Assignment #4 -- SVM

題目要求

1. Use the linear model y = 2x +ε with zero-mean Gaussian noise ε~ N(0, 1) to generate 500 data points with (equal spacing) x ∈ [−100, 100].
2. 訓練一個RBF SVM
3. 利用5-fold cross validation，至少找三組C and γ 參數，挑一組最好的
4. 用上題找到的參數，比較scaling方法、或不用scaling，影響準確率的差異為何。

我的答案

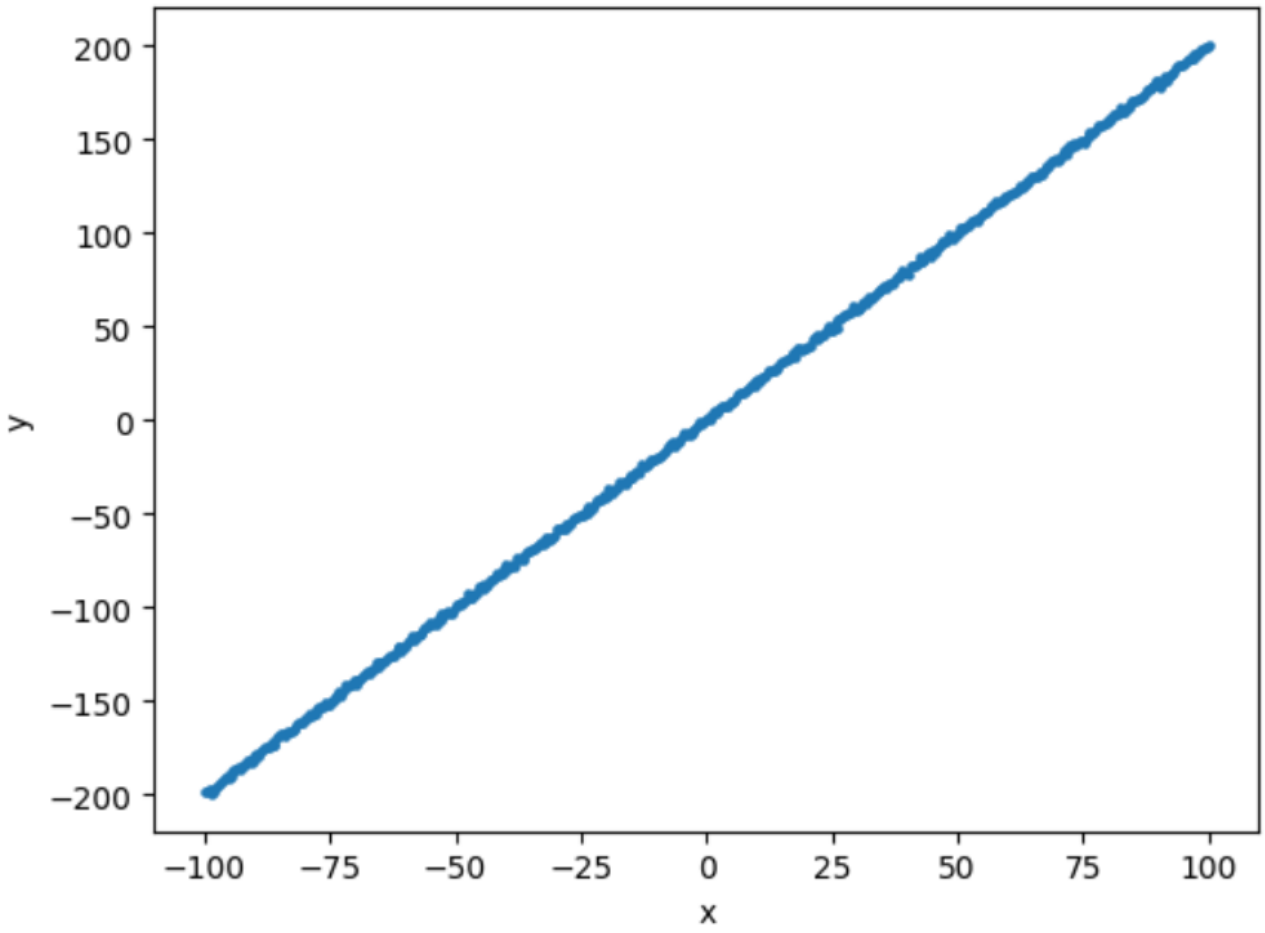
如何執行 (Execution Description)

1. 把 `libsvm_code.ipynb` 檔案以 GOOGLE COLAB 開啟。
2. 點選 **執行階段** > **全部執行** 。

實驗結果 (Experimental Results)

題目要求 1 ：

- 用 $y = 2x + \varepsilon$ 且 $\varepsilon \in N(0,1)$ 這個模型產生 500 筆資料， $x \in [-100, 100]$ ，相鄰的 x 值之間有相同的間距：



題目要求 2 ：

- 訓練一個 RBF SVM 。
- 這邊我使用參數 `-s 1 -t 2 -d 2 -c 128 -g 0.0001220703125 -w1 1 -w-1 6 -v 5` 。
- 使用以上參數所得的訓練結果截圖：

```
CVA: 92.75
para_str: -s 1 -t 2 -d 2 -c 128 -g 0.0001220703125 -w1 1 -w-1 6 -v 5
Accuracy = 93% (93/100) (classification)
```

  - `CVA` (5-fold cross validation) 值為 `92.75` 。
  - 將模型套用到 testing set 的結果為 `Accuracy = 93%` 。

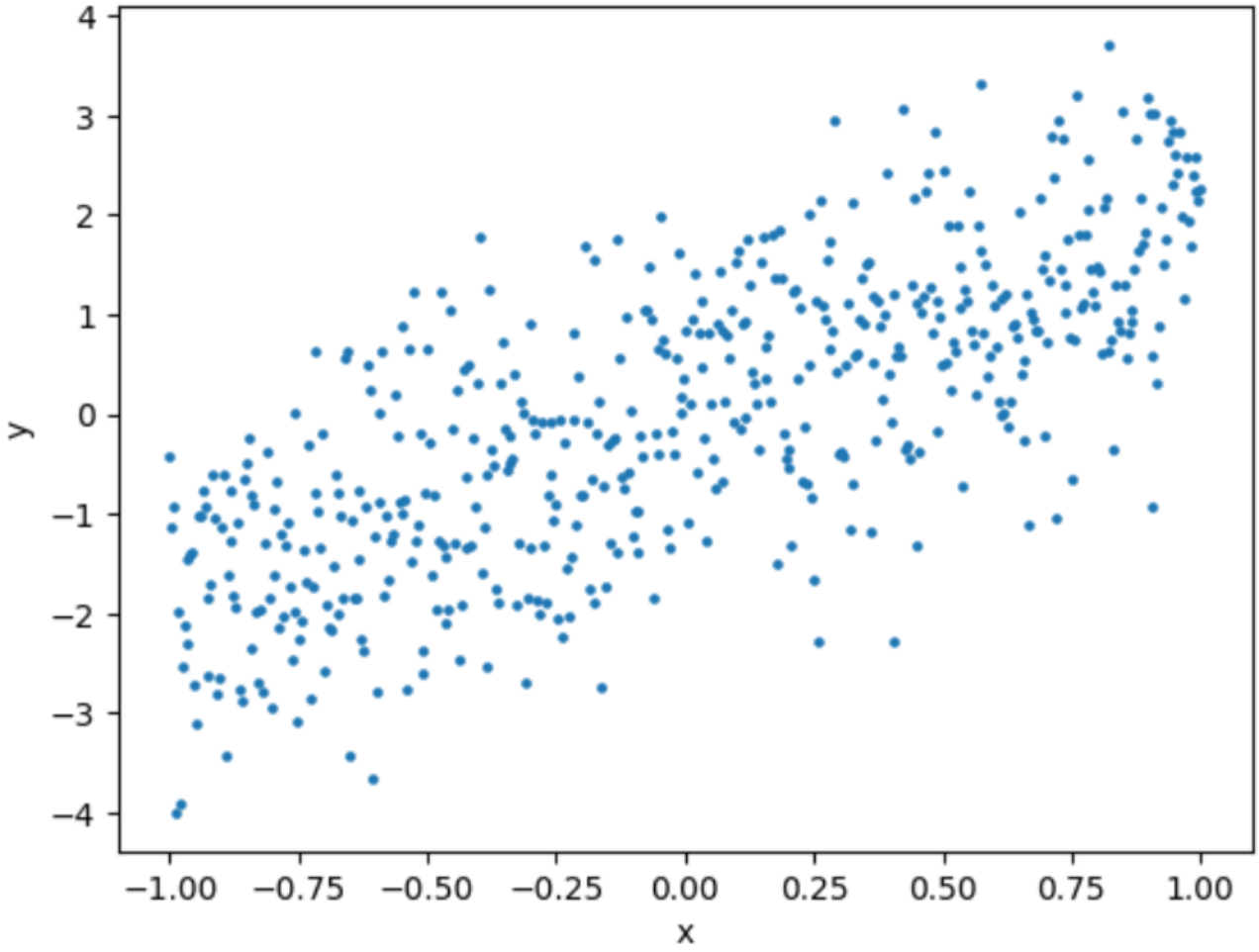
題目要求 3 ：

- 利用 5-fold cross validation 來找出三組 C 和 γ ，並且挑一組最好的。
- 我先分別假設了一些 C 和 γ 的值，讓程式慢慢去測試每種組合的結果，最後取出 3 組使 5-fold cross validation 值最好的參數，訓練結果截圖如下：

```
TOP 3 results:
< 1 >
CVA: 92.75
para_str: -s 1 -t 2 -d 2 -c 128 -g 0.0001220703125 -w1 1 -w-1 6 -v 5
Accuracy = 93% (93/100) (classification)
< 2 >
CVA: 91.5
para_str: -s 1 -t 2 -d 2 -c 0.001953125 -g 0.0001220703125 -w1 1 -w-1 6 -v 5
Accuracy = 93% (93/100) (classification)
< 3 >
CVA: 91.5
para_str: -s 1 -t 2 -d 2 -c 256 -g 0.0001220703125 -w1 1 -w-1 6 -v 5
Accuracy = 93% (93/100) (classification)
```

題目要求 4 ：

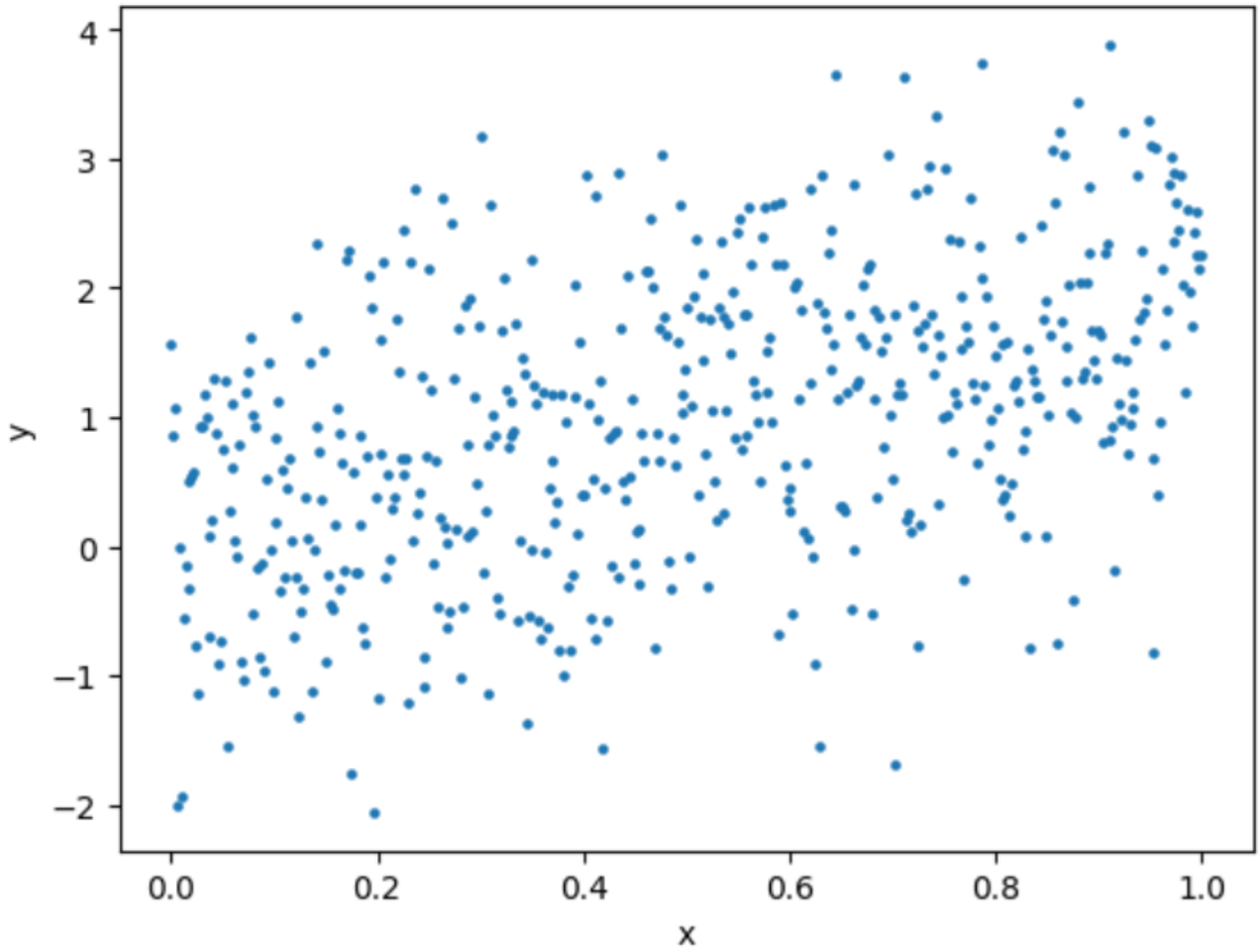
- 這邊我使用兩種對 x 的 scaling ，分別是 $[-1,1]$ 和 $[0,1]$ 。
- $[-1,1]$ scaling 資料分布截圖：



- $[-1,1]$ scaling 訓練結果截圖：

```
Cross Validation Accuracy = 93.75%
Accuracy = 96% (96/100) (classification)
```

- $[0,1]$ scaling 資料分布截圖：



- $[0,1]$ scaling 訓練結果截圖：

```
Cross Validation Accuracy = 87.5%
Accuracy = 92% (92/100) (classification)
```

結果觀察 (Conclusion)

1. 在使用 $(C,\gamma) = (128, 0.0001220703125)$ 這組參數的時候有較好的效果 (5-fold cross validation 較好)，不過也發現，有時候 5-fold cross validation 較好但還不是裡面最佳者在 testing 的時候會有更好的準確度。

2. 使用我從 $[-100,100]$ 這個區間找到的較好的參數時，將數據 (x) scale 至 $[-1,1]$ 這個區間會比將數據 scale 至 $[0,1]$ 這個區間或不 scale (維持 $[-100,100]$) 還要好。

相關討論 (Disscussion)

在選擇 C 和 γ 的值的時候，我有想過是不是應該要看 5-fold cross validation 以及當前的 train error 的值綜合判斷才可以決定我們的參數是否是目前最好的，不過我翻了 libsvm 專案的 source code 後發現好像沒有同時產生 5-fold cross validation 以及當前的 train error 的值的功能。

程式碼 (Code)

```
!pip install libsvm
```

```
import torch

# Set the number of data points
n = 500

# Set the range of x
x_min, x_max = -100, 100

# Generate equally spaced values of x
x = torch.linspace(x_min, x_max, n)

# store x val for future scaling
x_init = x

# store rand_perm for future scaling
rand_perm = torch.randperm(n)

# store split info for future scaling
split_rate = 0.8
split_num = int(n*split_rate)

# Generate noise
noise = torch.randn(n)

# Generate y values
y = 2 * x + noise
```

```
import matplotlib.pyplot as plt

plt.scatter(x, y, s=5)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
# random perm
# split train test

x = x.unsqueeze(1)
y = y.unsqueeze(1)

x = x[rand_perm]
y = y[rand_perm]
xy = torch.cat((x, y), dim=1)
print(xy.shape)

train_data = xy.numpy()[:split_num]
test_data = xy.numpy()[split_num:]
```

```
print(len(train_data))

# y=2x
train_label = torch.sign(2 * x - y).numpy().squeeze().tolist()[:split_num]
test_label = torch.sign(2 * x - y).numpy().squeeze().tolist()[split_num:]
```

```
from libsvm.svmutil import *
from libsvm import svmutil
import scipy
import scipy.constants
```

```
svm_model = svmutil.svm_train(train_label, train_data, "-s 1 -t 2 -d 2 -c 0.125 -g
0.000030517578125 -w1 1 -w-1 6 -v 5")
svm_model = svmutil.svm_train(train_label, train_data, "-s 1 -t 2 -d 2 -c 0.125 -g
0.000030517578125 -w1 1 -w-1 6")
label, acc, vals = svmutil.svm_predict(test_label, test_data, svm_model)
```

```
Cross_Validation_Accuracy = []
for i in range(-15, 16):
    c = 2 ** i
    for j in range(-15, 16):
        g = 2 ** j
        print(i, j)
        print("(c, g) = ( " + str(c) + ", " + str(g) + " )")
        para_str = '-s 1 -t 2 -d 2 -c ' + str(c) + ' -g ' + str(g) + ' -w1 1 -w-1
6 -v 5'
        svm_model = svmutil.svm_train(train_label, train_data, para_str)
        Cross_Validation_Accuracy.append((svm_model, para_str, c, g))
```

```
sorted_CVA = sorted(Cross_Validation_Accuracy, key=lambda x: x[0], reverse=True)
```

```
# TOP 3 (cross fold) and predicted results
print("TOP 3 results:")
for i in range(3):
    print("< " + str(i+1) + " >")
    print("CVA:", sorted_CVA[i][0])
    print("para_str:", sorted_CVA[i][1])
    para_str = '-s 1 -t 2 -d 2 -c ' + str(sorted_CVA[i][2]) + ' -g ' +
str(sorted_CVA[i][3]) + ' -w1 1 -w-1 6'
    svm_model = svmutil.svm_train(train_label, train_data, para_str)
    label, acc, vals = svmutil.svm_predict(test_label, test_data, svm_model)
```

```
# scaling [-100, 100] --> [-1, 1]
old_upper = 100
old_lower = -100
new_upper = 1
new_lower = -1
x = ( new_upper - new_lower ) * ( x_init - old_lower ) / ( old_upper - old_lower )
+ new_lower
```

```
# Generate y values
y = 2 * x + noise
```

```
plt.scatter(x, y, s=5)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
# random perm
# split train test

x = x.unsqueeze(1)
y = y.unsqueeze(1)

x = x[rand_perm]
y = y[rand_perm]
xy = torch.cat((x, y), dim=1)
print(xy.shape)

train_data = xy.numpy()[:split_num]
test_data = xy.numpy()[split_num:]
print(len(train_data))

# y=2x
train_label = torch.sign(2 * x - y).numpy().squeeze().tolist()[:split_num]
test_label = torch.sign(2 * x - y).numpy().squeeze().tolist()[split_num:]
```

```
para_str = '-s 1 -t 2 -d 2 -c ' + str(sorted_CVA[0][2]) + ' -g ' +
str(sorted_CVA[0][3]) + ' -w1 1 -w-1 6 -v 5'
svm_model = svmutil.svm_train(train_label, train_data, para_str)
para_str = '-s 1 -t 2 -d 2 -c ' + str(sorted_CVA[0][2]) + ' -g ' +
str(sorted_CVA[0][3]) + ' -w1 1 -w-1 6'
svm_model = svmutil.svm_train(train_label, train_data, para_str)
label, acc, vals = svmutil.svm_predict(test_label, test_data, svm_model)
```

```
# scaling [-100, 100] --> [0, 1]
old_upper = 100
old_lower = -100
new_upper = 1
new_lower = 0
x = ( new_upper - new_lower ) * ( x_init - old_lower ) / ( old_upper - old_lower )
+ new_lower
```

```
# Generate y values
y = 2 * x + noise
```

```
plt.scatter(x, y, s=5)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
# random perm
# split train test

x = x.unsqueeze(1)
y = y.unsqueeze(1)

x = x[rand_perm]
y = y[rand_perm]
xy = torch.cat((x, y), dim=1)
print(xy.shape)

train_data = xy.numpy()[:split_num]
test_data = xy.numpy()[split_num:]
print(len(train_data))

# y=2x
train_label = torch.sign(2 * x - y).numpy().squeeze().tolist()[:split_num]
test_label = torch.sign(2 * x - y).numpy().squeeze().tolist()[split_num:]
```

```
para_str = '-s 1 -t 2 -d 2 -c ' + str(sorted_CVA[0][2]) + ' -g ' +
str(sorted_CVA[0][3]) + ' -w1 1 -w-1 6 -v 5'
svm_model = svmutil.svm_train(train_label, train_data, para_str)
para_str = '-s 1 -t 2 -d 2 -c ' + str(sorted_CVA[0][2]) + ' -g ' +
str(sorted_CVA[0][3]) + ' -w1 1 -w-1 6'
svm_model = svmutil.svm_train(train_label, train_data, para_str)
label, acc, vals = svmutil.svm_predict(test_label, test_data, svm_model)
```