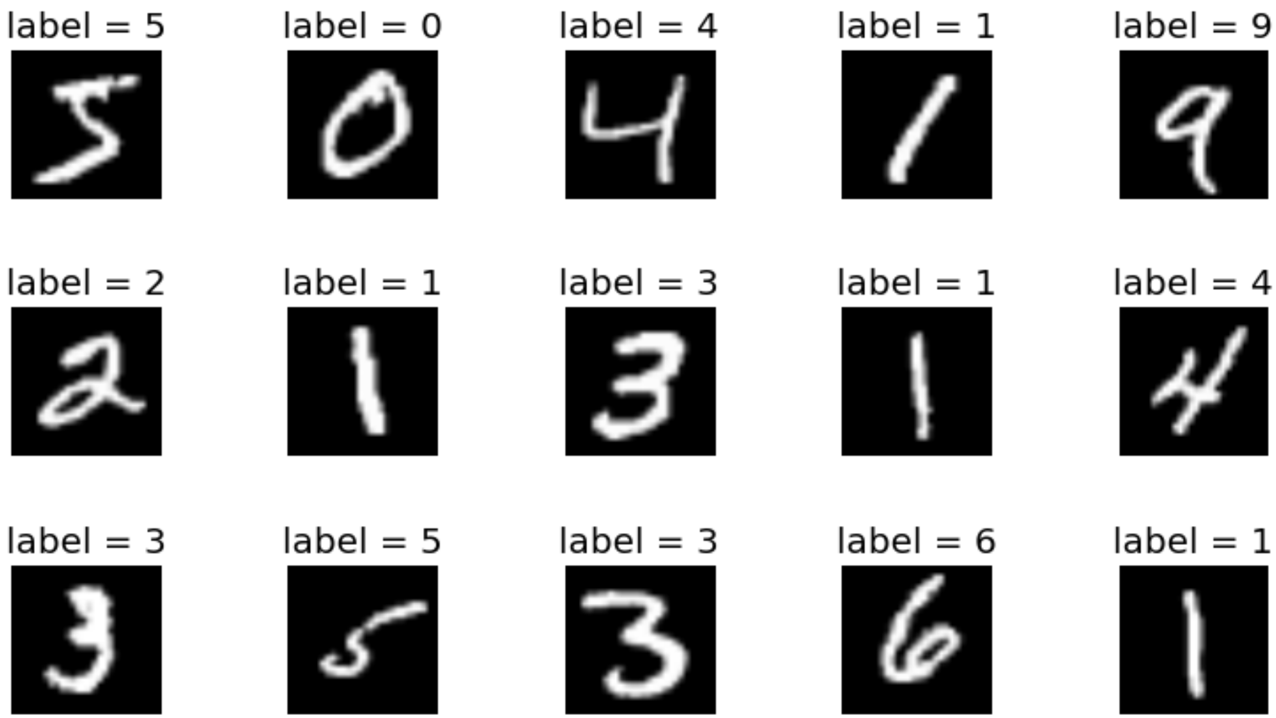
Oseok Hall

Intro to image data



https://3months.tistory.com/512

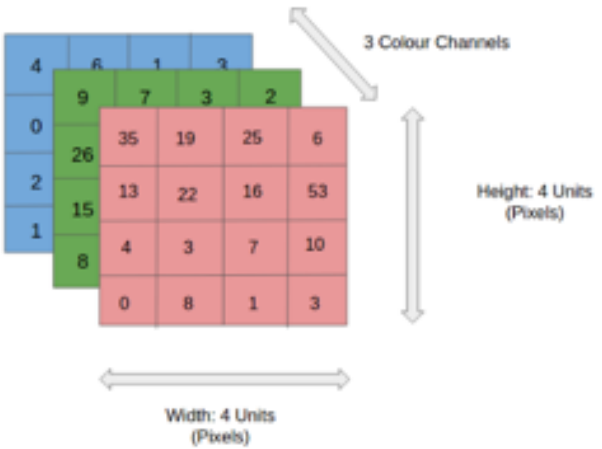
Intro to image data



What is image data

- Gray scale
  - 2-dim array
  - 0 represents black and as the number increases, the brightness increases and becomes white.

- RGB(Red-Green-Blue)
  - 3-dim array
  - It is expressed as a vector of three numbers that mean the brightness of three colors of red, green, and blue.



What is image data

- PIL(Python Imaging Library)
  - provides general image handling and lots of useful basic image operations

```python
from PIL import Image
from numpy import asarray
```

```python
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

- Load image

```python
In [69]: img = Image.open('./data/school.jpg')

         # asarray() class is used to convert PIL images into NumPy arrays
         numpydata = asarray(img)

         #  shape
         print(numpydata.shape)
         print(type(numpydata))
         plt.imshow(numpydata)

(666, 1000, 3)
<class 'numpy.ndarray'>

Out[69]: <matplotlib.image.AxesImage at 0x12e49cd10>
```

What is image data

- RGB(Red-Green-Blue)
  - example

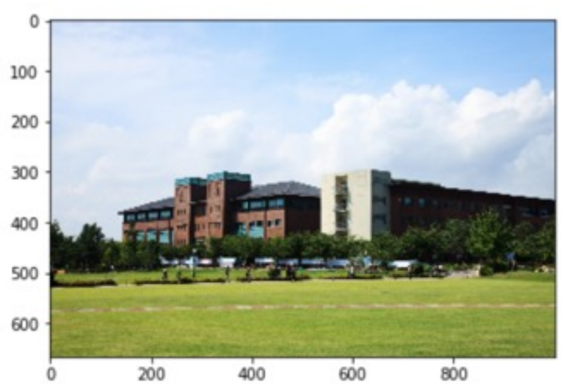
```
In [6]: plt.figure(figsize=(20,5))

        print('shape:', numpydata.shape)
        print('type:', type(numpydata))

        plt.subplot(1,4,1)
        plt.imshow(numpydata[300:600, 300:600, :])
        plt.axis("off")

        plt.subplot(1,4,2)
        plt.imshow(numpydata[300:600, 300:600, 0])
        plt.axis("off")

        plt.subplot(1,4,3)
        plt.imshow(numpydata[300:600, 300:600, 1])
        plt.axis("off")

        plt.subplot(1,4,4)
        plt.imshow(numpydata[300:600, 300:600, 2])
        plt.axis("off")

        shape: (666, 1000, 3)
        type: <class 'numpy.ndarray'>

Out[6]: (-0.5, 299.5, 299.5, -0.5)
```

Image processing methods

- Plot image *plt.imshow(np.ndarray)*
- Change color

```
In [84]: img = Image.open('./data/school.jpg').convert('L')

         # asarray() class is used to convert PIL images into NumPy arrays
         numpydata = asarray(img)

         # <class 'numpy.ndarray'>
         print(type(numpydata))

         #  shape
         print(numpydata.shape)
         plt.imshow(numpydata, cmap='gray')
```

```
<class 'numpy.ndarray'>
(666, 1000)
```



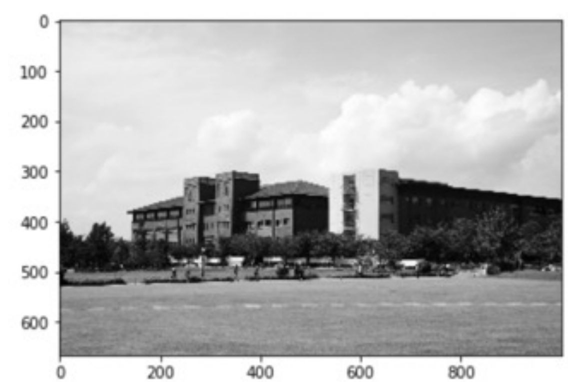


Image.convert('L'): convert into 256 level – gray image
cmap: colormap to use, try others 'Blues' or 'autumn'

Image processing methods

- Resize
  - Call resize() with a tuple giving the new size

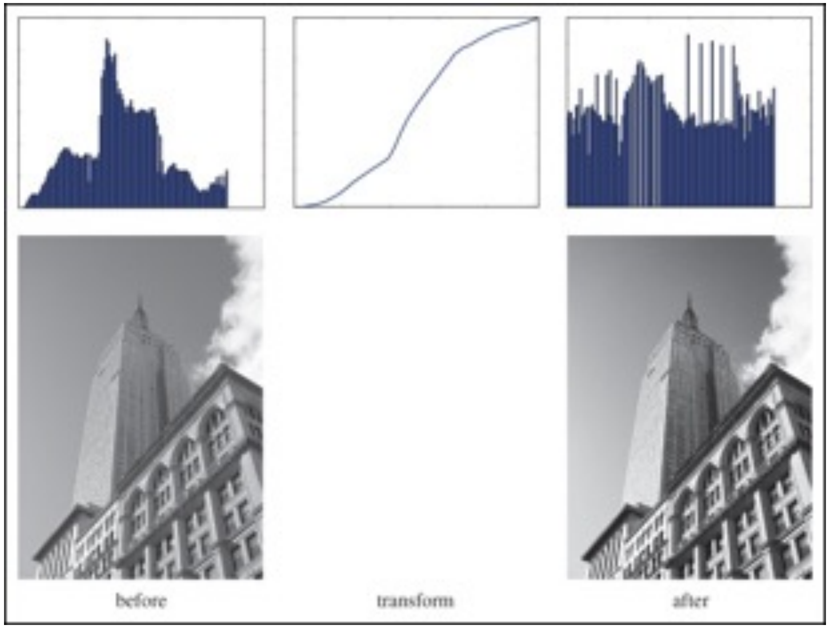
In [97]:
```
print(img.size)
img2 = img.resize((300, 200))
print(img2.size)
img2
```
```
(1000, 666)
(300, 200)
```
Out[97]:

- Rotate image
  - Call rotate() with counterclockwise angles giving the rotated image

In [100]:
```
img3 = img2.rotate(45)
img3
```
Out[100]:

Image processing methods (optional)

- Histogram Equalization
  - a very useful example of a gray-level transform
  - flatten the gray-level histogram of an image so that all intensities are as equally common as possible
  - normalize image intensity before other processing and increase image contrast



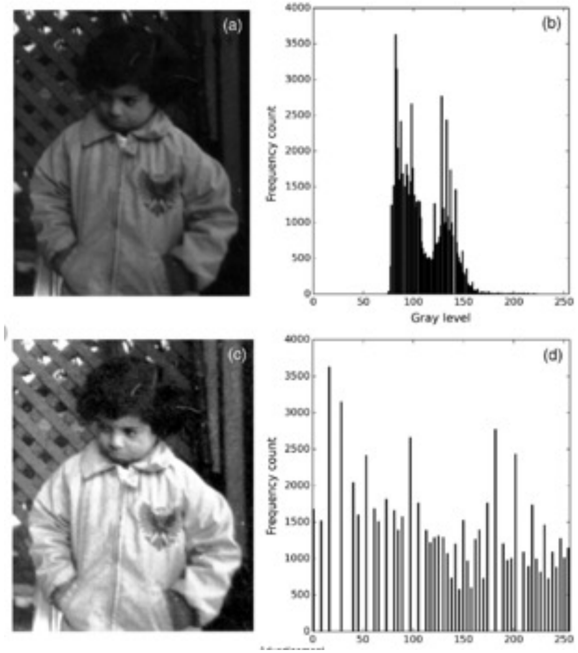
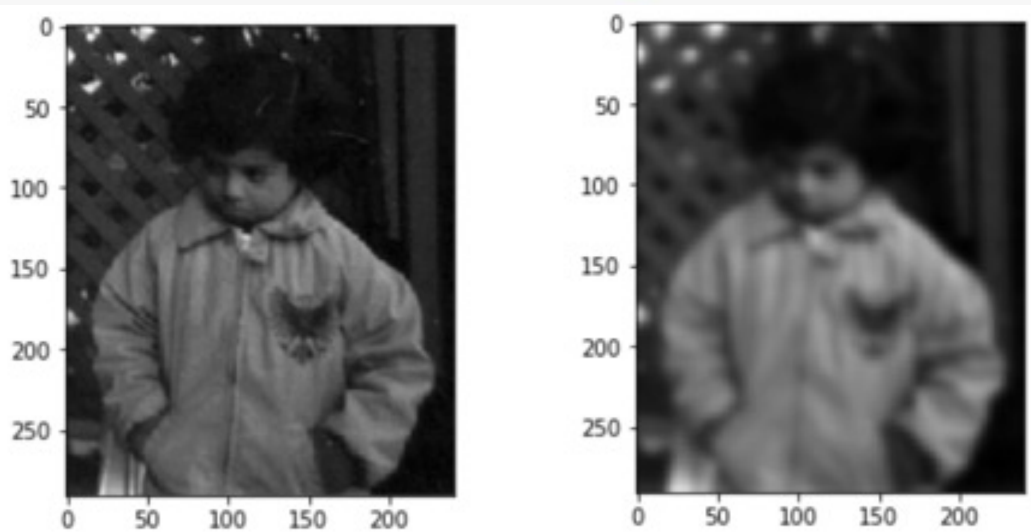


Image processing methods (optional)

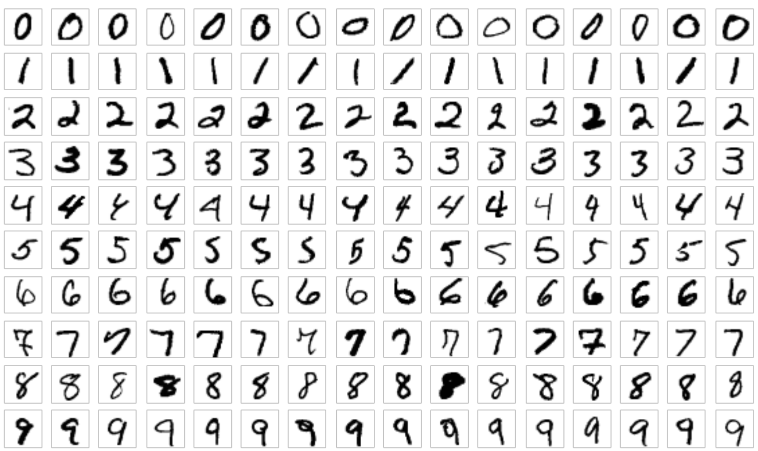
- Blurring images
  - Gaussian blurring of images
  - the (grayscale) image I is convolved with a Gaussian kernel
  - $$I_\sigma = I * G_\sigma \;, \quad G_\sigma = \frac{1}{2\pi\sigma} e^{-(x^2+y^2)/2\sigma^2}.$$

```
im = array(Image.open('./data/girl.jpg').convert('L'))
im2 = filters.gaussian_filter(im, 3)
```



MNIST data

- Handwritten digits
- A training set of 50,000 and a test set of 10,000.
- Each digit image was centered in a 28 x 28 image.
- 28x28 image is represented as a 784 dim vector.



http://yann.lecun.com/exdb/mnist/

Exercise(1) - Processing MNIST data

- Load MNIST data

```
import pickle
import pandas as pd

print('... loading data')
with open('data/mnist.pkl', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
```

```
train_x, train_y = train_set
test_x, test_y = test_set

train_x = pd.DataFrame(train_x)
train_y = pd.DataFrame(train_y, columns=['label'])
test_x = pd.DataFrame(test_x)
test_y = pd.DataFrame(test_y, columns=['label'])

train_data = pd.concat([train_x, train_y], axis=1)
test_data = pd.concat([test_x, test_y], axis=1)
```

```
print(train_data.shape, test_data.shape)
```

```
(50000, 785) (10000, 785)
```

Exercise(1) - Processing MNIST data

- Check the data and Plot the image

In [7]:
```
print(train_data.shape)
train_data.head()
```

```
(50000, 785)
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9

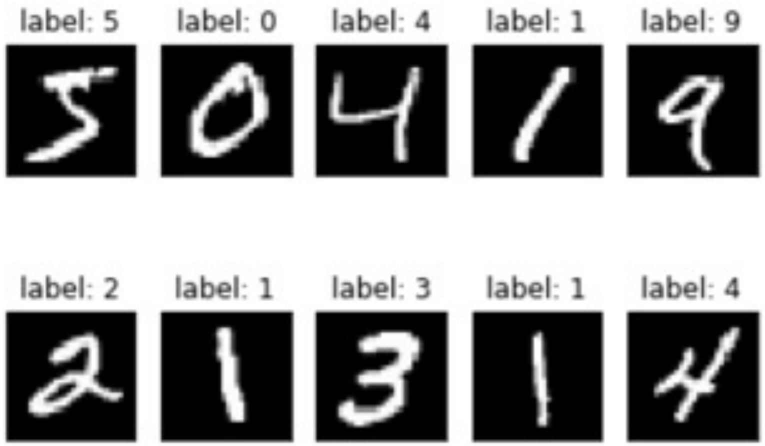
5 rows × 785 columns

Exercise(1) - Processing MNIST data

- Check the data and Plot the image

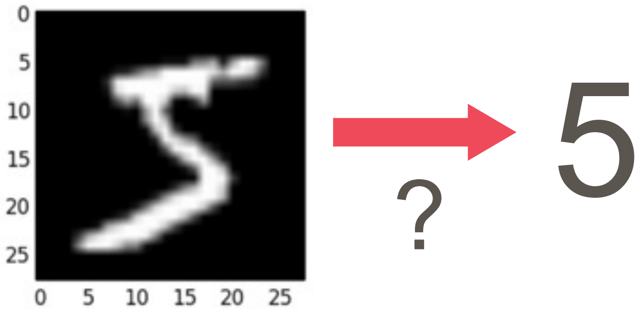
```
subset_images_X = train_data.iloc[:10, :-1]
subset_images_Y = train_data.iloc[:10, -1]
print(subset_images.shape)

for i, row in subset_images_X.iterrows():
    ax = plt.subplot(2, 5, i+1)
    pixels = row.values.reshape((28, 28))
    plt.imshow(pixels, cmap='gray')
    plt.title('label: {}'.format(subset_images_Y[i]))

    plt.xticks([]) # erase the ticks
    plt.yticks([])
```

'values' return *np.ndarray*



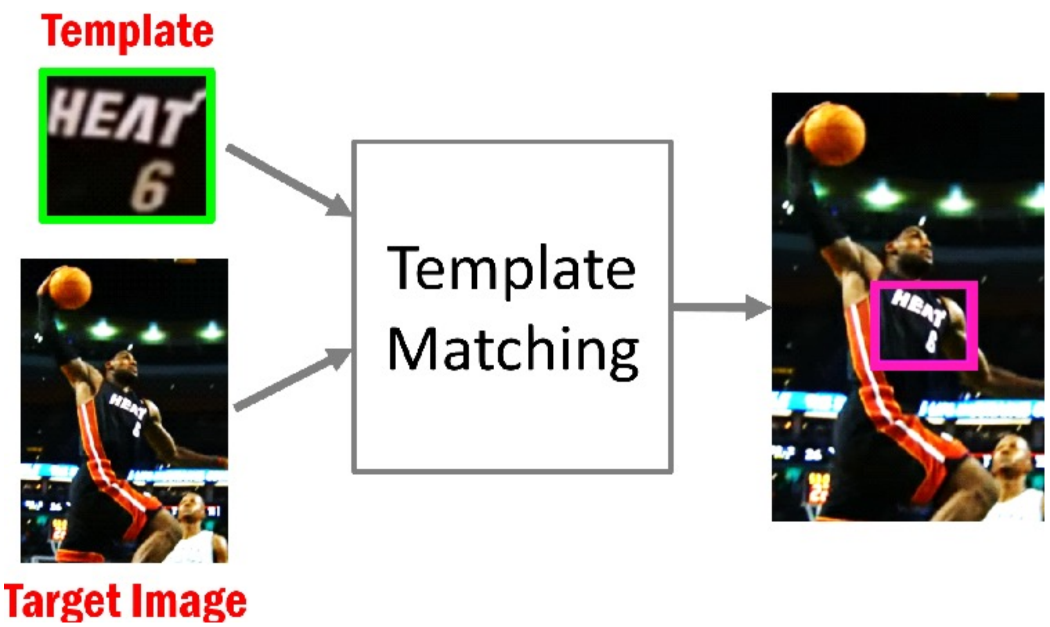
Exercises with MNIST

- Predict the label of test MNIST data
  1. Template matching
  2. Dimension Reduction
     - Principal Component Analysis (PCA): Dimension reduction Algorithm
     - Visualization of PCA
- Explained variance
- Image reconstruction



Template matching

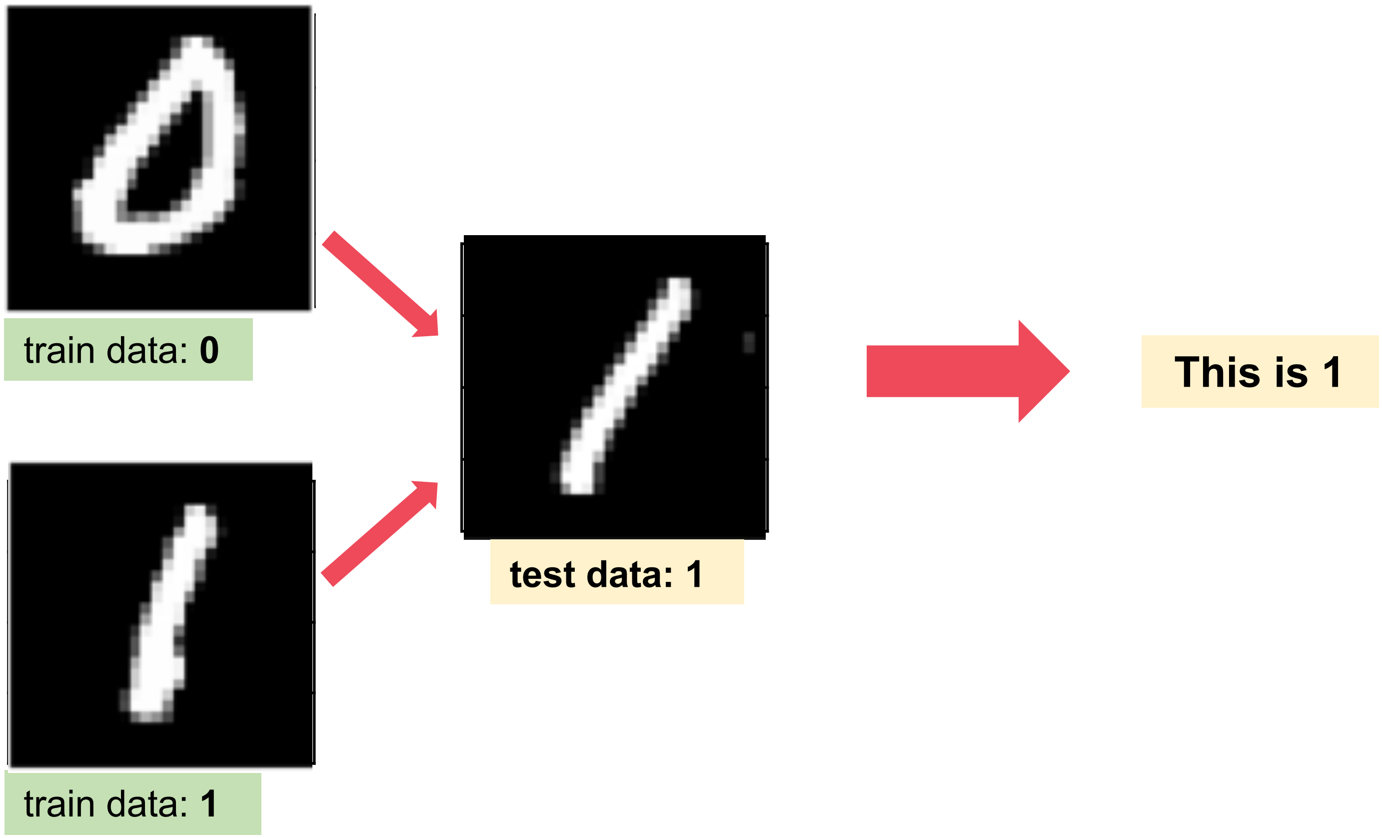
- What is template matching
  - a method for searching and finding the location of a template image in a larger image.



https://www.semanticscholar.org/paper/Template-Matching-with-Deformable-Diversity-Talmi-Mechrez/a8b8f5fca65d64617dba2dcc467603ffaaf7cdeb

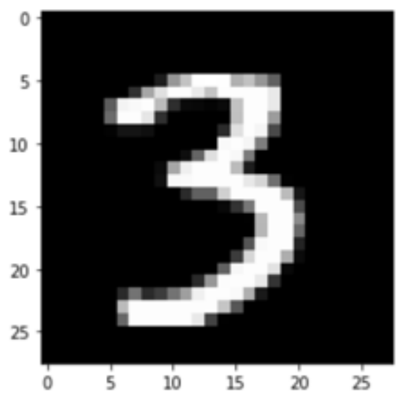
Template matching

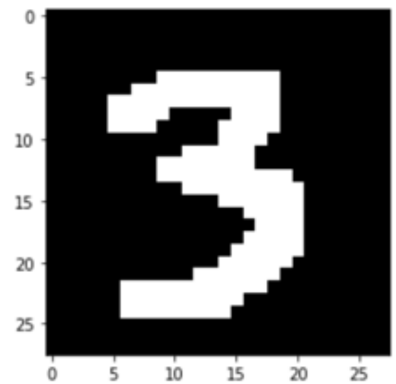
- For simplicity, let's use a binary image (represented by 0 or 1)
- A binary image in 784 dimensions (= 28 x 28)



Template matching

- Converting a gray scale *G* to a binary scale *B*.
- Ex) $B(x,y) = 0$ if $G(x,y) = 0$, otherwise $B(x,y) = 1$.

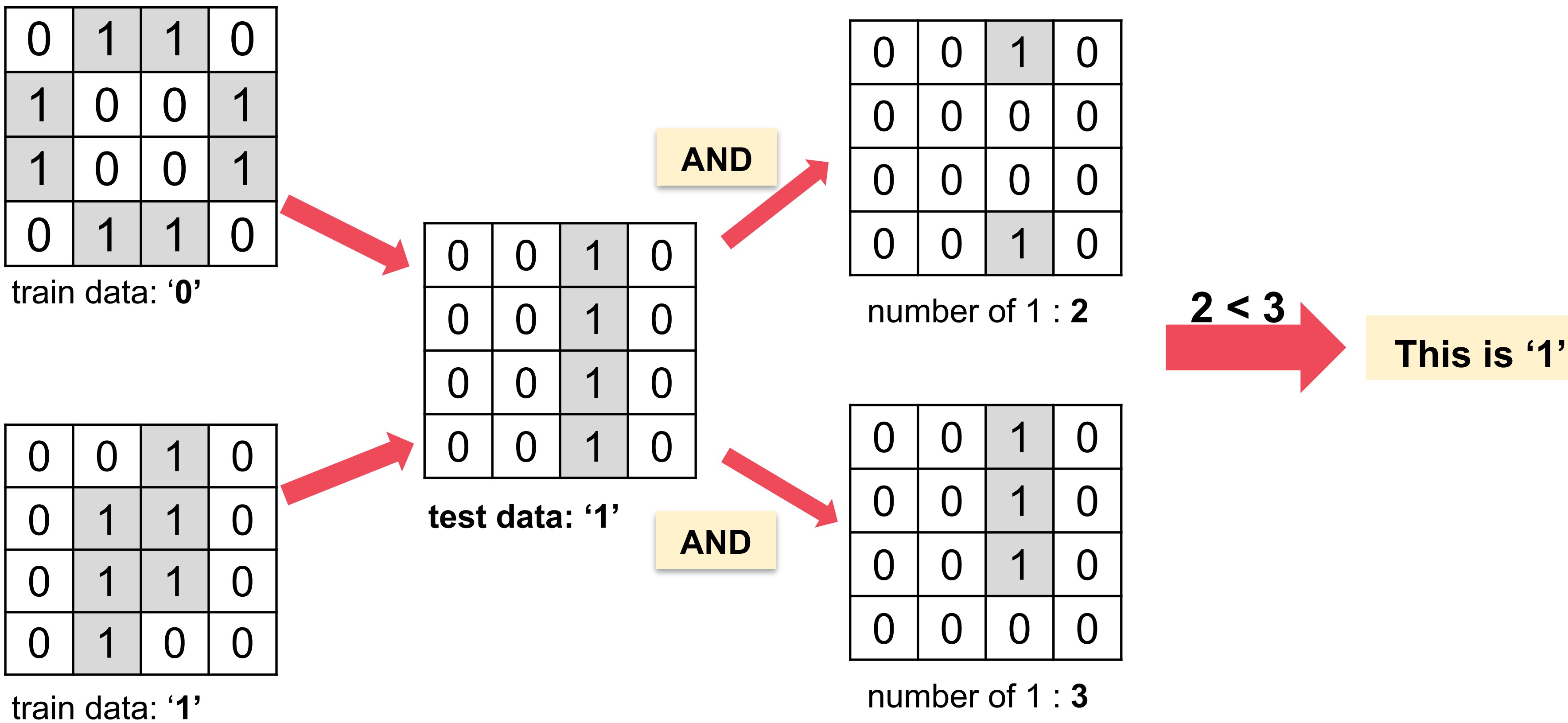





```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Template matching

- Template matching
  - 784-dim



Exercise(2) – Template matching with MNIST data

1) Pick two representative images from a set of '0' images and a set of '1' images in the training set.
2) Given a test image (should be '0' or '1' image), execute AND operation with the representative '0' image, and also '1' image.
3) Compute the number of 1s after AND operation
4) Assign the label yeilding more 1s.
5) Repeat the above steps for every '0' or '1' images in test set, then compute accuracy.

Exercise(2) – Template matching with MNIST data

- Result


```
Num of 1s after AND with 0-label train data:  94
Num of 1s after AND with 1-label train data:  47
0 -lable yields larger number, thus  1 th test image is classfied as  0  digit.
```

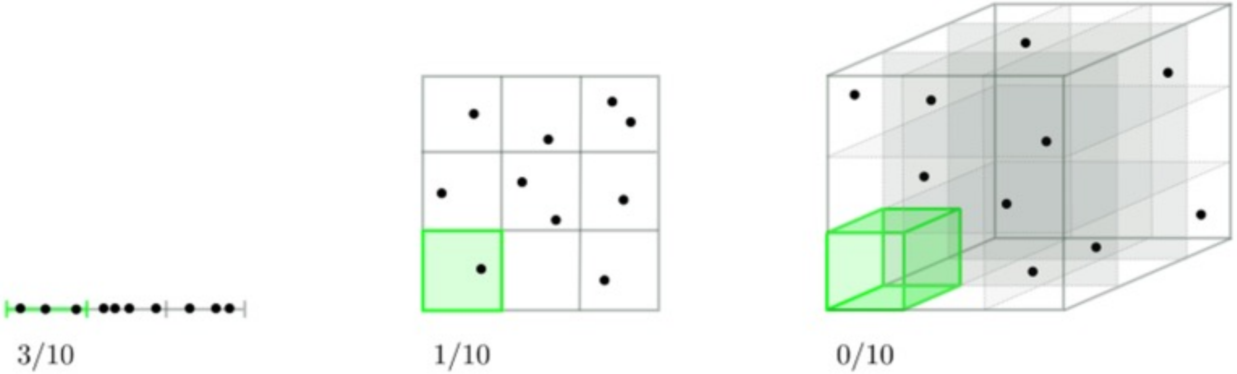
Example

Total accuracy on Test data is 0.9560

Accuracy

Dimension reduction

- High dimension & Curse of dimensionality
  - Machine learning often demands hundreds or thousands of dimensions.
    - MNIST data is 784 dimensional.
  - But, when dimension is too high, the volumes of the space increases so fast that the available data become sparse. curse of dimensionality!

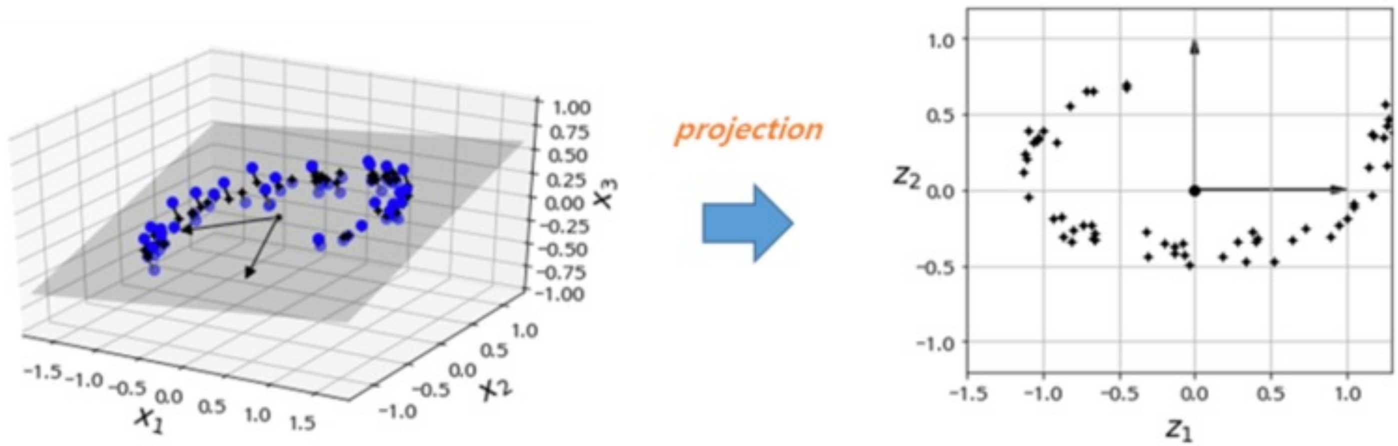


https://excelsior-cjh.tistory.com/167

How can we make this high-dimensional data more useful in machine learning and visualize the high-dimensional data? ➔ Dimension Reduction

Dimension reduction

- Dimension reduction
  - The representative method is “projection”

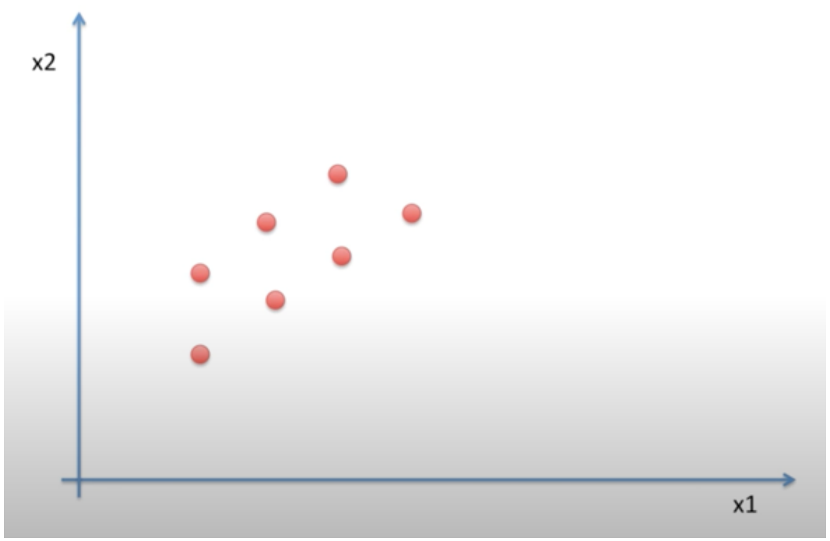


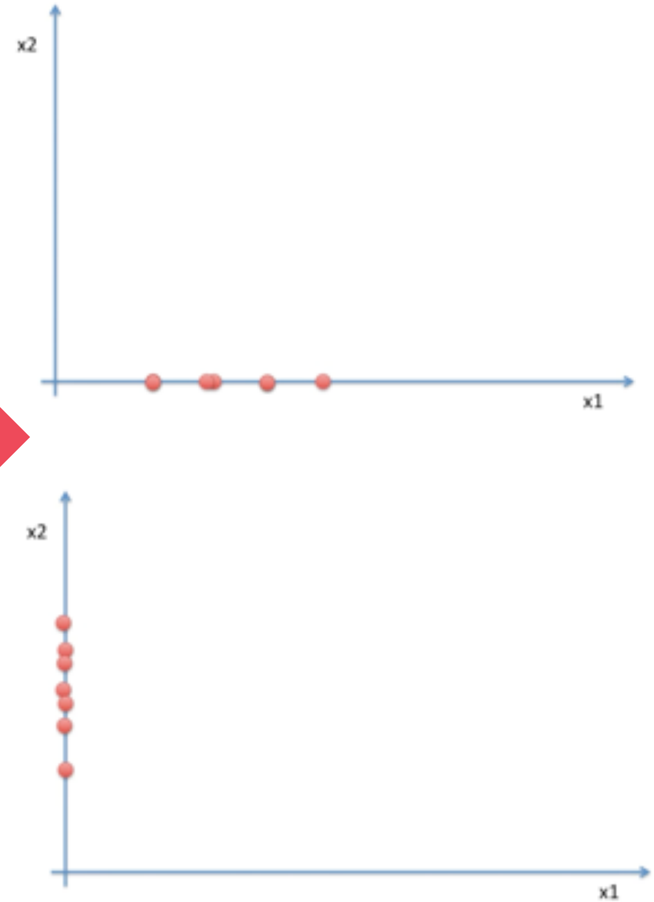
https://excelsior-cjh.tistory.com/167

  - Dimension reduction algorithm
    - **PCA(Principal Component Analysis)**
      - Reduce the number of variables (or features) of a data set, while preserving as much information as possible

Dimension reduction: PCA

- Example of PCA
  - Let's reduce 2-dim to 1-dim!



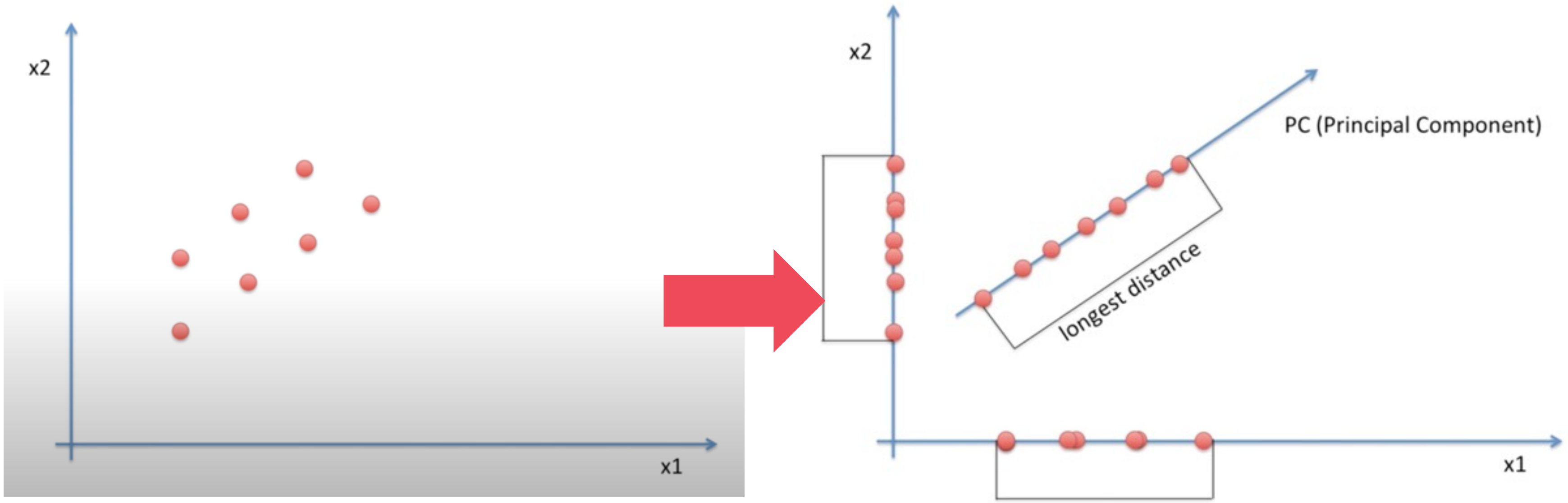


We miss information!

https://www.youtube.com/watch?v=xebPVQ1f7nM&t=273s

Dimension reduction: PCA

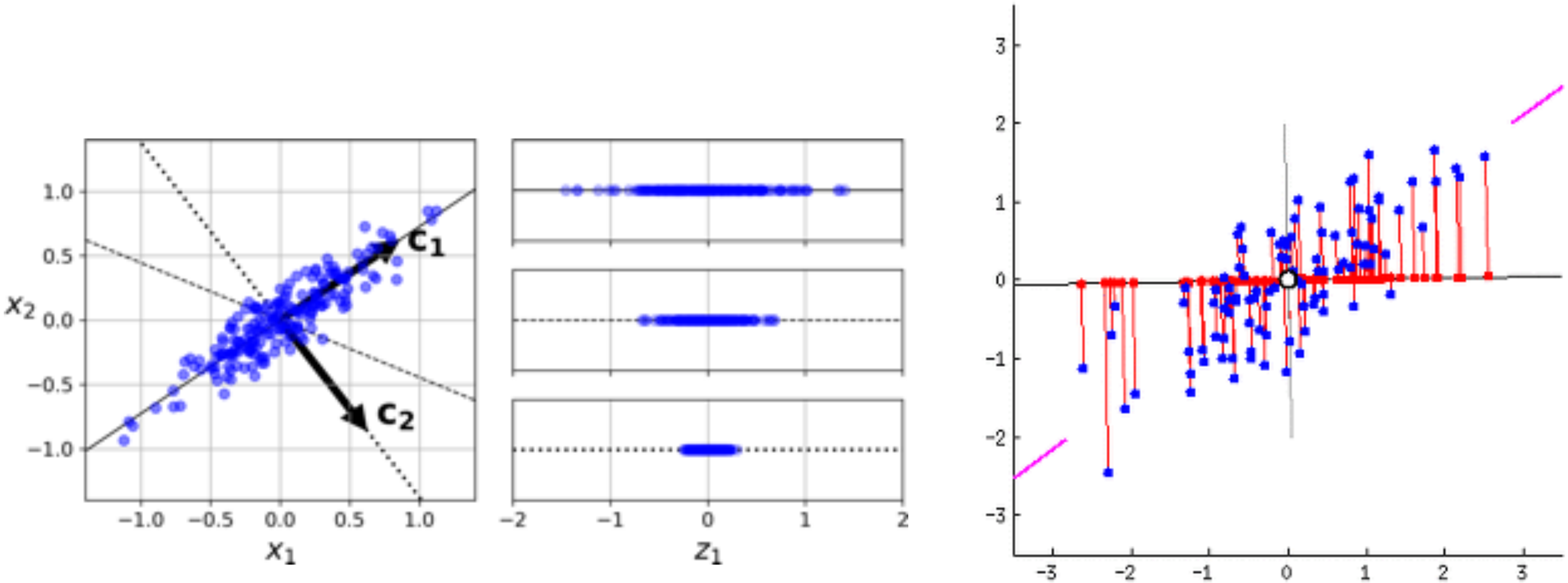
- Example of PCA
  - For example, let's reduce 2-dim to 1-dim!



The new axis on the right figure, directing the right-top, would be a better plane to maximize information (larger variance of samples).

Dimension reduction: PCA

- Example of PCA
  - For example, let's reduce 2-dim to 1-dim!



PC(Principal Component)
= Minimizing Reconstruction Error
= Eigenvector of Covariance Matrix

Dimension reduction: PCA

- PCA of Image data: 784-dim → 2-dim
  - Step
    1. **Calculate covariance matrix**
    2. **Calculate eigenvalues and eigenvectors from the covariance matrix**
    3. Project data onto two PCA planes (two eigenvectors)

Exercise (4) – Calculate covariance, Eigen value/vector

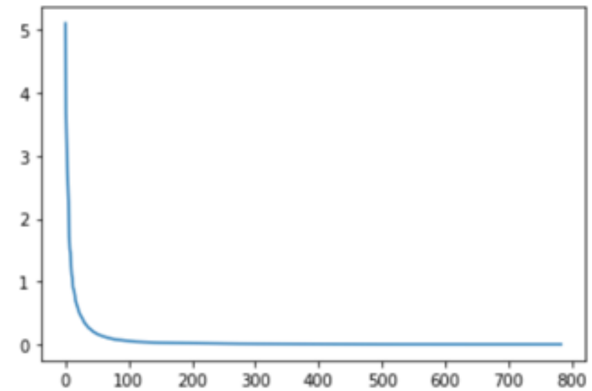
```
## dimension reduction scratch
train_x = train_data.iloc[:, :-1]
train_y = train_data.iloc[:, -1]

# calcualte covariance matrix, eigen values and eigen vectors
cov_matrix = np.cov(train_x.T)
eig_val, eig_vec = np.linalg.eig(cov_matrix)

# each column in eig_vec represents each eigen vector
# we want row vectors, thus perform transpose operation.
eig_vec = eig_vec.T
print('20 eigen values of 784 eigen values: ', eig_val[:20])

print('val:', eig_val.shape)
print('vec:', eig_vec.shape)
```

```
20 eigen values of 784 eigen values:  [5.10829281 3.70097988 3.25867822 2.8200844
2 2.54673474 2.26446711
 1.71820047 1.51312696 1.45150445 1.24028893 1.10062981 1.05915625
 0.89946813 0.88164617 0.82789811 0.78254504 0.69102204 0.66920675
 0.62200547 0.60339874]
val: (784,)
vec: (784, 784)
```

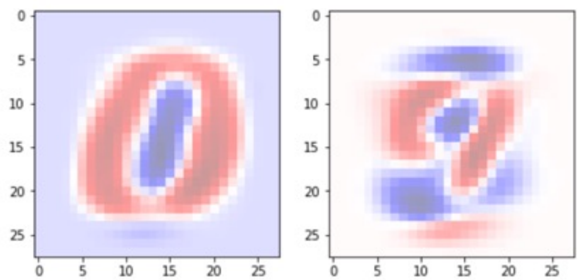
Dimension reduction: PCA

```
plt.plot(eig_val.real)
```

```
[<matplotlib.lines.Line2D at 0x7f800a9cf8e0>]
```



- PCA of Image data: 784-dim → 2-dim
  - Step
    1. Calculate covariance matrix
    2. Calculate eigenvalues and eigenvectors from the covariance matrix



⇒ Visualization of two eigen vectors with the largest eigen values.

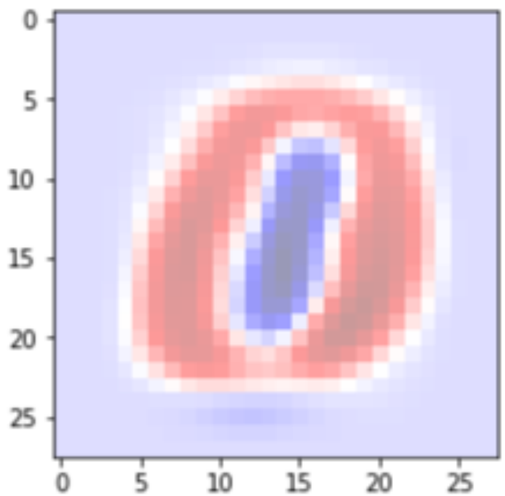
⇒ The two eigen vectors that best represent 784-dimensional MNIST data and have the least loss of information

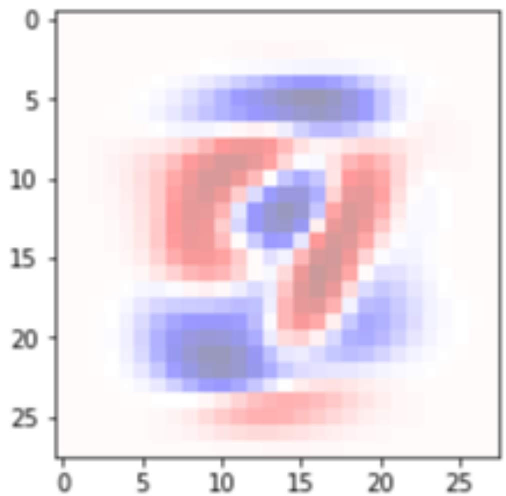
3. Project data onto two PCA planes (two eigenvectors)

https://colah.github.io/posts/2014-10-Visualizing-MNIST/

Exercise (5) – Visualize eigen vectors

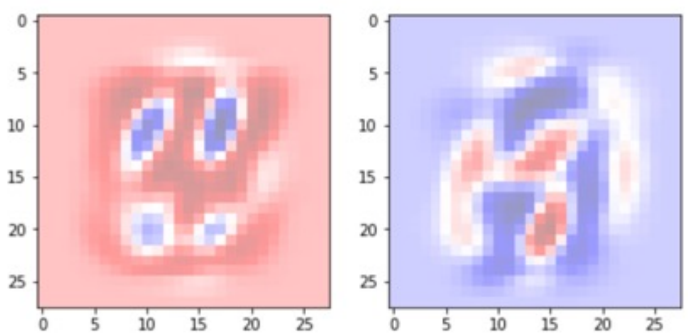
```
import matplotlib.pyplot as plt
import numpy as np

## Choose the 2 largest eigenvectors from eig_vec
good_vecs = ____________

plt.figure(figsize=(10, 5))
for i, vec in enumerate(good_vecs):
    vec = ____________
    vec = ____________

    ax = plt.subplot(2, 5, i+1)
    fig = plt.imshow(vec, alpha=0.4, cmap='seismic')
```





Dimension reduction: PCA

- PCA of Image data: 784-dim → 2-dim
  - Step
    1. Calculate covariance matrix
    2. Calculate eigenvalues and eigenvectors from the covariance matrix



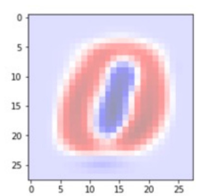
⇒ Visualization 2 eigen vectors with the 11th and 12th largest eigen values

⇒ Unlike the previous picture, you can see that MNIST data in 784 dimensions is not well represented and information loss is high.

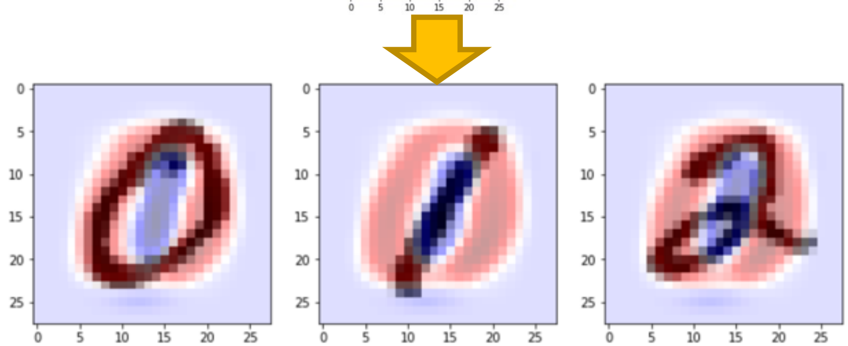
3. Project data onto two PCA planes (two eigenvectors)

Dimension reduction: PCA

- PCA of Image data: 784-dim → 2-dim
  - Step
    1. Calculate covariance matrix
    2. Calculate eigenvalues and eigenvectors from the covariance matrix
    3. Project data onto two PCA planes (two eigenvectors)



the largest eigenvector



inner product between
the largest eigenvector
and images ‘0’, ‘1’, and ‘2’

Dimension reduction: PCA

- PCA of Image data: 784-dim → 2-dim
  - Step
    1. Calculate covariance matrix
    2. Calculate eigenvalues and eigenvectors from the covariance matrix
    3. **Project data onto two PCA planes**

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.02294	-0.016899	-0.012524	-0.009867	-0.006321	0.0	0.0	0.0	0.0	5.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.02294	-0.016899	-0.012524	-0.009867	-0.006321	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.02294	-0.016899	-0.012524	-0.009867	-0.006321	0.0	0.0	0.0	0.0	4.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.02294	-0.016899	-0.012524	-0.009867	-0.006321	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.02294	-0.016899	-0.012524	-0.009867	-0.006321	0.0	0.0	0.0	0.0	9.0

	1st_pca	2nd_pca	label
0	-0.942522	4.901851	5.0
1	8.674059	7.854967	0.0
2	2.375962	-9.281880	4.0
3	-6.651204	3.597166	1.0
4	-5.128342	-2.863716	9.0

```
projected_x = train_x.dot(eig_vec[0:2].T).to_numpy()
```

Exercise (6) – Project data onto PCA planes

```
## Check the original array
train_data.head(3)
```

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4

3 rows × 785 columns

```
projected_x = 
print("new data points' shape: ", train_x.shape, "X", eig_vec[0:2].T.shape, "=", projected_x.shape)
print("projected_x.shape: ", projected_x.shape, " train_y.shape: ", train_y.shape)
```

```
new data points' shape:  (50000, 784) X (784, 2) = (50000, 2)
projected_x.shape:  (50000, 2)  train_y.shape:  (50000,)
```

```
new_coordinates = np.vstack((projected_x.T, train_y)).T
dataframe = 
dataframe.head(3)
```

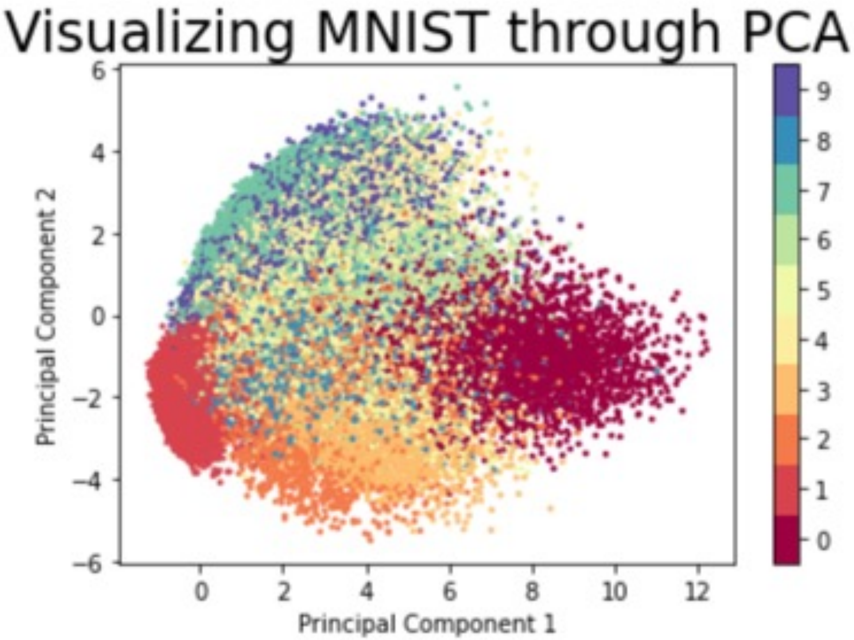
	1st_pca	2nd_pca	label
0	3.466855	-1.348822	5.0
1	6.926997	-1.353609	0.0
2	2.801635	1.445926	4.0

Exercise(7) - Visualize 2-dim MNIST data

```
plt.scatter(projected_x[:, 0].real, projected_x[:, 1].real, s=3, c=train_y, cmap='Spectral')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))

plt.title('Visualizing MNIST through PCA', fontsize=24);
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

In scatter()
s : Marker size
c : Marker colors

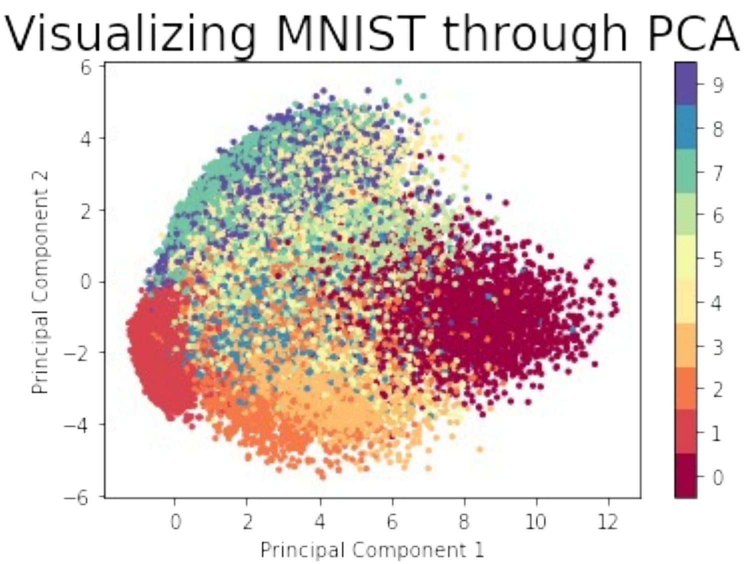
```
Text(0, 0.5, 'Principal Component 2')
```

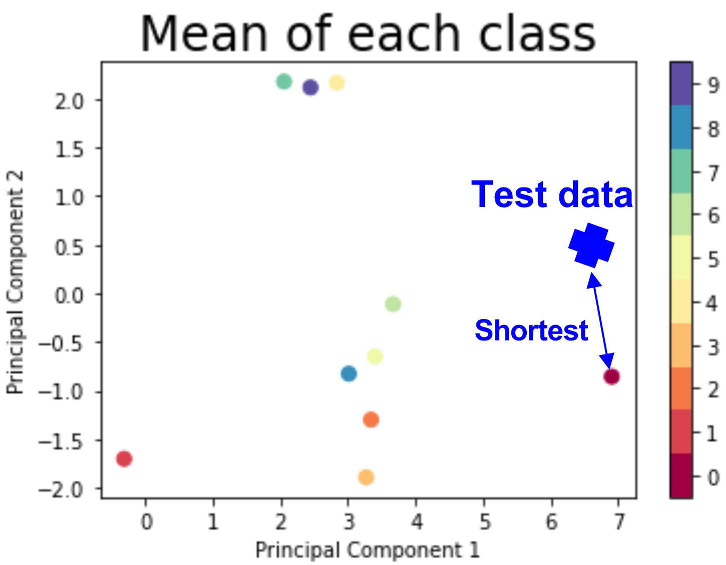


submit the exercises upto #7.

Prediction

- Then, how can we predict the label of test data?
  - Let's use the mean of each class (digit)

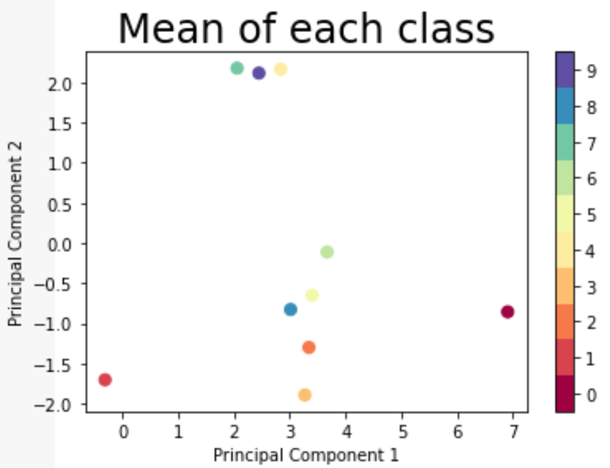




Mean of each class can be calculated by averaging over samples.

Advanced: Exercise(9) - Clustering & Visualization

```python
avg_point = []
X = []
Y = []

## calculate each label's mean value
for i in range(0, 10):
    mean_data = projected_x[train_data['label']==i].mean(axis=0)
    X.append(mean_data[0])
    Y.append(mean_data[1])

plt.scatter(X, Y, s=50, c=[0,1,2,3,4,5,6,7,8,9], cmap='Spectral')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))

plt.title('Mean of each class', fontsize=24);
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```




```python
di = {'x':X, 'y':Y}
avg_point_df = pd.DataFrame(di, columns=['x', 'y'])

avg_point_df
```

	x	y
0	6.905061	-0.855276
1	-0.311792	-1.701008
2	3.342736	-1.298553
3	3.269754	-1.890416
4	2.836307	2.165313
5	3.400326	-0.649264
6	3.668817	-0.110837
7	2.055240	2.177586
8	3.013611	-0.826211
9	2.445987	2.116579

Exercise(9) - Clustering & Visualization

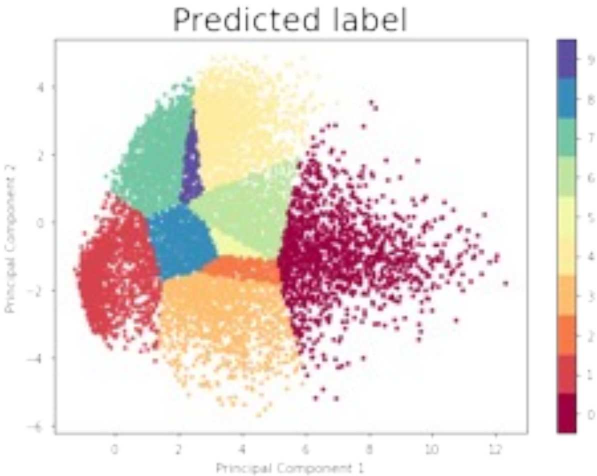
```
principal_df['pred_label'] = pred_label
principal_df['label'] = test_y

principal_df.head()
```

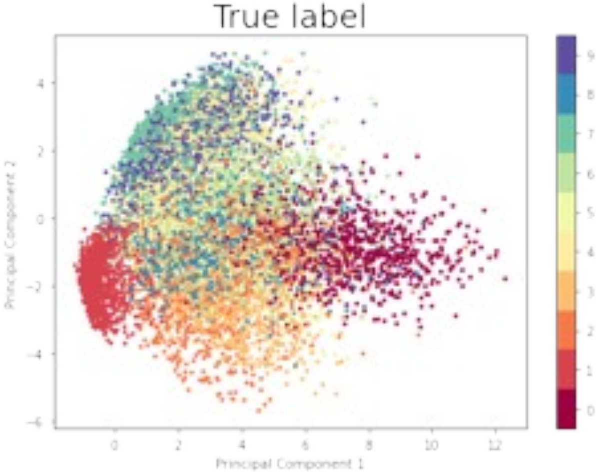
	PC 1	PC 2	pred_label	label
0	1.752303	2.798627	7	7
1	3.016745	-3.857681	3	2
2	-0.731315	-1.717984	1	1
3	7.836766	0.229712	0	0
4	3.765541	2.689341	4	4

Exercise(9) - Clustering & Visualization

```python
plt.figure(figsize=(15,5))


plt.subplot(1,2,1)
plt.scatter(principal_df['PC 1'], principal_df['PC 2'],
            s= 5, c=principal_df['pred_label'], cmap='Spectral')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title('Predicted label', fontsize=24);
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')


plt.subplot(1,2,2)
plt.scatter(principal_df['PC 1'], principal_df['PC 2'],
            s= 5, c=principal_df['label'], cmap='Spectral')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title('True label', fontsize=24);
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
Text(0, 0.5, 'Principal Component 2')
```





2-dim or 3-dim can be visualized.

Then, how about 100, 200 ... 784 dimension?

Exercise(9) - Clustering & Visualization

```
acc = len(principal_df[principal_df['label']==principal_df['pred_label']])/len(principal_df['label'])
print('\n Total accuracy on Test data is {:.4f}'.format(acc))
print('----------------------------------------')
print(principal_df)
```

```
 Total accuracy on Test data is 0.4348
----------------------------------------
         PC 1      PC 2  pred_label  label
0     1.752303  2.798627           7      7
1     3.016745 -3.857681           3      2
2    -0.731315 -1.717984           1      1
3     7.836766  0.229712           0      0
4     3.765541  2.689341           4      4
...        ...       ...         ...    ...
9995  4.165988 -2.221928           3      2
9996  5.197934 -2.853791           3      3
9997  2.032374  2.269081           7      4
9998  1.792168 -0.512353           8      5
9999  7.068762 -0.509858           0      6

[10000 rows x 4 columns]
```

Accuracy is calculated as
#correctely classified samples
Divided by #all samples

Exercise(9) - Clustering & Visualization

- Distributions look different. Why is it so?
- How can it be improved?

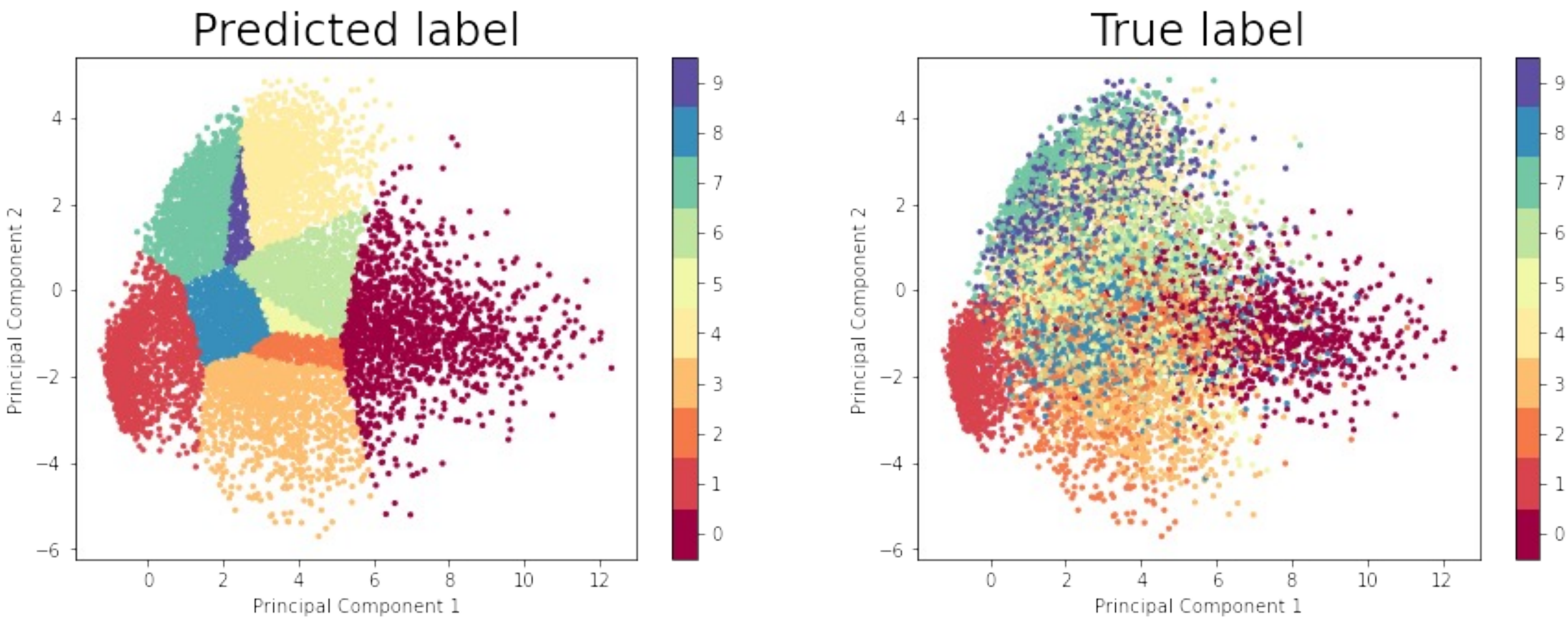


Image processing

- Explained Variance
  - The proportion of variance of the results projected on the axis of each principal component vector
  - The first k principal components PC1, ... , PCk explain $\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{q} \lambda_j}$ percent of the total variance.
  - The proportion of each eigen value

https://m.blog.naver.com/PostView.nhn?blogId=sw4r&logNo=221031465518&proxyReferer=https:%2F%2Fwww.google.com%2F

Image processing

- Explained Variance
  - Example
    - When we reduce the dimension from **3 to 2**, **how much information is preserved?**
    - variance(==eigenvalue) : [0.765 0.132 0.010]
    - explained variance ratio : [0.842 0.146 0.012]
      - 84.2% of the variance of dataset lies on the first principal component axis.
      - 14.6% of the variance of dataset lies on the second principal component axis.
      - In other words, you lose 1.2% of original data.

Exercise(10) - Explained variance

```
nor_val = [] # Normalized eigen values
explained_variances = [] # explained_variances: accumulated eigen value's proportion
sums = np.sum(eig_val)

for i, v in enumerate(eig_val):
    nor_val.append(v/sums)
    explained_variances.append(sum(nor_val))

dic = {'eig_val': eig_val, 'nor_val':nor_val, 'explained_variance':explained_variances}
ev = pd.DataFrame(dic)
print(ev.head())

plt.figure(figsize=(10,3))
plt.subplot(1,3,1); plt.plot(eig_val); plt.title("eig_val")
plt.subplot(1,3,2); plt.plot(nor_val); plt.title("nor_val")
plt.subplot(1,3,3); plt.plot(explained_variances); plt.title("explained_variances")
```

```
    eig_val   nor_val  explained_variance
0  5.108293  0.097444            0.097444
1  3.700980  0.070598            0.168042
2  3.258678  0.062161            0.230204
3  2.820084  0.053795            0.283999
4  2.546735  0.048581            0.332579
```

Let's calculate normalized eigen values and its cumulative distribution (= explained variance)

Exercise(10) - Explained variance

```
# Find the dimension when the 'explained variance ratio' is 95%

expvar_threshold = 0.95
for i, v in enumerate(explained_variances):
    if v >= expvar_threshold:
        print('#choson PCs : ', i+1)
        break


print('784 dim(pixel): {:.4f}% is explained in 784-dim.'.format(explained_variances[784-1]*100))
print('329 dim(pixel): {:.4f}% is explained in 329-dim.'.format(explained_variances[329-1]*100))
print('2 dim(pixel): {:.4f}% is explained in 2-dim.'.format(explained_variances[2-1]*100))
```

```
#choson PCs :  154
784 dim(pixel): 100.0000% is explained in 784-dim.
329 dim(pixel): 98.9805% is explained in 329-dim.
2 dim(pixel): 16.8042% is explained in 2-dim.
```

- In conclusion, more PCs are better to describe the original data.

Image processing

- Image reconstruction from compressed representation
  - Change image from a compressed representation

    back to an approximation of the original high dimensional data

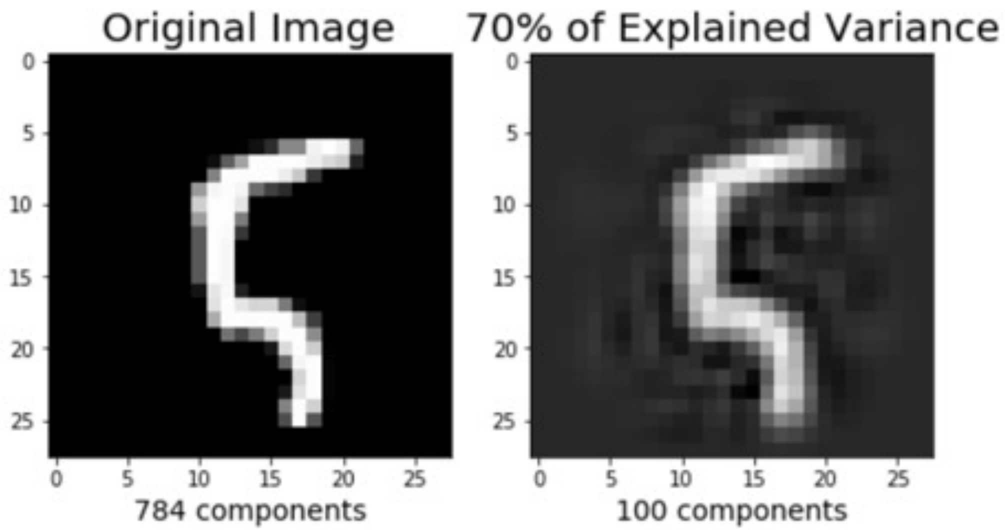


Image processing

- PCA with library

```
from sklearn.decomposition import PCA
eig_dim = [2, 5, 10]

for d in eig_dim:
    eig_train_data = PCA(n_components = d).fit_transform(train_x)
    print(f"dim: {d}, data₩'s shape: {eig_train_data.shape}")
```

```
dim: 2, data's shape: (50000, 2)
dim: 5, data's shape: (50000, 5)
dim: 10, data's shape: (50000, 10)
```

scikit learn

scikit-learn

- Simple and efficient tools for predictive data analysis
- Built on NumPy, SciPy, and matplotlib

Exercise(11) - Image reconstruction

```
plt.figure(figsize=(10, 5));

# Original image
plt.subplot(1, 3, 1);
plt.imshow(train_x.iloc[100, :].values.reshape(28,28), cmap = 'gray')
plt.title('Original Image', fontsize = 20);

# Image reconstruction from PCA data
pca = PCA(n_components = 200); eig_train_data = pca.fit_transform(train_x)
approximation = pca.inverse_transform(eig_train_data)
plt.subplot(1, 3, 2);
plt.imshow(approximation[100, :].reshape(28, 28), cmap = 'gray')
plt.title('N = 200', fontsize = 20);

pca = PCA(n_components = 100); eig_train_data = pca.fit_transform(train_x)
approximation = pca.inverse_transform(eig_train_data)
plt.subplot(1, 3, 3);
plt.imshow(approximation[100, :].reshape(28, 28), cmap = 'gray')
plt.title('N = 100', fontsize = 20);
```