紀老師程式教學網

https://www.facebook.com/teacherchi

機器學習

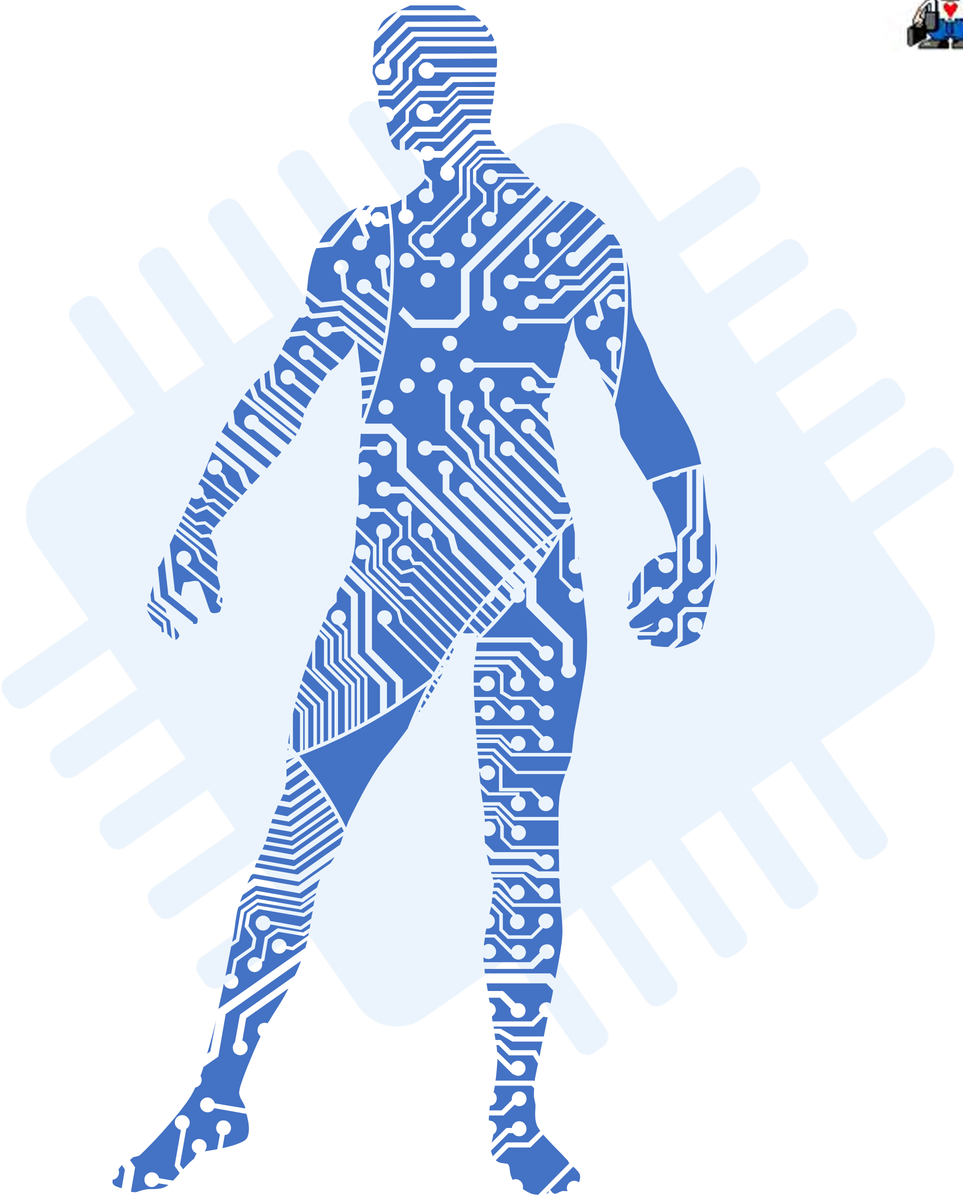
第 11 章 支援向量機

(Support Vector Machine)

講師：紀俊男

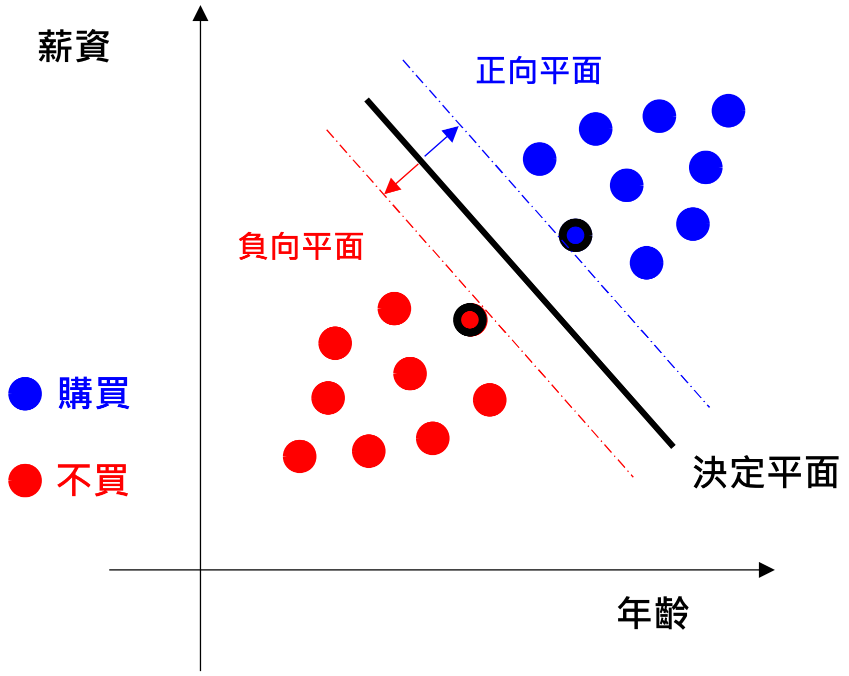
本章大綱

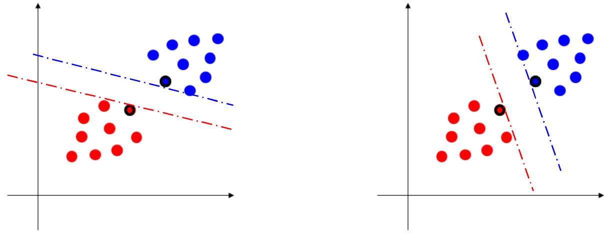
- 理論說明
- 資料前處理
- 實作支援向量機
- 參數優化的方法
- 本章總結



AI 理論說明

什麼是「支援向量機」？

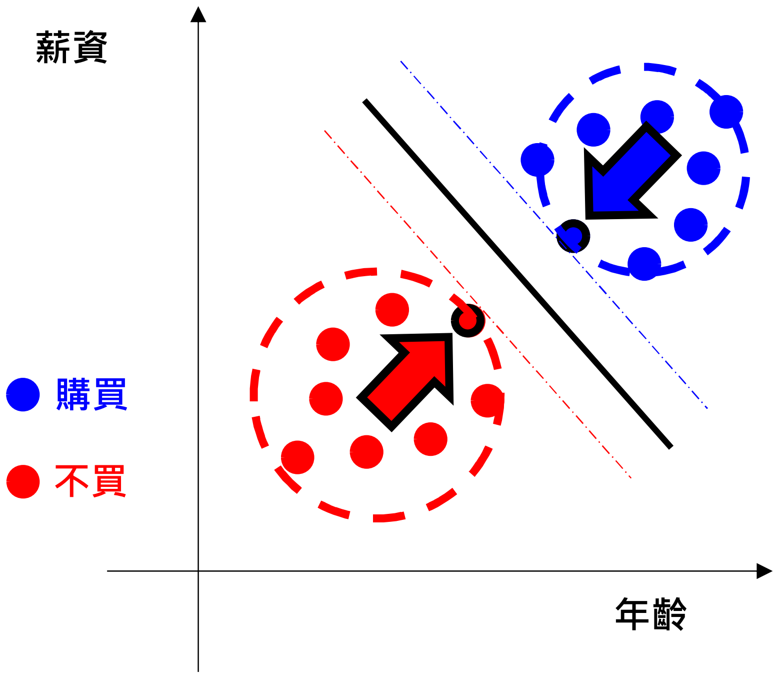


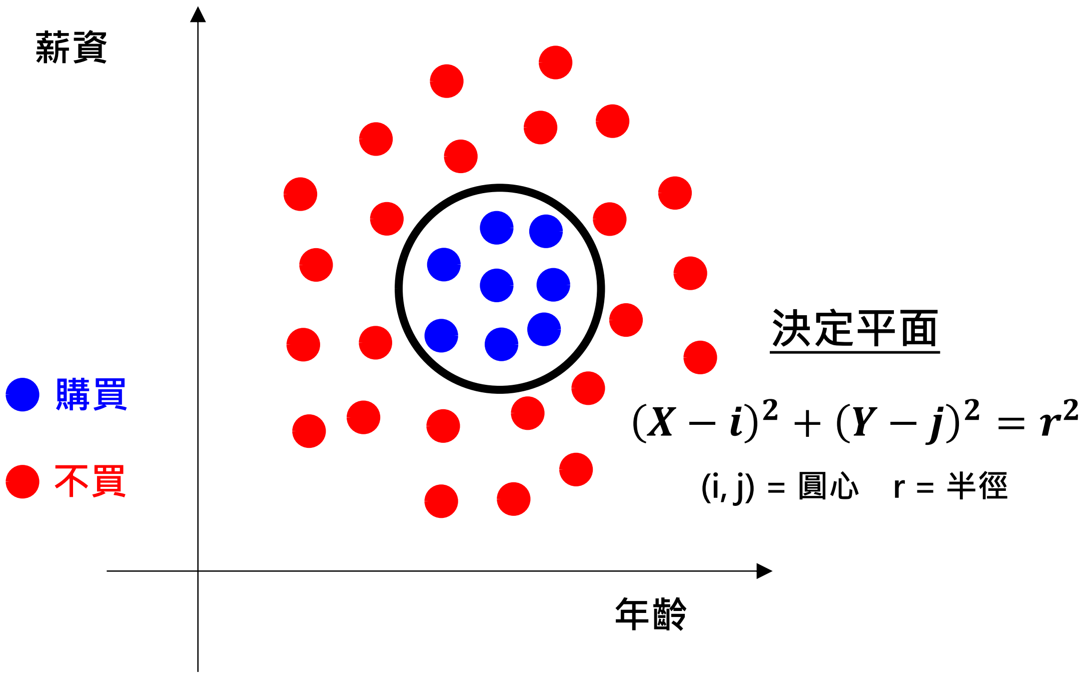


1. 找出「兩軍對峙」中，站在最前方的將士。
   （這兩個點 = 支援向量 Support Vector）
2. 畫出兩條「對峙線」，使得距離為最寬。
3. 這兩條對峙線中央，就是「決定平面」。
   （Decision Hyperplane，或「分割平面」）
4. 決定平面正向法向量所指的對峙線，叫做「正向平面」（Positive Hyperplane）。
5. 決定平面負向法向量所指的對峙線，叫做「負向平面」（Negative Hyperplane）。
6. 之後的任何分類，就靠「決定平面」來分。

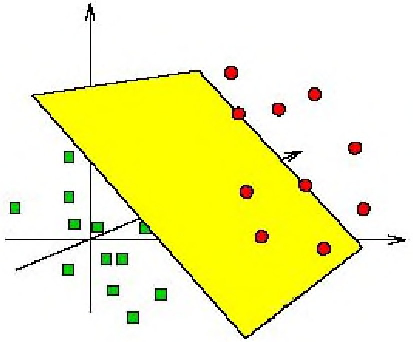
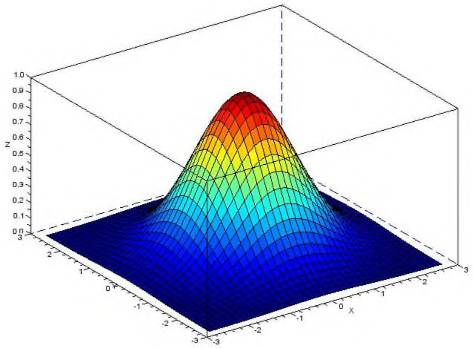
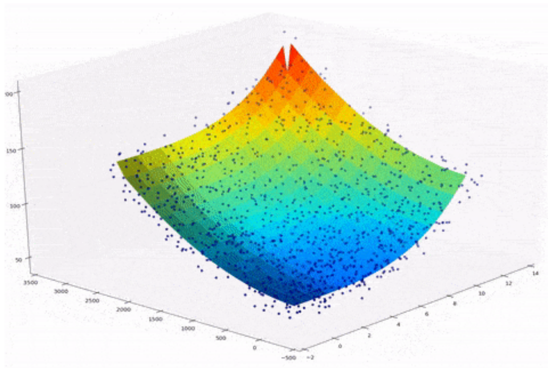
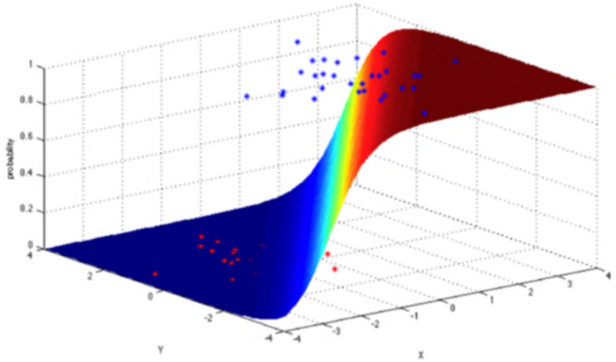
支援向量機特別之處

- 以「**離群值**」（支援向量）做為分類標準
  - 「離群值」=「長得很像，但不同類」
  - ∴ SVM 分類效能**特別好**！
- **內核函數（Kernel Function）**可以更換
  - 亦即：分類平面**不一定**要是直線，甚至**可自訂**！
  - 換成非線性內核的 SVM = Kernel SVM
  - Kernel SVM 有「**萬能**分類器」之稱





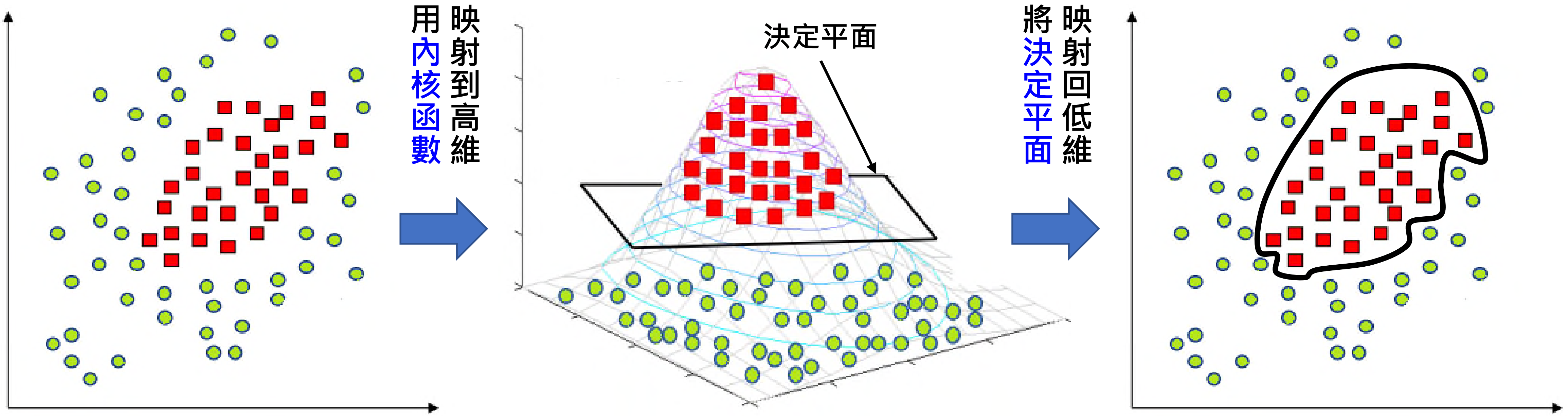
支援向量機常見的「內核函數」

Linear	Gaussian Radial Basis Function	Polynomial	Sigmoid
（'linear'）	（'rbf'）	（'poly'）	（'sigmoid'）
			
$k(X,Y) = X^T \cdot Y + c$	$k(X,Y) = e^{-\frac{\|X-Y\|^2}{2\sigma^2}}$	$k(X,Y) = \left(X^T \cdot Y + c\right)^d$	$k(X,Y) = tanh\left(\alpha X^T \cdot Y + c\right)$

完整的「內核函數（Kernel Function）」列表 & 解說：https://bit.ly/3CORtrj

「內核函數」如何幫助分類（Kernel Trick）

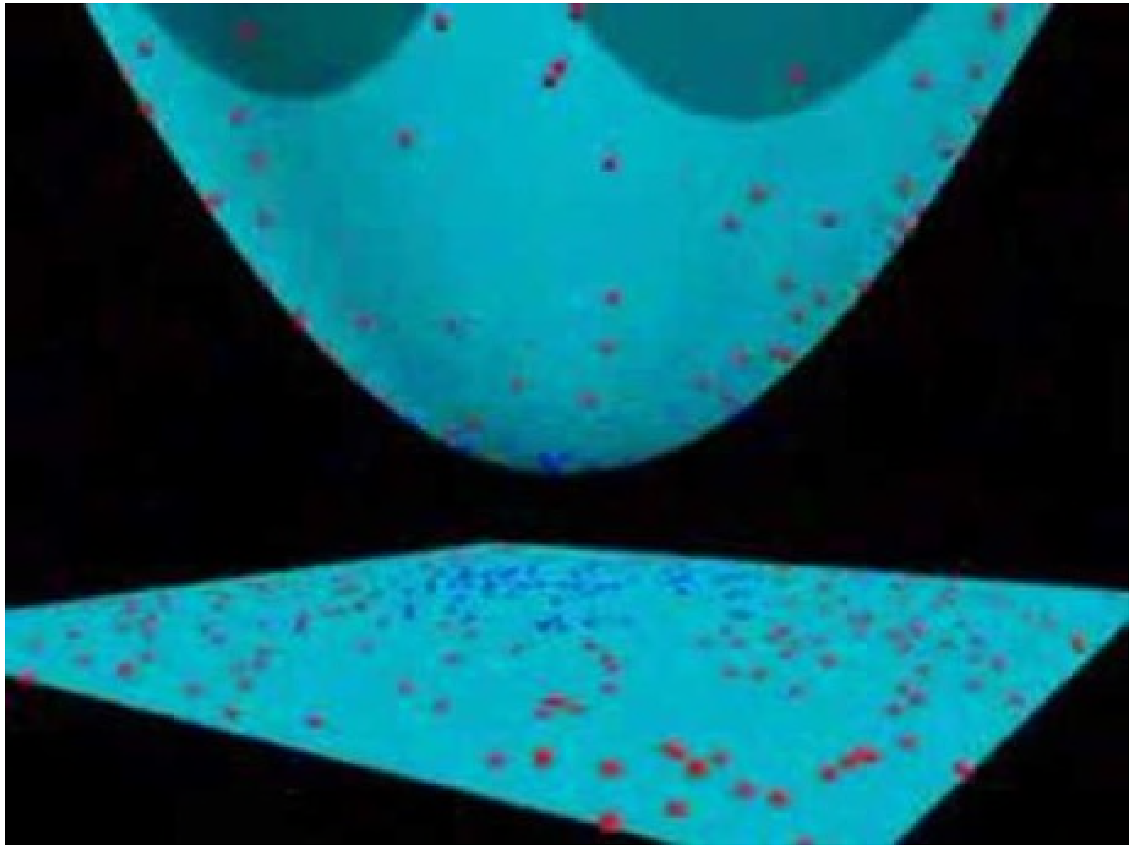
- 以 Gaussian RBF 為例：



SVM → 計算量超級龐大！但可解決許多「線性不可分」問題！

「內核函數」如何幫助分類（Kernel Trick）

- 以多項式內核函數（Polynomial）為例：



SVM with polynomial kernel visualization

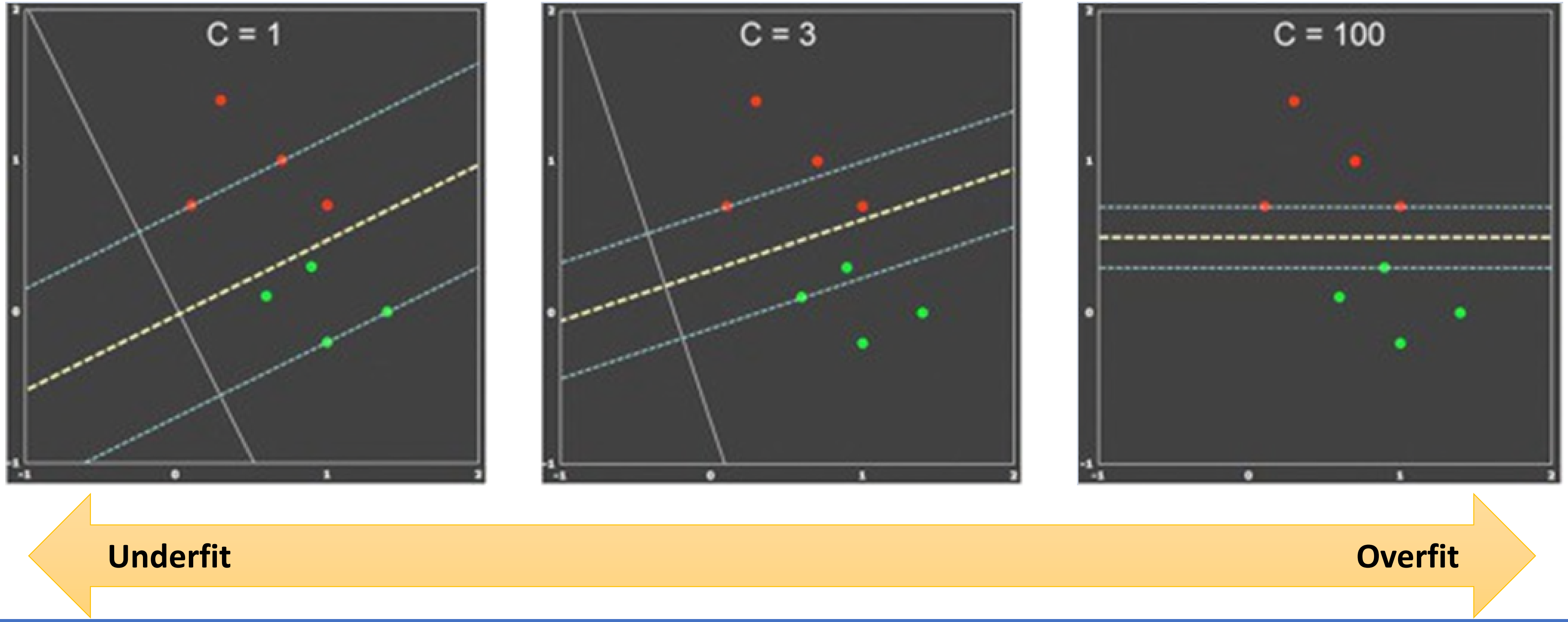
https://youtu.be/3liCbRZPrZA

支援向量機的重要參數

- C：懲罰參數（對所有內核函數皆有效）

C 小 → 懲罰小 → 允許雜點

C 大 → 懲罰大 → 不允許雜點

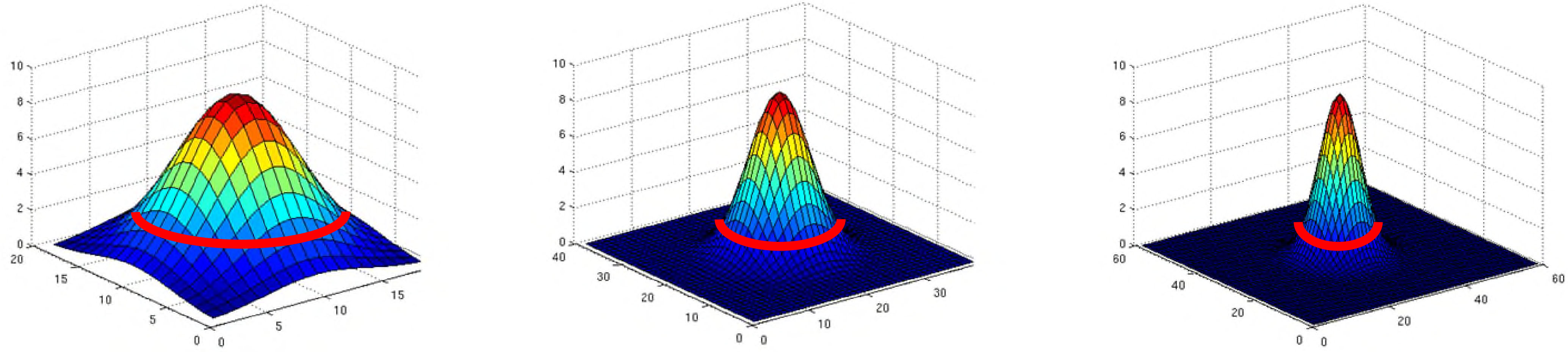


支援向量機的重要參數

- γ（Gamma）：離散參數（對 rbf, poly, sigmoid 有效）

$$\gamma = \frac{1}{2\sigma^2}$$

σ = 標準差



σ ↑ γ ↓ 涵蓋範圍廣，**Recall** 高

σ ↓ γ ↑ 涵蓋範圍窄，**Precision** 高

支援向量機的重要參數

from sklearn.svm import SVC

SVC (C=1.0, kernel= “rbf” , degree=3, gamma= “scale” , coef0=0.0)

方程式	內核函數名稱	懲罰參數 C	離散參數 γ	次方項 degree	常數項 coef0
$X^T \cdot Y + c$	“linear”	✓			✓
$e^{-\gamma\|X-Y\|^2}$	“rbf”	✓	✓		
$(X^T \cdot Y + c)^d$	“poly”	✓	✓	✓	✓
$tanh(\alpha X^T \cdot Y + c)$	“sigmoid”	✓	✓		✓

附註： **gamma=“scale” → gamma==** $\frac{1}{\#features * X.var()} = \frac{1}{自變數個數 \times 自變數變異數}$

資料前處理

載入「鳶尾花 (Iris) 」資料集

```python
from sklearn.datasets import load_iris

dataset = load_iris()
```

dataset - Dictionary (6 elements)

Key	Type	Size	Value
DESCR	str	1	.. _iris_dataset:
data	float64	(150, 4)	[[5.1 3.5 1.4 0.2] [4.9 3. 1.4 0.2]
feature_names	list	4	['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal ...
filename	str	1	D:\bin\anaconda3\lib\site-packages\sklearn\datasets\data\iris.csv
target	int32	(150,)	[0 0 0 ... 2 2 2]
target_names	str320	(3,)	ndarray object of numpy module

Close

- **DESCR**
  - 本資料集的文字敘述。
- **data**
  - NDArray，所有自變數 X。
- **feature_names**
  - 串列 (list) ，自變數欄位名稱。
- **target**
  - NDArray，應變數 Y。
- **target_name**
  - NDArray，應變數 Y 數字所對應的意義。
  - 如： 0=setosa, 1=versicolor... 等。

載入「自變數」與「應變數」

載入 NDArray，
並包裹成 DataFrame

```python
import pandas as pd

X = pd.DataFrame(dataset.data, columns=dataset.feature_names)
Y = pd.DataFrame(dataset.target, columns=["Iris_Type"])
Y_name = dataset.target_names.tolist()
```

↑ 雖然有載入，但我的程式碼沒用到

自變數 X

Index	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5	3.6	1.4	0.2

應變數 Y

Index	Iris_Type
0	0
1	0
2	0
3	0
4	0

Y_name

Index	Type	Size	Value
0	str	1	setosa
1	str	1	versicolor
2	str	1	virginica

其它資料前處理程式碼

```python
from HappyML.preprocessor import KBestSelector
import HappyML.preprocessor as pp

# Feature Selection
selector = KBestSelector(best_k='auto')
X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)

# Split Training / TEsting Set
X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)

# Feature Scaling
X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

特徵選擇

切分訓練集、
測試集

特徵縮放

執行結果

X_train

Index	sepal length (cm)	petal length (cm)	petal width (cm)
96	-0.21	0.20	0.08
102	1.43	1.15	1.14
75	0.84	0.31	0.21
115	0.61	0.81	1.41
10	-0.57	-1.32	-1.38

Y_train

Index	Iris_Type
96	1
102	2
75	1
115	2
10	0

X_test

Index	sepal length (cm)	petal length (cm)	petal width (cm)
63	0.26	0.48	0.21
97	0.37	0.25	0.08

Y_test

Index	Iris_Type
63	1
97	1

隨堂練習：資料前處理

- 請輸入、並執行下列程式碼，以執行「資料前處理」：

```python
from sklearn.datasets import load_iris

# Load Data
dataset = load_iris()

# X, Y
import pandas as pd
X = pd.DataFrame(dataset.data, columns=dataset.feature_names)
Y = pd.DataFrame(dataset.target, columns=["Iris_Type"])
Y_name = dataset.target_names.tolist()

# Load HappyML
from HappyML.preprocessor import KBestSelector
import HappyML.preprocessor as pp

# Feature Selection
selector = KBestSelector(best_k='auto')
X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)

# Split Training / TEsting Set
X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)

# Feature Scaling
X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

實作支援向量機

使用「標準函式庫」實作

程式碼

完全跟預設值一模一樣

產生物件本身→

訓練→

預測→

```
from sklearn.svm import SVC
import time

classifier = SVC(C=1.0, kernel="rbf", gamma="scale", random_state=int(time.time()))
classifier.fit(X_train, Y_train.values.ravel())
Y_pred = classifier.predict(X_test)
```

執行結果

Y_test

Index	Iris_Type
40	0
15	0
94	1
88	1
29	0
81	1
136	2
125	2
120	2
80	1

Y_pred

	0
0	0
1	0
2	1
3	1
4	0
5	1
6	2
7	2
8	2
9	1

隨堂練習：使用「標準函式庫」實作

- 請撰寫下列程式碼，並執行之：

```python
from sklearn.svm import SVC
import time

classifier = SVC(C=1.0, kernel="rbf", gamma="scale", random_state=int(time.time()))
classifier.fit(X_train, Y_train.values.ravel())
Y_pred = classifier.predict(X_test)
```

- 執行完畢後，請比較 Y_test（真實值）與 Y_pred（預測值）的差異。

使用「快樂版函式庫」實作

- 程式碼解說（1）：

/HappyML/classification.py

引入必要套件

類別的成員變數

建構函數

懲罰參數　內核函數　次方項　離散參數　常數項　亂數種子

```python
from sklearn.svm import SVC
import time

class SVM:
    __classifier = None

    __penalty_C = None
    __kernel = None
    __degree = None
    __gamma = None
    __coef0 = None
    __y_columns = None

    def __init__(self, C=1.0, kernel="rbf", degree=3, gamma="scale", coef0=0.0, random_state=int(time.time())):
        self.__penalty_C = C
        self.__kernel = kernel
        self.__degree = degree
        self.__gamma = gamma
        self.__coef0 = coef0

        self.__classifier = SVC(C=self.__penalty_C, kernel=self.__kernel,
                                degree=self.__degree, gamma=self.__gamma,
                                coef0=self.__coef0, random_state=random_state)
```

使用「快樂版函式庫」實作

- 程式碼解說（2）：

/HappyML/classification.py

classifier 的 getter & setter

```python
    @property
    def classifier(self):
        return self.__classifier

    @classifier.setter
    def classifier(self, classifier):
        self.__classifier = classifier
```

訓練

↓ Y_train 變成 NDArray 後「踩平」成一維

```python
    def fit(self, x_train, y_train):
        self.classifier.fit(x_train, y_train.values.ravel())
        self.__y_columns = y_train.columns

        return self
```

← 保存 Y_train 的欄位名稱

預測

↓ 把預測出來的 Y_pred 重新包裝回 DataFrame 再傳回

```python
    def predict(self, x_test):
        return pd.DataFrame(self.classifier.predict(x_test), index=x_test.index, columns=self.__y_columns)
```

使用「快樂版函式庫」實作

- 呼叫範例

```python
from HappyML.classification import SVM

classifier = SVM()
Y_pred = classifier.fit(X_train, Y_train).predict(X_test)
```

- 執行結果

Y_test

Index	Iris_Type
40	0
15	0
94	1
88	1
29	0
81	1
136	2
125	2
120	2
80	1

Y_pred

Index	Iris_Type
40	0
15	0
94	1
88	1
29	0
81	1
136	2
125	2
120	2
80	1

隨堂練習：使用「快樂版函式庫」實作

- 請撰寫下列程式碼，並執行之：

```
from HappyML.classification import SVM

classifier = SVM()
Y_pred = classifier.fit(X_train, Y_train).predict(X_test)
```

- 執行完畢後，請比較 **Y_test（真實值）** 與 **Y_pred（預測值）** 的差異。

計算效能 & 結果視覺化

- **程式碼解說**

K 次交叉驗證

```
# Performance
K = 10
kfp = KFoldClassificationPerformance(x_ary=X, y_ary=Y, classifier=classifier.classifier, k_fold=K)

print("----- SVM Classification -----")
print("{} Folds Mean Accuracy: {}".format(K, kfp.accuracy()))
print("{} Folds Mean Recall: {}".format(K, kfp.recall()))
print("{} Folds Mean Precision: {}".format(K, kfp.precision()))
print("{} Folds Mean F1-Score: {}".format(K, kfp.f_score()))
```

注意！

記得將自變數壓制到兩個維度

```
selector = KBestSelector(best_k=2)
```

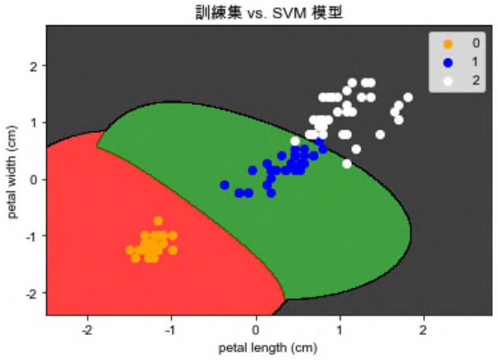
將結果視覺化

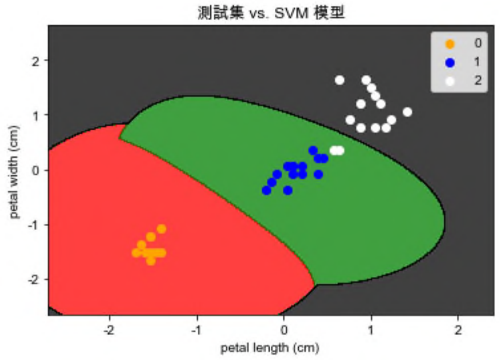
```
# Visualization
import HappyML.model_drawer as md

md.classify_result(x=X_train, y=Y_train, classifier=classifier.classifier,
    fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
    title="訓練集 vs. SVM 模型", font="DFKai-sb")
md.classify_result(x=X_test, y=Y_test, classifier=classifier.classifier,
    fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
    title="測試集 vs. SVM 模型", font="DFKai-sb")
```

計算效能 & 結果視覺化

- 執行結果

```
----- SVM Classification -----
10 Folds Mean Accuracy: 0.96
10 Folds Mean Recall: 0.96
10 Folds Mean Precision: 0.9644444444444445
10 Folds Mean F1-Score: 0.9597306397306398
```





隨堂練習：計算效能 & 結果視覺化

- 請修改卡方降維程式碼，將特徵值強制壓縮至二維：

```python
selector = KBestSelector(best_k=2)
```

- 請撰寫下列程式碼，並將程式從頭到尾執行看看：

```python
# Performance
K = 10
kfp = KFoldClassificationPerformance(x_ary=X, y_ary=Y, classifier=classifier.classifier, k_fold=K)

print("----- SVM Classification -----")
print("{} Folds Mean Accuracy: {}".format(K, kfp.accuracy()))
print("{} Folds Mean Recall: {}".format(K, kfp.recall()))
print("{} Folds Mean Precision: {}".format(K, kfp.precision()))
print("{} Folds Mean F1-Score: {}".format(K, kfp.f_score()))

# Visualization
import HappyML.model_drawer as md

md.classify_result(x=X_train, y=Y_train, classifier=classifier.classifier,
    fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
    title="訓練集 vs. SVM 模型", font="DFKai-sb")
md.classify_result(x=X_test, y=Y_test, classifier=classifier.classifier,
    fg_color=("orange", "blue", "white"), bg_color=("red", "green", "black"),
    title="測試集 vs. SVM 模型", font="DFKai-sb")
```

還可以更好嗎？

- 可以！只要找到適合此資料集的「超參數（Hyper Parameters）」即可！

超參數 → 機器無法決定、需靠人類幫忙決定的參數

SVC (C=1.0, kernel= "rbf" , degree=3, gamma= "scale" , coef0=0.0)

方程式	內核函數名稱	懲罰參數 C	離散參數 γ	次方項 degree	常數項 coef0
$X^T \cdot Y + c$	"linear"	✓			✓
$e^{-\gamma\|X-Y\|^2}$	"rbf"	✓	✓		
$\left(X^T \cdot Y + c\right)^d$	"poly"	✓	✓	✓	✓
$tanh\left(\alpha X^T \cdot Y + c\right)$	"sigmoid"	✓	✓		✓

參數優化的方法

網格搜尋法（Grid Search）

簡介：「網格搜尋法」

- 窮舉所有可能，讓電腦去決定哪種組合效能比較好

"linear" 可能組合	C：[1, 10, 100, 1000]
	coef0：[0, 10, 100, 1000]

"poly" 可能組合	C：[1, 10, 100, 1000]
	gamma：[0.1, 0.01, 0.001, 0.0001]
	degree：[2, 3, 4, 5]
	coef0：[0, 10, 100, 1000]

"rbf" 可能組合	C：[1, 10, 100, 1000]
	gamma：[0.1, 0.01, 0.001, 0.0001]

"sigmoid" 可能組合	C：[1, 10, 100, 1000]
	gamma：[0.1, 0.01, 0.001, 0.0001]
	coef0：[0, 10, 100, 1000]

以「標準函式庫」實作

```
from HappyML.classification import SVM
classifier = SVM()

# GridSearch without HappyML
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(estimator=classifier.classifier, param_grid=params_list, verbose=10, cv=10)
grid_search.fit(X, Y.values.ravel())

print("Best Parameters: {}  Best Score: {}".format(grid_search.best_params_, grid_search.best_score_))
classifier.classifier = grid_search.best_estimator_
```

產生 SVM 分類器（使用預設參數）

← 引入必要套件

是否逐步顯示
（0:不顯示～100:顯示很多訊息）

↓ K-Fold

開始尋找（擬合）→

欲評估「超參數」的模型

「超參數」組合（後述）

1

2

3

1. .best_params_：最佳「超參數」組合（如：kernel=“rbf”, C=1000, gamma=0.001）。
2. .best_score_：用最佳超參數，做出來的效能（預設：Accuracy）。
3. .best_estimator_：已經組合了最佳超參數的模型（可直接拿去 fit & predict）。

以「標準函式庫」實作

- 如何製作「**超參數組合**」（ param_grid=params_list ）

params_list = **超參數組合 = 串列 包 字典 包 串列**

```
                    1                      4                              4
[ { "kernel" : [ "linear" ],  "C" : [1, 10, 100, 1000],  "coef0" : [0, 10, 100, 1000]}
                    1                      4                              4
  { "kernel" : [ "rbf" ],  "C" : [1, 10, 100, 1000],  "gamma" : [0.001, 0.01, 0.1, 1]}
                    1                      4                              4                              4
  { "kernel" : [ "sigmoid" ],  "C" : [1, 10, 100, 1000],  "gamma" : [0.001, 0.01, 0.1, 1],  "coef0" : [0, 10, 100, 1000]} ]
```

(1×4 ×4) ×10-Folds + (1×4 ×4) ×10-Folds + (1×4 ×4 × 4) ×10-Folds = 960 次

以「標準函式庫」實作

- 「超參數」組合程式碼

產生 $10^3 \sim 10^6$ 共 4 個數字

各參數的可能數值

製造出各種參數組合

合成最後的超參數串列

```python
import numpy as np

# Ranges of Hyper Parameters
C_range = np.logspace(3, 6, 4) # Create [1000, 10000, 100000, 1000000]
Gamma_range = np.logspace(-4, -1, 4) # Create [0.0001, 0.001, 0.01, 0.1]
Coef0_range = np.logspace(0, 3, 4) # Create [1, 10, 100, 1000]

# Combination of Hyper Parameters
Linear_dict = dict(kernel=["linear"], C=C_range, coef0=Coef0_range)
RBF_dict = dict(kernel=["rbf"], C=C_range, gamma=Gamma_range)
Sigmoid_dict = dict(kernel=["sigmoid"], C=C_range, gamma=Gamma_range, coef0=Coef0_range)

# Collect all Combinations for Grid Search
params_list = [Linear_dict, RBF_dict, Sigmoid_dict]
```

以「標準函式庫」實作

完整程式碼

參數準備

```python
# Parameters --------------------
import numpy as np

# Ranges of Hyper Parameters
C_range = np.logspace(3, 6, 4) # Create [1000, 10000, 100000, 1000000]
Gamma_range = np.logspace(-4, -1, 4) # Create [0.0001, 0.001, 0.01, 0.1]
Coef0_range = np.logspace(0, 3, 4) # Create [1, 10, 100, 1000]

# Combination of Hyper Parameters
Linear_dict = dict(kernel=["linear"], C=C_range, coef0=Coef0_range)
RBF_dict = dict(kernel=["rbf"], C=C_range, gamma=Gamma_range)
Sigmoid_dict = dict(kernel=["sigmoid"], C=C_range, gamma=Gamma_range, coef0=Coef0_range)

# Collect all Combinations for Grid Search
params_list = [Linear_dict, RBF_dict, Sigmoid_dict]
```

網格搜尋

```python
# GridSearch --------------------
from HappyML.classification import SVM
classifier = SVM()

# GridSearch without HappyML
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(estimator=classifier.classifier, param_grid=params_list, verbose=10, cv=10)
grid_search.fit(X, Y.values.ravel())

print("Best Parameters: {}  Best Score: {}".format(grid_search.best_params_, grid_search.best_score_))
classifier.classifier = grid_search.best_estimator_
```

逐步顯示 & 10-Folds 驗證

以「標準函式庫」實作

- 執行結果

```
……（前略）……

[CV] C=1000000.0, coef0=1000.0, gamma=0.1, kernel=sigmoid ............
[CV]  C=1000000.0, coef0=1000.0, gamma=0.1, kernel=sigmoid, score=0.3333333333333333, total=   0.0s
[CV] C=1000000.0, coef0=1000.0, gamma=0.1, kernel=sigmoid ............
[CV]  C=1000000.0, coef0=1000.0, gamma=0.1, kernel=sigmoid, score=0.3333333333333333, total=   0.0s
[CV] C=1000000.0, coef0=1000.0, gamma=0.1, kernel=sigmoid ............
[CV]  C=1000000.0, coef0=1000.0, gamma=0.1, kernel=sigmoid, score=0.3333333333333333, total=   0.0s

Best Parameters: {'C': 100000.0, 'gamma': 0.1, 'kernel': 'rbf'}  Best Score: 0.9666666666666667
```

最佳超參數求出來了！

隨堂練習：以「標準函式庫」實作

- 請撰寫、並執行下列程式碼。以取得最佳的「超參數」：

```
# Parameters --------------------
import numpy as np

# Ranges of Hyper Parameters
C_range = np.logspace(3, 6, 4) # Create [1000, 10000, 100000, 1000000]
Gamma_range = np.logspace(-4, -1, 4) # Create [0.0001, 0.001, 0.01, 0.1]
Coef0_range = np.logspace(0, 3, 4) # Create [1, 10, 100, 1000]

# Combination of Hyper Parameters
Linear_dict = dict(kernel=["linear"], C=C_range, coef0=Coef0_range)
RBF_dict = dict(kernel=["rbf"], C=C_range, gamma=Gamma_range)
Sigmoid_dict = dict(kernel=["sigmoid"], C=C_range, gamma=Gamma_range, coef0=Coef0_range)

# Collect all Combinations for Grid Search
params_list = [Linear_dict, RBF_dict, Sigmoid_dict]

# GridSearch --------------------
from HappyML.classification import SVM
classifier = SVM()

# GridSearch without HappyML
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(estimator=classifier.classifier, param_grid=params_list, verbose=10, cv=10)
grid_search.fit(X, Y.values.ravel())

print("Best Parameters: {}  Best Score: {}".format(grid_search.best_params_, grid_search.best_score_))
classifier.classifier = grid_search.best_estimator_
```

以「快樂版函式庫」實作

- 程式碼解說（1）：

類別的成員變數

建構函數

verbose 的
getter & setter

```python
from sklearn.model_selection import GridSearchCV

class GridSearch:
    __validator = None
    __estimator = None
    __parameters = None
    __scorer = None
    __k_fold = None
    __best_score = None
    __best_parameters = None
    __best_estimator = None
    __verbose = None

    def __init__(self, estimator, parameters, scorer=None, k_fold=10, verbose=False):
        self.__estimator = estimator
        self.__parameters = parameters
        self.__scorer = scorer
        self.__k_fold = k_fold

        self.verbose = verbose
        self.__validator = GridSearchCV(estimator=self.__estimator, param_grid=self.__parameters,
                                        scoring=self.__scorer, cv=self.__k_fold, verbose=self.verbose)

    @property
    def verbose(self):
        return self.__verbose

    @verbose.setter
    def verbose(self, verbose):
        if verbose:
            self.__verbose = 10
        else:
            self.__verbose = 0
```

以「快樂版函式庫」實作

- 程式碼解說（2）：

validator 的 getter 函數

取得最佳分數

取得最佳參數

取得最佳分類器

擬合、尋找最佳參數

```python
    @property
    def validator(self):
        return self.__validator

    @property
    def best_score(self):
        return self.__best_score

    @property
    def best_parameters(self):
        return self.__best_parameters

    @property
    def best_estimator(self):
        return self.__best_estimator

    def fit(self, x_ary, y_ary):
        self.validator.fit(x_ary, y_ary.values.ravel())

        self.__best_parameters = self.validator.best_params_
        self.__best_score = self.validator.best_score_
        self.__best_estimator = self.validator.best_estimator_
```

以「快樂版函式庫」實作

- 呼叫範例

將這一段

換成這一段

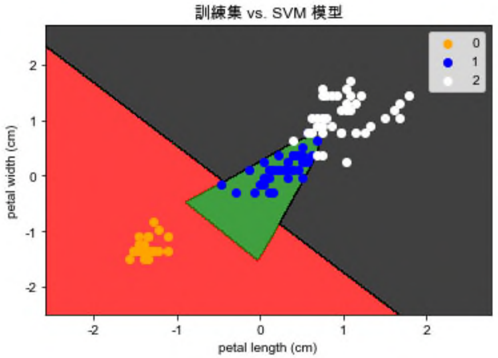
```python
# GridSearch --------------------
from HappyML.classification import SVM
classifier = SVM()

# GridSearch without HappyML
# from sklearn.model_selection import GridSearchCV
# grid_search = GridSearchCV(estimator=classifier.classifier, param_grid=params_list, verbose=10, cv=10)
# grid_search.fit(X, Y.values.ravel())
# print("Best Parameters: {}  Best Score: {}".format(grid_search.best_params_, grid_search.best_score_))
# classifier.classifier = grid_search.best_estimator_

# GridSearch with HappyML
from HappyML.performance import GridSearch
validator = GridSearch(estimator=classifier.classifier, parameters=params_list, verbose=True)
validator.fit(x_ary=X, y_ary=Y)
print("Best Parameters: {}  Best Score: {}".format(validator.best_parameters, validator.best_score))
classifier.classifier = validator.best_estimator

# Train & Predict
Y_pred_svm = classifier.fit(X_train, Y_train).predict(X_test)
```

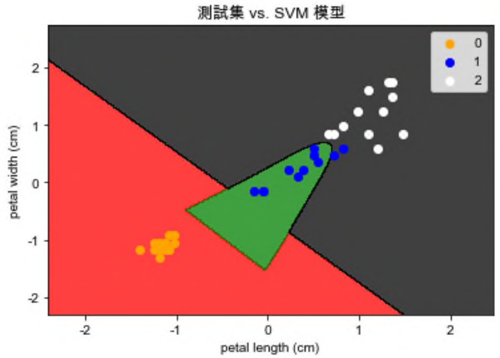
以「快樂版函式庫」實作

- 執行結果

超參數優化過的「支援向量機」效能

```
Best Parameters: {'C': 100000.0, 'gamma': 0.1, 'kernel': 'rbf'}  Best Score: 0.9666
----- SVM Classification -----
10 Folds Mean Accuracy: 0.9666666666666666
10 Folds Mean Recall: 0.9666666666666666
10 Folds Mean Precision: 0.9738095238095239
10 Folds Mean F1-Score: 0.9659090909090908
```





隨堂練習：以「快樂版函式庫」實作

- 請先將下列這幾行程式碼註解掉：

```
# GridSearch without HappyML
# from sklearn.model_selection import GridSearchCV
# grid_search = GridSearchCV(estimator=classifier.classifier, param_grid=params_list, verbose=10, cv=10)
# grid_search.fit(X, Y.values.ravel())
# print("Best Parameters: {}  Best Score: {}".format(grid_search.best_params_, grid_search.best_score_))
# classifier.classifier = grid_search.best_estimator_
```

- 然後在下方補上這些程式碼：

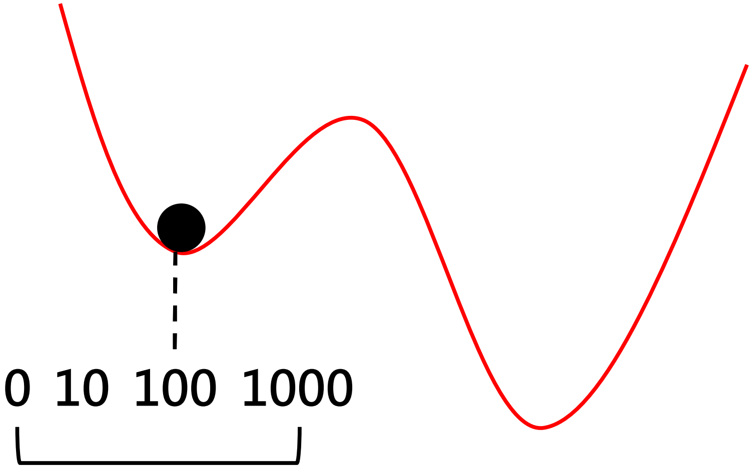
```
# GridSearch with HappyML
from HappyML.performance import GridSearch
validator = GridSearch(estimator=classifier.classifier, parameters=params_list, verbose=True)
validator.fit(x_ary=X, y_ary=Y)
print("Best Parameters: {}  Best Score: {}".format(validator.best_parameters, validator.best_score))
classifier.classifier = validator.best_estimator
```

- 將程式碼一口氣、從頭執行到尾，看看是否能用快樂版，優化模型的超參數組合？

「網格搜尋」的缺點

- 耗費運算時間 & 效能

- 僅能取得「局部極值」（Local Optimization）



<u>其它已知問題</u>：

- 執行「多項式內核函數（“poly”）」高於五次方項時，系統容易當機（解法：用 “rbf” 取代之）。
- 若使用 n_jobs 開啟多核心平行運作時（亦即 n_jobs > 1），容易當機，可使用單核心（n_jobs=1）運行之。（見：https://git.io/fj7FH）

課後作業：辨識「男聲」vs.「女聲」

- 資料集：
  - 請下載 Voice.csv 檔案。
  - 前 20 欄（[0] ~ [19]），是各種音質分析參數：
    - meanfreq：平均音頻
    - sd：最低音 ~ 最高音標準差
    - median：音頻中位數
    - Q25, IQR, Q75：音頻的第一、第二、第三、四分位數。
    - ......（以下略）
  - 第 21 欄（[20]），是判別結果（male=男性、female=女性）

- 要求：
  - 使用 **SVM** 做為分類器。
  - 必須挑選出「**顯著性高**」的欄位。
  - 請先用「SVM **預設超參數**」，執行訓練、預測。
  - 印出該次分類器 10 次交叉驗證後的「**確度、廣度、精度、F值**」。
  - 再利用「**網格搜尋**」，找出 SVM 分類器的最佳「超參數」。
  - 再印出該次分類器 10 次交叉驗證後的「**確度、廣度、精度、F值**」。
  - 比較兩次的效能，看看選擇**正確的超參數**，對 SVM 的影響有多大？

課後作業：辨識「男聲」vs.「女聲」

- 提示：
  - 應變數 Y 記得要做 Label Encoder。

- 輸出結果：

未調整超參數之前

```
The Significant Level: 0.05

--- The p-values of Feature Importance ---
TRUE  <0.05 0.00000000e+00 (kurt)
TRUE  <0.05 9.83873519e-67 (maxdom)
TRUE  <0.05 4.33796654e-65 (dfrange)
TRUE  <0.05 5.58809346e-10 (meandom)
TRUE  <0.05 2.27215909e-08 (sfm)
TRUE  <0.05 2.86631924e-07 (IQR)
TRUE  <0.05 8.11826484e-07 (skew)
TRUE  <0.05 6.02634872e-05 (meanfun)
TRUE  <0.05 1.85041429e-04 (Q25)
TRUE  <0.05 2.46987017e-03 (mindom)
FALSE >0.05 6.00809179e-02 (sd)
FALSE >0.05 6.64000790e-02 (mode)
FALSE >0.05 1.77517189e-01 (median)
FALSE >0.05 1.81665559e-01 (meanfreq)
FALSE >0.05 1.81665559e-01 (centroid)
FALSE >0.05 1.89369378e-01 (sp.ent)
FALSE >0.05 4.40885252e-01 (minfun)
FALSE >0.05 5.79710639e-01 (maxfun)
FALSE >0.05 6.19374212e-01 (modindx)
FALSE >0.05 8.51079560e-01 (Q75)

Number of Features Selected: 10
----- SVM Classification -----
10 Folds Mean Accuracy: 0.6582975081601783
10 Folds Mean Recall: 0.6582975081601783
10 Folds Mean Precision: 0.6764032221012367
10 Folds Mean F1-Score: 0.645261401304672
```

已調整超參數之後

```
The Significant Level: 0.05

--- The p-values of Feature Importance ---
TRUE  <0.05 0.00000000e+00 (kurt)
TRUE  <0.05 9.83873519e-67 (maxdom)
TRUE  <0.05 4.33796654e-65 (dfrange)
TRUE  <0.05 5.58809346e-10 (meandom)
TRUE  <0.05 2.27215909e-08 (sfm)
TRUE  <0.05 2.86631924e-07 (IQR)
TRUE  <0.05 8.11826484e-07 (skew)
TRUE  <0.05 6.02634872e-05 (meanfun)
TRUE  <0.05 1.85041429e-04 (Q25)
TRUE  <0.05 2.46987017e-03 (mindom)
FALSE >0.05 6.00809179e-02 (sd)
FALSE >0.05 6.64000790e-02 (mode)
FALSE >0.05 1.77517189e-01 (median)
FALSE >0.05 1.81665559e-01 (meanfreq)
FALSE >0.05 1.81665559e-01 (centroid)
FALSE >0.05 1.89369378e-01 (sp.ent)
FALSE >0.05 4.40885252e-01 (minfun)
FALSE >0.05 5.79710639e-01 (maxfun)
FALSE >0.05 6.19374212e-01 (modindx)
FALSE >0.05 8.51079560e-01 (Q75)

Number of Features Selected: 10
Best Parameters: {'C': 1000.0, 'gamma': 0.01}  Best Score: 0.9148594857097366
----- SVM Classification -----
10 Folds Mean Accuracy: 0.9148594857097366
10 Folds Mean Recall: 0.9148594857097366
10 Folds Mean Precision: 0.9198367858870806
10 Folds Mean F1-Score: 0.914489246727207
```

本章總結

- **原理解說**
  - 支援向量：離「決定平面」最近的兩個點。

- **分類原理**
  - 先用「內核函數」映射到高維空間。
  - 找到分割用的「決定平面」。
  - 將「決定平面」映射回低維空間。

- **常見的內核函數**
  - 線性（Linear）： $\boldsymbol{k(X,Y) = X^T \cdot Y + c}$
  - 高斯徑向基底函數（Gaussian RBF）： $\boldsymbol{k(X,Y) = e^{-\frac{\|X-Y\|^2}{2\sigma^2}}}$
  - 多項式內核（Polynomial）： $\boldsymbol{k(X,Y) = \left(X^T \cdot Y + c\right)^d}$
  - Sigmoid 內核： $\boldsymbol{k(X,Y) = tanh\left(\alpha X^T \cdot Y + c\right)}$

本章總結

- **優缺點**
  - 優點：能分辨**離群值**、**內核函數**可更換。
  - 缺點：**計算量**龐大、受**超參數**挑選影響大。

- **重要參數**
  - C：懲罰參數（C小：允許雜點、C大：不允許雜點）
  - gamma：離散參數。Gamma 小，離散程度越大，涵蓋範圍越大。
  - degree：次方項。多項式函數使用。
  - coef0：常數項。線性、多項式、Sigmoid 函數使用。

- **SVM 相關 Python 套件**
  - 分類器：sklearn.svm.SVC
  - 網格搜尋：sklearn.model_selection.GridSearchCV