第7章　K最近鄰

範例7-1 載入資料

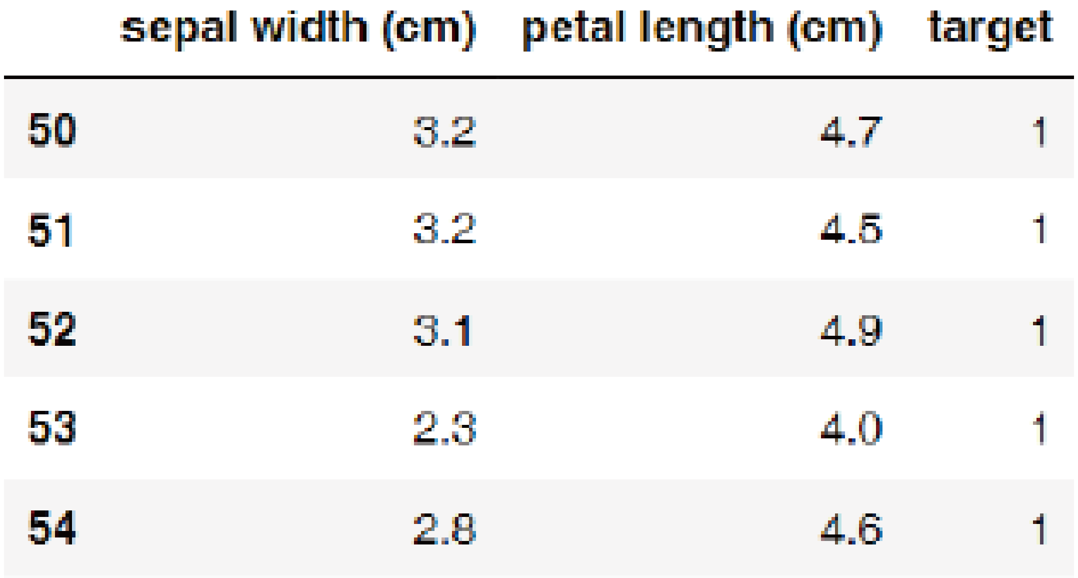
程式碼

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.rcParams['font.sans-serif'] =
['DFKai-sb']
plt.rcParams['axes.unicode_minus'
] = False
%config
InlineBackend.figure_format =
'retina‘
```

```
import warnings
warnings.filterwarnings('ignore')

from sklearn.datasets import
load_iris
iris = load_iris()
df = pd.DataFrame(iris['data'],
columns=iris['feature_names'])
df['target'] = iris['target']
df = df[['sepal width (cm)', 'petal
length (cm)','target']]
df = df.iloc[50:]
df.head()
```

執行結果



	sepal width (cm)	petal length (cm)	target
50	3.2	4.7	1
51	3.2	4.5	1
52	3.1	4.9	1
53	2.3	4.0	1
54	2.8	4.6	1

範例7-2 取出X和y

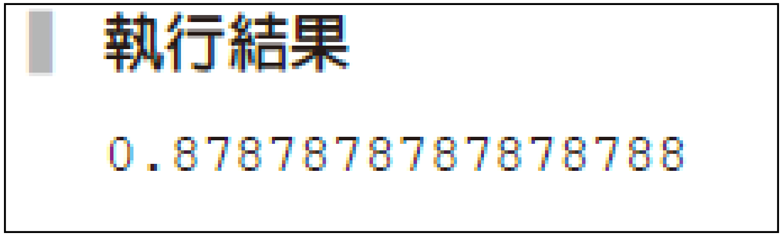
程式碼

```
X = df.drop('target', axis=1)
y = df['target']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                        test_size=0.33, random_state=42)
```

範例7-3 取出X和y

程式碼

```
from sklearn.neighbors import KNeighborsClassifier
# 初始物件
model = KNeighborsClassifier()
# 機器學習
model.fit(X_train, y_train)
#正確率的預測，model.score提供了簡便的正確率輸入方式
model.score(X_test, y_test)
```



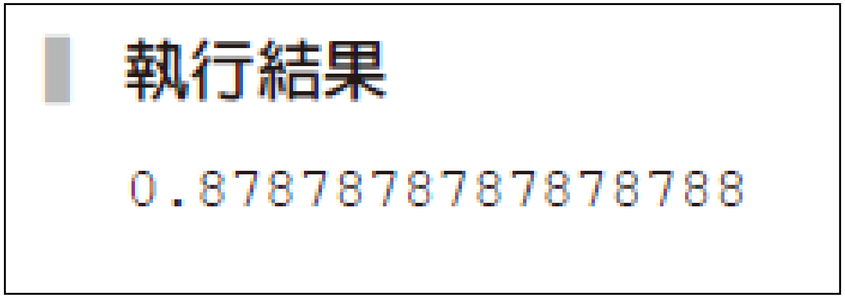
執行結果

```
0.8787878787878788
```

範例7-4 用標準化的資料來分析

程式碼

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
model_pl = make_pipeline(StandardScaler(),
                        KNeighborsClassifier())
model_pl.fit(X_train, y_train)
model_pl.score(X_test, y_test)
```



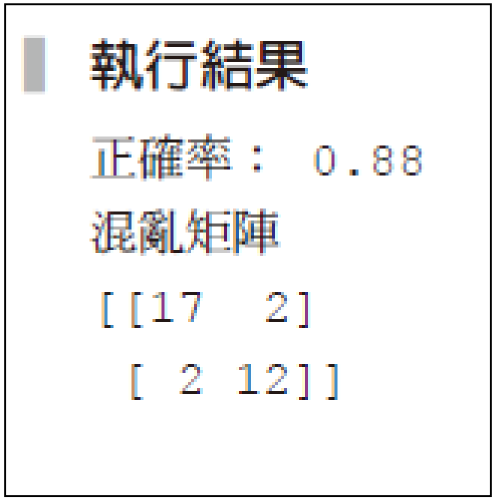
執行結果

```
0.8787878787878788
```

範例7-5 預測結果分析

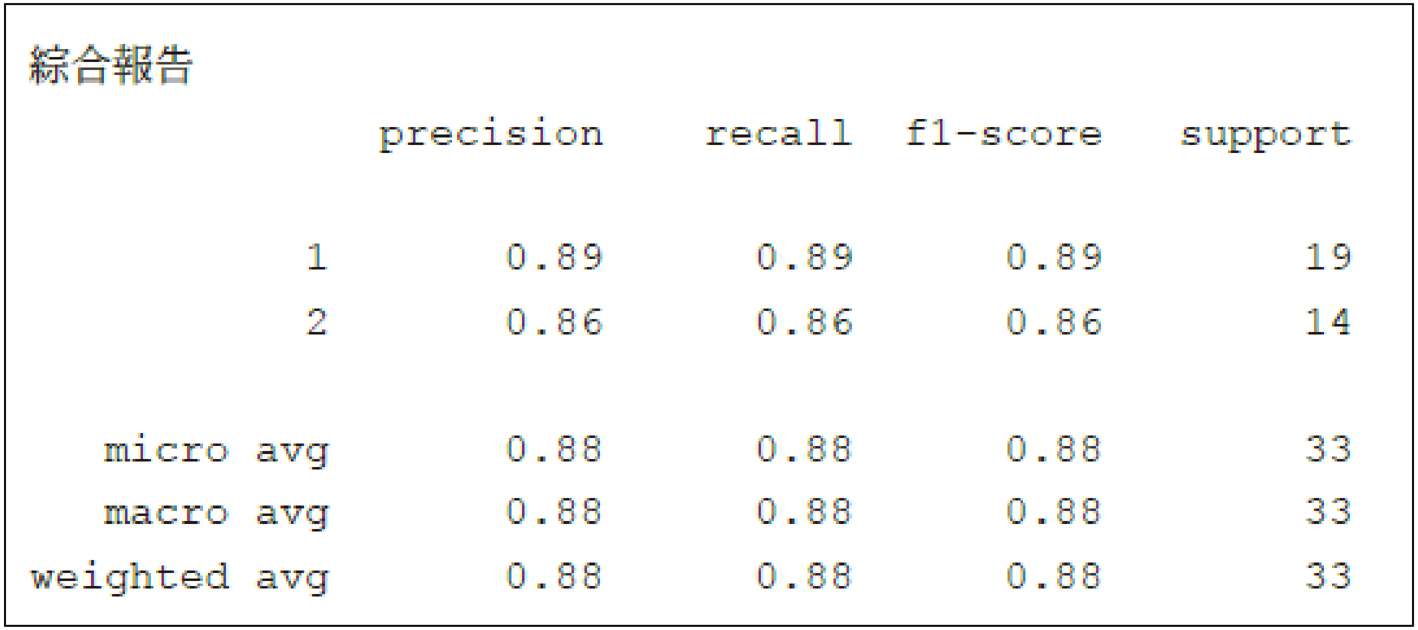
程式碼

```
from sklearn.metrics import confusion_matrix, accuracy_score,
                                              classification_report
y_pred = model_pl.predict(X_test)
print('正確率：', accuracy_score(y_test, y_pred).round(2))
print('混亂矩陣')
print(confusion_matrix(y_test, y_pred))
print('綜合報告')
print(classification_report(y_test, y_pred))
```



執行結果

```
正確率： 0.88
混亂矩陣
[[17  2]
 [ 2 12]]
```



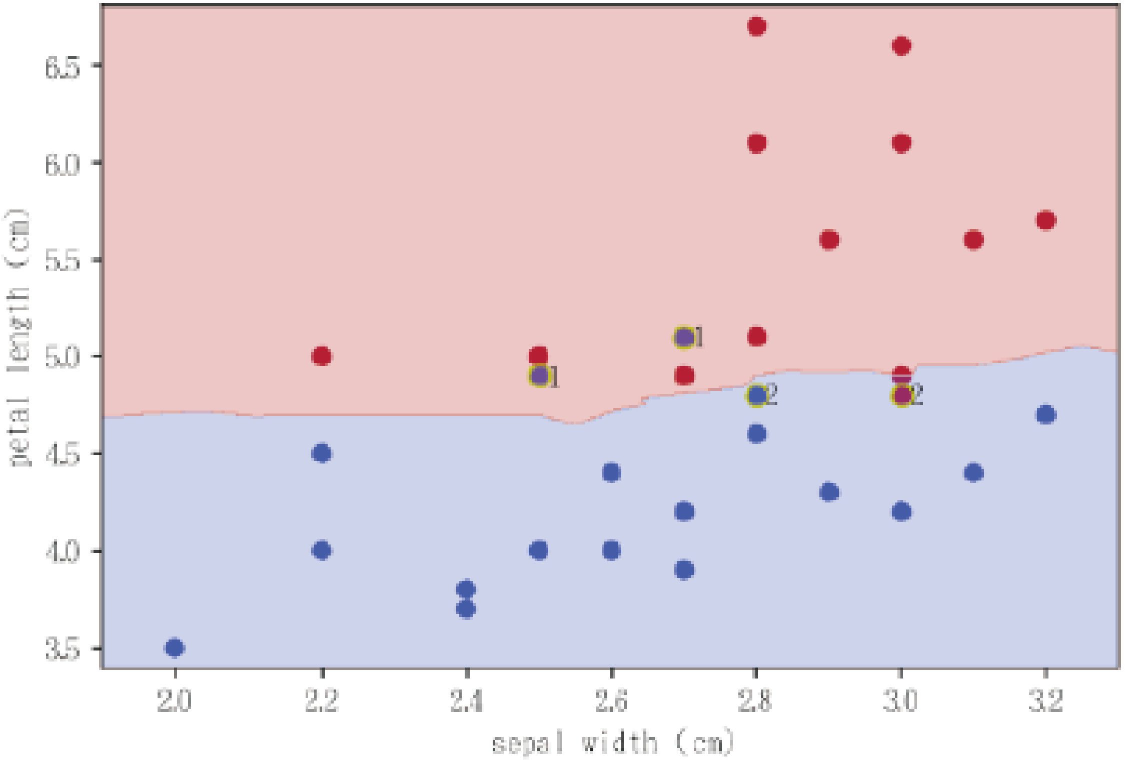
```
綜合報告
              precision    recall  f1-score   support

           1       0.89      0.89      0.89        19
           2       0.86      0.86      0.86        14

   micro avg       0.88      0.88      0.88        33
   macro avg       0.88      0.88      0.88        33
weighted avg       0.88      0.88      0.88        33
```

範例7-6 繪製未標準化結果的預測邊界

執行結果

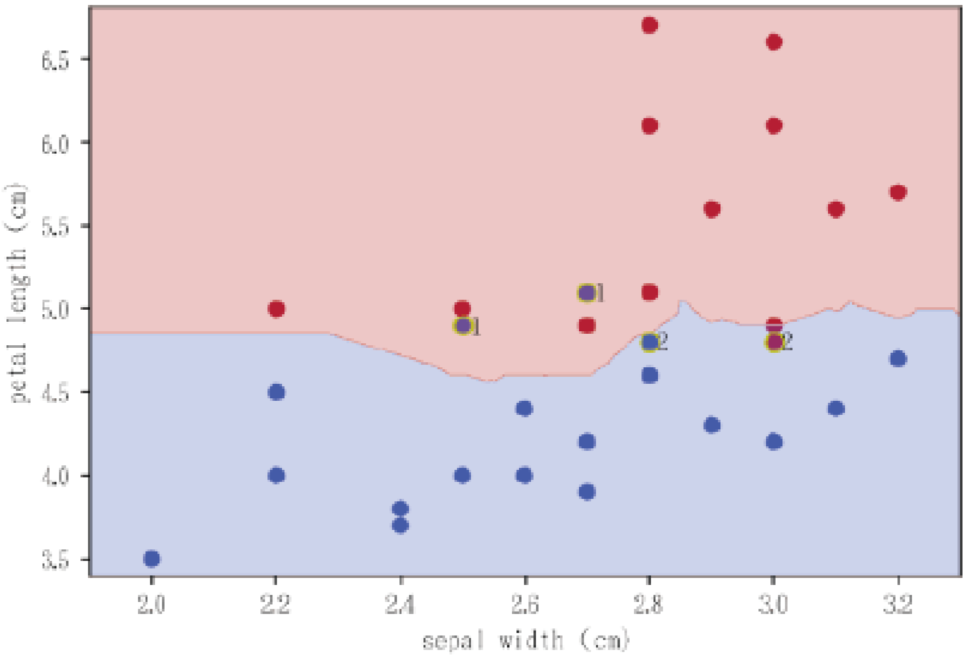


範例7-7　繪製標準化資料的預測邊界

程式碼

```
plot_decision_boundary(X_test, y_test, model_pl, True)
```

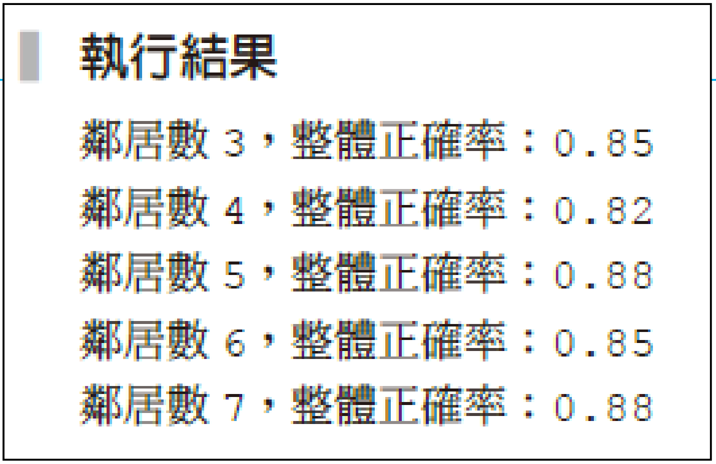
執行結果



範例7-8 n_neighbors數目的選擇

程式碼

```
accs = []
for n in range(3,8):
    model_pl = make_pipeline(StandardScaler(),
                             KNeighborsClassifier(n_neighbors=n))
    model_pl.fit(X_train, y_train)
    print(f'鄰居數{n}，整體正確率：{model_pl.score(X_test,
y_test).round(2)}')
```



執行結果

```
鄰居數 3，整體正確率：0.85
鄰居數 4，整體正確率：0.82
鄰居數 5，整體正確率：0.88
鄰居數 6，整體正確率：0.85
鄰居數 7，整體正確率：0.88
```

範例7-9 用全部特徵值來分析

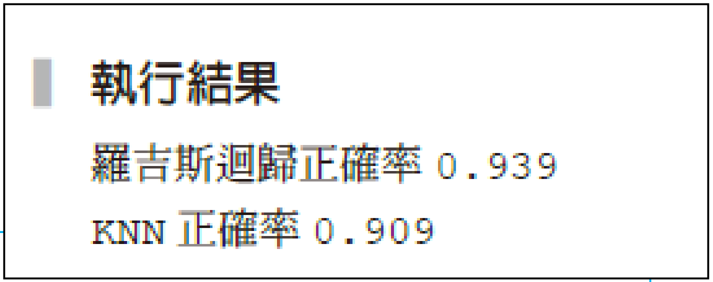
程式碼

```
iris = load_iris()
df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df['target'] = iris['target']
df = df.iloc[50:]
#資料分割
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.33,
                                                    random_state=42)
```

範例7-9 用全部特徵值來分析

承接上一頁

```
#羅吉斯迴歸
from sklearn.linear_model import LogisticRegression
model_pl_lr = make_pipeline(StandardScaler(), LogisticRegression
                                         (solver='liblinear'))
model_pl_lr.fit(X_train, y_train)
print(f'羅吉斯迴歸正確率{model_pl_lr.score(X_test,
y_test).round(3)}')
# KNN
model_pl_knn = make_pipeline(StandardScaler(),
KNeighborsClassifier())
model_pl_knn.fit(X_train, y_train)
print(f'KNN正確率{model_pl_knn.score(X_test,
y_test).round(3)}')
```



執行結果

```
羅吉斯迴歸正確率 0.939
KNN 正確率 0.909
```

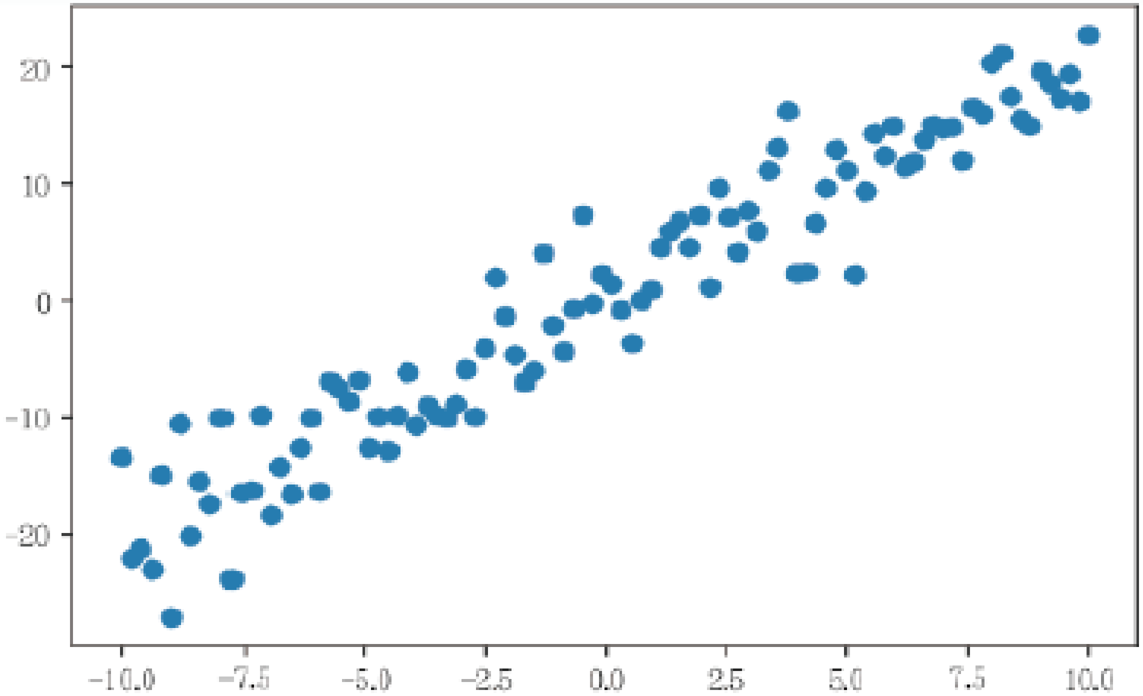
7-4　主成分分析

範例7-10　創建PCA用的資料

程式碼

```
np.random.seed(1)
x = np.linspace(-10, 10, 100)
y = 2 * x + 4*np.random.randn(100)
df_pca = pd.DataFrame(zip(x,y), columns=['x0','x1'])
plt.scatter(x, y);
```

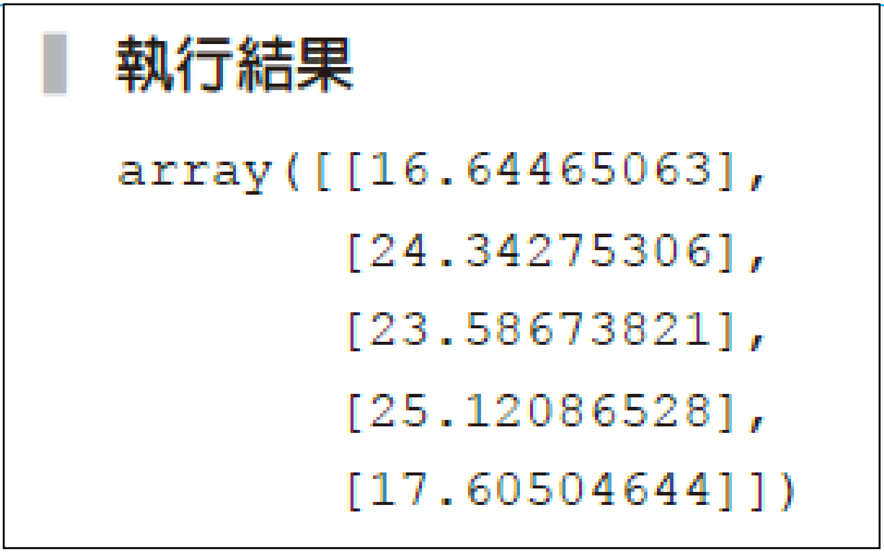
執行結果



範例7-11　如何選擇軸，能最大化的代表這份二維資料

程式碼

```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
X_pca = pca.fit_transform(df_pca)
X_pca[:5]
```



執行結果

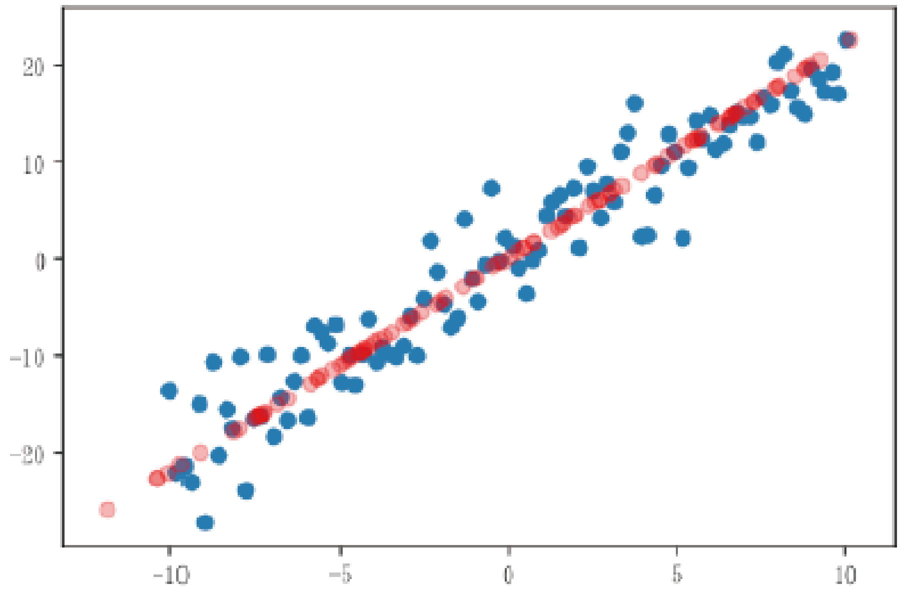
```
array([[16.64465063],
       [24.34275306],
       [23.58673821],
       [25.12086528],
       [17.60504644]])
```

範例7-12 範例7-11 得到的軸，繪製在原來的散布圖裡

程式碼

```
# 原來的資料
plt.scatter(x, y)
# 將X_pca 轉到原本的資料維度
X_new = pca.inverse_transform(X_pca)
plt.scatter(X_new[:,0], X_new[:,1], c='r', alpha=0.3);
```

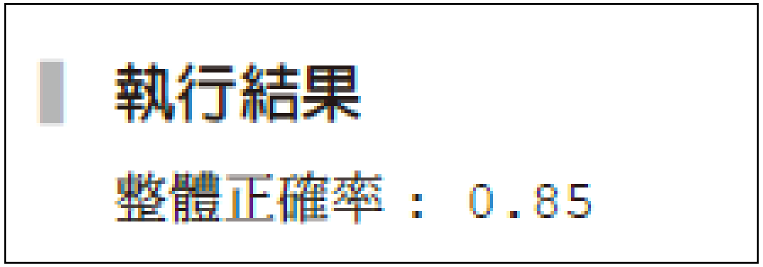
執行結果



範例7-14 將資料標準化，然後PCA(2)，再進行KNN預測

程式碼

```
model_pl = make_pipeline(StandardScaler(),
                         PCA(n_components=2),
                         KNeighborsClassifier())
model_pl.fit(X_train, y_train)
y_pred = model_pl.predict(X_test)
print('整體正確率:',accuracy_score(y_test, y_pred).round(2))
```



執行結果

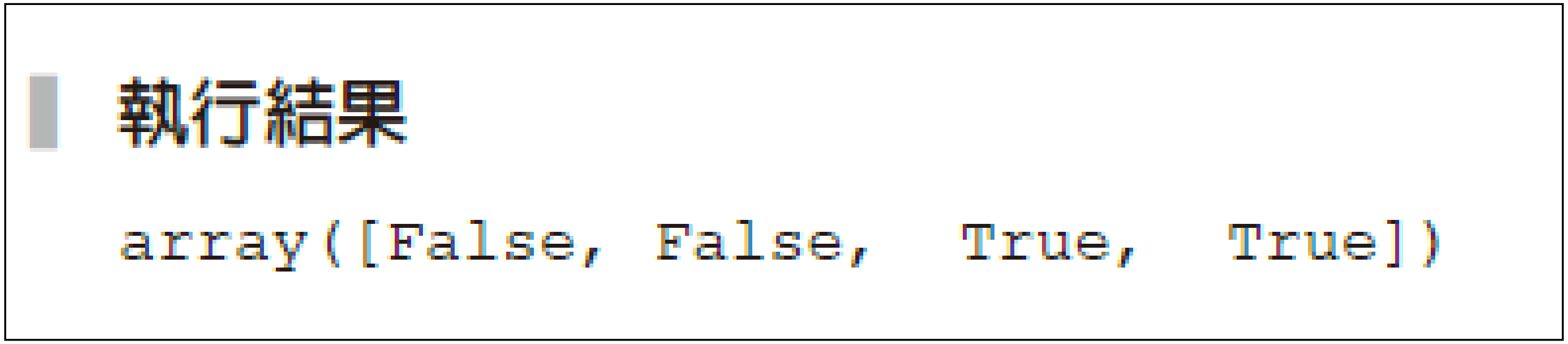
```
整體正確率： 0.85
```

7-5　SelectKBest

範例7-15　用SelectKBest選出最好的兩個特徵值，並指出是哪兩個欄位

程式碼

```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(f_classif, 2)
selector.fit(X_train, y_train)
selector.get_support()
```



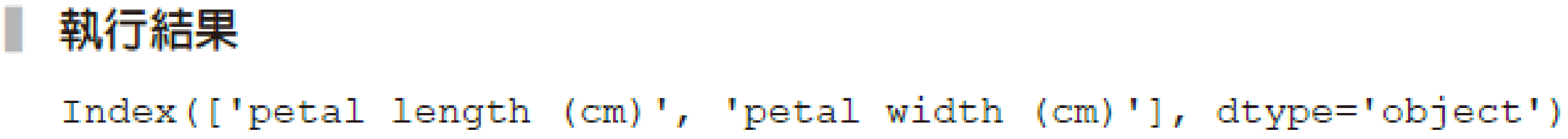
執行結果

```
array([False, False,  True,  True])
```

範例7-16 呈上例，將取出的欄位名稱列出

程式碼

```
X_test.columns[selector.get_support()]
```



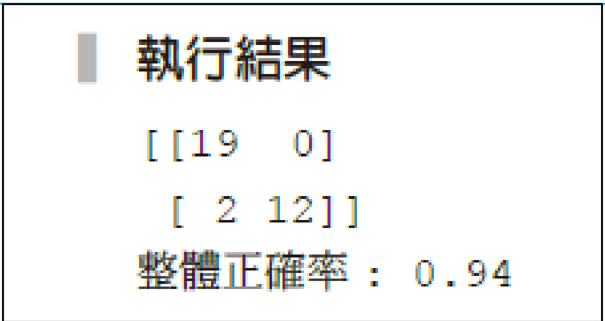
執行結果

```
Index(['petal length (cm)', 'petal width (cm)'], dtype='object')
```

範例7-17　創建管道器，連結標準化、SelectKBest和K最近鄰預測

程式碼

```
model_pl = make_pipeline(StandardScaler(),
                         SelectKBest(f_classif, 2),
                         KNeighborsClassifier())
model_pl.fit(X_train, y_train)
y_pred = model_pl.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print('整體正確率:',accuracy_score(y_test, y_pred).round(2))
```



執行結果

```
[[19  0]
 [ 2 12]]
整體正確率： 0.94
```