Codes

October 9, 2023

1 2.1

[2]:
```
import numpy as np
from sklearn.model_selection import train_test_split
x=np.array([
    10,11,12,13,14,
    15,16,17,18,19
])
y=np.array([
    0,1,2,3,4,
    5,6,7,8,9
])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
print(x)
print(y)
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```

```
[10 11 12 13 14 15 16 17 18 19]
[0 1 2 3 4 5 6 7 8 9]
[10 14 17 16 11 18 12]
[19 15 13]
[0 4 7 6 1 8 2]
[9 5 3]
```

2 2.2

[3]:
```
import numpy as np
from sklearn import preprocessing
X = np.array([[ 2, 2,  -1],
  [ 1,  2,  -2],
 [ 0,  -2, 2]])
scaler = preprocessing.MinMaxScaler()
X_processing = scaler.fit_transform(X)
```

```
print(X_processing)
```

```
[[1.   1.   0.25]
 [0.5  1.   0.  ]
 [0.   0.   1.  ]]
```

3 2.3

[4]:
```
import numpy as np
from sklearn import preprocessing
X = np.array([[ 2, 2,  -1],
  [ 1,  2,  -2],
 [ 0,  -2, 2]])
scaler = preprocessing.StandardScaler()
X_processing = scaler.fit_transform(X)
```

4 2.4

[5]:
```
import pandas as pd
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,168,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
data.duplicated()
print (data)
#
print (data[data.duplicated()])
data.drop_duplicates(subset=' ',inplace=True)
# data.drop_duplicates([' '],'first',inplace=True)
print(data)
```

```
0   1  172  70
1   2  162  62
2   3  175  75
3   4  170  68
4   5  168  67
5   6  160  58
6   7  164  64
7   7  164  64
8   8  160  53

7   7  164  64
```

```
0   1  172  70
1   2  162  62
2   3  175  75
3   4  170  68
4   5  168  67
5   6  160  58
6   7  164  64
8   8  160  53
```

5 2.5

[6]:
```
import pandas as pd
import numpy as np
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
#    5
data=data.dropna()
print(data)
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
# #
data=data.dropna(how='all')
print(data,'---all')
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
#
data.dropna(axis = 1)
print(data,'--1')
```

```
0   1  172.0  70
1   2  162.0  62
2   3  175.0  75
3   4  170.0  68
5   6  160.0  58
6   7  164.0  64
```

```
7   7  164.0  64
8   8  160.0  53

0   1  172.0  70
1   2  162.0  62
2   3  175.0  75
3   4  170.0  68
4   5    NaN  67
5   6  160.0  58
6   7  164.0  64
7   7  164.0  64
8   8  160.0  53 ---all

0   1  172.0  70
1   2  162.0  62
2   3  175.0  75
3   4  170.0  68
4   5    NaN  67
5   6  160.0  58
6   7  164.0  64
7   7  164.0  64
8   8  160.0  53 --1
```

6 2.6

[8]:
```
import pandas as pd
import numpy as np
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
data=data.fillna(170)
print(data)

data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
#    / ,   /
data = data.fillna(method='ffill')
print(data,'--ffill')
# bfill

data=pd.DataFrame({
```

```
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
data=data.fillna(method='bfill')
print(data,'--bfill')

data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,np.nan,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
data[' '].fillna(data[' '].mean(),inplace=True)
print(data)
```

```
0   1  172.0  70
1   2  162.0  62
2   3  175.0  75
3   4  170.0  68
4   5  170.0  67
5   6  160.0  58
6   7  164.0  64
7   7  164.0  64
8   8  160.0  53

0   1  172.0  70
1   2  162.0  62
2   3  175.0  75
3   4  170.0  68
4   5  170.0  67
5   6  160.0  58
6   7  164.0  64
7   7  164.0  64
8   8  160.0  53 --ffill

0   1  172.0  70
1   2  162.0  62
2   3  175.0  75
3   4  170.0  68
4   5  160.0  67
5   6  160.0  58
6   7  164.0  64
7   7  164.0  64
8   8  160.0  53 --bfill

0   1  172.000  70
```

```
1   2  162.000  62
2   3  175.000  75
3   4  170.000  68
4   5  165.875  67
5   6  160.000  58
6   7  164.000  64
7   7  164.000  64
8   8  160.000  53
```

7 2.7

[10]:
```
import pandas as pd
import numpy as np
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,1700,160,164,164,160],
' ':[70,62,75,68,67,58,64,64,53]
})
print("        ",any(data[' ']>240))
renew_value=data[' '][data[' ']<200].max()
print (renew_value,'---renew_value')
#   200cm  renew_value
data.loc[data[' ']>200,' ']=renew_value
print (data)
```

```
         True
175 ---renew_value

0   1  172  70
1   2  162  62
2   3  175  75
3   4  170  68
4   5  175  67
5   6  160  58
6   7  164  64
7   7  164  64
8   8  160  53
```

8 2.8

[13]:
```
import pandas as pd
import numpy as np
data=pd.DataFrame({
' ':[1,2,3,4,5,6,7,7,8],
' ':[172,162,175,170,17,160,164,164,160],
```

```
' ':[70,62,75,68,67,58,64,64,53]
})
print("        ",any(data[' ']<50))
renew_value=data[' '][data[' ']>50].min()
print (renew_value,'---renew_value')
data.loc[data[' ']<50,' ']=renew_value
print(data)
```

```
         True
160 ---renew_value

0   1  172  70
1   2  162  62
2   3  175  75
3   4  170  68
4   5  160  67
5   6  160  58
6   7  164  64
7   7  164  64
8   8  160  53
```

9 2.9

[35]:
```
import numpy as np
from sklearn.decomposition import PCA
X=np.array([
[1,2,1,2],
[7,2,2,4],
[3,7,3,6],
[2,5,2,3],
[3,2,2,9],
[5,0,3,5]])
print (X)
# pca=PCA(n_components=3)
pca=PCA(n_components=0.78)

X_pca=pca.fit_transform(X)# pca.fit(X)pca.transform(X)
print (X_pca,'---X_pca')
X_new=pca.inverse_transform(X_pca)#
# # explained_variance_ratio_
print(pca.explained_variance_ratio_)
print(X_new)
```

```
[[1 2 1 2]
 [7 2 2 4]
 [3 7 3 6]
```

```
 [2 5 2 3]
 [3 2 2 9]
 [5 0 3 5]]
[[-1.65093621 -3.23768946]
 [ 2.47917462 -1.04915797]
 [-2.87718111  2.84729259]
 [-2.96243102 -0.8737948 ]
 [ 1.80035114  3.23872099]
 [ 3.21102257 -0.92537137]] ---X_pca
[0.44274388 0.35663085]
[[2.42221652 2.87080095 1.55833678 1.4186813 ]
 [4.87649641 0.69089601 2.12107841 4.6982136 ]
 [1.96923712 6.36991978 2.48399953 6.44111185]
 [1.76809501 4.85831644 1.87595211 3.10065763]
 [4.66337293 3.01724635 2.78267584 8.29705627]
 [5.30058201 0.19282048 2.17795733 5.04427935]]
```

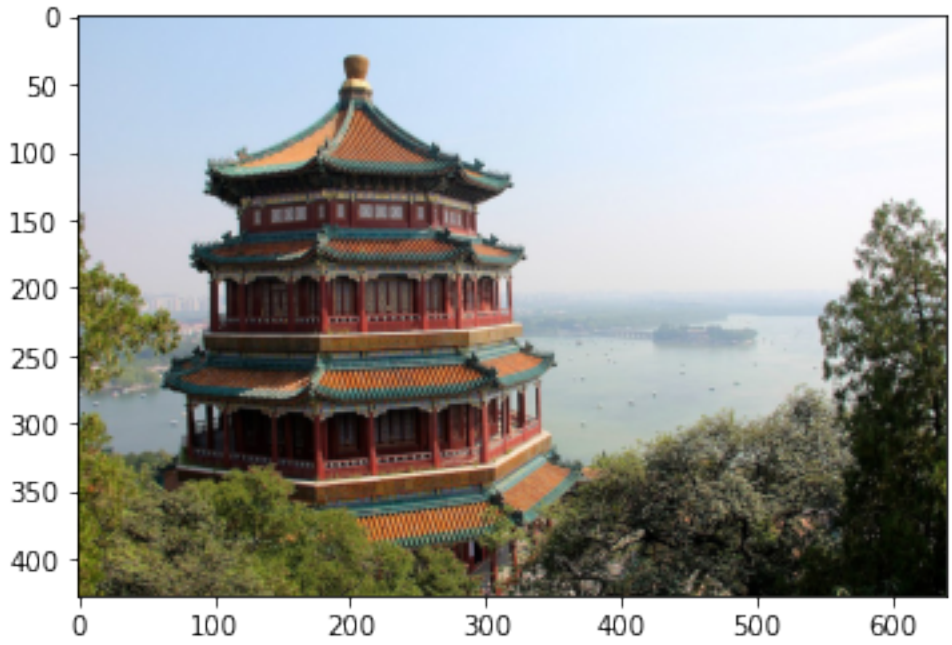
10 2.10

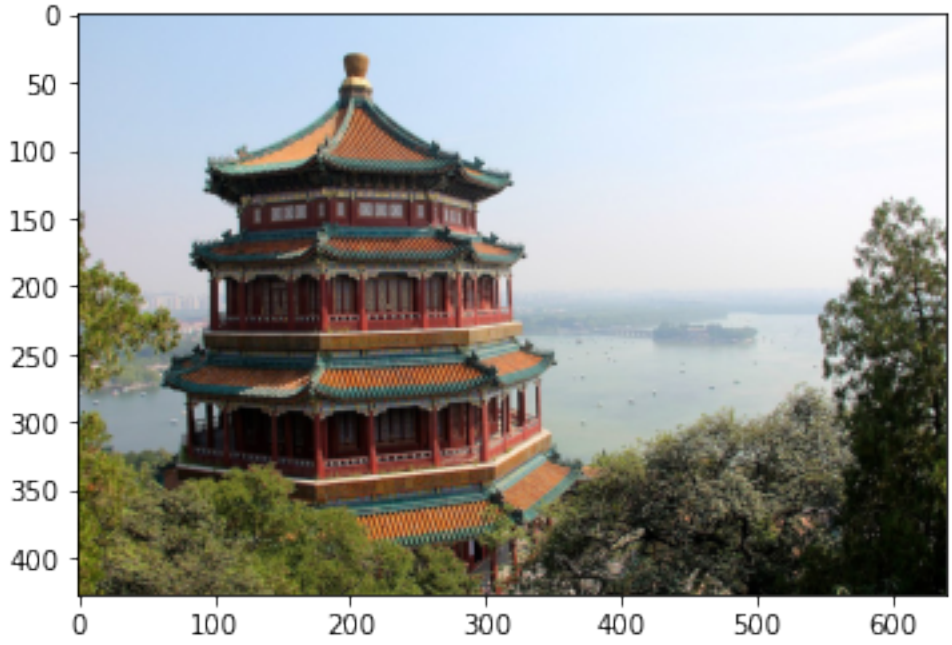
[31]:
```
from sklearn.decomposition import PCA
import numpy as np
from sklearn.preprocessing import StandardScaler

x=np.array([[10001,2,55], [16020,4,11], [12008,6,33], [13131,8,22]])
print (x)
# feature normalization (feature scaling)
X_scaler = StandardScaler()
print (X_scaler,'---X_scaler')
x = X_scaler.fit_transform(x)
print (x,'---x_X_scaler')
# PCA
pca = PCA(n_components=0.9)#       90%
pca.fit(x)
print(pca.explained_variance_ratio_)
pca.transform(x)
```

```
[[10001     2    55]
 [16020     4    11]
 [12008     6    33]
 [13131     8    22]]
StandardScaler() ---X_scaler
[[-1.2817325  -1.34164079  1.52127766]
 [ 1.48440157 -0.4472136  -1.18321596]
 [-0.35938143  0.4472136   0.16903085]
 [ 0.15671236  1.34164079 -0.50709255]] ---x_X_scaler
[0.74292812 0.25704179]
```

[31]:
```
array([[ 2.36863319,  0.38298087],
       [-1.50952734,  1.23481789],
       [ 0.14360068, -0.58040917],
       [-1.00270653, -1.03738959]])
```

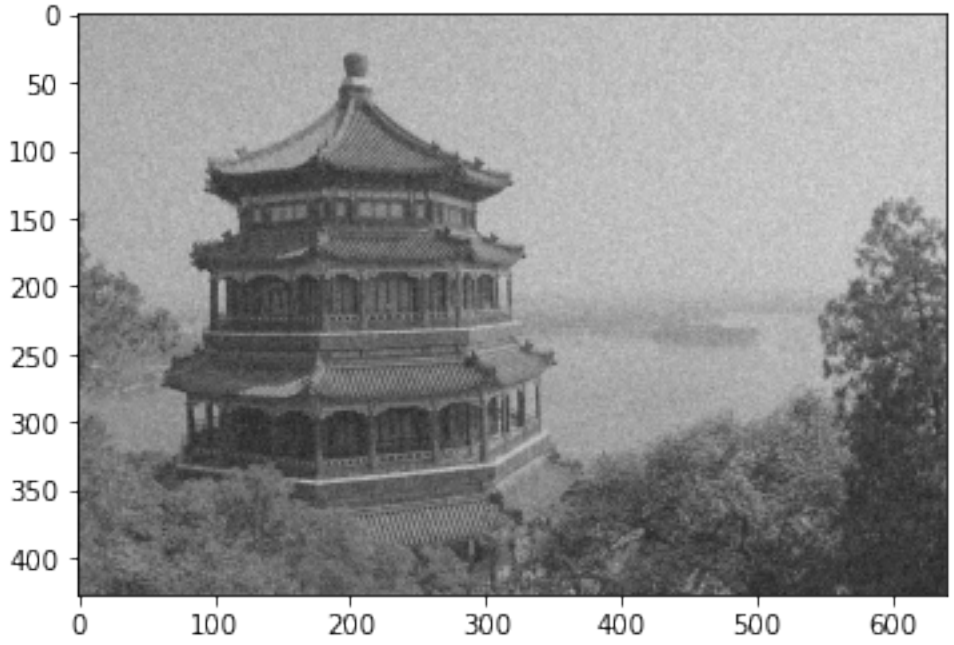
[24]:
```
#
from sklearn.datasets import load_sample_image
import matplotlib.pyplot as plt
import numpy as np
img1=load_sample_image("china.jpg")
plt.imshow(img1)
plt.show()
#             CNN
img1=np.array(img1,dtype=np.float64)/255
plt.imshow(img1)
plt.show()
#
from skimage import color,filters
img1=color.rgb2gray(img1)
print (img1,'--img1')
img2=np.random.normal(img1.data,0.1)
print (img2,'--img2')
plt.imshow(img2,cmap='gray')
plt.show()
#  PCA
from sklearn.decomposition import PCA
pca=PCA(0.88)
img2_pca=pca.fit_transform(img2)
img3=pca.inverse_transform(img2_pca)
#
plt.imshow(img3,cmap='gray')
plt.show()
```

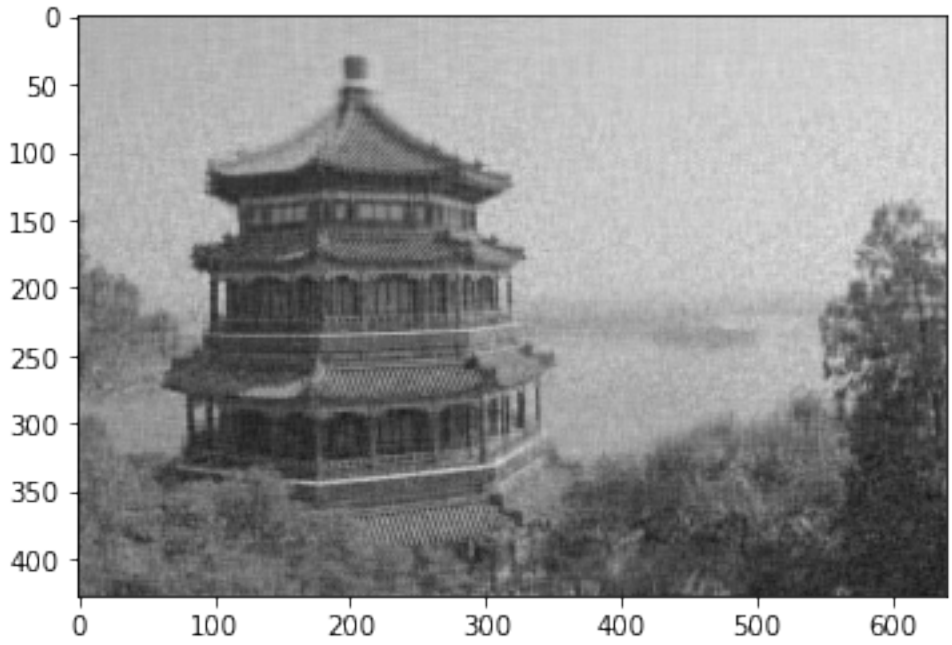





```
[[0.77421765 0.77421765 0.77421765 … 0.98461137 0.98461137 0.98461137]
 [0.76637451 0.77029608 0.77029608 … 0.9882502  0.9882502  0.9882502 ]
 [0.77421765 0.77421765 0.77421765 … 0.99188902 0.99188902 0.99188902]
 …
```

```
 [0.29975176 0.52916706 0.43784784 … 0.15966118 0.04650902 0.03866588]
 [0.42747412 0.45914431 0.42272706 … 0.04201412 0.05043059 0.0832098 ]
 [0.39552824 0.35634235 0.38407686 … 0.18432941 0.0714451  0.08181098]] --img1
[[ 0.58589786  0.81298327  0.73906598 …  1.09118132  0.98740222
   0.91922625]
 [ 0.7617418   0.96810661  0.8908024  …  0.93036503  0.97867562
   0.94110189]
 [ 0.72301049  0.78762327  0.71407962 …  1.02362806  0.85647
   0.92453856]
 …
 [ 0.15549228  0.48782626  0.41147707 …  0.18051708 -0.14398469
  -0.02777253]
 [ 0.45237093  0.43078286  0.37640427 … -0.05067856  0.18102432
   0.0775044 ]
 [ 0.41359739  0.25942559  0.49515873 …  0.11641252 -0.14726962
   0.0916538 ]] --img2
```



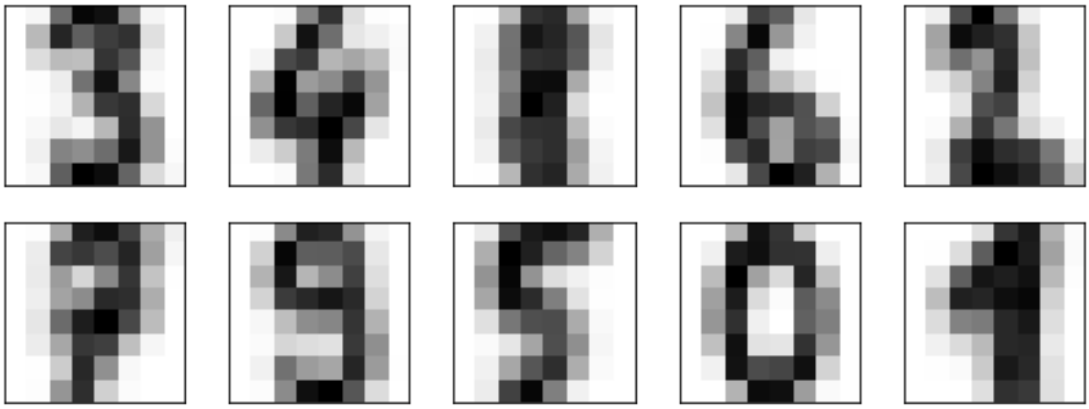


11 3.1

[40]:
```
#
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
y = digits.target
print (X,'---X')
print (X.shape)
print (y,'---y')
print (len(y))
#
import time
start = time.process_time()
#   K-means
from sklearn.cluster import KMeans
KM = KMeans(n_clusters=10)
c = KM.fit_predict(X)
print (c,'----c')
end = time.process_time()
print('Time is %.3f' % (end - start))
#
import numpy as np
# np.zeros_like     ,       ,       0
```

```
y_predict = np.zeros_like(c)
print (y_predict,'---y_predcit')
from scipy.stats import mode
for i in range(10):
    mask = (c == i)
    y_predict[mask] = mode(y[mask])[0]
print (y_predict,'---mask_y_predcit')
KM.cluster_centers_.shape
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
fig, ax = plt.subplots(2, 5, figsize = (8, 3))
centers = KM.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
   axi.set(xticks=[], yticks = [])
   axi.imshow(center, cmap = plt.cm.binary)
#
from sklearn.metrics import accuracy_score
print('     %.4f %%' % accuracy_score(y, y_predict))
```

```
[[ 0.  0.  5. … 0.  0.  0.]
 [ 0.  0.  0. … 10.  0.  0.]
 [ 0.  0.  0. … 16.  9.  0.]
 …
 [ 0.  0.  1. …  6.  0.  0.]
 [ 0.  0.  2. … 12.  0.  0.]
 [ 0.  0. 10. … 12.  1.  0.]] ---X
(1797, 64)
[0 1 2 … 8 9 8] ---y
1797
[8 2 2 … 2 6 6] ----c
Time is 0.782
[0 0 0 … 0 0 0] ---y_predcit
[0 8 8 … 8 9 9] ---mask_y_predcit
     0.7880 %
```



12 3.2

[17]:
```
#
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data;
y = digits.target
#
import time
from sklearn.decomposition import PCA
start = time.process_time()
#  PCA
pca = PCA(n_components=10)
pca.fit(X)
X_reduction = pca.transform(X)
print(X_reduction.shape)
end = time.process_time()
print('PCA Time is %.3f' % (end - start))
#  K-means
from sklearn.cluster import KMeans
start = time.process_time()
KM = KMeans(n_clusters=10)
c = KM.fit_predict(X_reduction)
end = time.process_time()
print('K-means Time is %.3f' % (end - start))
#
from scipy.stats import mode
import numpy as np
y_predict = np.zeros_like(c)
for i in range(10):
```

```
    mask = (c == i)
    y_predict[mask] = mode(y[mask])[0]
#
from sklearn.metrics import accuracy_score
print('     %.4f %%' % accuracy_score(y, y_predict))
```

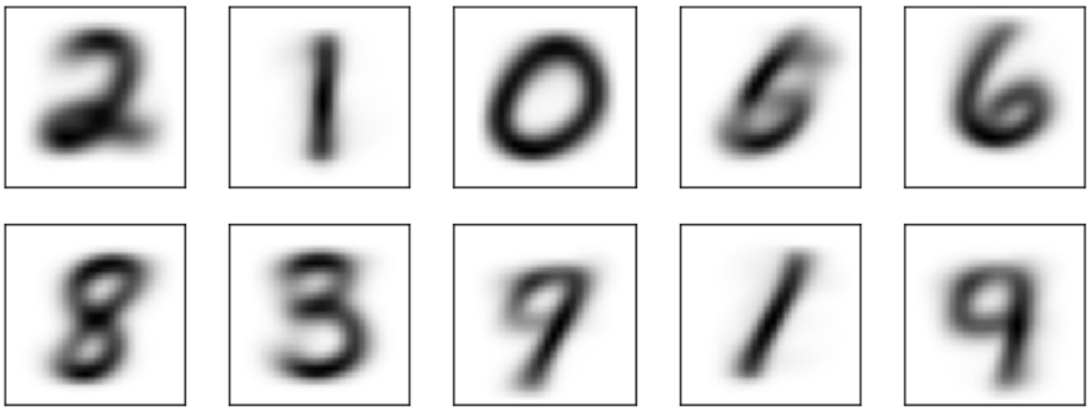
```
(1797, 10)
PCA Time is 0.053
K-means Time is 0.520
     0.7858 %
```

13 3.3

[22]:
```
#
from sklearn.datasets import load_digits
from sklearn.datasets import fetch_openml
digits = fetch_openml('mnist_784')
X = digits.data
y = digits.target
#
import time
start = time.process_time()
#  K-means
from sklearn.cluster import KMeans
KM = KMeans(n_clusters=10)
c = KM.fit_predict(X)
end = time.process_time()
print('Time is %.3f' % (end - start))
#
from scipy.stats import mode
import numpy as np
y_predict = np.zeros_like(c)
for i in range(10):
    mask = (c == i)
    y_predict[mask] = mode(y[mask])[0]
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
fig, ax = plt.subplots(2, 5, figsize = (8, 3))
centers = KM.cluster_centers_.reshape(10, 28, 28)
for axi, center in zip(ax.flat, centers):
   axi.set(xticks=[], yticks = [])
   axi.imshow(center, cmap = plt.cm.binary)
import pandas as pd
y=y.astype(np.int8)
```

```
y_predict=pd.DataFrame(y_predict)
y_predict=y_predict.astype(np.int8)
#
from sklearn.metrics import accuracy_score
print('     %.4f %%' % accuracy_score(y, y_predict))
```

```
Time is 138.769
     0.5851 %
```



14 3.4

[16]:
```
#
from sklearn.datasets import load_digits
from sklearn.datasets import fetch_openml
digits = fetch_openml('mnist_784')
X = digits.data
y = digits.target
#  PCA
import time
start = time.process_time()
from sklearn.decomposition import PCA
pca = PCA(n_components=64)
pca.fit(X)
X_reduction = pca.transform(X)
print(X_reduction.shape)

from sklearn.cluster import KMeans
KM = KMeans(n_clusters=10)
c = KM.fit_predict(X_reduction)
end = time.process_time()
```

```
print('Time is %.3f' % (end - start))
#
from scipy.stats import mode
import numpy as np
y_predict = np.zeros_like(c)
for i in range(10):
    mask = (c == i)
    y_predict[mask] = mode(y[mask])[0]
#
from sklearn.metrics import accuracy_score
y=y.astype(np.int8)
import pandas as pd
y_predict=pd.DataFrame(y_predict)
y_predict=y_predict.astype(np.int8)
accuracy_score(y, y_predict)
```

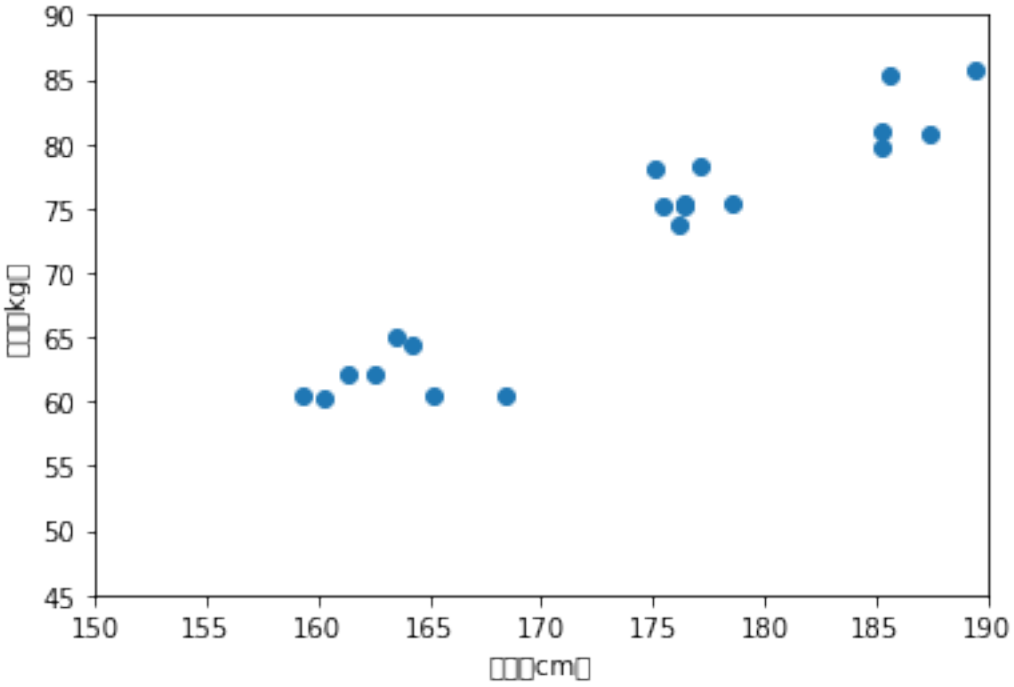
```
(70000, 64)
Time is 34.500
```

[16]:
```
0.5845142857142858
```

15 3.5

[21]:
```
#
import numpy as np
x=np.array([
159.3,160.3,165.2,162.5,
175.4,178.6,177.1,176.4,
189.4,176.2,185.3,161.3,
164.2,163.5,176.4,185.6,
175.1,168.4,187.4,185.2
])
y=np.array([
60.5,60.2,60.4,62.1,
75.1,75.3,78.2,75.4,
85.8, 73.7, 81,62.2,
64.4, 65.1, 75.1,85.3,
78,60.4,80.8,79.7
])
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.scatter(x, y)
plt.xlim([150,190])
```

```
plt.ylim([45,90])
plt.xlabel('  cm ')
plt.ylabel('  kg ')
plt.show()
```

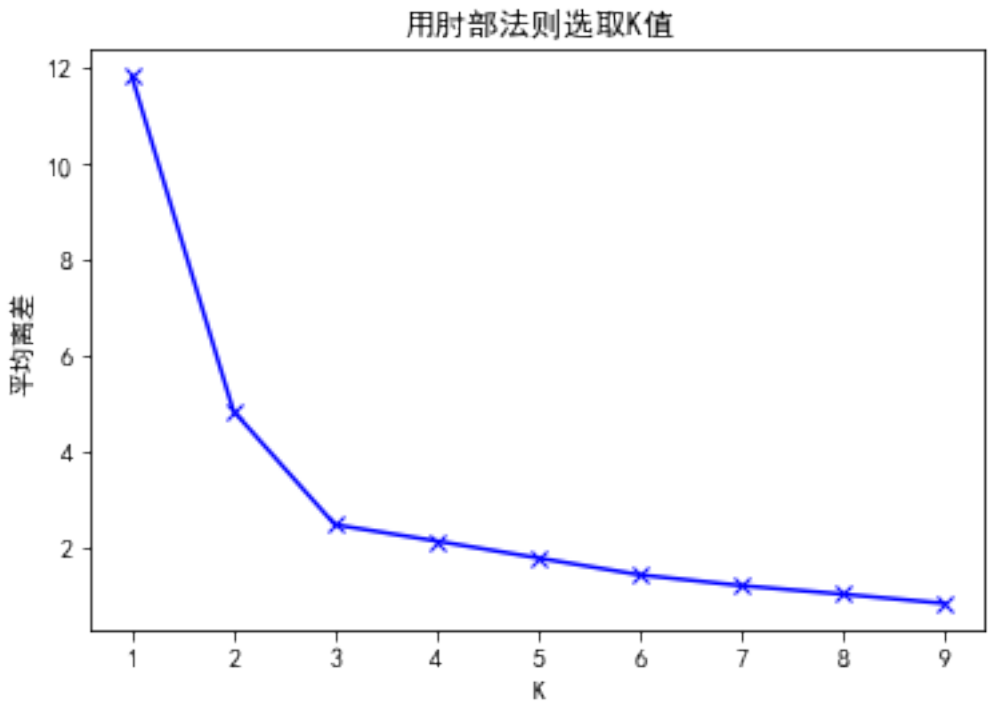


16 3.6

[18]:
```
#
import numpy as np
x=np.array([
159.3,160.3,165.2,162.5,
175.4,178.6,177.1,176.4,
189.4,176.2,185.3,161.3,
164.2,163.5,176.4,185.6,
175.1,168.4,187.4,185.2
])
y=np.array([
60.5,60.2,60.4,62.1,
75.1,75.3,78.2,75.4,
85.8, 73.7, 81,62.2,
64.4, 65.1, 75.1,85.3,
78,60.4,80.8,79.7
```

```
])
X = np.array(list(zip(x, y))).reshape(len(x), 2)
K = range(1, 10)
meandistortions = []
#
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
for k in K:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    meandistortions.append(sum(np.min(cdist(X, kmeans.cluster_centers_,␣
 ↪'euclidean'), axis=1))/X.shape[0])
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.plot(K, meandistortions, 'bx-')
plt.xlabel('K')
plt.ylabel('  ')
plt.title('    K ')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

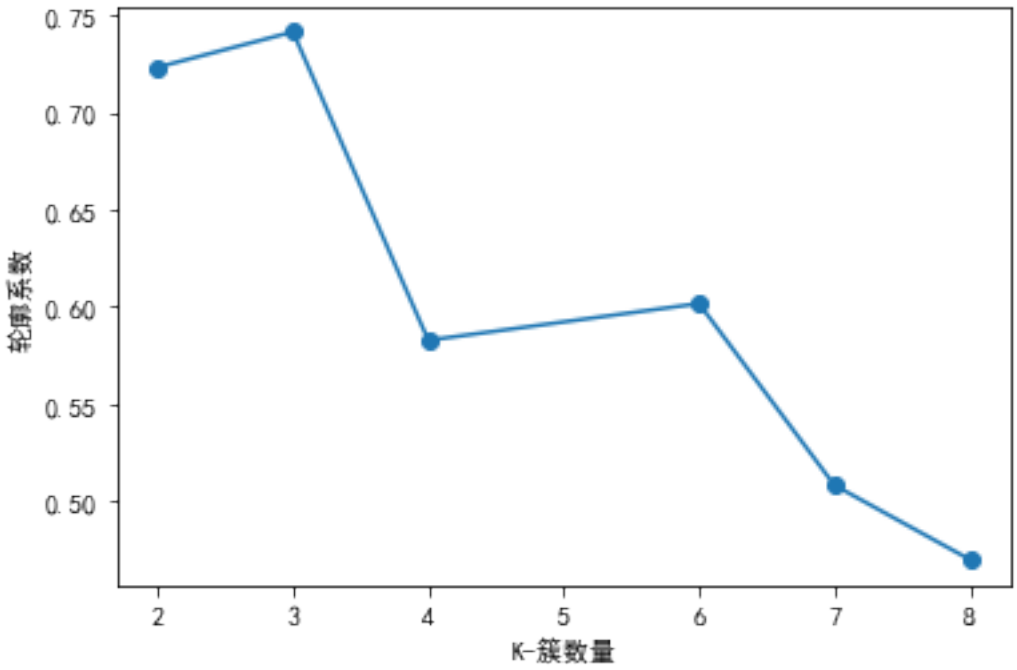


17 3.7

[19]:
```
#
import numpy as np
x=np.array([
159.3,160.3,165.2,162.5,
175.4,178.6,177.1,176.4,
189.4,176.2,185.3,161.3,
164.2,163.5,176.4,185.6,
175.1,168.4,187.4,185.2
])
y=np.array([
60.5,60.2,60.4,62.1,
75.1,75.3,78.2,75.4,
85.8,73.7,81,62.2,64.4,
65.1,75.1,85.3,78,
60.4,80.8,79.7
])
sc_scores = []
X = np.array(list(zip(x, y))).reshape(len(x), 2)
#
clusters_number = [2,3,4,6,7,8]
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
for t in clusters_number:
    kmeans_model = KMeans(n_clusters=t).fit(X)
    sc_scores.append(silhouette_score(X, kmeans_model.labels_,␣
 ↪metric='euclidean'))
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.xlabel('K-  ')
plt.ylabel('  ')
plt.plot(clusters_number,sc_scores,'o-')
```

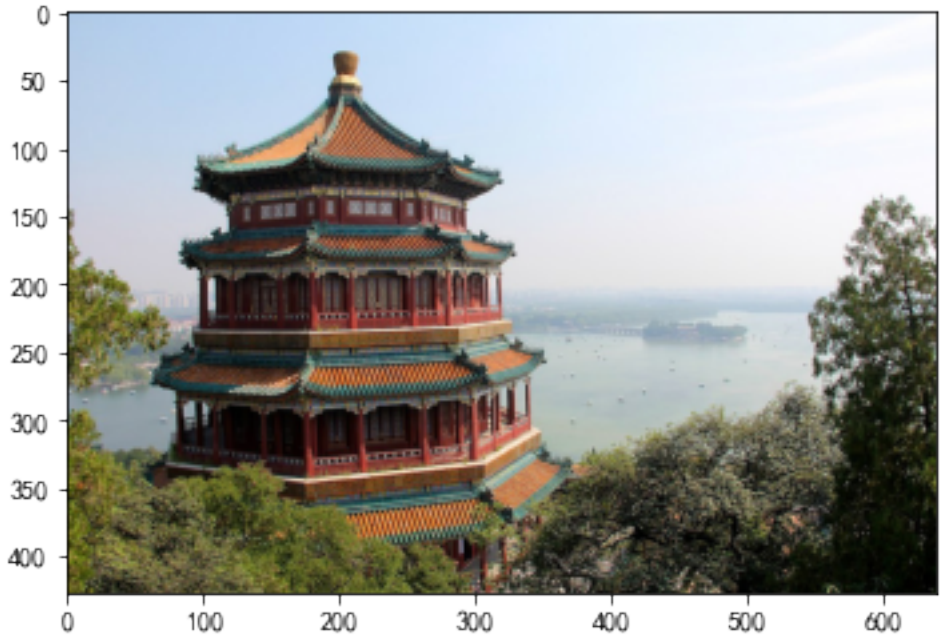
[19]: [<matplotlib.lines.Line2D at 0x24ae18c4f10>]



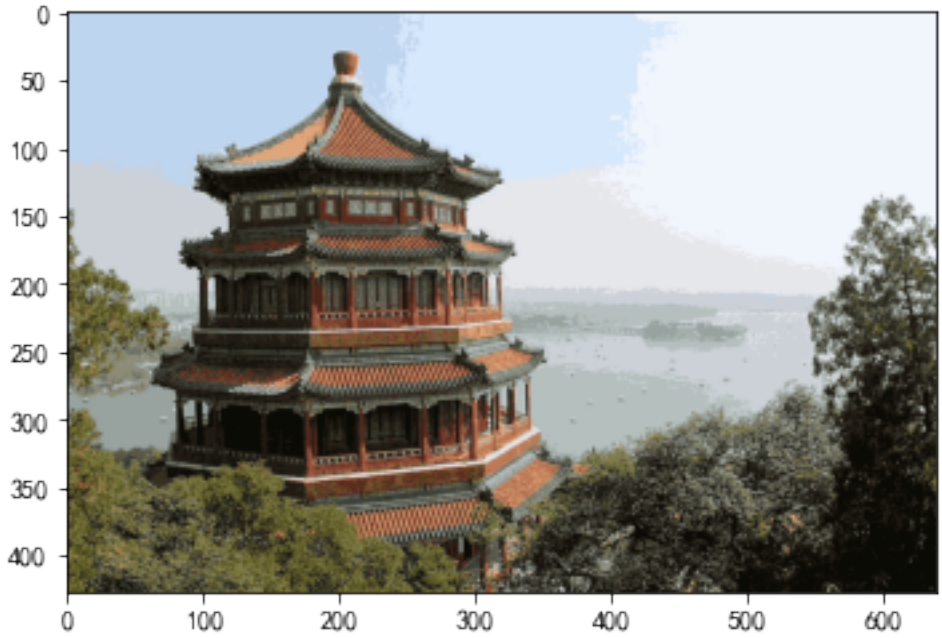
18 3.8

[20]:
```
#
from sklearn.datasets import load_sample_image
img1 = load_sample_image("china.jpg")
import numpy as np
img1 = np.array(img1, dtype=np.float64) / 255
```

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.imshow(img1)
plt.show()
m, n, p = img1.shape;
x = img1.reshape(-1,p)
print(img1.shape,x.shape)
#
from sklearn.cluster import KMeans
colors = 20
KM = KMeans(colors)
labels = KM.fit_predict(x)
color = KM.cluster_centers_
#
def recreate_img1(codebook, labels, m, n, p):
    img1 = np.zeros((m, n, p))
    label_idx = 0
    for i in range(m):
        for j in range(n):
            img1[i][j] = codebook[labels[label_idx]]
            label_idx += 1
    return img1
#
img2 = recreate_img1(KM.cluster_centers_, labels, m, n, p)
plt.imshow(img2)
plt.show()
num = np.unique(img1.reshape(-1,1))
```



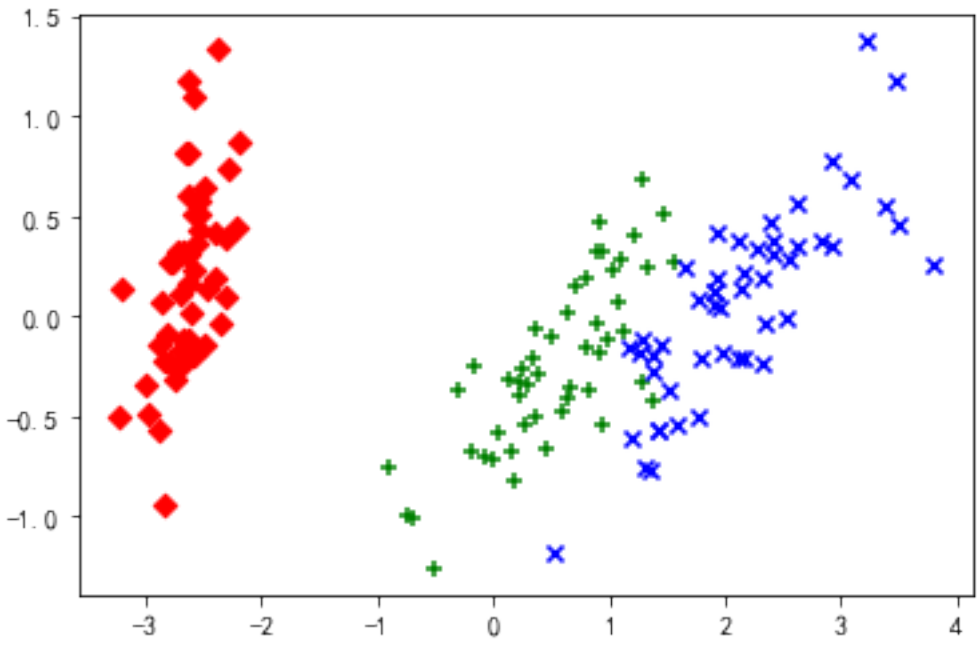
```
(427, 640, 3) (273280, 3)
```



19 4.1

[21]:
```
#
from sklearn.datasets import load_iris
data = load_iris()
X=data.data
y=data.target
#  PCA
from sklearn.decomposition import PCA
PCA_X=PCA(n_components=2)
reduced_X=PCA_X.fit_transform(X)
#
import matplotlib.pyplot as plt
plt.scatter(reduced_X[y==0,0],reduced_X[y==0,1],color='r',marker='D')
plt.scatter(reduced_X[y==1,0],reduced_X[y==1,1],color='g',marker='+')
plt.scatter(reduced_X[y==2,0],reduced_X[y==2,1],color='b',marker='x')
```

[21]:
```
<matplotlib.collections.PathCollection at 0x24bad99ab50>
```



20 4.2

[23]:
```
#
from sklearn.datasets import load_iris
load_data = load_iris()
x = load_data.data
y = load_data.target
print(x[:10])
#
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25)
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
x_train = std.fit_transform(x_train)
x_test = std.transform(x_test)
#  KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
result = knn.predict(x_test)
#
r_result = knn.score(x_test,y_test)
print("    ",result)
print("    ",y_test)
```

```
print("    ",r_result)
#    5cm 3cm   1cm 0.5cm
X_new = np.array([[5,3,1,0.5]])
prediction = knn.predict(X_new)
print("     {}".format(load_data ['target_names'][prediction]))
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
    [2 2 2 0 1 0 1 0 1 2 0 1 2 1 1 1 2 1 0 2 0 1 2 2 1 0 1 0 0 2 1 2 0 0 0 0
0
 2]
    [2 2 2 0 1 0 1 0 1 2 0 1 2 1 1 1 2 1 0 2 0 1 2 2 1 0 1 0 0 2 1 2 0 0 0 0
0
 2]
    1.0
     ['virginica']
```

21 4.3

[24]:
```
#
import numpy as np
X = np.array([
[182, 80, 1],
[177, 70, 1],
[160, 59, 0],
[154, 54, 0],
[165, 65, 1],
[192, 90, 1],
[174, 64, 0],
[176, 70, 0],
[158, 54, 0],
[172, 76, 1]
])
y = [44, 43, 38, 37, 40, 47, 39, 40, 37, 42]
k = 5
#  KNN
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(k)
```

```
knn.fit(X,y)
#
X_test = np.array([
[174, 59, 0],
[174, 75, 1]
])
predictions = knn.predict(X_test)
print("    ", predictions)
```

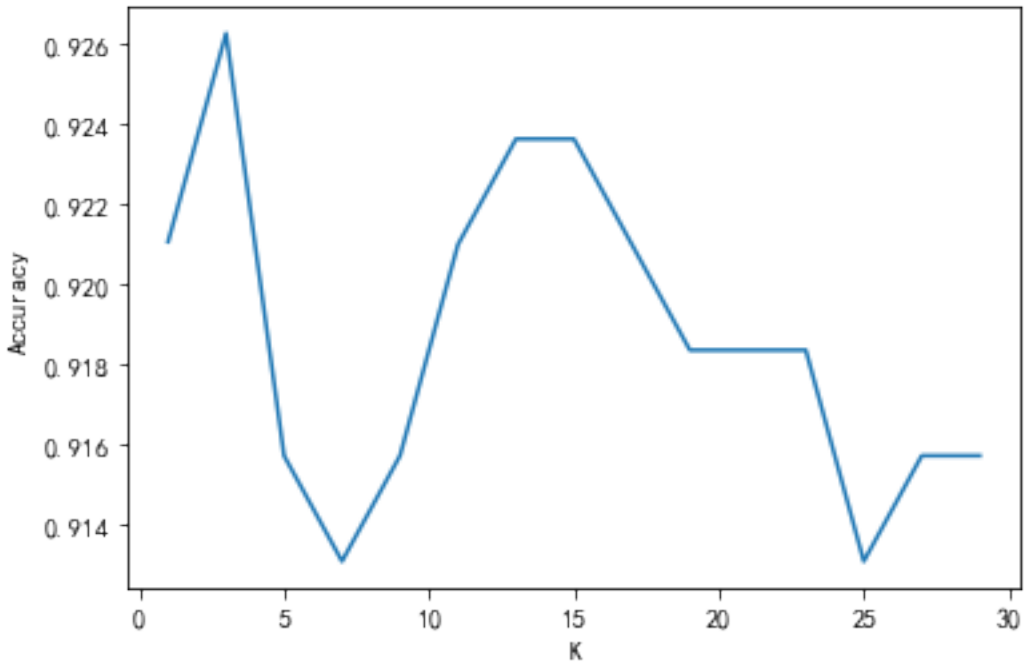
```
    [40.  41.6]
```

22 4.4

[25]:
```
#
from sklearn import datasets
breast_cancer = datasets.load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target
#
from sklearn.model_selection import train_test_split,cross_val_score
train_X,test_X,train_y,test_y = train_test_split(X,y,test_size=1/
 ↪3,random_state=3)
#     K
k_range = [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29]
cv_scores = []
from sklearn.neighbors import KNeighborsClassifier
for n in k_range:
    knn = KNeighborsClassifier(n)
    scores = cross_val_score(knn,train_X,train_y,cv=10,scoring='accuracy')
    cv_scores.append(scores.mean())
    print("     :%.2f" % scores.mean()," K    :%d" %n)
#
import matplotlib.pyplot as plt
plt.plot(k_range,cv_scores)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```

```
    :0.92  K    :1
    :0.93  K    :3
    :0.92  K    :5
    :0.91  K    :7
    :0.92  K    :9
    :0.92  K    :11
    :0.92  K    :13
    :0.92  K    :15
```

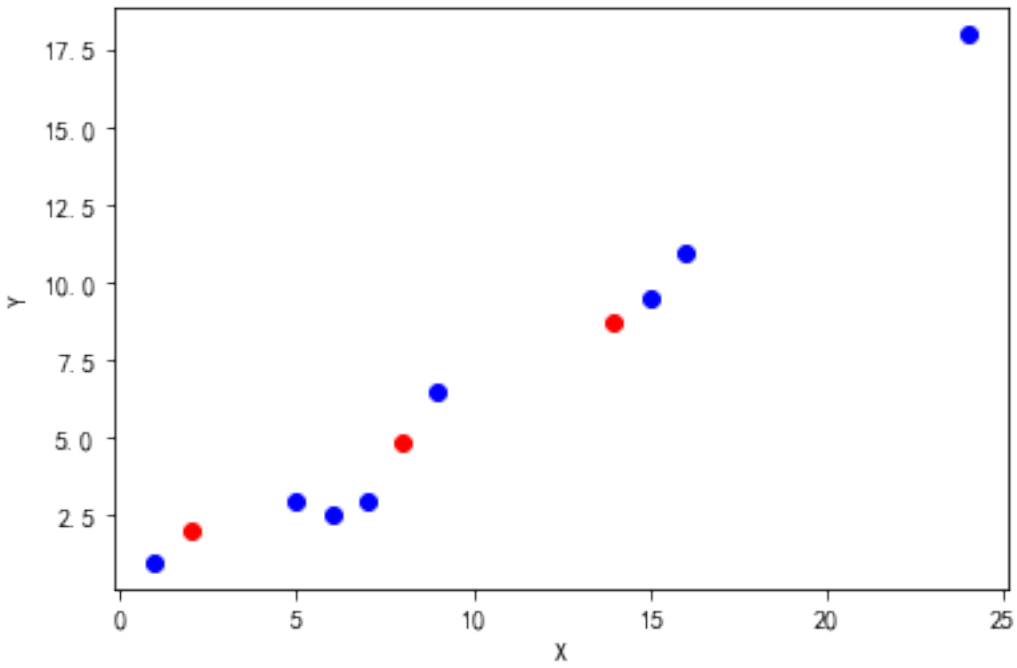
```
:0.92  K   :17
:0.92  K   :19
:0.92  K   :21
:0.92  K   :23
:0.91  K   :25
:0.92  K   :27
:0.92  K   :29
```

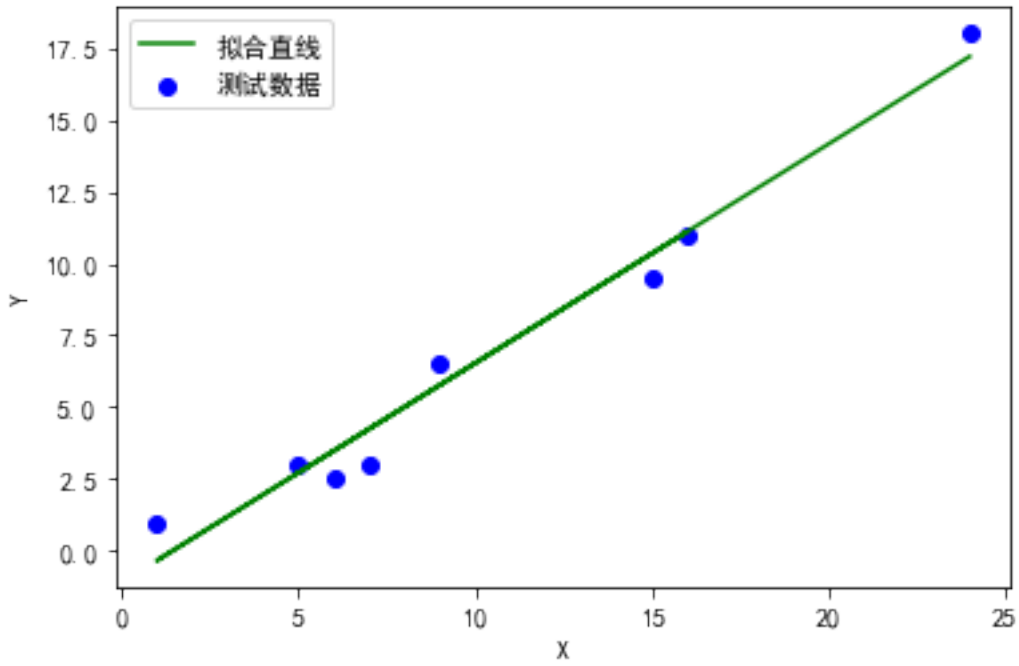


23 5.1

[26]:
```
#
import numpy as np
X = np.array([
           [1], [2], [5], [6], [7], [8], [9],[14], [15], [16], [24]
])
Y = [1,2,3,2.5,3,4.9,6.5,8.7,9.5,11,18]
#
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,␣
 ↪random_state=10010)
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
```

```
plt.rcParams['axes.unicode_minus'] = False
plt.scatter(x_train,y_train,label='train',color='b')
plt.scatter(x_test,y_test,label='test',color='r')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
#
from sklearn.linear_model import LinearRegression
LR=LinearRegression()
LR.fit(x_train, y_train)
#
plt.scatter(x_train,y_train,color='b',label='  ')
y_train_pred=LR.predict(x_train)
plt.plot(x_train,y_train_pred,color='green',label='  ')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
#
a=LR.intercept_
b=LR.coef_
print('    a=',a,' b=',b)
```






```
a= -1.0932968095391553  b= [0.76200451]
```

24 5.2

[27]:
```
#
from sklearn import datasets
boston = datasets.load_boston()
X, Y = boston.data, boston.target
#
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,␣
 ↪random_state=1001)
#
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train, y_train)
y_predict = reg.predict(x_test)
#
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_predict))
print(reg.score(x_train, y_train))
print(reg.score(x_test, y_test))
```

```
29.824006898863182
```

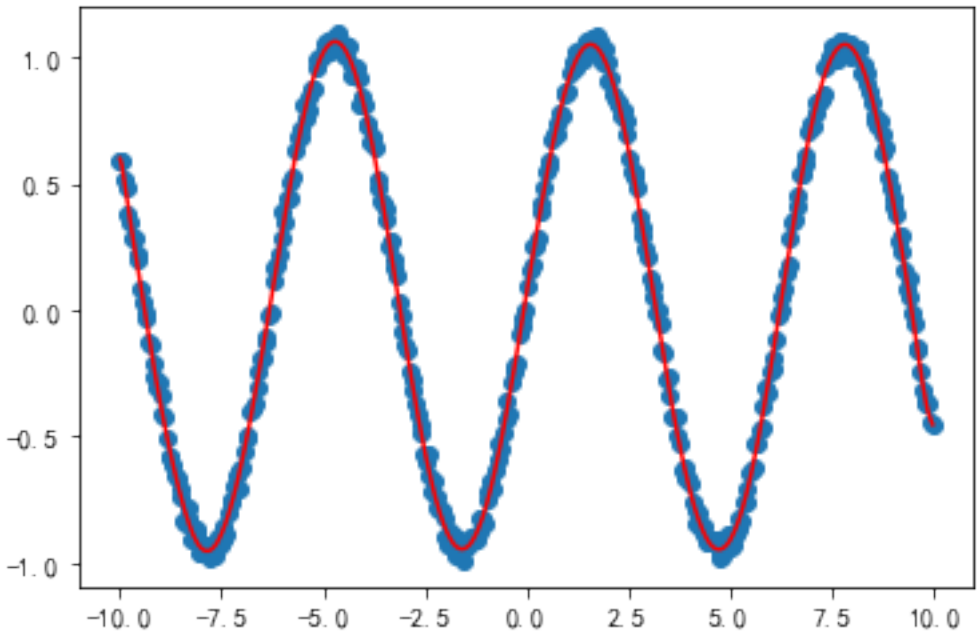
```
0.7570401119326386
0.6783942923302058
```

25 5.3

[28]:
```
#
import numpy as np
X = np.linspace(-10, 10, 400)
Y = np.sin(X) + 0.1*np.random.rand(len(X))
X = X.reshape(-1, 1)
Y = Y.reshape(-1, 1)
#
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
dim = 30
def polynomial_LR(degree=1):
    polynomial_features = PolynomialFeatures(degree=degree,include_bias=False)
    linear_regression = LinearRegression(normalize=True)
    pipeline = Pipeline([("polynomial_features",␣
 ↪polynomial_features),("linear_regression", linear_regression)])
    return pipeline
from sklearn.metrics import mean_squared_error
model= polynomial_LR(degree=dim)
model.fit(X, Y)
#
train_score = model.score(X, Y)
mse = mean_squared_error(Y, model.predict(X))
print(train_score)
print(mse)
import matplotlib.pyplot as plt
plt.scatter(X, Y)
plt.plot(X, model.predict(X), 'r-')
```

```
0.9984693855748482
0.0007321031979295839
```

[28]:
```
[<matplotlib.lines.Line2D at 0x24baca41dc0>]
```



26 6.1

[29]:
```
import pandas as pd
#
credit_card = pd.read_csv('d:/creditcard.csv')
#
print(credit_card.shape)
# 3
print(credit_card.head(3))
```

```
(284807, 31)
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
```

```
2 -0.139097 -0.055353 -0.059752  378.66      0

[3 rows x 31 columns]
```

27 6.2

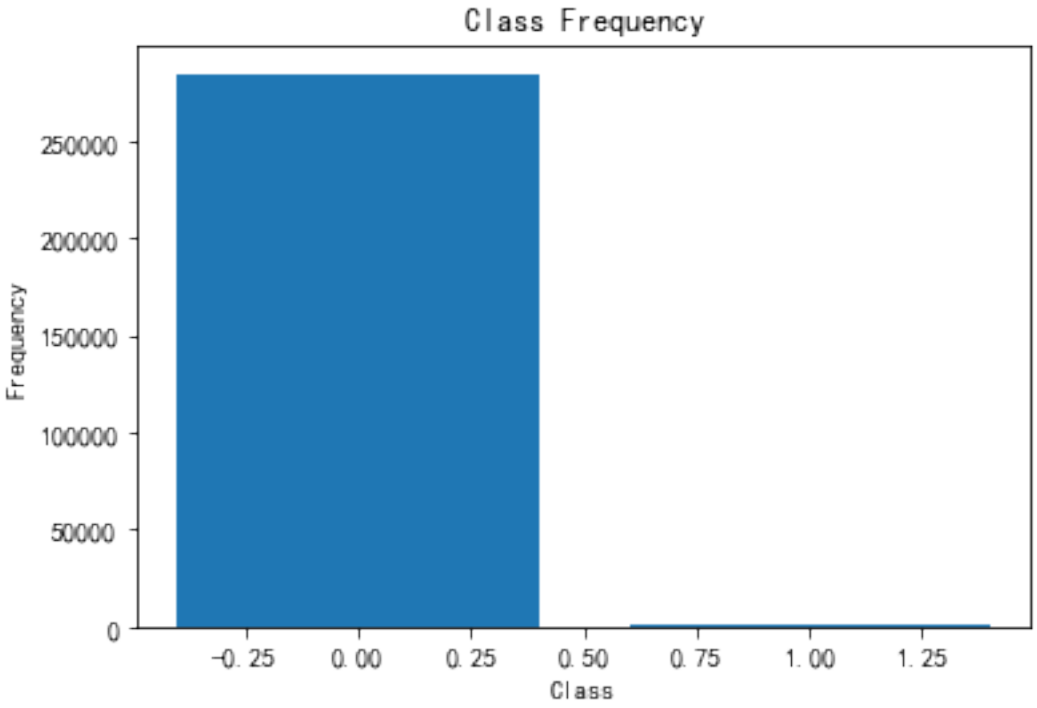
[30]:
```
#
import pandas as pd
credit_card = pd.read_csv('d:/creditcard.csv')
X = credit_card.drop(columns='Class', axis=1)
y = credit_card.Class.values
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
X_train = std.fit_transform(X_train)
X_test = std.transform(X_test)
#
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(max_iter=1000)
LR.fit(X_train, y_train)
y_train_hat = LR.predict(X_train)
y_train_hat_probs = LR.predict_proba(X_train)[:,1]
#
from sklearn.metrics import roc_curve, roc_auc_score, classification_report,␣
 ↪accuracy_score, confusion_matrix
train_accuracy = accuracy_score(y_train, y_train_hat)*100
train_auc_roc = roc_auc_score(y_train, y_train_hat_probs)*100
print('  :\n', confusion_matrix(y_train, y_train_hat))
print('  AUC: %.4f %%' % train_auc_roc)
print('   : %.4f %%' % train_accuracy)
#
y_test_hat = LR.predict(X_test)
y_test_hat_probs = LR.predict_proba(X_test)[:,1]
test_accuracy = accuracy_score(y_test, y_test_hat)*100
test_auc_roc = roc_auc_score(y_test, y_test_hat_probs)*100
print('  :\n', confusion_matrix(y_test, y_test_hat))
print('   AUC: %.4f %%' % test_auc_roc)
print('    : %.4f %%' % test_accuracy)
print(classification_report(y_test, y_test_hat, digits=6))
```

```
  :
[[213209     31]
[   141    224]]
 AUC: 98.0441 %
```

```
  : 99.9195 %
 :
[[71062    13]
[   46    81]]
 AUC: 97.6237 %
  : 99.9171 %
              precision    recall  f1-score   support

           0   0.999353  0.999817  0.999585     71075
           1   0.861702  0.637795  0.733032       127

    accuracy                       0.999171     71202
   macro avg   0.930528  0.818806  0.866308     71202
weighted avg   0.999108  0.999171  0.999110     71202
```

28 6.3

[31]:
```
unique, counts = np.unique(y, return_counts=True)
plt.bar(unique,counts)
plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```



29 6.4

[34]:
```
#
pip install imbalanced-learn
```

```
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.9.0-py3-none-any.whl (199 kB)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.0.1)
Collecting scikit-learn>=1.0.1
  Downloading scikit_learn-1.0.2-cp38-cp38-win_amd64.whl (7.2 MB)
Note: you may need to restart the kernel to use updated packages.
```

```
ERROR: Exception:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 171, in
_merge_into_criterion
    crit = self.state.criteria[name]
KeyError: 'scikit-learn'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\urllib3\response.py", line 438, in _error_catcher
    yield
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\urllib3\response.py", line 519, in read
    data = self._fp.read(amt) if not fp_closed else b""
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\cachecontrol\filewrapper.py", line 62, in read
    data = self.__fp.read(amt)
  File "C:\ProgramData\Anaconda3\lib\http\client.py", line 458, in read
    n = self.readinto(b)
  File "C:\ProgramData\Anaconda3\lib\http\client.py", line 502, in readinto
    n = self.fp.readinto(b)
  File "C:\ProgramData\Anaconda3\lib\socket.py", line 669, in readinto
    return self._sock.recv_into(b)
  File "C:\ProgramData\Anaconda3\lib\ssl.py", line 1241, in recv_into
    return self.read(nbytes, buffer)
  File "C:\ProgramData\Anaconda3\lib\ssl.py", line 1099, in read
    return self._sslobj.read(len, buffer)
socket.timeout: The read operation timed out
```

```
During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\cli\base_command.py", line 189, in _main
    status = self.run(options, args)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\cli\req_command.py", line 178, in wrapper
    return func(self, options, args)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\commands\install.py", line 316, in run
    requirement_set = resolver.resolve(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\resolver.py", line 121, in resolve
    self._result = resolver.resolve(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 453, in resolve
    state = resolution.resolve(requirements, max_rounds=max_rounds)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 347, in resolve
    failure_causes = self._attempt_to_pin_criterion(name, criterion)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 207, in
_attempt_to_pin_criterion
    criteria = self._get_criteria_to_update(candidate)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 199, in
_get_criteria_to_update
    name, crit = self._merge_into_criterion(r, parent=candidate)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 173, in
_merge_into_criterion
    crit = Criterion.from_requirement(self._p, requirement, parent)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\resolvers.py", line 82, in from_requirement
    if not cands:
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\resolvelib\structs.py", line 124, in __bool__
    return bool(self._sequence)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\found_candidates.py", line 143, in
__bool__
    return any(self)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\found_candidates.py", line 38, in
_iter_built
    candidate = func()
```

```
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\factory.py", line 167, in
_make_candidate_from_link
    self._link_candidate_cache[link] = LinkCandidate(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\candidates.py", line 300, in
__init__
    super().__init__(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\candidates.py", line 144, in
__init__
    self.dist = self._prepare()
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\candidates.py", line 226, in
_prepare
    dist = self._prepare_distribution()
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\resolution\resolvelib\candidates.py", line 311, in
_prepare_distribution
    return self._factory.preparer.prepare_linked_requirement(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\operations\prepare.py", line 457, in
prepare_linked_requirement
    return self._prepare_linked_requirement(req, parallel_builds)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\operations\prepare.py", line 480, in
_prepare_linked_requirement
    local_file = unpack_url(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\operations\prepare.py", line 230, in unpack_url
    file = get_http_url(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\operations\prepare.py", line 108, in get_http_url
    from_path, content_type = download(link, temp_dir.path)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\network\download.py", line 163, in __call__
    for chunk in chunks:
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\cli\progress_bars.py", line 159, in iter
    for x in it:
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_internal\network\utils.py", line 64, in response_chunks
    for chunk in response.raw.stream(
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\urllib3\response.py", line 576, in stream
    data = self.read(amt=amt, decode_content=decode_content)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\urllib3\response.py", line 541, in read
```

```
    raise IncompleteRead(self._fp_bytes_read, self.length_remaining)
  File "C:\ProgramData\Anaconda3\lib\contextlib.py", line 131, in __exit__
    self.gen.throw(type, value, traceback)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\pip\_vendor\urllib3\response.py", line 443, in _error_catcher
    raise ReadTimeoutError(self._pool, None, "Read timed out.")
pip._vendor.urllib3.exceptions.ReadTimeoutError:
HTTPSConnectionPool(host='files.pythonhosted.org', port=443): Read timed out.
```

[ ]:
```
#
import pandas as pd
credit_card = pd.read_csv('d:/creditcard.csv')
X = credit_card.drop(columns='Class', axis=1)
y = credit_card.Class.values
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
#   SMOTE
from imblearn.over_sampling import SMOTE
OS=SMOTE(random_state=1)
X_train,y_train=OS.fit_resample(X_train,y_train)
#
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
X_train = std.fit_transform(X_train)
X_test = std.transform(X_test)
#
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(max_iter=1000)
LR.fit(X_train, y_train)
y_train_hat = LR.predict(X_train)
y_train_hat_probs = LR.predict_proba(X_train)[:,1]
#
from sklearn.metrics import roc_curve, roc_auc_score, classification_report,␣
 ↪accuracy_score, confusion_matrix
train_accuracy = accuracy_score(y_train, y_train_hat)*100
train_auc_roc = roc_auc_score(y_train, y_train_hat_probs)*100
print('  :\n', confusion_matrix(y_train, y_train_hat))
print('  AUC: %.4f %%' % train_auc_roc)
print('   : %.4f %%' % train_accuracy)
y_test_hat = LR.predict(X_test)
y_test_hat_probs = LR.predict_proba(X_test)[:,1]
test_accuracy = accuracy_score(y_test, y_test_hat)*100
test_auc_roc = roc_auc_score(y_test, y_test_hat_probs)*100
print('  :\n', confusion_matrix(y_test, y_test_hat))
print('   AUC: %.4f %%' % test_auc_roc)
print('    : %.4f %%' % test_accuracy)
```

```
print(classification_report(y_test, y_test_hat, digits=6))
```

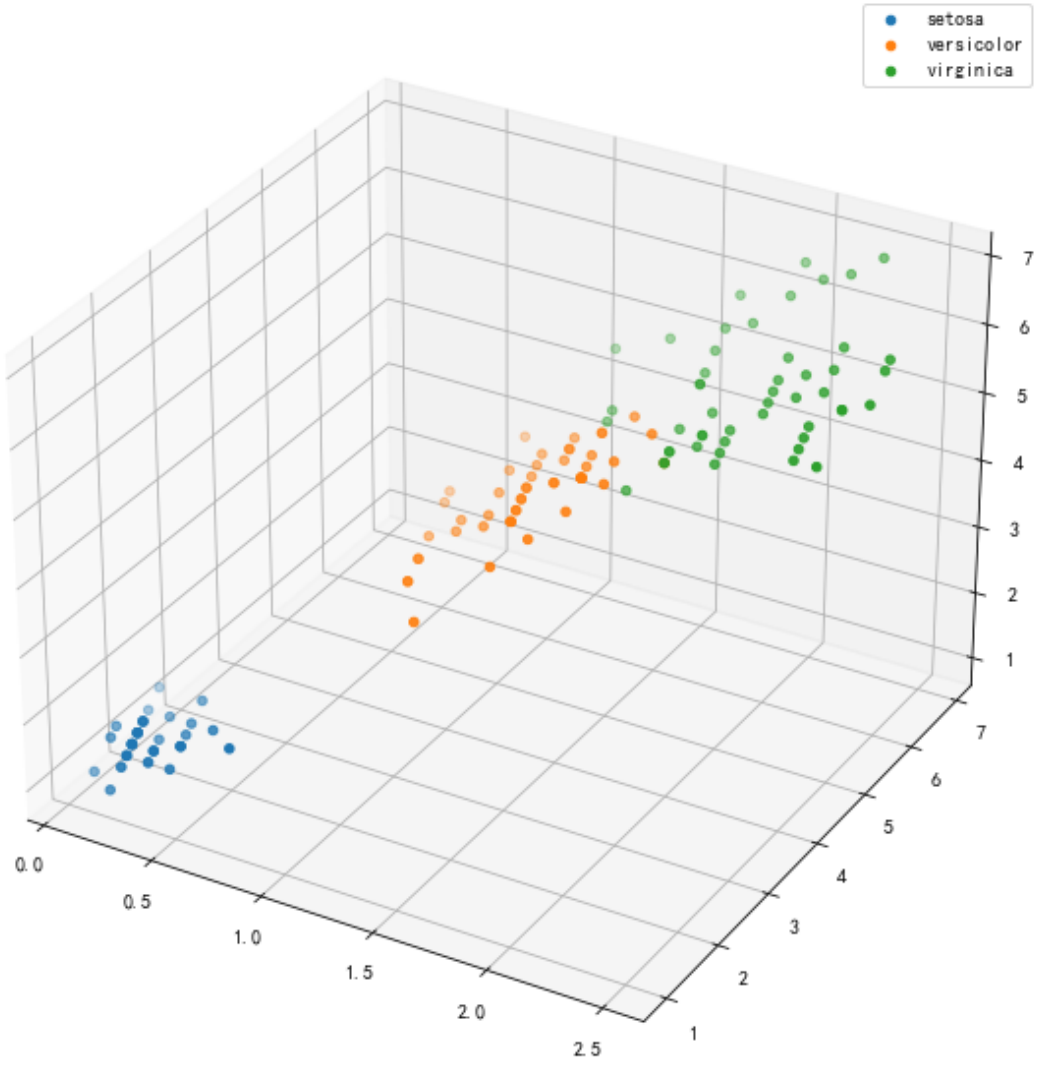
30 6.5

[35]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size = 0.25)
#
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
#
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(penalty='l2',C=100,multi_class='ovr')
LR.fit(X_train,y_train)
y_predict = LR.predict(X_test)
#
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
print(y_test)
print(y_predict)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.92      0.96        13
           2       0.92      1.00      0.96        12

    accuracy                           0.97        38
   macro avg       0.97      0.97      0.97        38
weighted avg       0.98      0.97      0.97        38

[0 0 0 0 2 2 1 1 0 1 2 2 0 1 1 1 2 2 1 1 0 2 0 0 2 1 1 1 2 0 1 1 0 0 0 2 2
 2]
[0 0 0 0 2 2 1 1 0 1 2 2 0 1 1 1 2 2 1 1 0 2 0 0 2 1 1 2 2 0 1 1 0 0 0 2 2
 2]
```

31 7.1

[36]:
```
import pandas as pd
from sklearn import datasets
data = datasets.load_iris()
X = pd.DataFrame(data=data.data, columns=data.feature_names)
#   DataFrame
X['target'] = data.target
C0 = X[X['target'] == 0].values
C1 = X[X['target'] == 1].values
C2 = X[X['target'] == 2].values
#    scatter()  x y z
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(C0[:, 3], C0[:, 2], C0[:, 2], label='setosa')
ax.scatter(C1[:, 3], C1[:, 2], C1[:, 2], label='versicolor')
ax.scatter(C2[:, 3], C2[:, 2], C2[:, 2], label='virginica')
#
plt.legend()
plt.show()
```

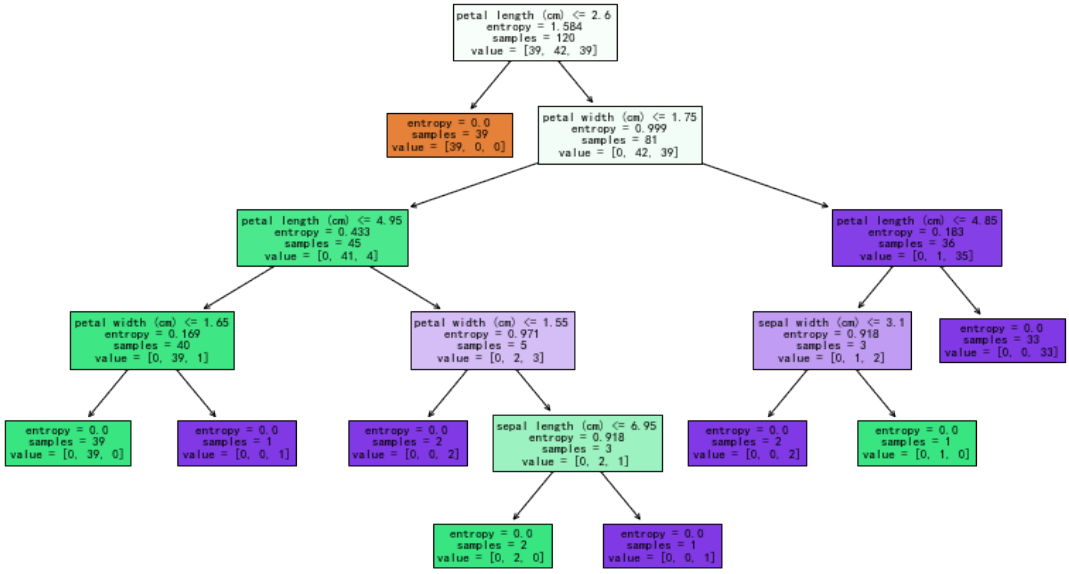


32 7.2

[37]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris['data']
y = iris['target']
feature_names = iris.feature_names
#
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=1024)
```

```
#
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_ = clf.predict(X_test)
#
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_))
#
import matplotlib.pyplot as plt
plt.figure(figsize=(14, 8))
from sklearn import tree
tree.plot_tree(clf, filled=True, feature_names=feature_names)
plt.show()
```

```
1.0
```

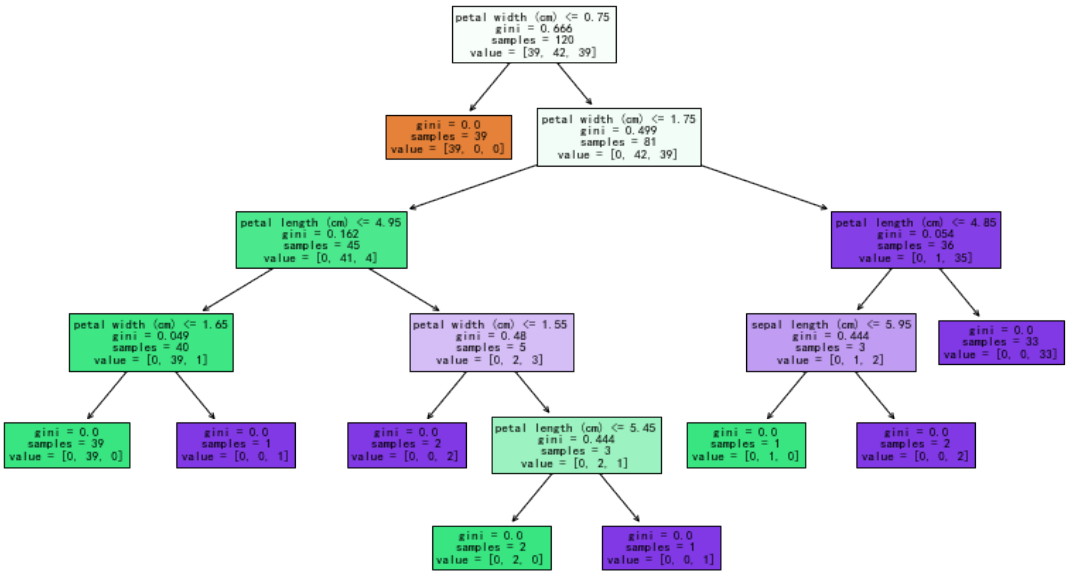


33 7.3

[38]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris['data']
y = iris['target']
feature_names = iris.feature_names
#
```

```
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=1024)
#
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='gini')
clf.fit(X_train, y_train)
y_ = clf.predict(X_test)
#
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_))
#
import matplotlib.pyplot as plt
plt.figure(figsize=(14, 8))
from sklearn import tree
tree.plot_tree(clf, filled=True, feature_names=feature_names)
plt.show()
```

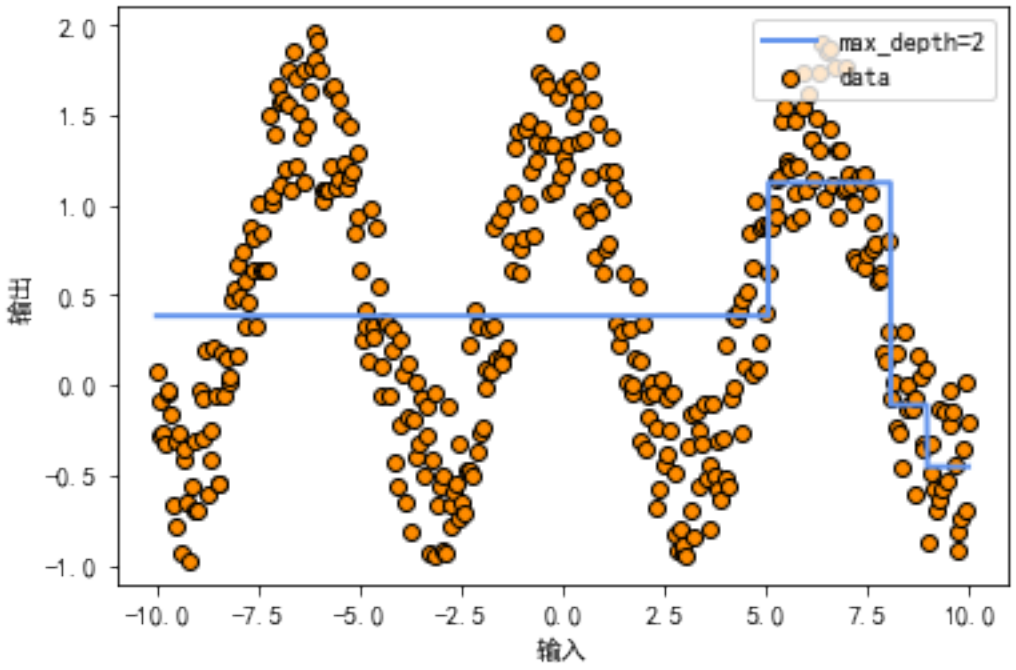
```
1.0
```



34 7.4

[39]:
```
import numpy as np
num=400
X=np.linspace(-10,10,num)
```

```
X=X.reshape(num,1)
y=np.cos(X).ravel()+np.random.rand(len(X))
# 2
from sklearn.tree import DecisionTreeRegressor
DTR=DecisionTreeRegressor(max_depth=2)
DTR.fit(X,y)
#
X_test=np.arange(-10,10.0,0.01)[:,np.newaxis]
y_predict=DTR.predict(X_test)
#
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.figure()
plt.scatter(X,y,edgecolor="black",c="darkorange",label="data")
plt.
 ↪plot(X_test,y_predict,color="cornflowerblue",label="max_depth=2",linewidth=2)
plt.xlabel(" ")
plt.ylabel(" ")
plt.legend()
plt.show()
```



35 7.5

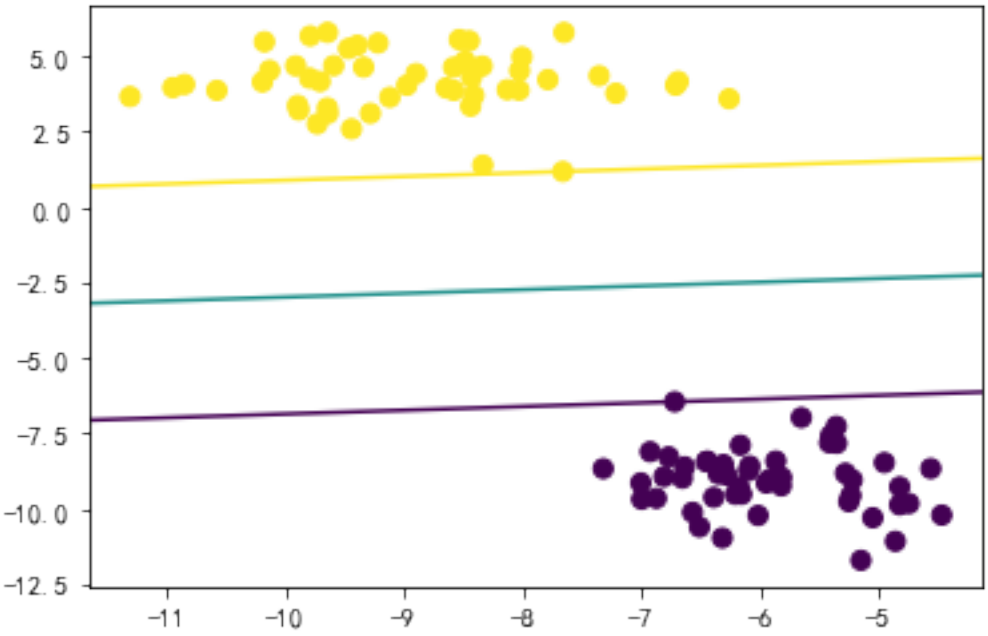
[40]:
```
from sklearn import datasets
boston = datasets.load_boston()
X, Y = boston.data, boston.target
print(X.shape)
print(Y.shape)
#
from sklearn.feature_selection import SelectPercentile, f_regression
selector = SelectPercentile(f_regression, percentile=10)

X_new = selector.fit_transform(X, Y)
print(X_new.shape)
X = X_new
#
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,␣
 ↪random_state=1001)
#
from sklearn.tree import DecisionTreeRegressor
reg = DecisionTreeRegressor(max_depth=8)
reg.fit(x_train, y_train)
y_predict = reg.predict(x_test)
#
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_predict))
print(reg.score(x_train, y_train))
print(reg.score(x_test, y_test))
```

```
(506, 13)
(506,)
(506, 2)
29.809854701224847
0.9609178265285536
0.6785469018555453
```

36 8.1

[41]:
```
from sklearn.datasets import make_blobs
data,label = make_blobs(n_samples=100, centers=2)
#SVC     data label SVM
from sklearn import svm
SVM = svm.SVC(kernel ="linear",C=1000)
SVM.fit(data,label)
#
import matplotlib.pyplot as plt
plt.scatter(data[:, 0],data[:, 1],c=label,s=50)
ax = plt.gca()
```

```
xlim = ax.get_xlim()
ylim = ax.get_ylim()
import numpy as np
xx = np.linspace(xlim[0],xlim[1],50)
yy = np.linspace(ylim[0],ylim[1],50)
Y,X = np.meshgrid(yy,xx)
xy = np.vstack([X.ravel(),Y.ravel()]).T
Z = SVM.decision_function(xy).reshape(X.shape)
ax.contour(X,Y,Z,levels=[-1,0,1])
ax.scatter(SVM.support_vectors_[:, 0],
          SVM.support_vectors_[:, 1],
           s=100,linewidth=1,facecolors="none")
plt.show()
```



37 8.2

[42]:
```
pip install tensorflow
pip install matplotlib
pip install sklearn
```

```
  File "<ipython-input-42-53def1ffb674>", line 1
    pip install tensorflow
        ^
SyntaxError: invalid syntax
```

[ ]:
```
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
#
X_train = X_train / 255.0
X_test = X_test / 255.0
#
label_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
#
import matplotlib.pyplot as plt
plt.figure(figsize=(10,4))
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.imshow(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(label_names[y_train[i]])
#
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train_flat = X_train.reshape(X_train.shape[0],  X_train.shape[1]* X_train.
 ↪shape[2])
X_test_flat = X_test.reshape(X_test.shape[0],  X_test.shape[1] * X_test.shape[2])
#SVC
from sklearn.svm import SVC
svc = SVC(C=1, kernel='linear', gamma="auto")
svc.fit(X_train_flat, y_train)
y_pred_svc = svc.predict(X_test_flat)
#
from sklearn import metrics
F1 = metrics.f1_score(y_test, y_pred_svc, average= "weighted")
Accuracy = metrics.accuracy_score(y_test, y_pred_svc)
CM = metrics.confusion_matrix(y_test, y_pred_svc)
print("  : {}".format(Accuracy))
print("  : \n", CM)
print(metrics.classification_report(y_test, y_pred_svc))
```

38 8.3

[ ]:
```
import numpy as np
X = np.sort(5 * np.random.rand(100, 1), axis=0)
y = np.sin(X).ravel()
y[::4] += 3 * (0.5 - np.random.rand(25))
#
```

```
from sklearn.svm import SVR
RBF = SVR(kernel='rbf')
RBF_clf=RBF.fit(X,y).predict(X)
Linear = SVR(kernel='linear')
Linear_clf=Linear.fit(X,y).predict(X)
Poly = SVR(kernel='poly')
Poly_clf=Poly.fit(X,y).predict(X)
#
import matplotlib.pyplot as plt
plt.scatter(X,y,c="black")
plt.plot(X,RBF_clf,c="blue")
plt.scatter(X,y,c="black")
plt.plot(X,Linear_clf,c="green")
plt.scatter(X,y,c="black")
plt.plot(X,Poly_clf,c="red")
```

39 8.4

[ ]:
```
from sklearn import datasets
data = datasets.load_boston()
X = data.data
y = data.target
#
from sklearn.model_selection import train_test_split
Xtrain,Xtest,Ytrain,Ytest=train_test_split(X, y, test_size=0.02)
#
from sklearn.preprocessing import StandardScaler
Std_X = StandardScaler()
#
Xtrain=Std_X.fit_transform(Xtrain)
Xtest=Std_X.transform(Xtest)
Std_y = StandardScaler()
#
Ytrain=Std_y.fit_transform(Ytrain.reshape(-1, 1))
Ytest=Std_y.transform(Ytest.reshape(-1, 1))
#     rbf
from sklearn.svm import SVR
RBF = SVR(kernel='rbf')
RBF_clf=RBF.fit(Xtrain,Ytrain).predict(Xtest)
#     linear
Linear = SVR(kernel='linear')
Linear_clf=Linear.fit(Xtrain,Ytrain).predict(Xtest)
#     poly
Poly = SVR(kernel='poly')
Poly_clf=Poly.fit(Xtrain,Ytrain).predict(Xtest)
```

```
#   rbf
from sklearn import metrics
print(metrics.mean_squared_error(Ytest, RBF_clf))
print(RBF_clf)
#   linear
from sklearn import metrics
print(metrics.mean_squared_error(Ytest, Linear_clf))
print(Linear_clf)
#   poly
from sklearn import metrics
print(metrics.mean_squared_error(Ytest, Poly_clf))
```

40 9.1

[ ]:
```
import numpy as np
X = np.linspace(-10, 10, 400)
X_new = X.reshape(-1,1)
y = X*X + 1*np.random.rand(len(X))
#
from sklearn.neural_network import MLPRegressor
MLP = MLPRegressor(alpha=1e-6,hidden_layer_sizes=(10, 20), random_state=10,␣
 ↪max_iter=100000,activation='relu')
MLP.fit(X_new,y)
y_new = MLP.predict(X_new)
#
import matplotlib.pyplot as plt
plt.scatter(X,y,c="yellow")
plt.plot(X_new,y_new,c="black")
```

41 9.2

[ ]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size = 0.25)
#
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
#
from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(hidden_layer_sizes=(20,8), max_iter=100000)
MLP.fit(X_train,y_train)
y_predict = MLP.predict(X_test)
#
from sklearn.metrics import roc_curve, roc_auc_score, classification_report,␣
↪accuracy_score, confusion_matrix
train_accuracy = accuracy_score(y_test, y_predict)*100
print('  :\n', confusion_matrix(y_test, y_predict))
print(train_accuracy)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
```

42 10.1

[ ]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
#
from sklearn.ensemble import AdaBoostClassifier
Boosting = AdaBoostClassifier(n_estimators=50,learning_rate=1)
model = Boosting.fit(X_train, y_train)
y_pred = model.predict(X_test)
#
from sklearn import metrics
print("  :",metrics.accuracy_score(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))
```

43 10.2

[ ]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
# SVM
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
svc=SVC(probability=True, kernel='linear')
Boosting =AdaBoostClassifier(n_estimators=50,  base_estimator=svc,␣
 ↪learning_rate=1)
model = Boosting.fit(X_train, y_train)
y_pred = model.predict(X_test)
#
from sklearn import metrics
print("  :",metrics.accuracy_score(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))
```

44 10.3

[ ]:
```
#
from sklearn.datasets import fetch_lfw_people
Data = fetch_lfw_people(min_faces_per_person=70)
x=Data.data
n_features=x.shape[1]
y=Data.target
target_names=Data.target_names
#
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.imshow(x[i].reshape(62,47))
    plt.xlabel(target_names[y[i]])
#
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.6)
#
from sklearn.decomposition import PCA
PCA=PCA(n_components=100).fit(x_train)
x_train_pca = PCA.transform(x_train)
x_test_pca = PCA.transform(x_test)
#   knn
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train_pca, y_train)
#   adaboost
from sklearn.ensemble import AdaBoostClassifier
Ada_DTC = AdaBoostClassifier(n_estimators=100, learning_rate=0.2)
```

```
Ada_DTC.fit(x_train_pca,y_train)
#   SVC
from sklearn.svm import SVC
svc=SVC(probability=True, kernel='linear')
Ada_SVC =␣
 ↪AdaBoostClassifier(base_estimator=svc,n_estimators=100,learning_rate=0.2)
Ada_SVC.fit(x_train_pca,y_train)
#
y_pred1=knn.predict(x_test_pca)
y_pred2=Ada_DTC.predict(x_test_pca)
y_pred3=Ada_SVC.predict(x_test_pca)
#
from sklearn import metrics
print("###################KNN  ###################")
print(knn.score(x_test_pca, y_test))
print(metrics.classification_report(y_test,y_pred1))
print(metrics.confusion_matrix(y_test,y_pred1))
print("########Adaboost+      #########")
print(Ada_DTC.score(x_test_pca, y_test))
print(metrics.classification_report(y_test,y_pred2))
print(metrics.confusion_matrix(y_test,y_pred2))
print("###########Adaboost+SVC    ###########")
print(Ada_SVC.score(x_test_pca, y_test))
print(metrics.classification_report(y_test,y_pred3))
print(metrics.confusion_matrix(y_test,y_pred3))
```

45 10.4

[ ]:

```
import numpy as np
X = np.linspace(-10, 10, 300)
data = X.reshape(-1,1)
target = X + 0.4*np.random.rand(len(X))
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.plot(data, target)
plt.xlabel(' ')
plt.ylabel(' ')
plt.show()
#
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,target,train_size =0.98)
#
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.ensemble import AdaBoostRegressor
regr_1 = DecisionTreeRegressor(max_depth=20)
from sklearn.svm import SVR
regr_2 = AdaBoostRegressor(base_estimator=SVR(kernel='rbf'),n_estimators=50)
regr_3 = SVR(kernel='rbf')
regr_1.fit(X_train, y_train)
regr_2.fit(X_train, y_train)
regr_3.fit(X_train,y_train)
#
y_pred1 = regr_1.predict(X_test)
y_pred2 = regr_2.predict(X_test)
y_pred3 = regr_3.predict(X_test)
#
from sklearn.metrics import mean_squared_error
print('###############  ##################')
print(y_test)
print('################   ################')
print(y_pred1)
print(mean_squared_error(y_test, y_pred1))
print('############Adaboost+SVR#################')
print(y_pred2)
print(mean_squared_error(y_test, y_pred2))
print('##################SVR####################')
print(y_pred3)
print(mean_squared_error(y_test, y_pred3))
```