Machine Learning HW1 608410166 楊京樺

書籤:

- **執行說明:**

  在 python 環境中執行 (此作業是在 anaconda3 上完成)

- **作法:**

  運算的部分都寫成 function，再將不同 data 代進去

所有的 Function:

```
regression(x_nd, y_1d)
trainingError(x, y, degree)
leaveOneOut(x, y, degree)
fiveFold(x, y, degree)
testingError(x, y, w, degree)

regularization(x_nd, y_1d, Lambda, dim)
trainingError_14d_regularization(x, y, Lambda)
leaveOneOut_14d_regularization(x, y, Lambda)
fiveFold_14d_regularization(x, y, Lambda)
testingError_14d_regularization(x, y, w, Lambda)

singlePlot(point_x, point_y, range_x, w, lim_x, lim_y)
integrationMap(point_x, point_y, range_x, w_1d, w_5d, w_10d, w_14d, li
m_x, lim_y)
integrationMap_14d(point_x, point_y, range_x, Lambda1, Lambda2, La
mbda3, Lambda4, lim_x, lim_y):
```

有 5 個 有關 regression 的 function, 5 個有關 regularization 的 function, 3 個畫圖的 function，接下來一一的介紹各個 function。

```
1. regression(x_nd, y_1d)
```

```
# get the function weight  W = (X^T X)^-1 X^T y
def regression(x_nd, y_1d):
    return np.dot(np.dot(np.linalg.inv(np.dot(x_nd.T, x_nd)), x_nd.T), y_1d)
```

x_nd: training 的 x 化成的矩陣 (如, 5 維時 x_nd = [x^5, x^4, x^3, x^2, x^1, 1])

y_1d: training 的 y

return regression 後的 Weight

計算 regression 公式的 function，利用這 function 可以得到線性回歸後的權重。

2. `trainingError(x, y, degree)`

```
# testing error
def trainingError(x, y, degree):
    error = 0
    dim = degree+1
    num = y.size  # data number
    x_nd = np.zeros((num, dim))
    for i in range(num):
        for j in range(dim):
            x_nd[i][j] = pow(x[i], j)
    global weight
    weight = regression(x_nd, y)

    # Ein(w) = 1/N * |Xw-y|^2
    for i in range(num):
        error += math.pow((np.dot(x_nd[i].T, weight) - y[i]), 2)
    error /= num
    return error
```

X: training data 的 x

Y: training data 的 y

Degree: 要找的模型空間是多少維度

Return training error E_{in}

計算 training error 的 function，將我們的 training data 和要求的維度代進去後，他會回傳 training error。

這個 function 一進去會先把我們給他的 X 陣列根據我們的維度轉成矩陣

例如, degree=5, 我們會先把 X 轉成[x^5, x^4, x^3, x^2, x^1, 1]。

之後將這個矩陣和我們給的 y 丟進 regression function, 得到 weight

```
weight = regression(x_nd, y)
```

最後再利用這個 weight 得到我們的模型，再利用這個模型將 x 值重新代進去算 y 值，之後將 y 值扣去實際的 y 值找他的誤差，並加總取平均，這就是 function return 的值。

3. leaveOneOut(x, y, degree)

```
# Leave one out
def leaveOneOut(x, y, degree):
    error = 0
    dim = degree+1
    num = y.size
    x_nd = np.zeros((num-1, dim))
    train_y = np.zeros(num-1)

    for i in range(num):
        k = 0
        for j in range(num):
            if i == j:
                test_x = x[j]
                test_y = y[j]
                continue
            else:
                for l in range(dim):
                    x_nd[k][l] = pow(x[j], l)
                train_y[k] = y[j]
                k=k+1
            #end of for loop
        w = regression(x_nd, train_y)
        y_prediction = 0
        for k in range(dim):
            y_prediction += w[k] * pow(test_x, k)
        error += math.pow((y_prediction - test_y), 2)
        #end of for loop
    error /= num
    return error
```

x: training data x

y: training data y

degree: 維度

return Leave-One-Out 的 E_{in}

---

cross-validation errors 的 function 之一，這個 function 會先將我們給的 x 和 y 先抽出一組出來當作 testing data，再將剩下的 data 做 regression 後產出模型，再利用剛抽出來的 testing data 找他的誤差。

有幾組 data 就會做幾次，之後再取平均，就是 function 要 return 的值。

例如，有 15 組 x, y 傳 function，會重複上面動作 15 次(每一組都當過一次 testing data)，再將其誤差加總取平均。

4. `fiveFold(x, y, degree)`

```
# five-fold
def fiveFold(x, y, degree):
    error = 0
    dim = degree + 1
    num = y.size
    groupNum = 5
    testNum = int(num / groupNum)
    trainNum = num - testNum
    test_x = np.zeros(testNum)
    test_y = np.zeros(testNum)
    x_nd = np.zeros((trainNum, dim))
    train_y = np.zeros(trainNum)
    # split in 5 group
    for i in range(groupNum):
        k = 0
        l = 0
        for j in range(num):
            if j % groupNum == i:
                test_x[k] = x[j]
                test_y[k] = y[j]
                k = k + 1
            else:
                for m in range(dim):
                    x_nd[l][m] = pow(x[j], m)
                train_y[l] = y[j]
                l = l + 1
            #end of for loop
        w = regression(x_nd, train_y)
        for k in range(testNum):
            y_prediction = 0
            for l in range(dim):
                y_prediction += w[l] * pow(test_x[k], l)
            error += math.pow((y_prediction - test_y[k]), 2)
        #end of for loop
    error /= num
    return error
```

x: training data x

y: training data y

degree: 維度

return Five-fold 的 E_{in}

這個 function 和上一個 function 一樣是做 cross-validation errors 的 function。

首先會先分成 5 組，其中一組拿來做 testing data，剩下四組 regression 後用先前抽出的 testing data 算他誤差，做五次後取平均(每組都做過一次 testing data)。

因為是分五組，所以每次算誤差加總後是除以 M/5 次(假設 M 我們給 function 的資料數)，而總共 5 次所以會再除以 5，所以這個 function 是直接在最外面除以總資料數。

```
error /= num
```

PS. 分組方式是 x, y 的 index 與 5 取餘數，相同的為一組，

假設有 15 筆 data => group1 = {(x[0], y[0]), (x[5], y[5], x[10], y[10])},

group2 = {(x[1], y[1]), (x[6], y[6], x[11], y[11])}.... 以此類推

5. `testingError(x, y, w, degree)`

```
def testingError(x, y, w, degree):
    num = y.size
    dim = degree + 1
    error = 0
    for i in range(num):
        y_prediction = 0
        for j in range(dim):
            y_prediction += w[j]*pow(x[i], j)
        error += math.pow((y_prediction - y[i]), 2)
    error /= num
    return error
```

x: testing data x

y: testing data y

w: training error 那個 function 算出來的 weight

degree: 維度

return testing error 的 E_{in}

這個 function 不會做 regression，他直接拿我們給的 weight 和我們給的 testing data 來算 E_{in}。

6. `regularization(x_nd, y_1d, Lambda, dim)`

```
# get the function weight W = (X^T X + λI)^-1 X^T y
def regularization(x_nd, y_1d, Lambda, dim):
    return np.dot(np.dot(np.linalg.inv((np.dot(x_nd.T, x_nd) + Lambda*np.eye(dim))), x_nd.T), y_1d)
```

x_nd: training 的 x 化成的矩陣 (如, 5 維時 x_nd = [x^5, x^4, x^3, x^2, x^1, 1])

y_1d: training 的 y

Lamdba: 我們 Regularization 的 **λ**

Dim: x 矩陣的大小(degree+1) => 算 I 時使用

return regularization 後的 Weight

$$\mathbf{w} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y}$$

7. `trainingError_14d_regularization(x, y, Lambda)`

```
def trainingError_14d_regularization(x, y, Lambda):
    error = 0
    dim = 15
    num = y.size  # data number
    x_nd = np.zeros((num, dim))
    for i in range(num):
        for j in range(dim):
            x_nd[i][j] = pow(x[i], j)
    global weight
    weight = regularization(x_nd, y, Lambda, dim)
    # error
    for i in range(num):
        error += math.pow((np.dot(x_nd[i].T, weight) - y[i]), 2) + Lambda * np.dot(weight.T, weight)

    error /= num
    return error
```

x: training data x

y: training data y

Lamdba: 我們 Regularization 的 **λ**

return training error 的 E_{in}

這個 function 做的動作與前面介紹的 trainingError function 差不多，不同在於這邊已經直接訂為 14 維，且是使用 regularization 的方式來求 weight 及算誤差。

E_{in}:

$$J_m(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(\mathbf{x}_i;\mathbf{w}))^2 + \lambda\|\mathbf{w}\|^2$$

Weight:

$$\mathbf{w} = \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

8. leaveOneOut_14d_regularization(x, y, Lambda)

```
def leaveOneOut_14d_regularization(x, y, Lambda):
    error = 0
    dim = 15
    num = y.size
    x_nd = np.zeros((num-1, dim))
    train_y = np.zeros(num-1)

    for i in range(num):
        k = 0
        for j in range(num):
            if i == j:
                test_x = x[j]
                test_y = y[j]
                continue
            else:
                for l in range(dim):
                    x_nd[k][l] = pow(x[j], l)
                train_y[k] = y[j]
                k=k+1
            #end of for loop
        w = regularization(x_nd, train_y, Lambda, dim)
        y_prediction = 0
        for k in range(dim):
            y_prediction += w[k] * pow(test_x, k)
        error += (math.pow((y_prediction - test_y), 2) + Lambda * np.dot(w.T, w))
        #end of for loop
    error /= num
    return error
```

x: training data x

y: training data y

Lamdba: 我們 Regularization 的 **λ**

return leave-one-out 的 E_{in}

與前面的 function 狀況相同，所以只貼 function 就不介紹了。詳細步驟與 LeaveOneOut 的部分差不多。不同的只在於是用 regularization 算 weight 和 E_{in}

9. `fiveFold_14d_regularization(x, y, Lambda)`

```
def fiveFold_14d_regularization(x, y, Lambda):
    error = 0
    dim = 15
    num = y.size
    groupNum = 5
    testNum = int(num / groupNum)
    trainNum = num - testNum
    test_x = np.zeros(testNum)
    test_y = np.zeros(testNum)
    x_nd = np.zeros((trainNum, dim))
    train_y = np.zeros(trainNum)
    # split in 5 group
    for i in range(groupNum):
        k = 0
        l = 0
        for j in range(num):
            if j % groupNum == i:
                test_x[k] = x[j]
                test_y[k] = y[j]
                k = k + 1
            else:
                for m in range(dim):
                    x_nd[l][m] = pow(x[j], m)
                train_y[l] = y[j]
                l = l + 1
            #end of for loop
        w = regularization(x_nd, train_y, Lambda, dim)
        # error
        for k in range(testNum):
            y_prediction = 0
            for l in range(dim):
                y_prediction += w[l] * pow(test_x[k], l)
            error += (math.pow((y_prediction - test_y[k]), 2) + Lambda * np.dot(w.T, w))
        #end of for loop
    error /= num
    return error
```

x: training data x

y: training data y

Lamdba: 我們 Regularization 的 **λ**

return five-fold 的 E_{in}

與前面的 function 狀況相同，所以只貼 function 就不介紹了。詳細步驟與 Fivefold 的部分差不多，分為五組且一樣利用 index 取 5 的餘數相同的一組來分組。

不同的只在於是用 regularization 算 weight 和 E_{in}

10. `testingError_14d_regularization(x, y, w, Lambda)`

```
# testing error (Just only input test data)
def testingError_14d_regularization(x, y, w, Lambda):
    num = y.size
    dim = 15
    error = 0
#     w_norm = 0
#     for i in range(w.size):
#         w_norm += abs(w[i])
    for i in range(num):
        y_prediction = 0
        for j in range(dim):
            y_prediction += w[j]*pow(x[i], j)
        error += (math.pow((y_prediction - y[i]), 2) + Lambda * np.dot(w.T, w))
    error /= num
    return error
```

x: testing data x

y: testing data y

w: training error 那個 function 算出來的 weight

Lambda: Regularization 的 **λ**

return testing error 的 E_{in}

這個 function 不會最 regularization，他直接拿我們給的 weight 和我們給的 testing data 來算 E_{in}。

11. `singlePlot(point_x, point_y, range_x, w, lim_x, lim_y)`

```
def singlePlot(point_x, point_y, range_x, w, lim_x, lim_y):
    dim = w.size
    plt.xlim(lim_x)
    plt.ylim(lim_y)
    plt.title("single plot (degree %d)" %(dim-1))
    plt.scatter(point_x, point_y, c = 'r', alpha=0.6)
    paint_y = 0
    for i in range(dim):
        paint_y += w[i]*(range_x**i)
    plt.plot(range_x, paint_y)
    plt.show()
```

Point_x: 圖上點的 x 座標

Point_y: 圖上點的 y 座標

Range_x: 畫線 x 的範圍且間距為多少 => ex: `np.arange(-3, 3, 0.01)`

w: 利用 x 求 y 的 weight => ex: $y = w[2]*x^2 + w[1]*x^1 + w[0]*x^0$

lim_x: x 軸顯示的範圍

lim_y: y 軸顯示的範圍

畫單圖

12. integrationMap(point_x, point_y, range_x, w_1d, w_5d, w_10d, w_14d, lim_x, lim_y)

```
def integrationMap(point_x, point_y, range_x, w_1d, w_5d, w_10d, w_14d, lim_x, lim_y):
    plt.title("Integration map")
    plt.xlim(lim_x)
    plt.ylim(lim_y)

    plt.scatter(point_x, point_y, c = 'r', alpha=0.6)
    paint_y = w_1d[1]*range_x + w_1d[0]
    plt.plot(range_x, paint_y)

    paint_y = 0
    for i in range(6):
        paint_y += w_5d[i]*(range_x**i)
    plt.plot(range_x, paint_y)

    paint_y = 0
    for i in range(11):
        paint_y += w_10d[i]*(range_x**i)
    plt.plot(range_x, paint_y)

    paint_y = 0
    for i in range(15):
        paint_y += w_14d[i]*(range_x**i)
    plt.plot(range_x, paint_y)
    plt.show()
```

Point_x: 圖上點的 x 座標

Point_y: 圖上點的 y 座標

Range_x:畫線 x 的範圍且間距為多少 => ex: np.arange(-3, 3, 0.01)

W_1d: 利用 x 求 y 的 weight (1 維)

W_5d: 利用 x 求 y 的 weight (5 維)

W_10d: 利用 x 求 y 的 weight (10 維)

W_14d: 利用 x 求 y 的 weight (14 維)

lim_x: x 軸顯示的範圍

lim_y: y 軸顯示的範圍

畫整合圖 (1 維 5 維 10 維 14 維)

13. integrationMap_14d(point_x, point_y, range_x, Lambda1, Lambda2, Lambda3, Lambda4, lim_x, lim_y):

```
def integrationMap_14d(point_x, point_y, range_x, Lambda1, Lambda2, Lambda3, Lambda4, lim_x, lim_y):
    plt.title("Integration map")
    plt.xlim(lim_x)
    plt.ylim(lim_y)

    plt.scatter(point_x, point_y, c = 'r', alpha=0.6)

    paint_y = 0
    for i in range(15):
        paint_y += Lambda1[i]*(range_x**i)
    plt.plot(range_x, paint_y)

    paint_y = 0
    for i in range(15):
        paint_y += Lambda2[i]*(range_x**i)
    plt.plot(range_x, paint_y)

    paint_y = 0
    for i in range(15):
        paint_y += Lambda3[i]*(range_x**i)
    plt.plot(range_x, paint_y)

    paint_y = 0
    for i in range(15):
        paint_y += Lambda4[i]*(range_x**i)
    plt.plot(range_x, paint_y)

    plt.show()
```

Point_x: 圖上點的 x 座標

Point_y: 圖上點的 y 座標

Range_x: 畫線 x 的範圍且間距為多少 => ex: np.arange(-3, 3, 0.01)

Lambda1: 利用 x 求 y 的 weight (Lambda1)

Lambda2: 利用 x 求 y 的 weight (Lambda2)

Lambda3: 利用 x 求 y 的 weight (Lambda3)

Lambda4: 利用 x 求 y 的 weight (Lambda4)

lim_x: x 軸顯示的範圍

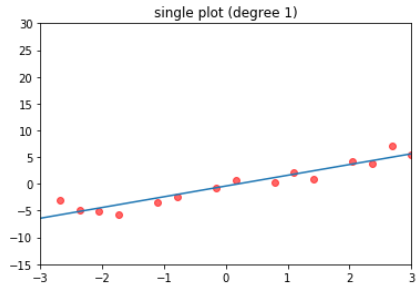
lim_y: y 軸顯示的範圍

畫整合圖 (不同 Lambda 值的圖)

● 結果

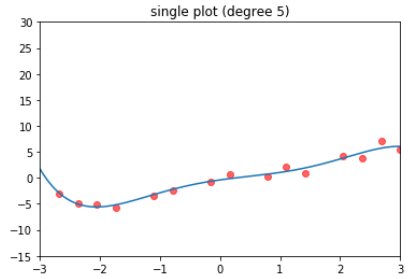
(a)

```
degree 1:
        training error
        Ein =  1.3866913898163131

        cross-validation errors [Leave one out]
        Ein =  2.0427282054046034

        cross-validation errors [five-fold]
        Ein =  2.1304391233561053

        testing error
        Ein =  0.8628309039545659
```

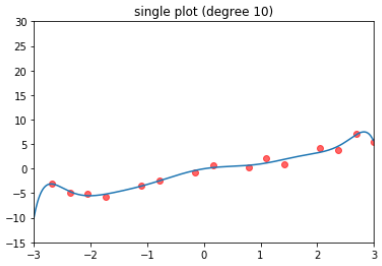


(b)

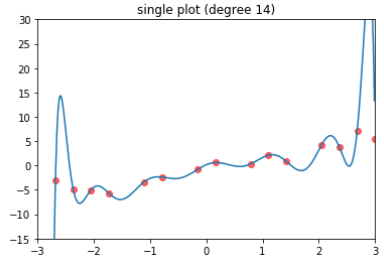
```
degree 5:
        training error
        Ein =  0.48002978426523507

        cross-validation errors [Leave one out]
        Ein =  2.7092139683151566

        cross-validation errors [five-fold]
        Ein =  5.0003041636632055

        testing error
        Ein =  12.040824692018976
```




```
degree 10:
        training error
        Ein =  0.30905398694509595

        cross-validation errors [Leave one out]
        Ein =  311.3479093240394

        cross-validation errors [five-fold]
        Ein =  8956.427181918847

        testing error
        Ein =  4.581530278919752
```



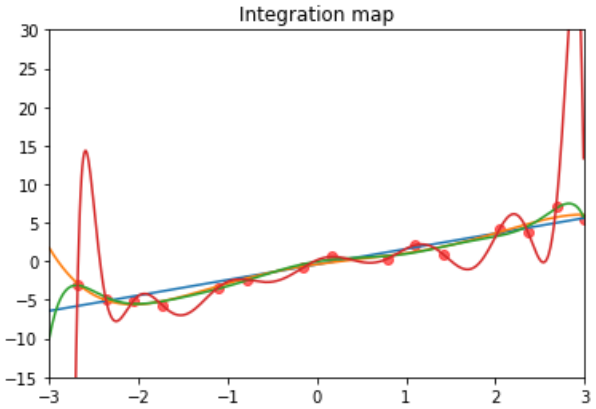

```
degree 14:
        training error
        Ein =  3.184675839652O6e-10

        cross-validation errors [Leave one out]
        Ein =  238719.7191503484

        cross-validation errors [five-fold]
        Ein =  393416.5905571226

        testing error
        Ein =  535270.7766524113
```



(a)(b) 整合後的圖與各個 E_{in}:

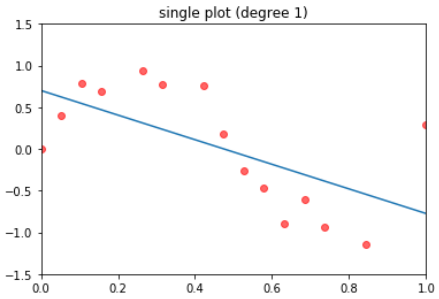
整合
y = 2x + ε:

degree	Training Error	Leave-One-Out	Five-Fold	Testing Error
1	1.386691e+00	2.042728	2.130439	0.862831
5	4.800298e-01	2.709214	5.000304	12.040825
10	3.090540e-01	311.347909	8956.427182	4.581530
14	3.184676e-10	238719.719150	393416.590557	535270.776652

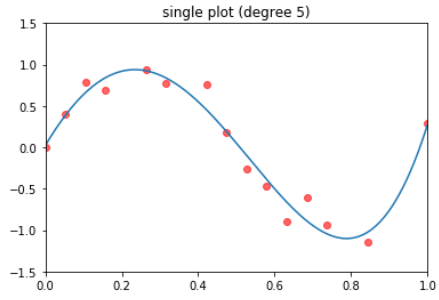


(c)

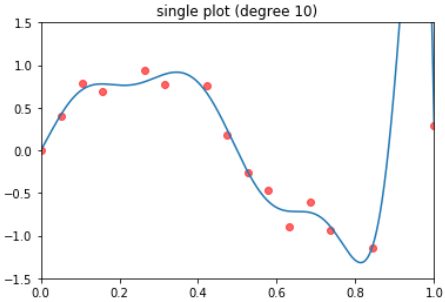
```
Degree = 1:
        training error
        Ein =  0.2855842330957012

        cross-validation errors [Leave one out]
        Ein =  0.4347907164855316

        cross-validation errors [five-fold]
        Ein =  0.4261943489427151

        testing error
        Ein =  0.21481116633169592
```



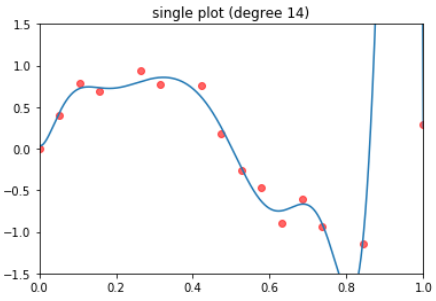

```
Degree = 5:
        training error
        Ein =  0.022749578412012257

        cross-validation errors [Leave one out]
        Ein =  0.45245518035841253

        cross-validation errors [five-fold]
        Ein =  0.31318001820231345

        testing error
        Ein =  0.017149595757925486
```



```
Degree = 10:
        training error
        Ein =  0.008543802283289638

        cross-validation errors [Leave one out]
        Ein =  19655.07221982198

        cross-validation errors [five-fold]
        Ein =  7727.102656358748

        testing error
        Ein =  3.27638855143219
```



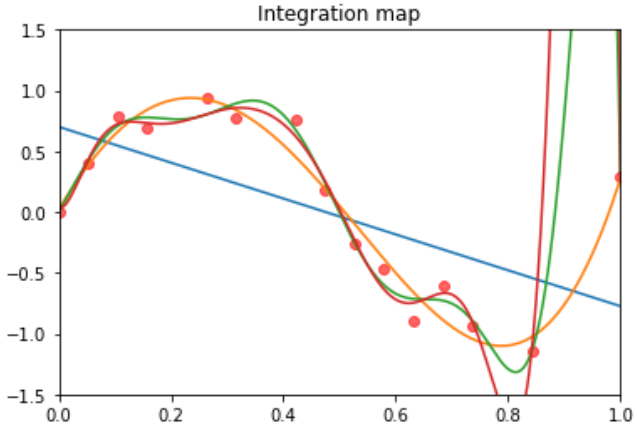

```
Degree = 14:
        training error
        Ein =  0.008722371661576267

        cross-validation errors [Leave one out]
        Ein =  61334525.39174279

        cross-validation errors [five-fold]
        Ein =  35554240.49534663

        testing error
        Ein =  47.84411899986448
```



整合的圖與 E_{in}:

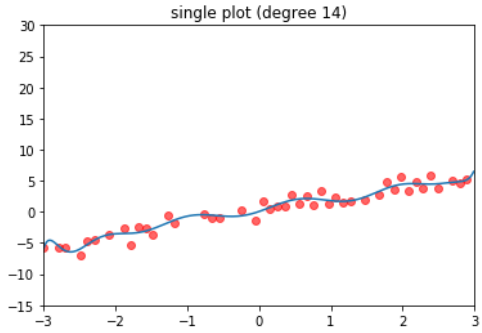
整合
$y = \sin(2\pi x) + \varepsilon$:

degree	Training Error	Leave-One-Out	Five-Fold	Testing Error
1	0.285584	4.347907e-01	4.261943e-01	0.214811
5	0.022750	4.524552e-01	3.131800e-01	0.017150
10	0.008544	1.965507e+04	7.727103e+03	3.276389
14	0.008722	6.133453e+07	3.555424e+07	47.844119

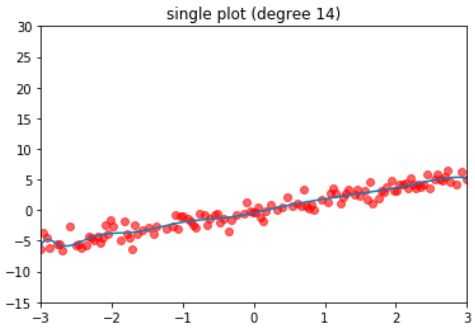


(d)

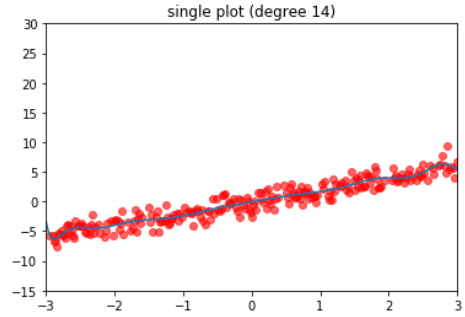
```
y = 2x + ε
degree = 14
m = 60:
        training error
        Ein =  0.6342411012999404

        cross-validation errors [Leave one out]
        Ein =  8.08143303088541

        cross-validation errors [five-fold]
        Ein =  4.2317957672072

        testing error
        Ein =  1.612615717384139
```




```
y = 2x + ε
degree = 14
m = 160:
        training error
        Ein =  0.9266215419636364

        cross-validation errors [Leave one out]
        Ein =  1.3493753366559922

        cross-validation errors [five-fold]
        Ein =  1.4506811309874483

        testing error
        Ein =  1.235031839377005
```




```
y = 2x + ε
degree = 14
m = 320:
        training error
        Ein =  1.1164959090931166

        cross-validation errors [Leave one out]
        Ein =  1.3023577450137498

        cross-validation errors [five-fold]
        Ein =  1.3059351941119934

        testing error
        Ein =  1.086129304456182
```



整理的 Five-Fold 和 Testing Error 的 E_{in}:

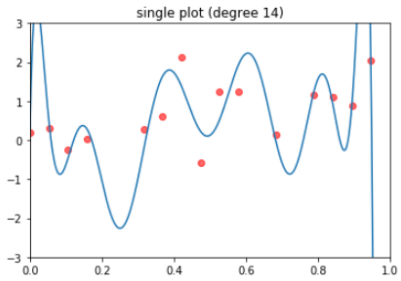
整合

```
y = 2x + ε (m = 60, m = 160, m = 320):
```

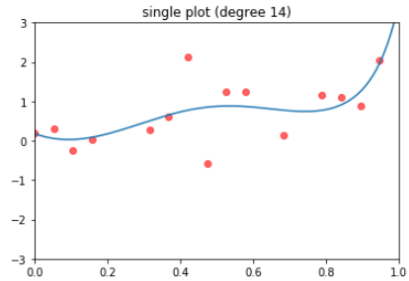
m	Five-Fold	Testing Error
60	4.231796	1.612616
160	1.450681	1.235032
320	1.305935	1.086129

(e)

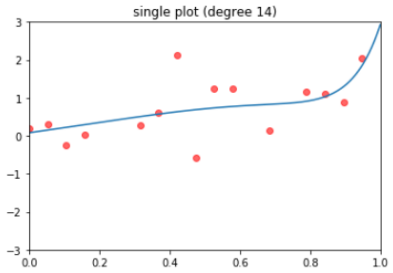
```
Lambda = 0

        training error
        Ein =  0.25121674041482744

        cross-validation errors [leave one out]
        Ein =  6221.2383428876465

        cross-validation errors [five-fold]
        Ein =  72753.8502270516

        testing error
        Ein =  0.2512167404533986
```



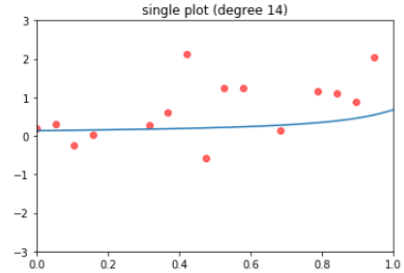

```
Lambda = 0.001/m

        training error
        Ein =  0.41235638924177087

        cross-validation errors [leave one out]
        Ein =  1.4726688386786786

        cross-validation errors [five-fold]
        Ein =  1.6661109924203121

        testing error
        Ein =  0.41235638924177087
```




```
Lambda = 1/m

        training error
        Ein =  0.5485837159232257

        cross-validation errors [leave one out]
        Ein =  0.7718614404478981

        cross-validation errors [five-fold]
        Ein =  0.7040501403777093

        testing error
        Ein =  0.5485837159232259
```



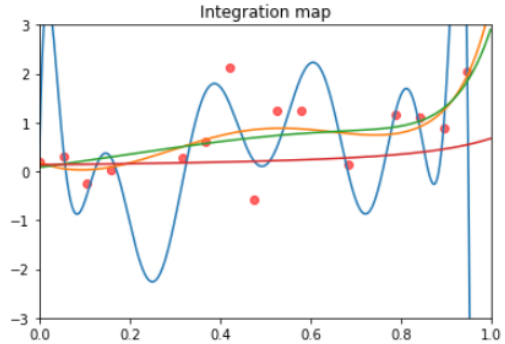

```
Lambda = 1000/m

        training error
        Ein =  2.9698190806429117

        cross-validation errors [leave one out]
        Ein =  2.848038833319791

        cross-validation errors [five-fold]
        Ein =  2.4696189089302867

        testing error
        Ein =  2.9698190806429117
```



整理各個用 Lambda 做 regularization 的 E_{in}:

整合
y = 2x + ε (regularization), m = 20:

Lambda (λ)	Training Error	Leave-One-Out	Five-Fold	Testing Error
0	0.251217	6221.238343	72753.850227	0.251217
0.001/m	0.412356	1.472669	1.666111	0.412356
1/m	0.548584	0.771861	0.704050	0.548584
1000/m	2.969819	2.848039	2.469619	2.969819



● 討　　論

從這幾題中可以觀察出

1. 當維度越大時，在做 testing error 越容易產生極大的誤差，原因是在於當在高維度且要符合每個 training 的 x, y 值時，很有可會使模型產生極大的弧度，而造成 overfitting。
2. 我們可以利用 cross-validation errors 評估出是否會發生 overfitting，從上面幾個圖形可以觀察出當我們的圖越曲折時，cross-validation errors 的值基本上都是偏大。
3. 當資料的樣本數越多時，模型預測會越準確。
   (在題型(d)的 testing error 可以看出，在 m=320 的 testing error 是最小的)
4. Regularization 和 regression 可以看出是有差別的，在題型(e) lambda=0 時基本上就是 regression。在 lambda=0 時的圖也是最為複雜的，而做了 regularization 的圖則是較為平緩。所以可以看出 regularization 可以一定程度的避免 overfitting 的發生，但如果 regularization 力度太大則會照成整個模型的誤差極大。
5. 在題型(e) 比較不同 lambda 值所產出的模型可以發現，當 lambda 值越大時圖是越平緩的，所以說當 lambda 值越大時 regularization 的力度越大，圖會變得越平緩但誤差也會跟著變大。