第11章　模型參數挑選和網格搜尋

第11章　模型參數挑選和網格搜尋

- 介紹網格搜尋
- 了解模型參數對預測結果的影響
- 如何設定網格搜尋參數
- 整理網格搜尋的過程分析
- 所有模型和其最佳參數一起比較
- 如何將不同模型放入網格搜尋裡
- 如何將資料預處理也放到網格搜尋裡

- 在上一章我們做了各種機器學習的大亂鬥，但事實上，我們都還沒有談到如何去修改機器的預設參數。因此，如果要比較各種機器學習的結果優劣，還必須考量不同參數的影響。
- 你可以預想，這個過程會變得相當複雜，因為針對每一個參數的組合，都必須進行所謂的交叉驗證。如此一來，程式會有多個迴圈，會耗費許多時間在執行，也容易出錯。
- 還好，sklearn提供了網格搜尋的方式，幫助我們解決這個問題。

範例11-1 定義邊界繪製函數

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')
```

範例11-1 定義邊界繪製函數

承接上一頁

```
def plot_decision_boundary(X_test, y_test, model, ax):
    points = 500
    x1_max, x2_max = X_test.max()
    x1_min, x2_min = X_test.min()
    X1, X2 = np.meshgrid(np.linspace(x1_min-0.1, x1_max+0.1
            points),np.linspace(x2_min-0.1, x2_max+0.1, points))
    x1_label, x2_label = X_test.columns
    X_test.plot(kind='scatter', x=x1_label, y=x2_label, c=y_test,
            cmap='coolwarm', colorbar=False, s=20, ax=ax)
    grids = np.array(list(zip(X1.ravel(), X2.ravel())))
    ax.contourf(X1, X2, model.predict(grids).reshape(X1.shape),
            alpha=0.3, cmap='coolwarm')
```

範例11-2 載入鳶尾花的資料和完成資料預處理

程式碼

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

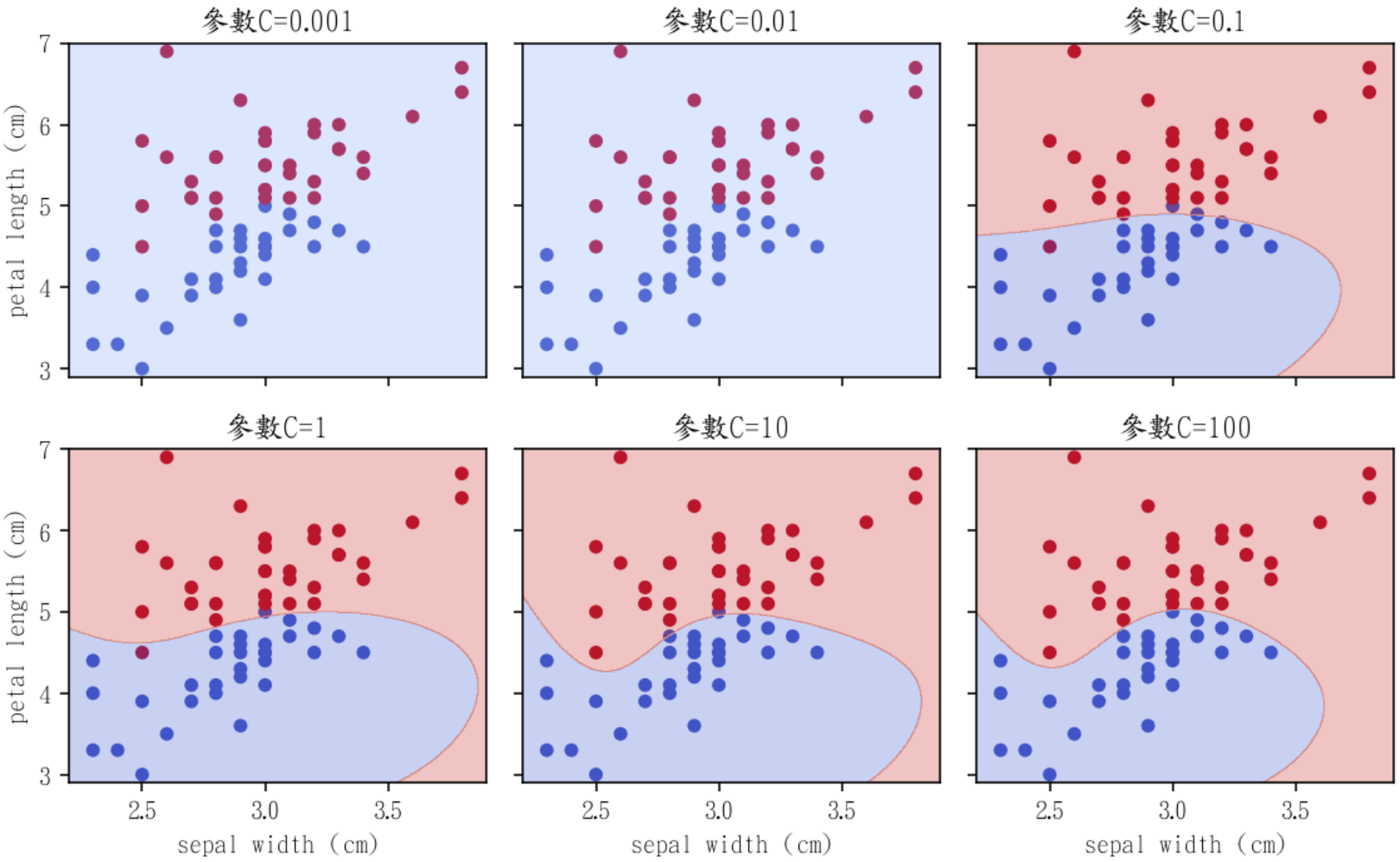
```
y_col = 'target'
X = df[X_cols]
y = df[y_col]
from sklearn.model_selection
import train_test_split
X_train, X_test, y_train, y_test
= train_test_split(X, y,

test_size=0.33,
random_state=42)
```

範例11-3　用預測邊界方式了解不同參數C對支持向量機的影響

程式碼

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
fig, axes = plt.subplots(2,3, figsize=(8,5), sharex=True, sharey=True)
scores = []
Cs = [0.001, 0.01, 0.1, 1, 10, 100]
```

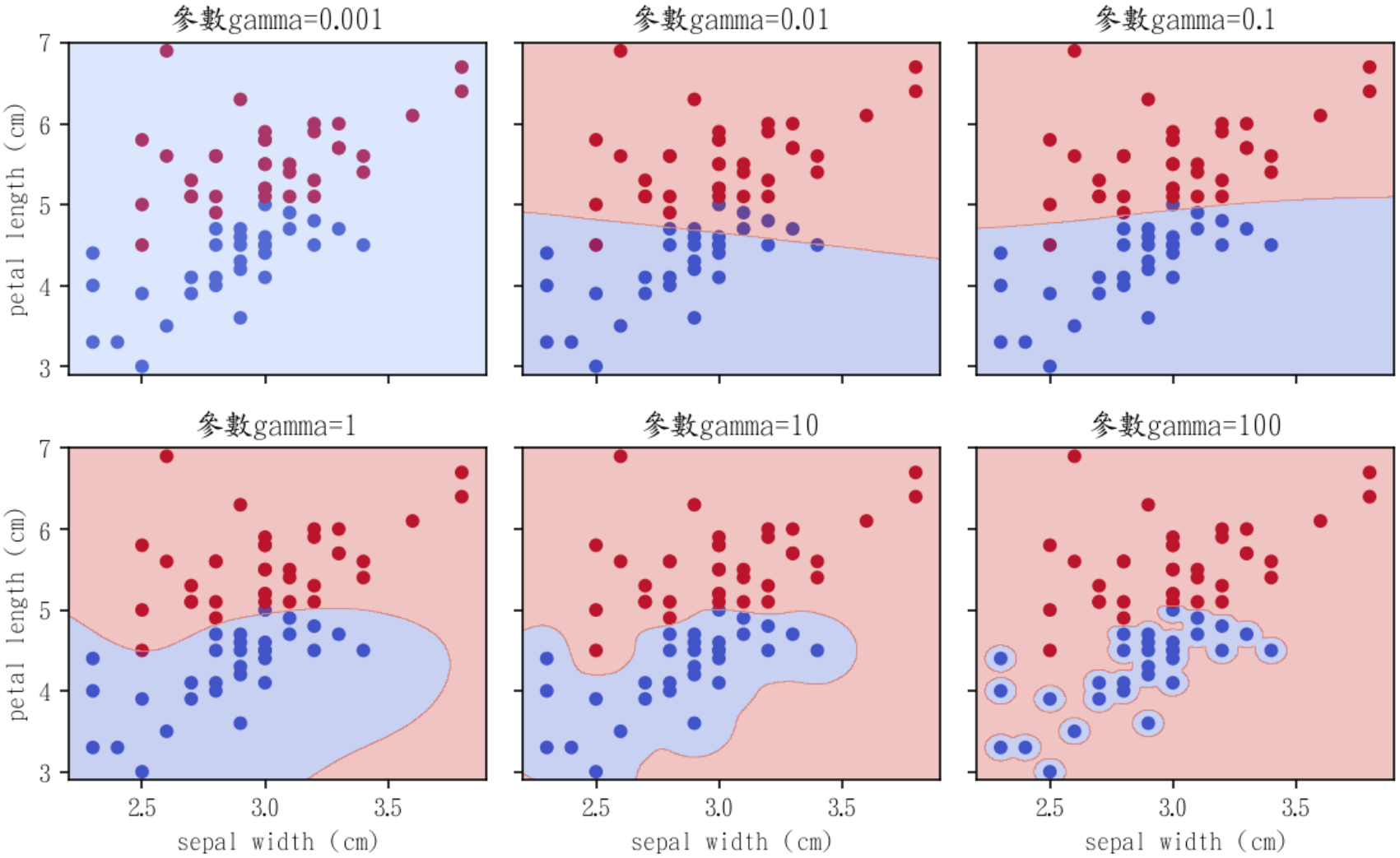
```
for ax, c in zip(axes.ravel(), Cs):
    model_pl = make_pipeline(StandardScaler(), SVC(C=c))
    model_pl.fit(X_train, y_train)
plot_decision_boundary(X_train, y_train, model_pl, ax)
    ax.set_title(f'參數C={c}')
plt.tight_layout()
```


參數C=0.001
參數C=0.01
參數C=0.1
參數C=1
參數C=10
參數C=100
petal length (cm)
sepal width (cm)
7
6
5
4
3
2.5
3.0
3.5

範例11-4 用邊界繪圖方式了解gamma參數對支持向量機的影響

程式碼

```
fig, axes = plt.subplots(2,3, figsize=(8,5), sharex=True,
sharey=True)
scores = []
gammas = [0.001, 0.01, 0.1, 1, 10, 100]
for ax, gamma in zip(axes.ravel(), gammas):
    model_pl = make_pipeline(StandardScaler(),
SVC(gamma=gamma))
    model_pl.fit(X_train, y_train)
    plot_decision_boundary(X_train, y_train, model_pl, ax)
    ax.set_title(f'參數gamma={gamma}')
plt.tight_layout()
```


參數gamma=0.001
參數gamma=0.01
參數gamma=0.1
參數gamma=1
參數gamma=10
參數gamma=100
petal length (cm)
sepal width (cm)
7
6
5
4
3
2.5
3.0
3.5

範例11-5 支持向量機能挑選的參數

程式碼

```
model_pl = make_pipeline(StandardScaler(), SVC())
keys = model_pl.get_params().keys()
keys
```

執行結果

```
dict_keys(['memory', 'steps', 'standardscaler', 'svc',
'standardscaler__copy', 'standardscaler__with_mean',
'standardscaler__with_std', 'svc__C', 'svc__cache_size', 'svc__
class_weight', 'svc__coef0', 'svc__decision_function_shape',
'svc__degree', 'svc__gamma', 'svc__kernel', 'svc__max_iter',
'svc__probability', 'svc__random_state', 'svc__shrinking',
'svc__tol', 'svc__verbose'])
```

範例11-6 支持向量機的參數組合偷懶版

程式碼

```
param_grid = {
    'svc_ _kernel': ['linear', 'rbf'],
    'svc_ _C':[0.1, 0.5, 0.8, 1, 5],
    'svc_ _gamma':np.arange(0.2, 1, 0.2)
}
print(param_grid)
```

執行結果

```
{'svc__kernel': ['linear', 'rbf'], 'svc__C': [0.1, 0.5, 0.8, 1,
5], 'svc__gamma': array([0.2, 0.4, 0.6, 0.8])}
```

- **初始化網格搜尋物件：網格搜尋物件初始完後，就像是一般的「預測器」一樣。只不過初始化時有幾個重要參數，**

1. 你要搜尋的預測器，
2. 網格的搜尋參數範圍，
3. 交叉驗證的折數。

範例11-8 進行網格搜尋

程式碼

```
model_pl = make_pipeline(StandardScaler(), SVC())
gs = GridSearchCV(model_pl, param_grid=param_grid,
                  cv=10, return_train_score=True)
# 進行網路搜尋學習
gs.fit(X_train, y_train)
```

執行結果

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('standardscaler', StandardScaler(copy=True,
            with_mean=True, with_std=True)), ('svc', SVC(C=1.0,
            cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'svc__kernel': ['linear'], 'svc__C': [0.1,
                   0.5, 0.8, 1, 5]}, {'svc__kernel': ['rbf'],
                   'svc__C': [0.1, 0.5, 0.8, 1, 5], 'svc__gamma':
                   array([0.2, 0.4, 0.6, 0.8])}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

範例11-9 取得網格搜尋的最佳參數

程式碼

```
gs.best_params_
```

執行結果

```
{'svc__C': 0.5, 'svc__kernel': 'linear'}
```

範例11-10 取得網格搜尋裡最佳預測的結果

程式碼

```
gs.best_score_
```

執行結果

```
0.9701492537313433
```

範例11-12 最佳參數運用在測試集的預測結果

程式碼

```
model_pl_svc = make_pipeline(StandardScaler(), SVC(C=0.5,
kernel='linear'))
model_pl_svc.fit(X_train, y_train)
model_pl_svc.score(X_test, y_test)
```

執行結果

```
0.8787878787878788
```

11-3 網格搜尋的過程分析

- 有時候我們會想更進一步了解究竟參數是如何影響預測結果好壞，這時就要在GridSearchCV裡，將參數return_train_score設為True。在cv_results_屬性裡存放著交叉驗證的所有分析結果，其中'mean_test_score'索引鍵提供了網格搜尋裡的交叉驗證的平均結果，而其對應的參數值則存放在'params'欄位。

11-3 網格搜尋的過程分析

- 因此我們先將gs.cv_results_轉換成DataFrame的格式，再進一步取出['params','mean_test_score']欄位，最後用sort_values依'mean_test_score'的值由大到小排序。我們列出前五筆最佳的參數值。觀察發現一個有趣的事實：原來同時有數筆資料在網格搜尋裡的交叉驗證結果是相同的，都是0.97。換言之，除了原本的最佳參數外，這些參數組合都是值得進一步探索的。

範例11-15 網格搜尋的過程分析

程式碼

```
# 第一行是為了能看欄位裡所有的值
pd.set_option('display.max_colwidth', -1)
df_cv =
pd.DataFrame(gs.cv_results_)[['params','mean_test_score']].\
sort_values(by = 'mean_test_score', ascending=False).head(12)
df_cv
```

	params	mean_test_score	accuracy
12	{'svc__C': 0.5, 'svc__gamma': 0.8, 'svc__kernel': 'rbf'}	0.970149	0.848485
16	{'svc__C': 0.8, 'svc__gamma': 0.8, 'svc__kernel': 'rbf'}	0.970149	0.848485
2	{'svc__C': 0.8, 'svc__kernel': 'linear'}	0.970149	0.848485
3	{'svc__C': 1, 'svc__kernel': 'linear'}	0.970149	0.848485
22	{'svc__C': 5, 'svc__gamma': 0.4, 'svc__kernel': 'rbf'}	0.970149	0.818182
21	{'svc__C': 5, 'svc__gamma': 0.2, 'svc__kernel': 'rbf'}	0.970149	0.848485
20	{'svc__C': 1, 'svc__gamma': 0.8, 'svc__kernel': 'rbf'}	0.970149	0.848485
19	{'svc__C': 1, 'svc__gamma': 0.6000000000000001, 'svc__kernel': 'rbf'}	0.970149	0.848485
18	{'svc__C': 1, 'svc__gamma': 0.4, 'svc__kernel': 'rbf'}	0.970149	0.848485
10	{'svc__C': 0.5, 'svc__gamma': 0.4, 'svc__kernel': 'rbf'}	0.970149	0.848485
11	{'svc__C': 0.5, 'svc__gamma': 0.6000000000000001, 'svc__kernel': 'rbf'}	0.970149	0.848485
1	{'svc__C': 0.5, 'svc__kernel': 'linear'}	0.970149	0.878788