Assignment2

April 25, 2025

CSE 242 Assignment 2, Spring 2025

3 Questions, 100 pts

`Your name: Mengxiao Hu         Student ID:mhu110(2172399)`

0.1 Instruction

- Submit your assignments onto **Canvas** by the due date. Upload a zip file containing:
  (1) The saved/latest .ipynb file.
  (2) Also save your file into a pdf version, if error appears, save an html version instead (easy to grade for written questions).
  (3) All other materials to make your .ipynb file runnable.

  For assignment related questions, please reach TA or grader through Slack/Discord/Email.
- This is an **individual** assignment. All help from others (from the web, books other than text, or people other than the TA or instructor) must be clearly acknowledged.
- Most coding parts can be finished with only 1-2 lines of codes.
- Make sure you have installed required packages: scikit-learn.
- Double click to edit each markdown cell.

0.2 Objective

- **Task 1:** Maximum likelihood estiamtion (math)
- **Task 2:** Linear Regression (with Scikit-learn)
- **Task 3:** Principle Component Analysis (with Scikit-learn)

1 Question 1 (Maximum likelihood estiamtion and Posterior, 20 pts)

Assume we have a coin that has some unknown probability h of coming up heads (and probability $1-h$ of coming up tails). If the coin is flipped five times getting three heads and two tails (**HHHTT**) then:

(a – 10pts) What is the maximum likelihood estimate for h?

(b – 10pts) Assume that (before flipping the coin) we have a prior density $p(h)$ for the various values of $h \in [0,1]$. Give the formula for the posterior probability density $p(h \mid \mathbf{HHHTT})$ as a function of the prior $p(h)$.

For Question 1(b), the solution is acceptable if the formula contains an integral.

Solution (a): The likelihood function is:

$$L(h) = P(\mathbf{HHHTT} \mid h) = h^3 \cdot (1-h)^2$$

To maximize this function, we take the derivative with respect to h and set it equal to zero:

$$\frac{d}{dh}L(h) = 3h^2 \cdot (1-h)^2 - 2h^3 \cdot (1-h) = 0$$

Which gives: $h = 0$, $h = 1$, or $h = \frac{3}{5}$.

As $h = 0$ and $h = 1$ both give a likelihood of 0 while $h = \frac{3}{5}$ gives a positive likelihood, the maximum likelihood estimate for h is $\frac{3}{5}$.

Solution (b): Given that:

$$p(h \mid \mathbf{HHHTT}) = \frac{P(\mathbf{HHHTT} \mid h) \cdot p(h)}{P(\mathbf{HHHTT})}$$

$$P(\mathbf{HHHTT}) = \int_0^1 P(\mathbf{HHHTT} \mid h) \cdot p(h)\, dh = \int_0^1 h^3 \cdot (1-h)^2 \cdot p(h)\, dh$$

We can infer that:

$$p(h \mid \mathbf{HHHTT}) = \frac{h^3 \cdot (1-h)^2 \cdot p(h)}{\int_0^1 h^3 \cdot (1-h)^2 \cdot p(h)\, dh}$$

2 Question 2 (Linear Regression, 40 pts)

In this question, you will be using **Scikit-learn** to empirically apply the Linear Regression model. We will adopt an advertisement dataset **Advertising.csv** from Kaggle.

2.0.1 Reading data using Pandas

[1]:

```python
# Read the dataset you will be working on
# The dataframe loaded with pandas is named as data

import pandas as pd
data = pd.read_csv('Advertising.csv', index_col=0)  # Modify your data path accordingly.

# Take a look at the first 5 rows
data.head()
```

```
[1]:        TV  Radio  Newspaper  Sales
     1  230.1   37.8       69.2   22.1
     2   44.5   39.3       45.1   10.4
     3   17.2   45.9       69.3    9.3
     4  151.5   41.3       58.5   18.5
     5  180.8   10.8       58.4   12.9
```

```
[2]: data.shape
```

```
[2]: (200, 4)
```

2.0.2 Features and responses

What are the features? - **TV:** advertising dollars spent on TV for a single product in a given market (in thousands of dollars) - **Radio:** advertising dollars spent on Radio - **Newspaper:** advertising dollars spent on Newspaper

What is the response? - **Sales:** sales of a single product in a given market (in thousands of items)

What else do we know? - There are 200 **observations**, and each observation (each row) is a single market. - **Your main task**: predict the sales of a single product in a given market. - Since the response variable is continuous, this is a **regression** problem.

2.0.3 Linear regression

Pros: fast, no tuning required, highly interpretable

Cons: adopting the linear regression is unlikely to always generate the best predictive accuracy, since Linear Regression presumes a linear relationship between the features and response. This assumption may not always make sense. (But if one can *transform* the features so that do have a linear relationship, sometimes progress can be made, for example by taking logs.)

Expression:

- y: the response
- β_0: the intercept
- β_1: the coefficient for x_1 (the first feature)
- β_i is the coefficient for x_i (the ith feature, $i \in \{1, 2, ..., n\}$)

For this regression task:

$\text{Sales} = \beta_0 + \beta_1 \times \text{TV} + \beta_2 \times \text{Radio} + \beta_3 \times \text{Newspaper}$

The β values ($\beta = [\beta_0, \beta_1, \beta_2, \beta_3]$) are called the **model coefficients**. These values are "learned" during the model fitting step using the "least squares" criterion. Then, we can make use of the fitted model to make predictions!

2.0.4 Prepare the dataset for training use

- Scikit-learn expects X (feature matrix) and y (response vector) to be NumPy arrays.
- Pandas is built on top of NumPy, and exposes numpy methods. Thus, X can be a pandas DataFrame, y can be a pandas Series!

[3]:

```
# Prepare the feature X
X = data[['TV', 'Radio', 'Newspaper']]

# check the type and shape of X
print(type(X))
print(X.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
(200, 3)
```

[4]:

```
# Prepare the response Y
y = data['Sales']
# check the type and shape of y
print(type(y))
print(y.shape)
```

```
<class 'pandas.core.series.Series'>
(200,)
```

2.1 Question 2.1 (Split the dataset into train and test parts, 5 pts)

For features X and response y, use **sklearn** to perform the the splitting of dataset: 80% for train data, 20% for the test data.

[5]:

```
################# Your answer for Question 2.1   #################

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)

################# Your code above #################
```

Double check the shape of train and test data:

[6]:

```
# default split is 80% for training and 20% for testing
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(160, 3)
(160,)
(40, 3)
(40,)
```

2.2 Question 2.2 (Train a Linear regression model via Scikit-learn, 10 pts)

Assume you model is named as **model**, and you train the linear regression model to fit on the training data.

[7]:
```
################# Your answer for Question 2.2   #################
# Name your linear regression model as 'model', so proceeding cells won't run into errors.

from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, y_train)

################# Your code above ################
```

[7]:
```
LinearRegression()
```

[8]:
```
# Take a look at your model coefficients
import numpy as np
print(np.round(model.intercept_, 4))
print([np.round(val, 4) for val in model.coef_])
```

```
2.9791
[np.float64(0.0447), np.float64(0.1892), np.float64(0.0028)]
```

[9]:
```
# Match the feature names with the trained coefficients
list(zip(data.columns[:3].tolist(), model.coef_))
```

[9]:
```
[('TV', np.float64(0.044729517468716326)),
 ('Radio', np.float64(0.18919505423437652)),
 ('Newspaper', np.float64(0.0027611143413671935))]
```

2.3 Question 2.3 (Interpreting model coefficients, 10 pts)

The coefficients of each feature are printed above.

2.3.1 Your tasks:

- What is the trained linear regression model?
- **(replace β_i below with the trained coefficients, reserve four decimal places, 5 pts)**

$$\text{Sales} = \beta_0 + \beta_1 \times \text{TV} + \beta_2 \times \text{Radio} + \beta_3 \times \text{Newspaper}$$

- **(Open) question (5 pts)**: how do you interpret the coefficient of **TV** (β_1)?

2.3.2 Your solution for Question 2.3:

Sales $= 2.9721 + 0.0447 \times$ TV $+ 0.1892 \times$ Radio $+ 0.0028 \times$ Newspaper

It indicates that, when holding all other variables (Radio and Newspaper advertising) constant, for each additional unit spent on TV advertising, the model predicts an increase of **0.0447 units** in sales.

2.3.3 Making predictions

[10]:

```
# make predictions on the testing set
y_pred = model.predict(X_test)
# Print the test accuracy score from your trained linear regression model
model.score(X_test,y_test)
```

[10]:

```
0.899438024100912
```

2.4 Model evaluation metrics for regression

Evaluation metrics for classification problems, such as **accuracy**, are not useful for regression problems. Instead, we need evaluation metrics designed for comparing continuous values. (Since predictions would hardly ever exactly match the measured quantities.)

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is a popular evaluation metric for regression, which is defined as the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

2.5 Question 2.4 (Evaluate your predictions on test features with RMSE metric, 5 pts)

Hint: you may calculate MSE using scikit-learn, but RMSE is a little bit different from MSE.

[11]:

```
from sklearn import metrics
################# Your answer for Question 2.4   #################

from sklearn.metrics import mean_squared_error
import numpy as np

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

```

```
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

################# Your code above #################
```

```
Mean Squared Error (MSE): 3.1741
Root Mean Squared Error (RMSE): 1.7816
```

2.6 Question 2.5 (Does the feature ‘Newspaper’ help? 10 pts)

Does the feature **Newspaper** contribute or impair the model prediction? Now repeat previous procedure:

- For features X, only select **TV** and **Radio**; response y is unchanged.
- Split the dataset as what you have done before.
- Fit/Train the model on the train data.
- Make predictions on the test data.
- Compute the RMSE of the predictions on the test data.
- **Show your observations!**

Reminder: if you use random_state while splitting the dataset, make sure the value of random_state is the same for the two splits. So that the test data only differs in the column of **Newspaper**.

[12]:

```
################# Your answer for Question 2.5   #################

X_reduced = data[['TV', 'Radio']]

X_train_reduced, X_test_reduced, y_train, y_test = train_test_split(X_reduced,␣
 ↪y, test_size=0.2, random_state=42)

model_reduced = LinearRegression()
model_reduced.fit(X_train_reduced, y_train)

print(f"Intercept: {model_reduced.intercept_:.4f}")
print("Coefficients:")
for feature, coef in zip(X_reduced.columns, model_reduced.coef_):
    print(f"  {feature}: {coef:.4f}")

y_pred_reduced = model_reduced.predict(X_test_reduced)
mse_reduced = mean_squared_error(y_test, y_pred_reduced)
rmse_reduced = np.sqrt(mse_reduced)

print(f"\nWith Newspaper - RMSE: {rmse:.4f}")
print(f"Without Newspaper - RMSE: {rmse_reduced:.4f}")
```

```
################# Your code above #################
```

```
Intercept: 3.0283
Coefficients:
  TV: 0.0447
  Radio: 0.1907

With Newspaper - RMSE: 1.7816
Without Newspaper - RMSE: 1.7714
```

2.6.1 Your observations:

As RMSE without Newspaper is slightly smaller, the feature Newspaper slightly impair the model prediction.

3 Question 3 (PCA, 40 pts)

In this question, we will explore how PCA helps with reducing the dimensionality through scikit-learn. We will adopt a simple digits dataset contained in scikit-learn.

[13]:

```
# import libraries
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns;
sns.set()
```

[14]:

```
from sklearn.datasets import load_digits
data_digit = load_digits()
data_digit.data.shape
```

[14]:

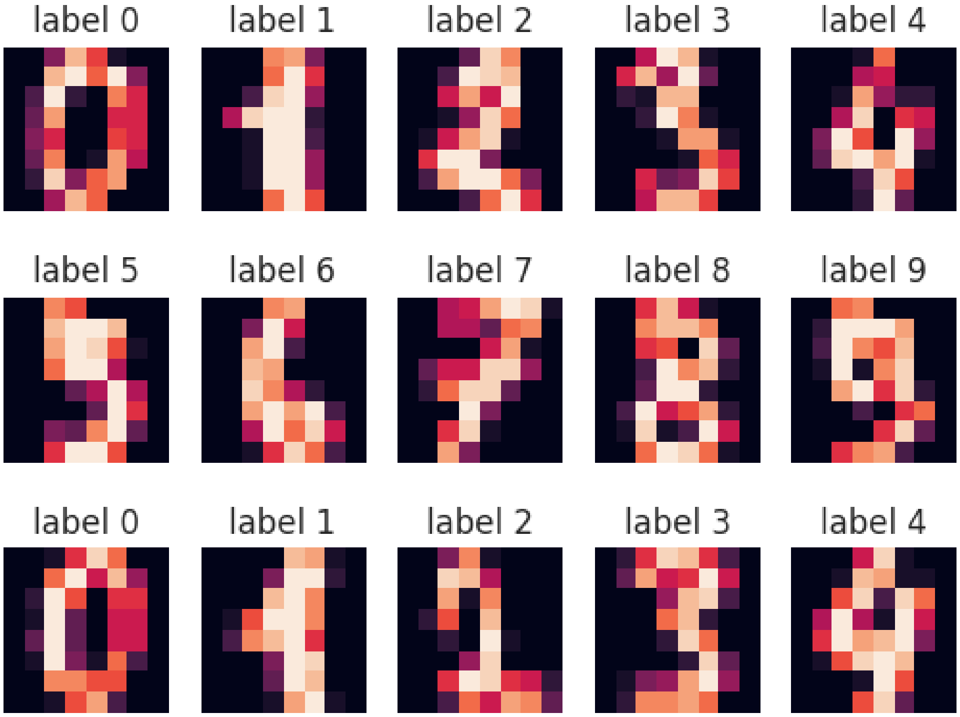
```
(1797, 64)
```

As you have seen above, there are 1797 rows and 64 columns in this dataset. Each row denotes the pixel values of an image. Since each image is 8*8 (in pixel), the dimension of each feature is 64. Let's take a look at a few images contained in this dataset.

[15]:

```
# A help function to display images with labels
def display_digits(imgs):
    h = 3
    w = 5
    fig, axes = plt.subplots(h, w)
    for i in range(h):
        for j in range(w):
            if imgs.shape[1] == 64:
                axes[i, j].imshow(imgs[i * w + j].reshape(8, 8))
            else:
                axes[i, j].imshow(imgs[i * w + j])
            axes[i, j].axis('off')
```

```python
            axes[i, j].set_title(f'label {data_digit.target[i * w + j]}')
# Take a look at a few examples
display_digits(data_digit.images)
```



PCA finds uncorrelated orthogonal axes (principal components) in a high dimensional space (i.e., 64) to project the feature onto principal components.

3.1 Question 3.1 (PCA and dimensionality deduction with Scikit-learn, 10 pts)

In this question, you may use the implementation of PCA from scikit-learn, as imported below. Detailed explanations of parameters settings are available in this link.

3.1.1 Your task in this question:

Fit the model with feature X (**data_digit.data** for this dataset) and apply the dimensionality reduction on X, keep 2 components.

[16]:

```python
from sklearn.decomposition import PCA

################# Your answer for Question 3.1   #################

pca = PCA(n_components=2)
pca.fit(data_digit.data)
```

```python
# name the transformed (dimension reduced) feature as projected
projected = pca.transform(data_digit.data)

################# Complete the code above #################


# The code below checks whether the feature dimension 64 is reduced to 2.
print(data_digit.data.shape)
print(projected.shape)
```

```
(1797, 64)
(1797, 2)
```

With the reduced dimension of features, we can now visualize the projected/reduced features in a 2-dimension plot as below. There are hardly any overlaps between digit 0 and any other digits. While in the central area of the figure, features are not well separable.

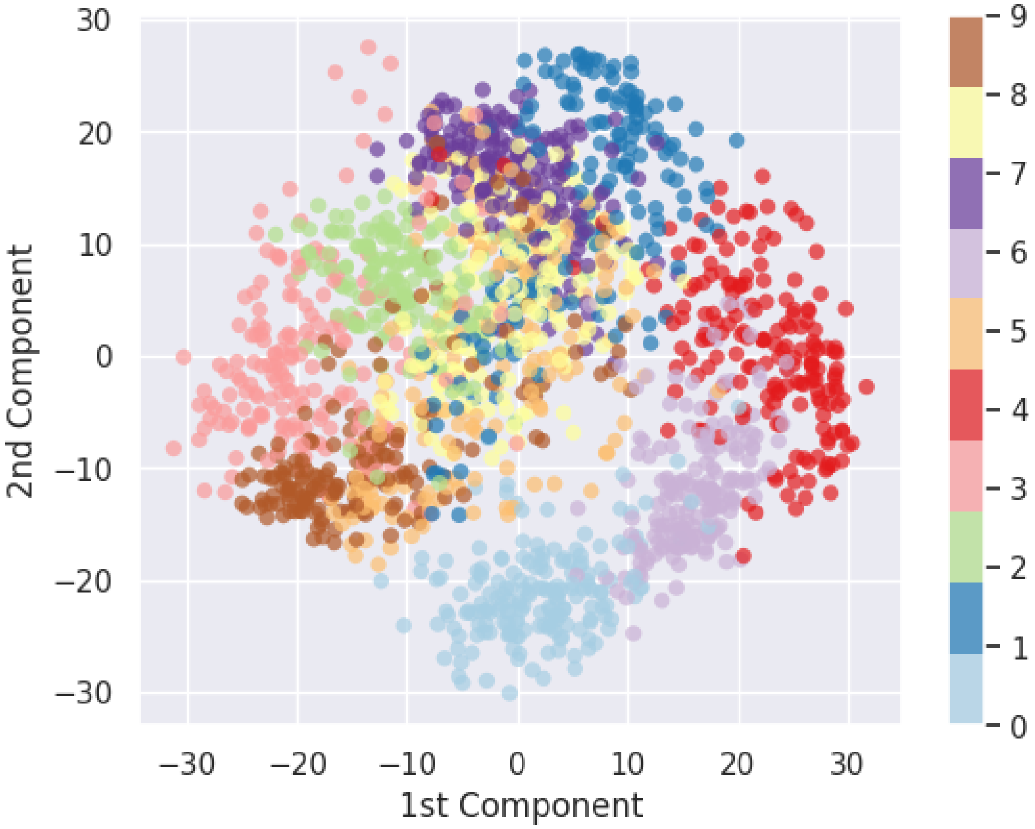
[17]:

```python
plt.scatter(projected[:, 0], projected[:, 1],
            c=data_digit.target, edgecolor='none', alpha=0.7,
            cmap=plt.cm.get_cmap('Paired', 10))
plt.xlabel('1st Component')
plt.ylabel('2nd Component')
plt.colorbar()
plt.figure(figsize=(18, 8))
```

```
/tmp/ipykernel_65344/284023998.py:3: MatplotlibDeprecationWarning: The get_cmap
function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use
``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or
``pyplot.get_cmap()`` instead.
  cmap=plt.cm.get_cmap('Paired', 10))
```

[17]:

```
<Figure size 1800x800 with 0 Axes>
```



```
<Figure size 1800x800 with 0 Axes>
```

3.2 Question 3.2 (How many components do we need? 15 pts)

In practice, suppose you are given a high-dimension dataset, and you are interested in performing a dimension reduciton with PCA. How many conponents do we need? One method is to look at the distribution of the explained variance ratio w.r.t. each component.

In this question, you may use the function explained_variance_ratio in **pca** (with scikit-learn) to address this question. Detailed explanations of parameters settings are available in this link.

3.2.1 Your task in this question:

Print the ratio of variance explained by each of the selected components;

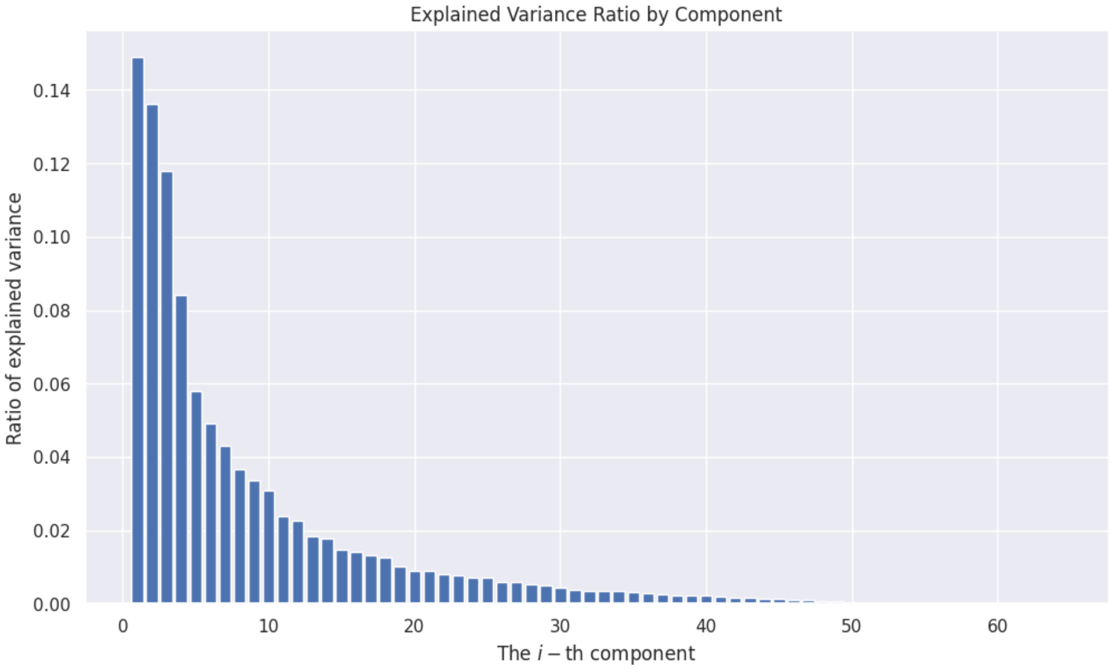
Visualize with matplotlib or seaborn (x axis: $i - th$ component; y axis: the corresponding ratio of explained variance).

[18]:

```
################# Your answer for Question 3.2   #################


pca = PCA()
pca.fit(data_digit.data)
```

```
explained_variance_ratio = pca.explained_variance_ratio_

plt.figure(figsize=(10, 6))
plt.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio)
plt.title('Explained Variance Ratio by Component')
plt.grid(True)
plt.tight_layout()

plt.xlabel('The $i-$th component')
plt.ylabel('Ratio of explained variance')
################# Complete the code above #################
```

```
[18]: Text(72.375, 0.5, 'Ratio of explained variance')
```



3.2.2 How to obtain eigen values and vectors from sklearn PCA

- Eigen values – your-PCA-model.explained_variance_
- Eigen vectors – your-PCA-model.components_

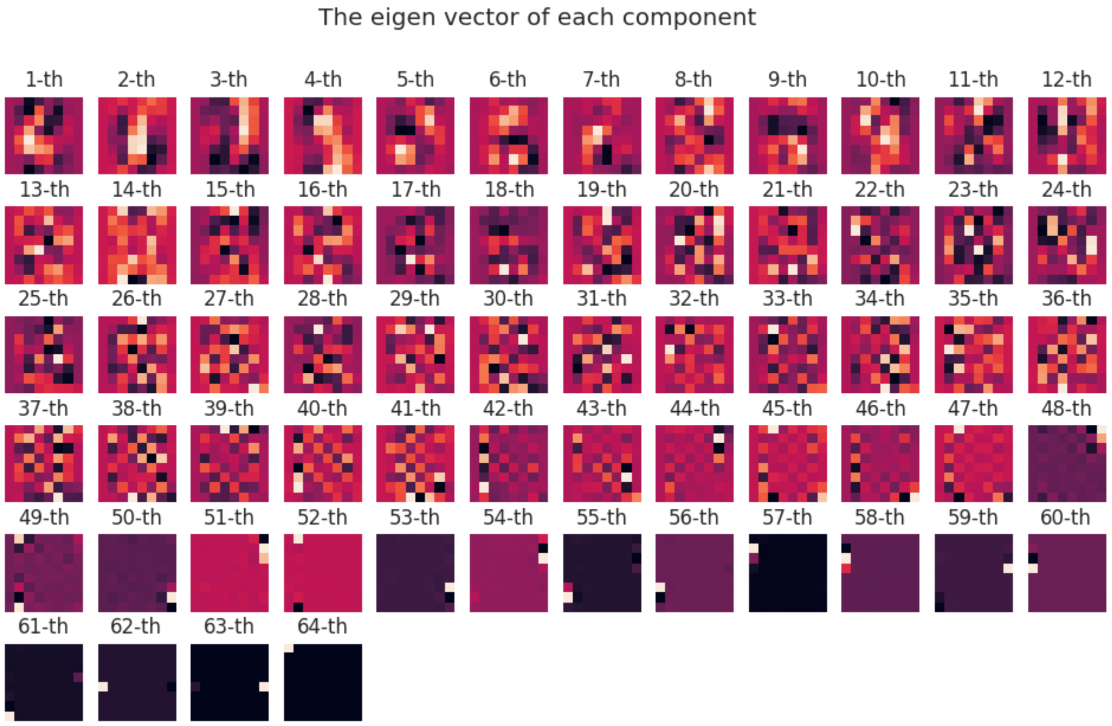
3.2.3 Eigen vectors (Eigen digits)

[19]:

```
# Default # of components is the dimension of feature space
pca = PCA()
pca.fit(data_digit.data)
h = 6
w = 12
fig, axes = plt.subplots(h, w)
fig.set_size_inches(12, 7)
for i in range(h):
    for j in range(w):
        idx = i * w + j
        if idx > 63:
            axes[i, j].axis('off')
        else:
            axes[i, j].imshow(pca.components_[idx].reshape(8, 8))
            axes[i, j].axis('off')
            axes[i, j].set_title(f'{idx + 1}-th')
fig.suptitle('The eigen vector of each component')
```

[19]:

```
Text(0.5, 0.98, 'The eigen vector of each component')
```



3.3 Question 3.3 What are your observations from the visualized eigen vectors? (5 pts)

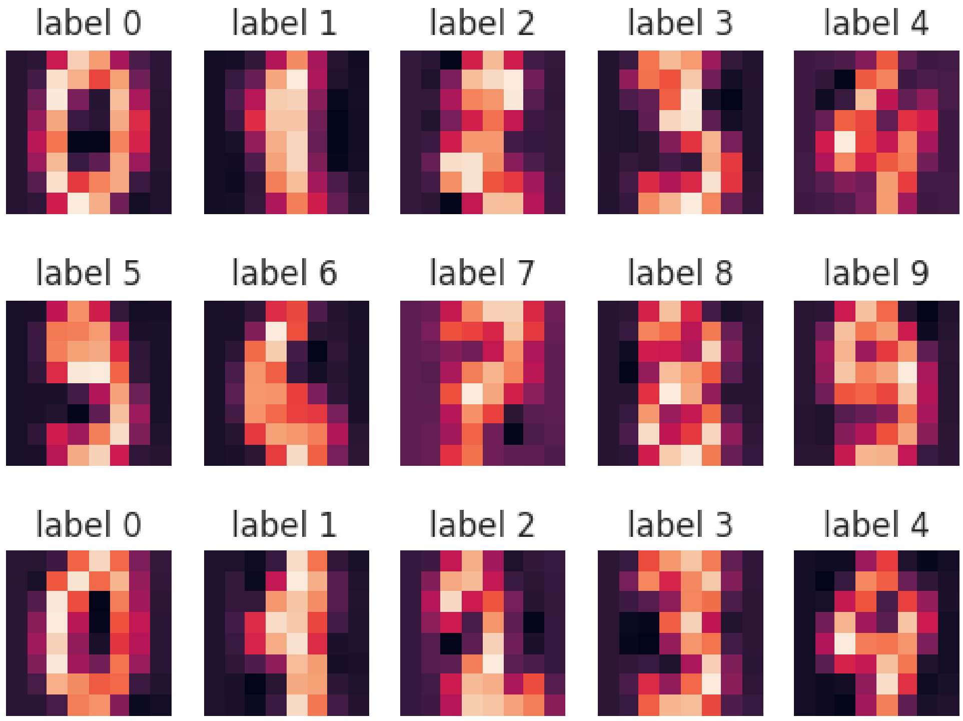
3.3.1 Your solutions for Question 3.3:

The eigen vectors closer to the beginning(correspond with larger eigen values) are more complex, while those at the end appear almost as solid colors after visualization.

3.3.2 Feature reconstruction:

Suppose you only want to preserve 80% variance after applying PCA for the dimension reduction, due to storage issues, etc.

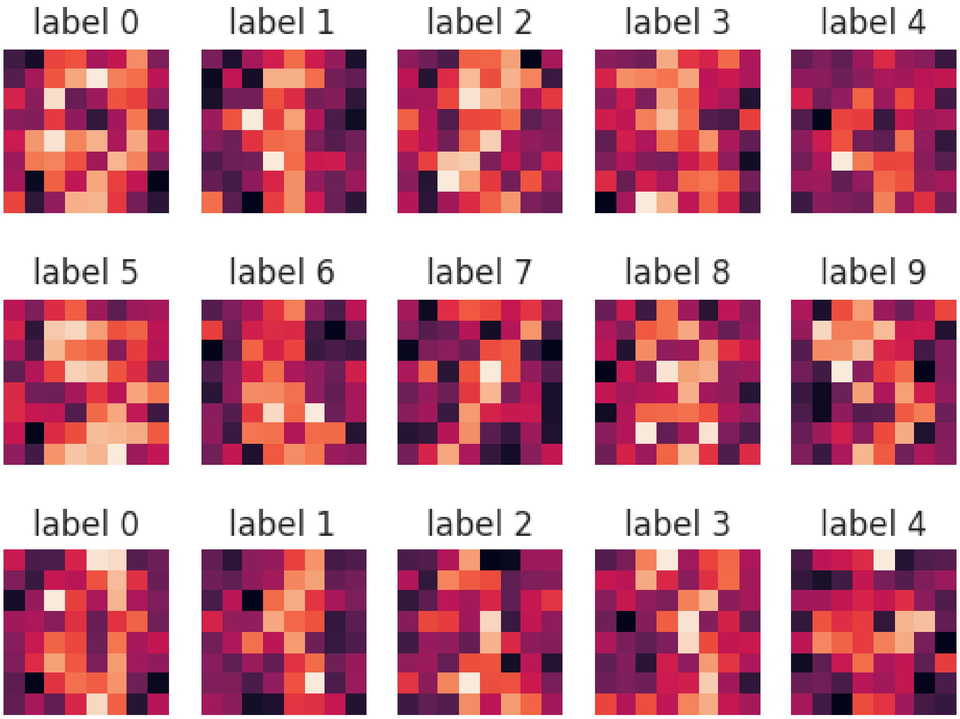
[20]:
```python
# Use pca from sklearn to fit on the feature space
pca_08 = PCA(0.8).fit(data_digit.data)

# Reconstruct
lower_dimension = pca_08.fit_transform(data_digit.data)
reconstructed = pca_08.inverse_transform(lower_dimension)

# Display the reconstructed images (with a much lower dimension)
display_digits(reconstructed)
```



Assume now you are given only a set of perturbed images (with some random noise). As shown below, the figures become much more difficult to recognize.

[21]:

```
# Apply random noise on clean images
noisy_imgs = np.random.normal(data_digit.data, 5)
# Display the noisy images
display_digits(noisy_imgs)
```



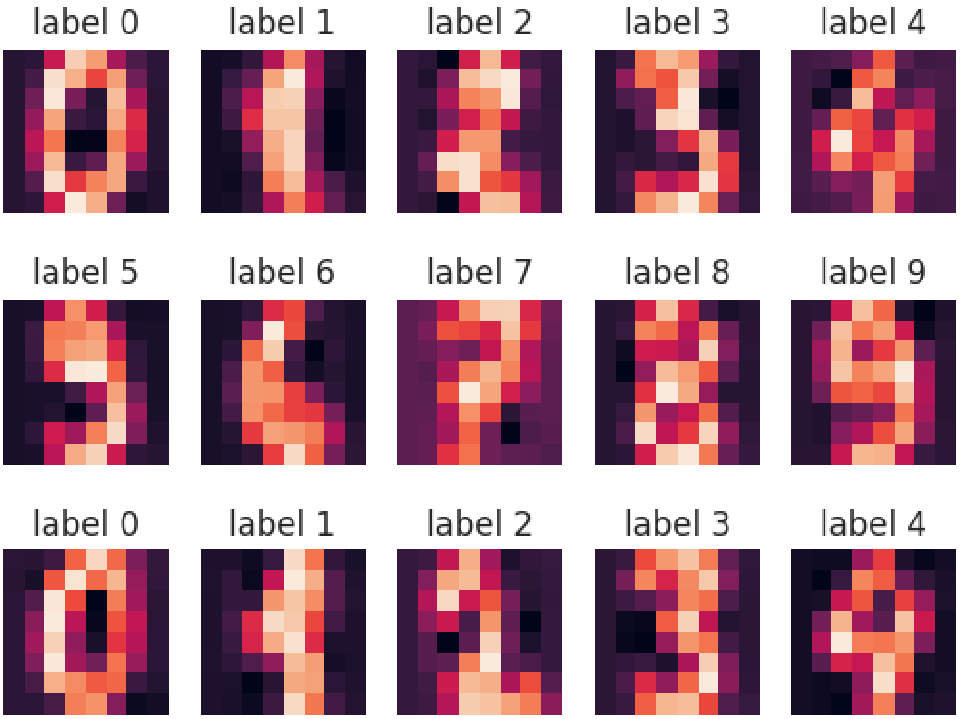
3.4 Question 3.4 Does PCA mitigate the random noise? (10 pts)

3.4.1 Your tasks:

- Perform feature reconstruction on noisy_imgs (preserving 80% variance after applying PCA for the dimension reduction)
- Explain your understandings on why PCA reconstructs cleaner images.

[22]:

```
################# Your answer for Question 3.4   #################

pca_denoising = PCA(0.8).fit(noisy_imgs)
lower_dimension = pca_denoising.fit_transform(data_digit.data)

# name the reconstructed images from noisy images as 'reconstructed_noise'
reconstructed_noise = pca_denoising.inverse_transform(lower_dimension)
# Display the reconstructed images
display_digits(reconstructed_noise)
################# Complete the code above #################
```



3.4.2 Your written answers for Question 3.4:

PCA identifies dimensions with the highest variance in the data. Since meaningful signal tends to be correlated and structured while noise spreads out approximately equally in all directions of the high-dimensional space, the top principal components capture mostly signal, while noise gets distributed across many lower-variance components that are discarded.