實驗5

實驗 5.1.1 – example241

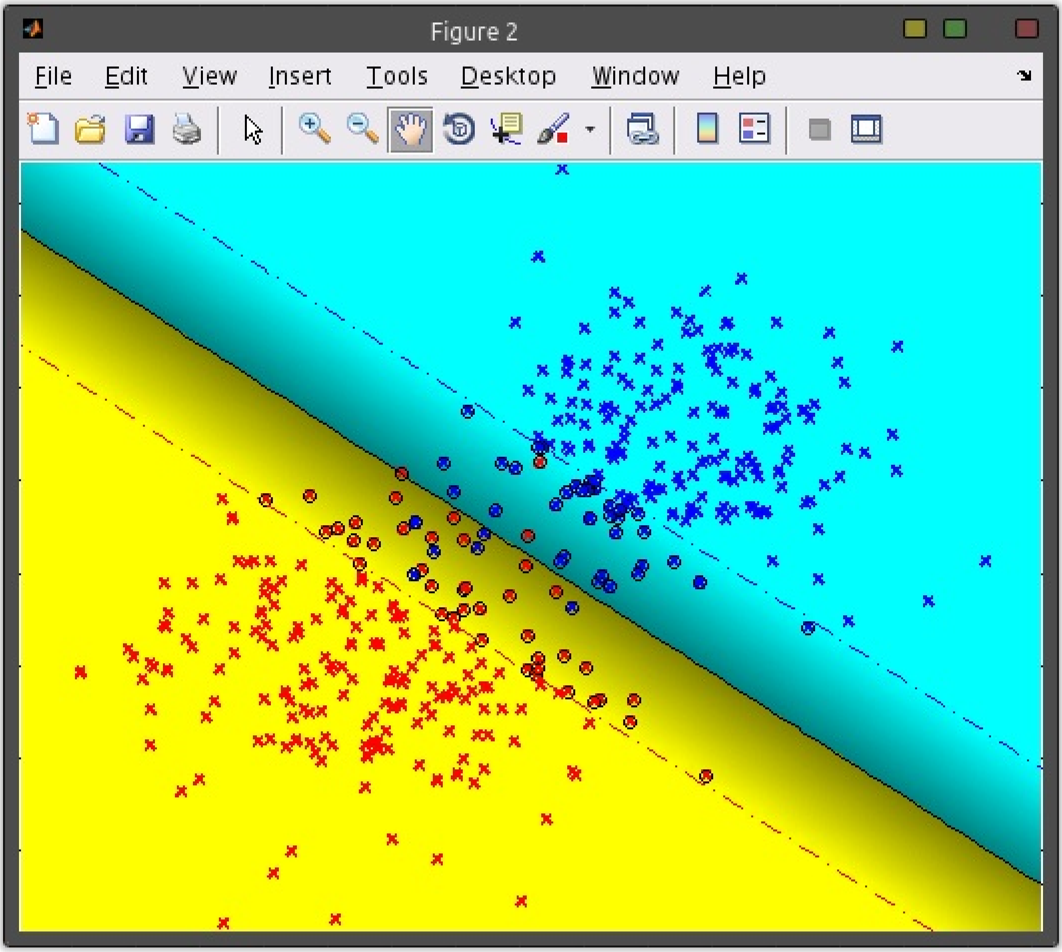
- 用SVM進行分類：使用Platt算法、線性、C=0.1的情況、忍耐度=0.001、最多100000次迭代、誤差=10^(-10)

```
kernel='linear';
kpar1=0;
kpar2=0;
C=0.1;
% C=0.2;
% C= 0.5;
% C=1;
% C=2;
% C=20;
tol=0.001;
steps=100000;
eps=10^(-10);
method=0;
[alpha, w0, w, evals, stp, glob] = SMO2(X1', y1',kernel, kpar1, kpar2, C, tol, steps,
```

- C=0.1時:

- 結果：

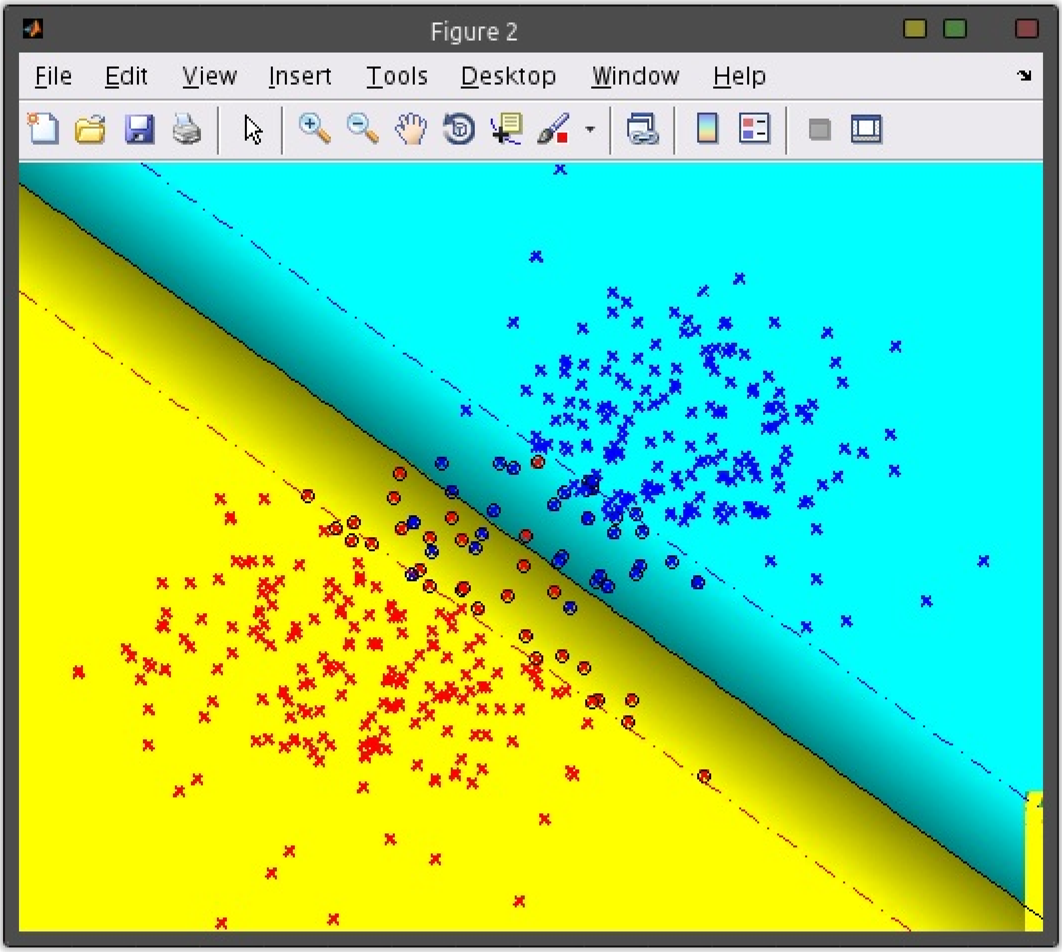
```
Pe_tr =
    0.0225
Pe_te =
    0.0325
sup_vec =
    82
marg =
    0.9410
```

- 

- C=0.2時：

- 結果：

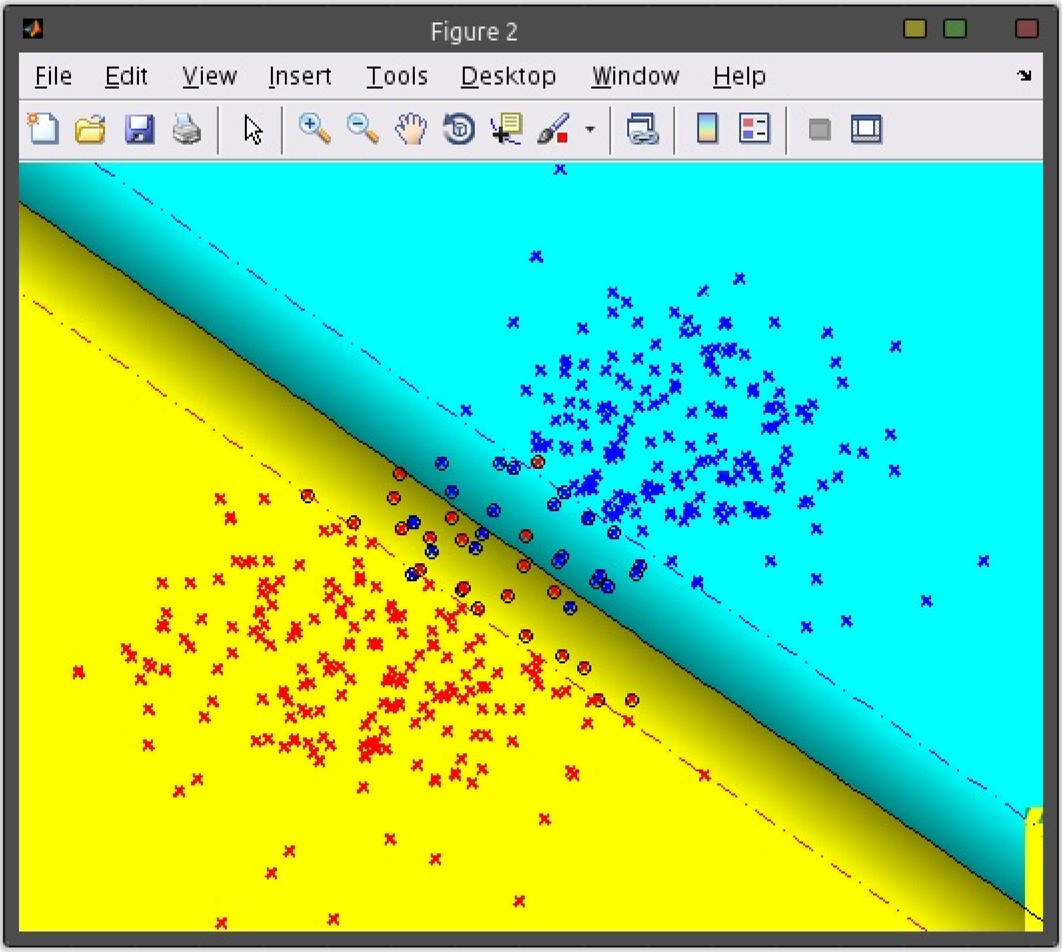
```
Pe_tr =
    0.0200
Pe_te =
    0.0300
sup_vec =
    61
marg =
    0.8219
```

* 

* C=0.5時：

* 結果：

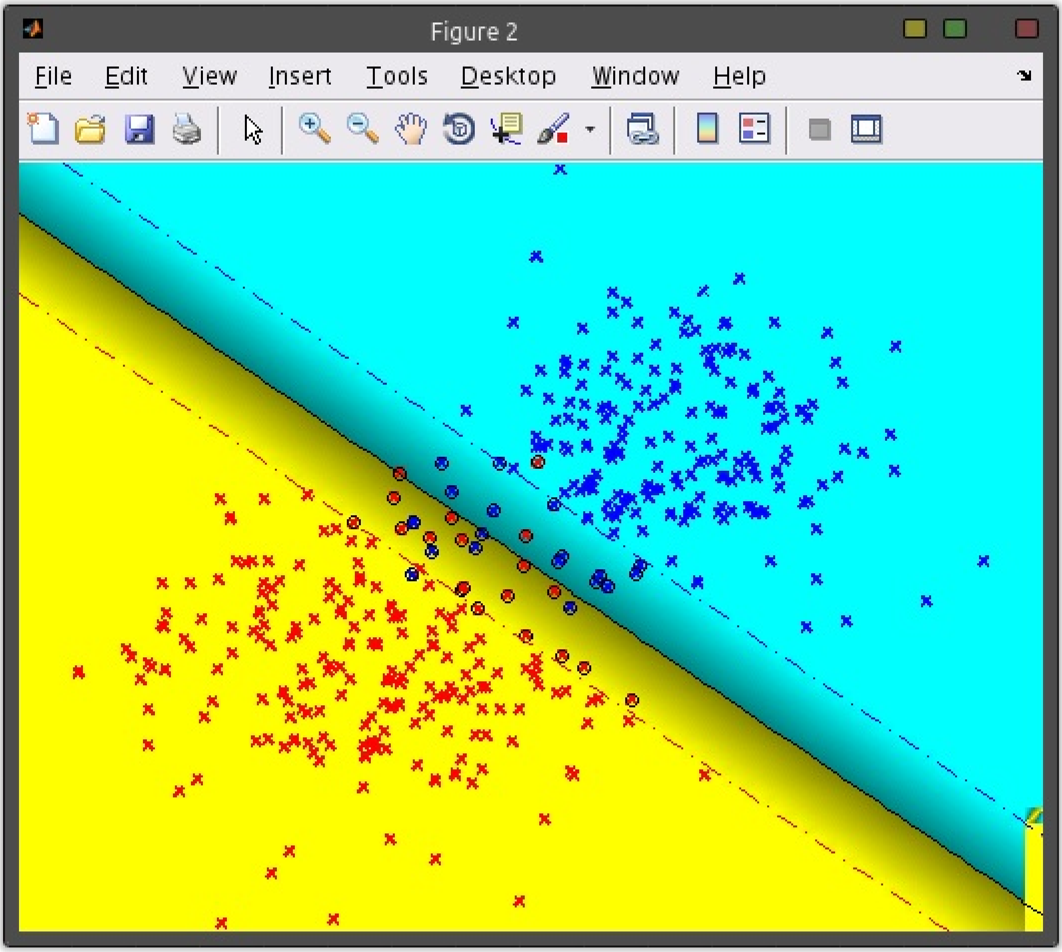
```
Pe_tr =
    0.0200
Pe_te =
    0.0325
sup_vec =
    44
marg =
    0.7085
```

- 

- C=1時：

- 結果：

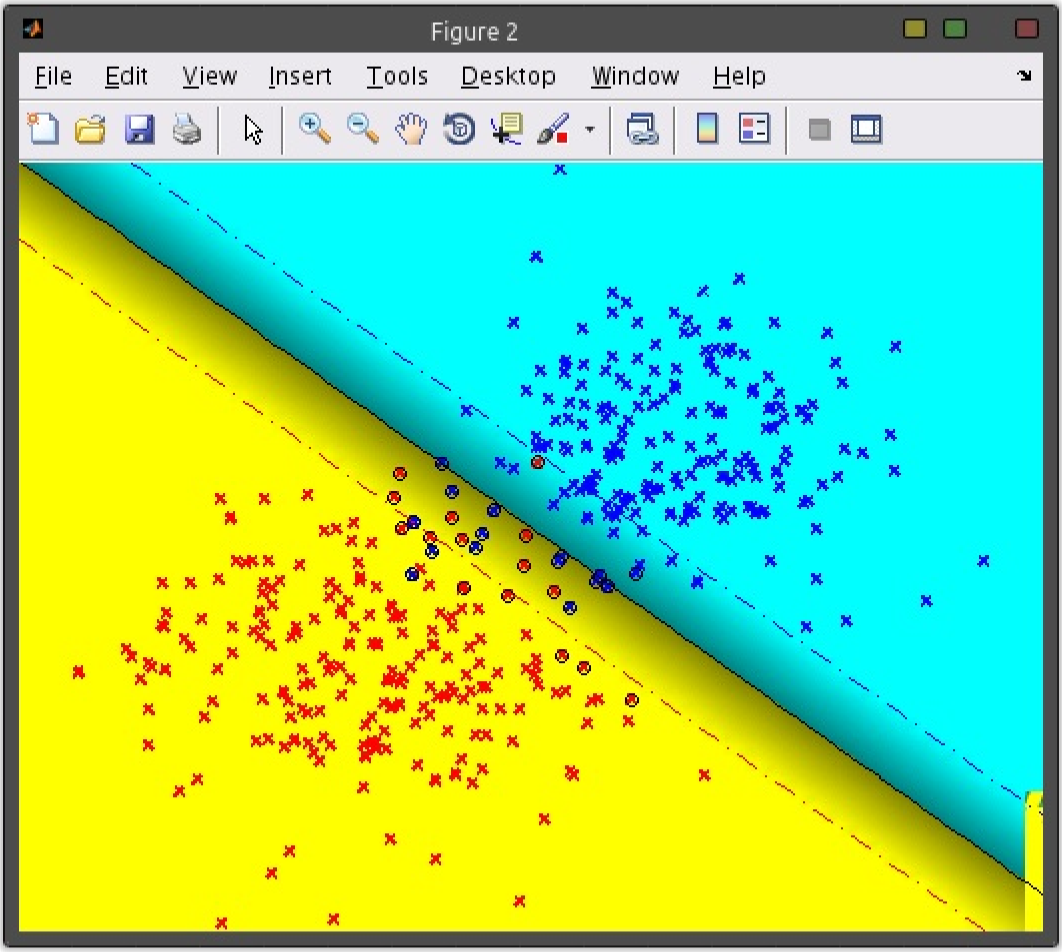
```
Pe_tr =
    0.0225
Pe_te =
    0.0325
sup_vec =
    37
marg =
    0.6319
```

- 

- C=2時：

- 結果：

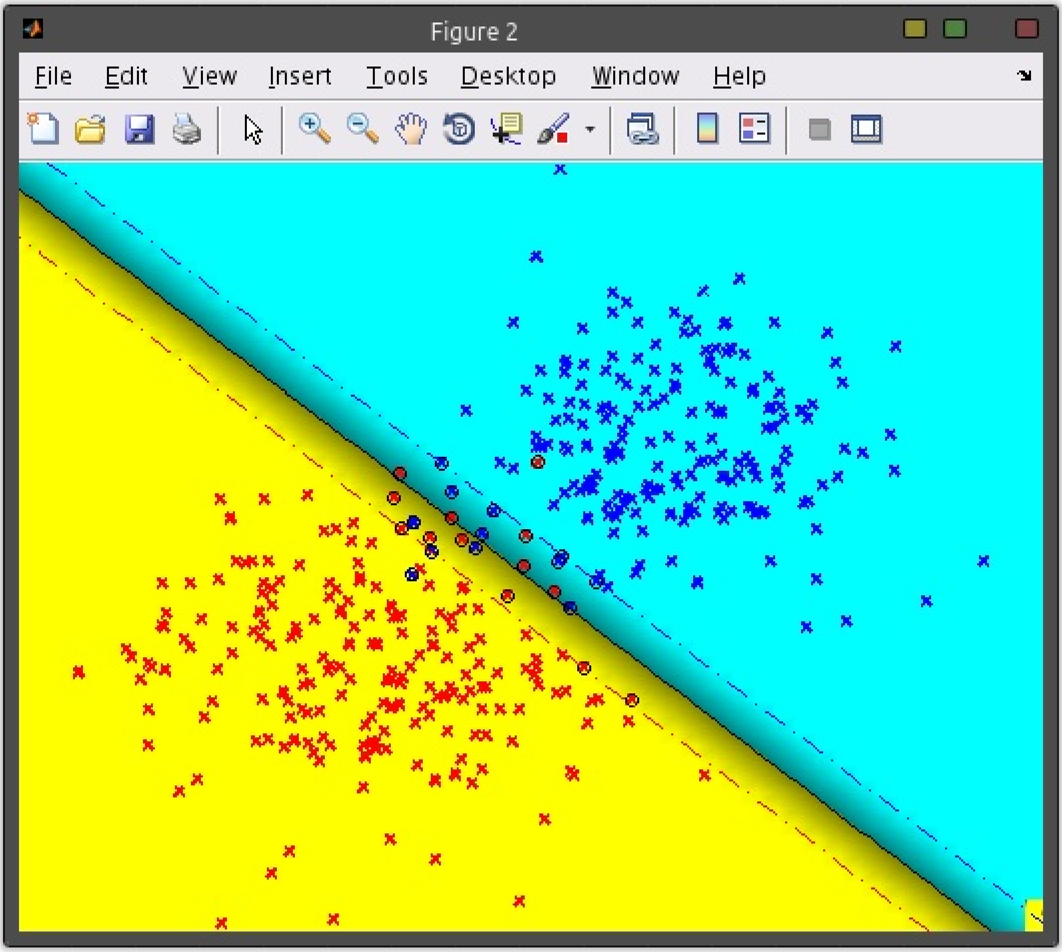
```
Pe_tr =
    0.0325
Pe_te =
    0.0350
sup_vec =
    31
marg =
    0.6047
```

- 

- C=20時：

- 結果：

```
Pe_tr =
    0.0250
Pe_te =
    0.0350
sup_vec =
    25
marg =
    0.3573
```

- 

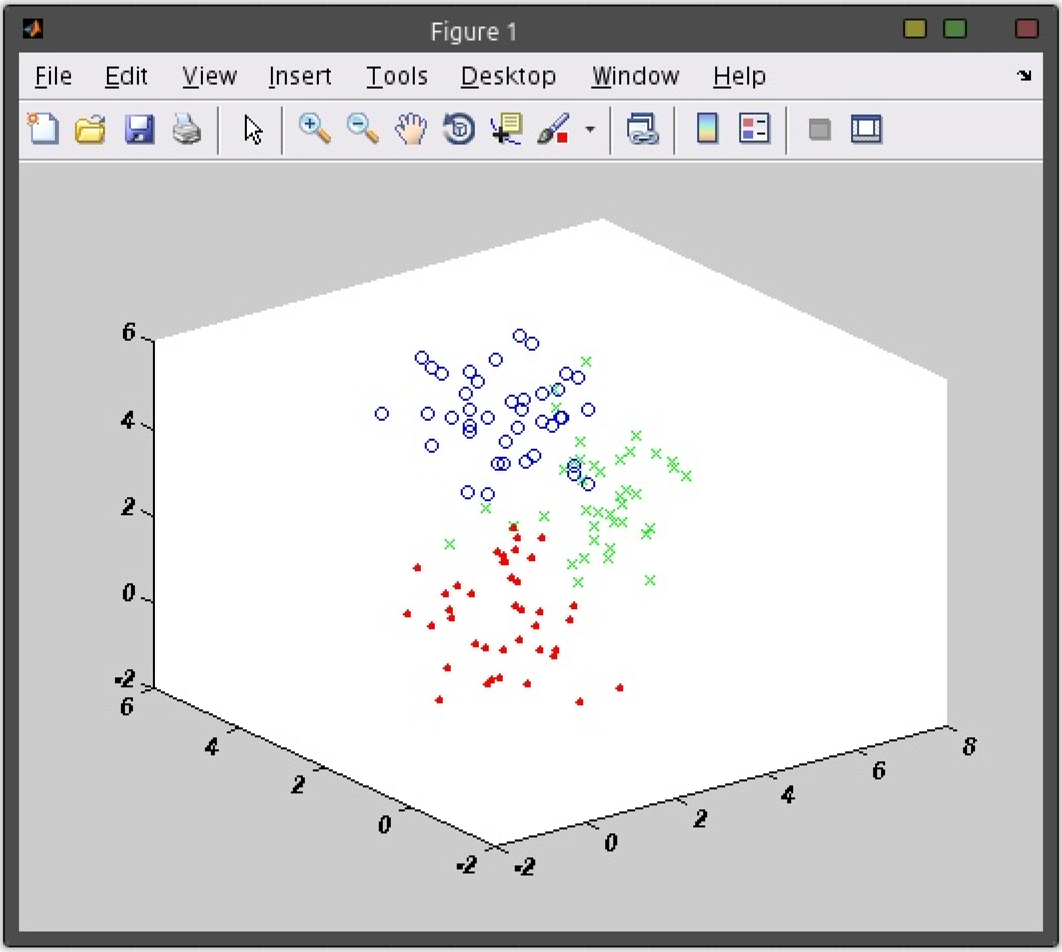
- 參數「C」對分類器的位置、斜率，以及邊界寬度都有很大的影響。 C越小，在邊界內允許被誤分的點就越多，從而擴大了邊界寬度等。 C越大，在邊界內允許被誤分的點就越少，從而減小了邊界寬度等。

實驗 5.1.2 – example242

- 代碼就是將`example241`「移植」到了3緯上
- 結果：

```
marg =
    0.3584    1.1198    0.6592
sup_vec =
     5    19    13
err_svm =
    0.0500
```

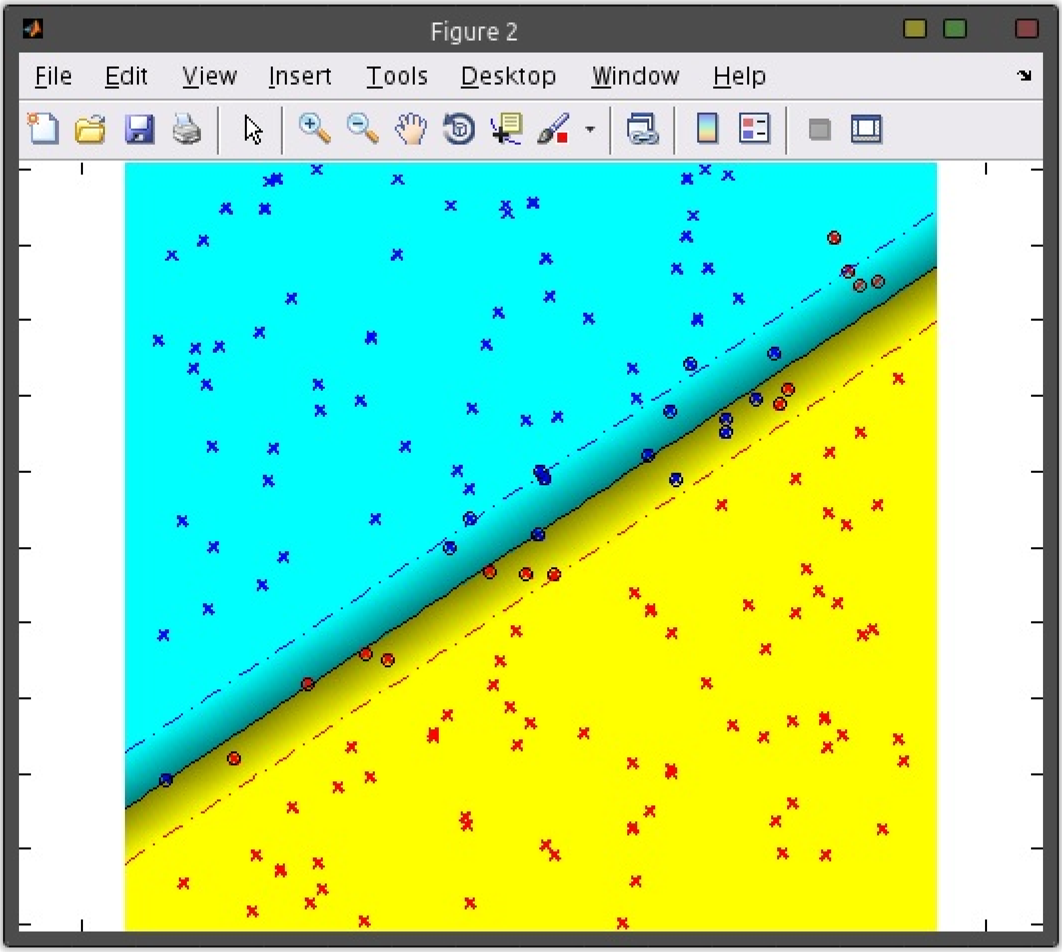
•



實驗 5.1.3 – example251

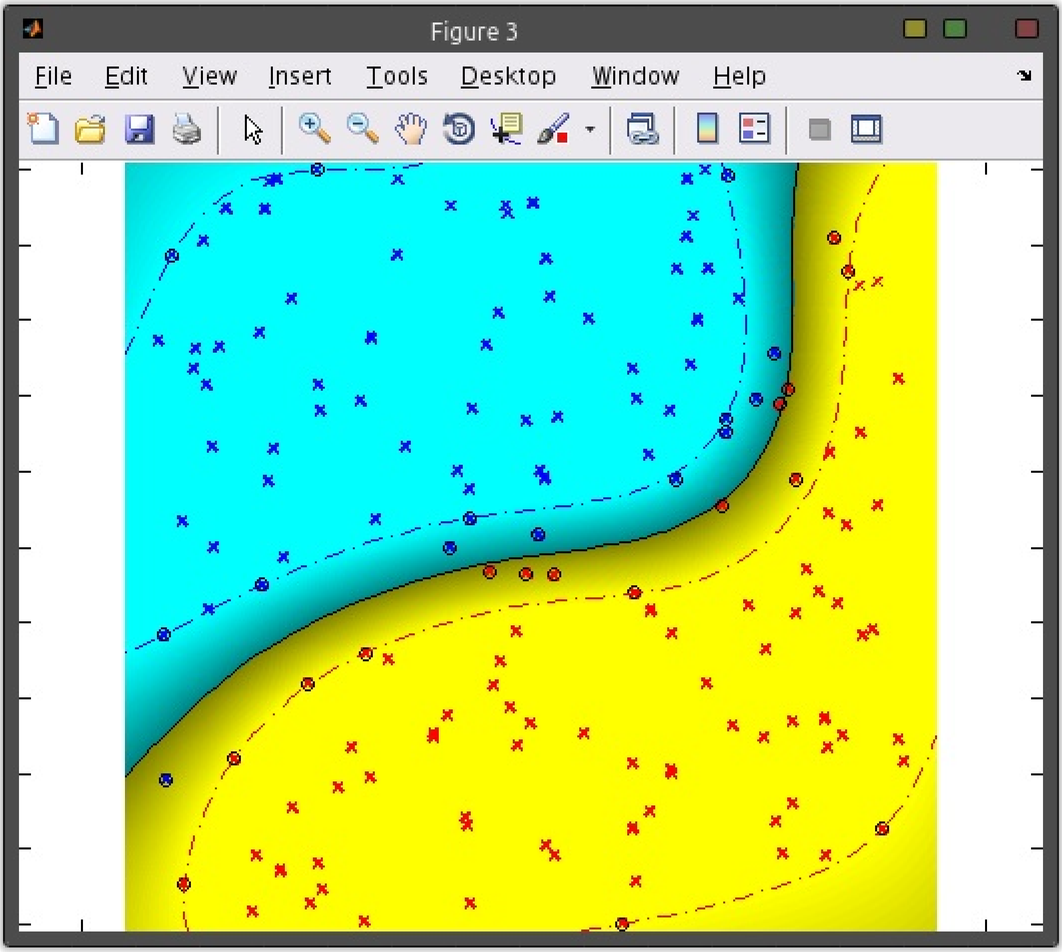
- 代碼分別使用了`Linear`，`Radial Basis Function`，`Polynomial`這三種kernel來進行
- linear:
  - result:

```
kernel =
linear
marg =
    0.7482
Pe_train =
    0.0667
Pe_test =
    0.0733
sup_vec =
    27
```

- 

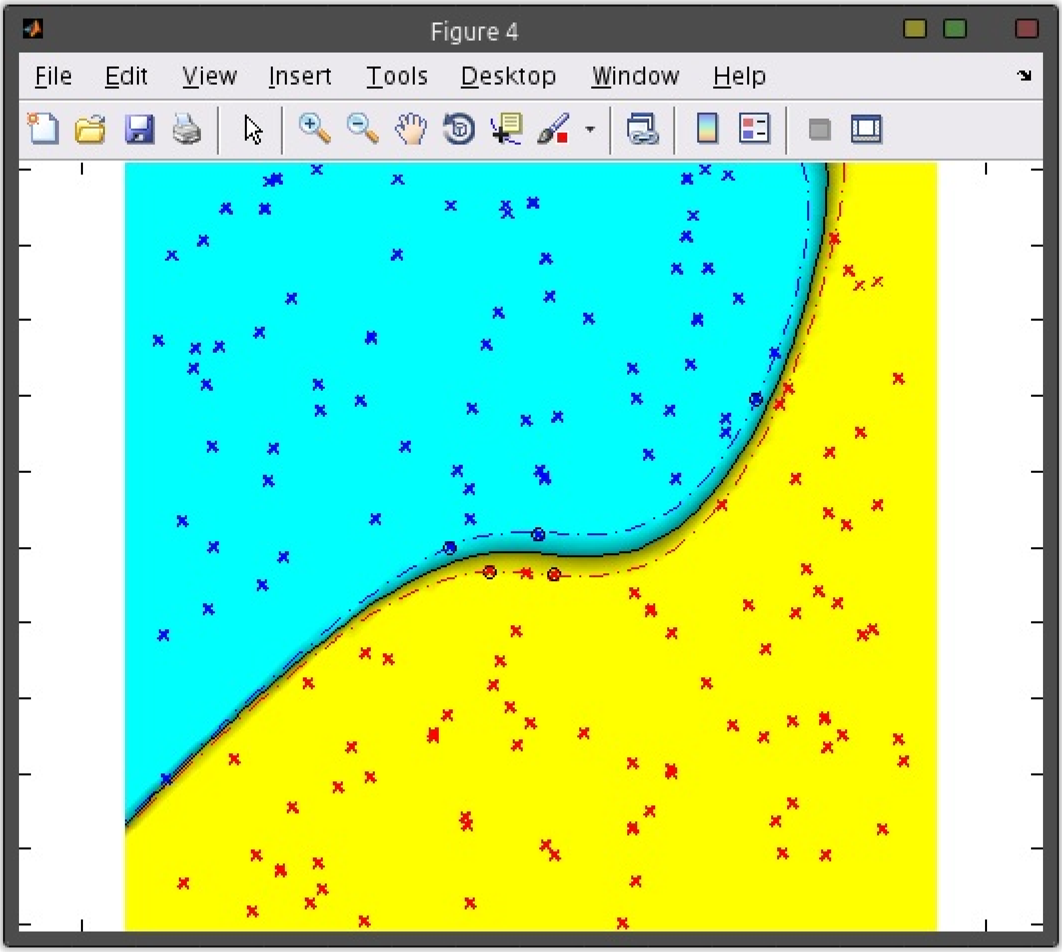
- rbf:
- result:

```
kernel =
rbf
Pe_train =
    0.0133
Pe_test =
    0.0333
sup_vec =
    30
```

- 

- polynomial:
  - result:

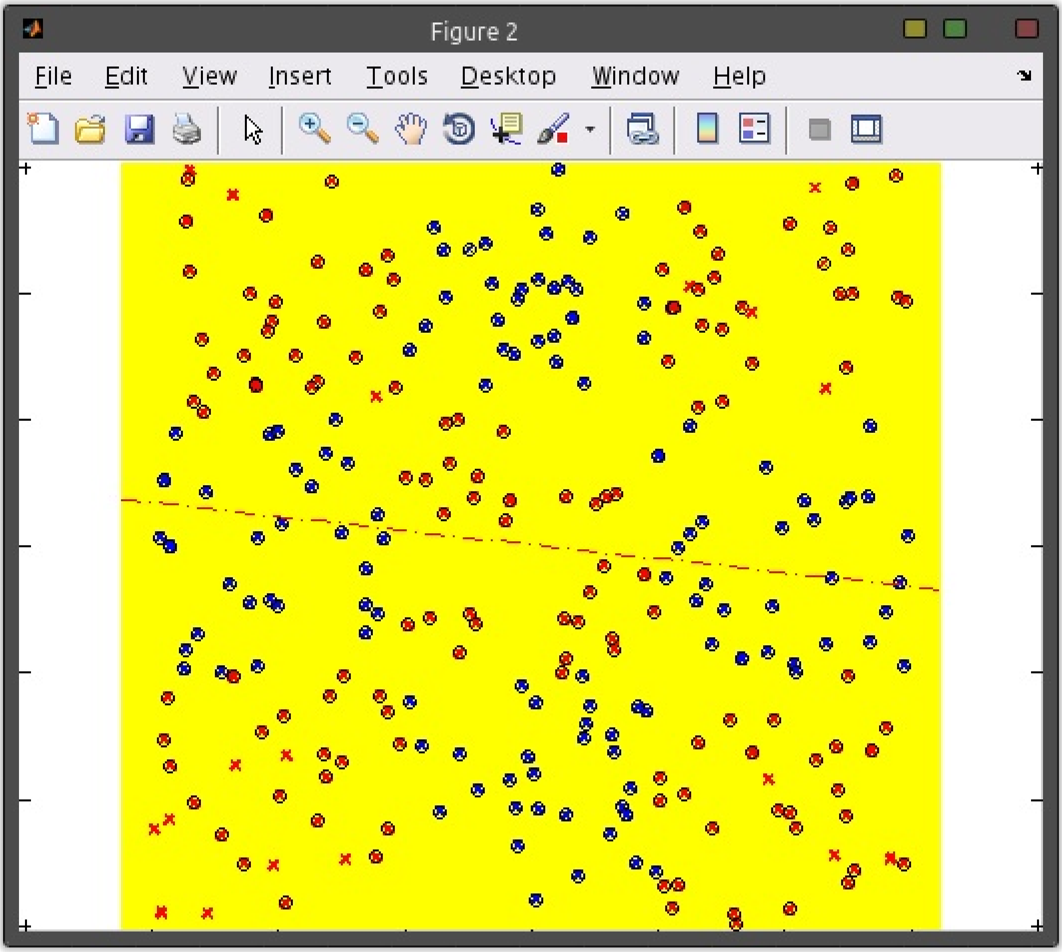
```
kernel =
poly
Pe_train =
     0
Pe_test =
    0.0267
sup_vec =
     8
```

- 

實驗 5.1.4 – example252

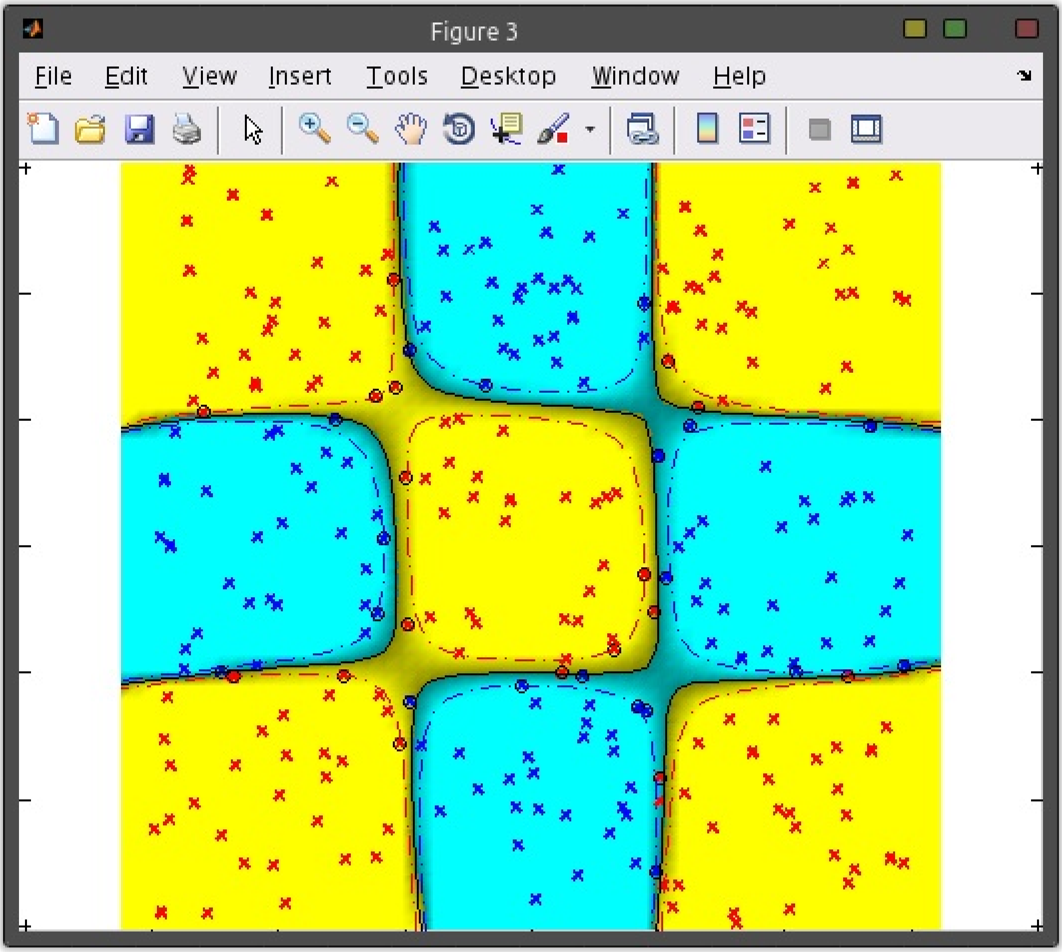
- 代碼分別使用了`Linear`，`Radial Basis Function`，`Polynomial`這三種kernel來進行，數據與example251不同。
- linear:
- result:

```
kernel =
linear
marg =
   3.3904e+08
Pe_train =
    0.4444
Pe_test =
    0.4444
sup_vec =
   255
```

- 

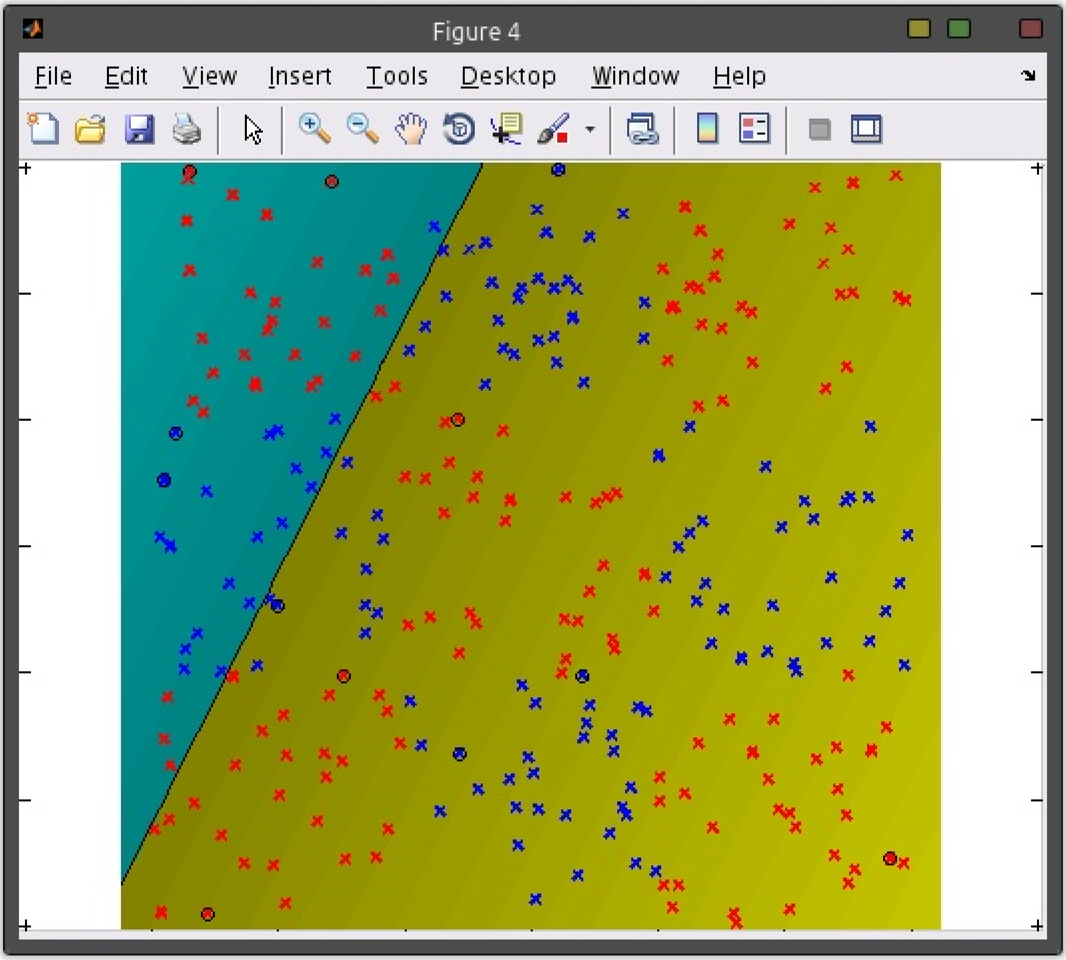
- rbf:
  - result:

```
kernel =
rbf
Pe_train =
    0.0185
Pe_test =
    0.0370
sup_vec =
    36
```

- 

- poly:
- result:

```
kernel =
poly
Pe_train =
    0.4852
Pe_test =
    0.5111
sup_vec =
    12
```

- 

example251 vs example252

- 兩種數據，多種kernel之間的特性：
    - 對於線性可分的情況，linear, rbf和poly的效果都還不錯
    - 對於線性不可分的情況下，rbf的效果遠遠勝於linear和poly。
- 總結：
    - 在線性不可分的情況下，應該優先選擇rbf kernel

實驗 5.2

操作流程(環境Ubuntu 64bit, g++-4.8)：

```
./libsvm_build/svm-train ./iris_train.scale iris.model

*
optimization finished, #iter = 10
nu = 0.145412
obj = -3.940271, rho = -0.000668
nSV = 9, nBSV = 6
*
optimization finished, #iter = 16
nu = 0.086859
obj = -2.287943, rho = 0.131067
nSV = 6, nBSV = 2
*
optimization finished, #iter = 31
nu = 0.568109
obj = -20.090069, rho = 0.033717
```

```
nSV = 31, nBSV = 27
Total nSV = 42

./libsvm_build/svm-predict iris_test.scale iris.model iris.result
Accuracy = 96% (72/75) (classification)
```