Machine Learning HW1 608410166 楊京樺

書籤:

* [**執行說明**](#執行說明)
* [**作 法**](#作法)
* [**結 果**](#結果)
* [**討 論**](#討論)
* **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)

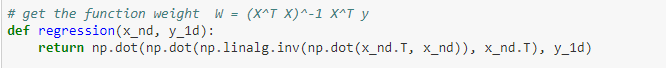
singlePlot(point\_x, point\_y, range\_x, w, lim\_x, lim\_y)

integrationMap(point\_x, point\_y, range\_x, w\_1d, w\_5d, w\_10d, w\_14d, lim\_x, lim\_y)

integrationMap\_14d(point\_x, point\_y, range\_x, Lambda1, Lambda2, Lambda3, 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)



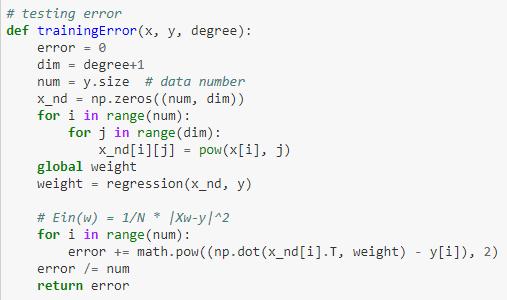
x\_nd: training的 x化成的矩陣 (如, 5維時 x\_nd = [x5, x4, x3, x2, x1, 1] )

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)



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 Ein

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

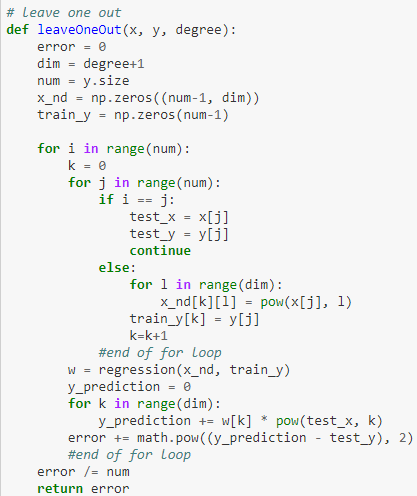
例如, degree=5, 我們會先把X轉成[x5, x4, x3, x2, x1, 1]。

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)



x: training data x

y: training data y

degree: 維度

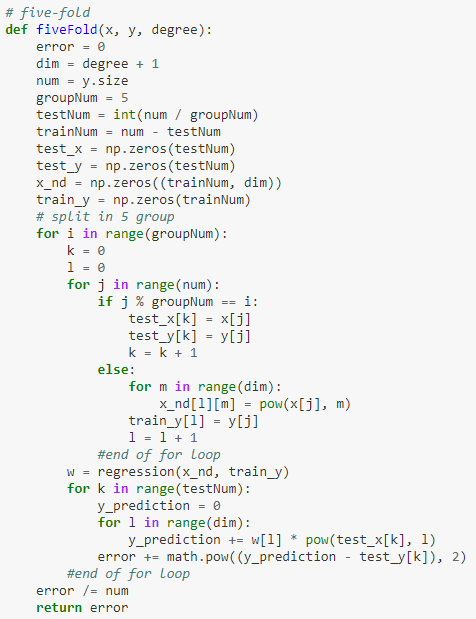
return Leave-One-Out的Ein

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)



x: training data x

y: training data y

degree: 維度

return Five-fold的Ein

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)。

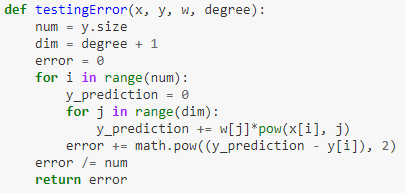
因為是分五組，所以每次算誤差加總後是除以M/5次(假設M我們給function的資料數)，而總共5次所以會再除以5，所以這個function是直接在最外面除以總資料數。 

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)



x: testing data x

y: testing data y

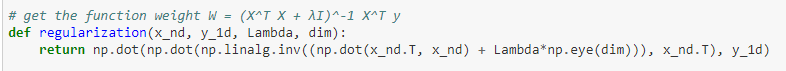
w: training error那個function算出來的weight

degree: 維度

return testing error的Ein

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Ein。

6. regularization(x\_nd, y\_1d, Lambda, dim)



x\_nd: training的 x化成的矩陣 (如, 5維時 x\_nd = [x5, x4, x3, x2, x1, 1] )

y\_1d: training的y

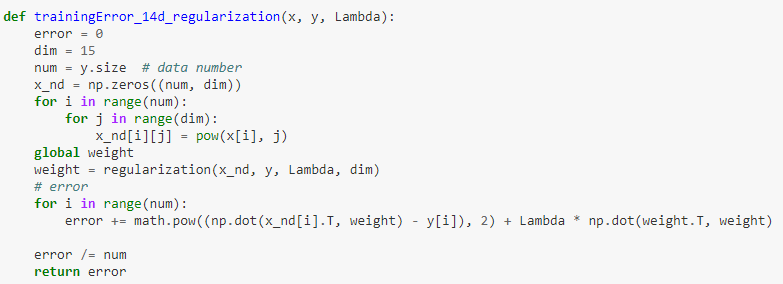
Lamdba: 我們Regularization的**λ**

Dim: x矩陣的大小(degree+1) => 算I時使用

return regularization後的Weight



7. trainingError\_14d\_regularization(x, y, Lambda)



x: training data x

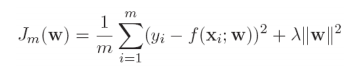
y: training data y

Lamdba: 我們Regularization的**λ**

return training error的Ein

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。

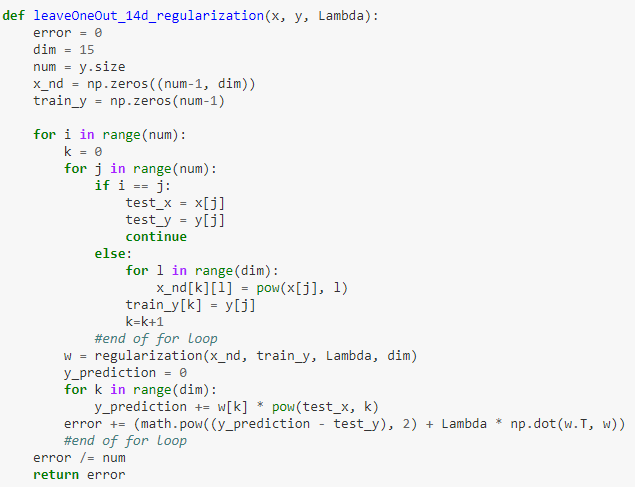
Ein:



Weight:



8. leaveOneOut\_14d\_regularization(x, y, Lambda)



x: training data x

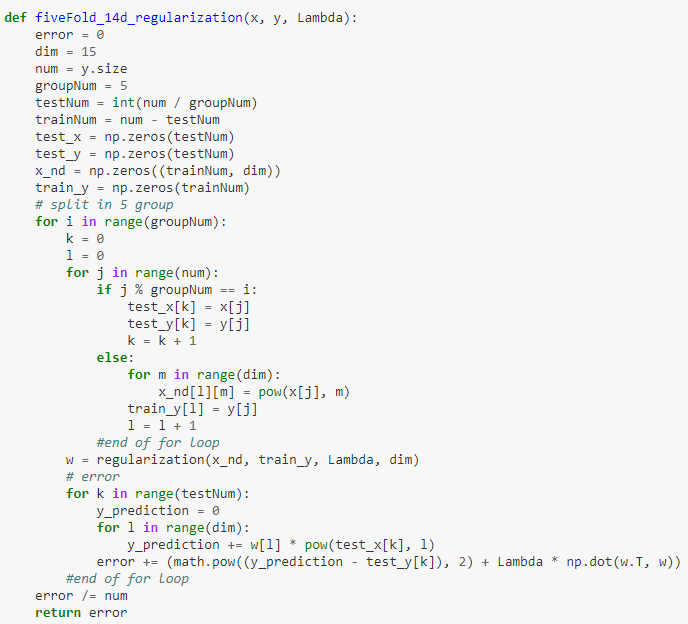
y: training data y

Lamdba: 我們Regularization的**λ**

return leave-one-out的Ein

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Ein

9. fiveFold\_14d\_regularization(x, y, Lambda)



x: training data x

y: training data y

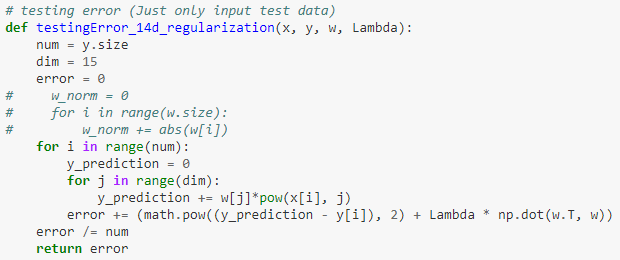
Lamdba: 我們Regularization的**λ**

return five-fold的Ein

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Ein

10. testingError\_14d\_regularization(x, y, w, Lambda)



x: testing data x

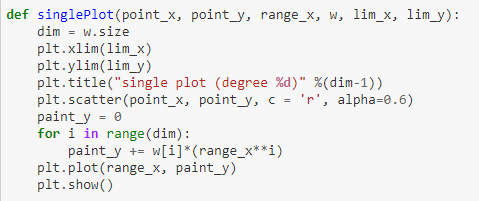
y: testing data y

w: training error那個function算出來的weight

Lambda: Regularization的**λ**

return testing error的Ein

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Ein。

11. singlePlot(point\_x, point\_y, range\_x, w, lim\_x, lim\_y)

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)

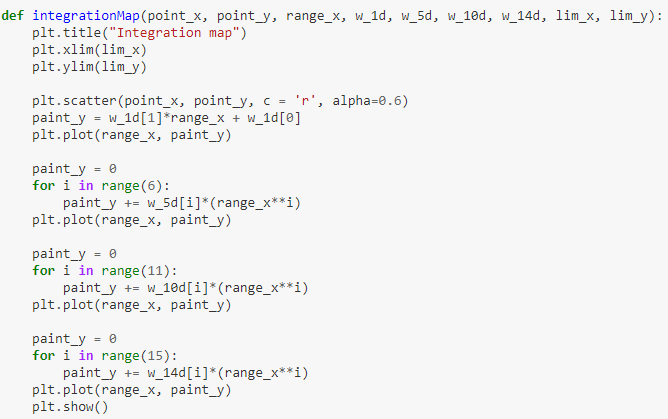
w: 利用x求y的weight => ex: y = w[2]\*x2 + w[1]\*x1+w[0]\*x0

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)



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)

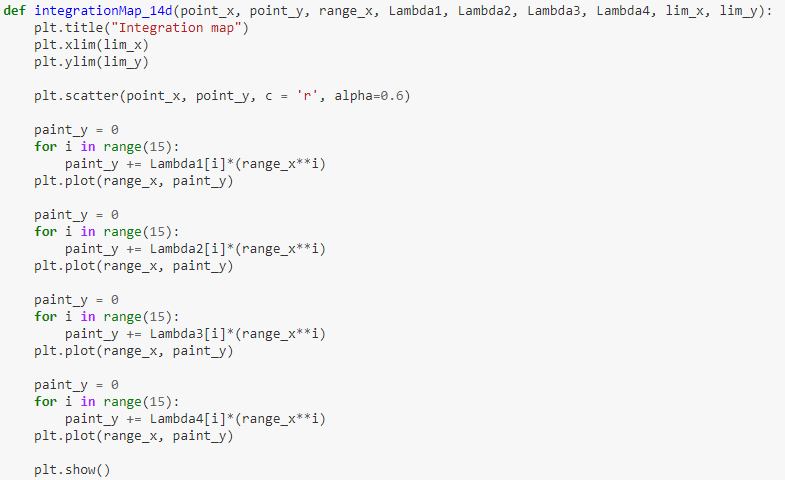
W\_14d: 利用x求y的weight (14維)

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):



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)

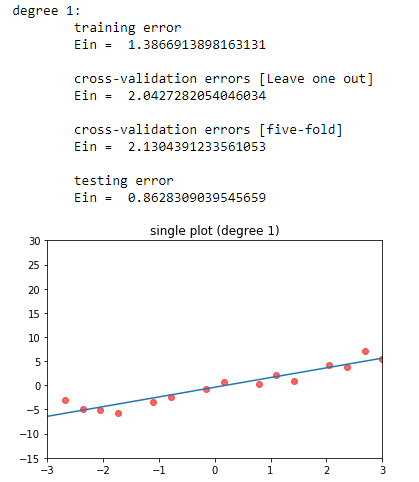
lim\_x: x軸顯示的範圍

lim\_y: y軸顯示的範圍

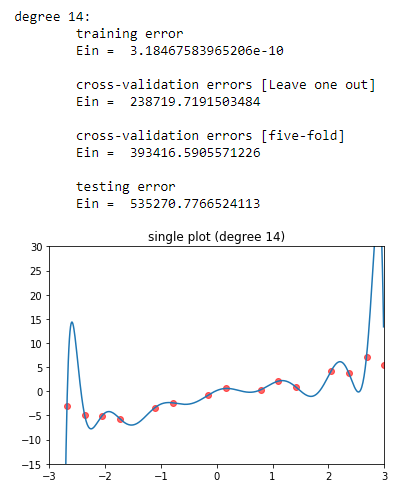
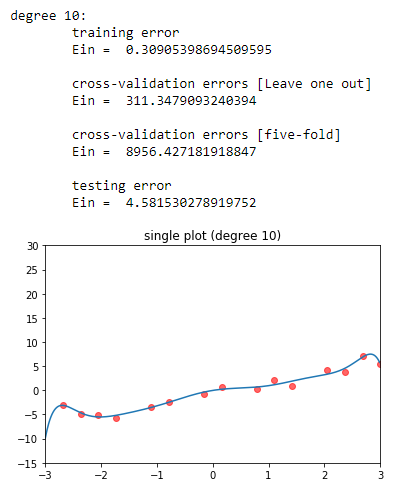
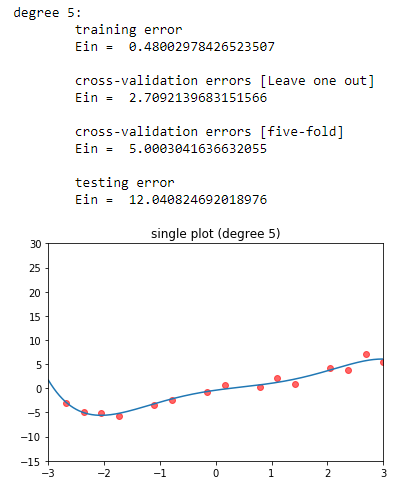
畫整合圖 (不同Lambda值的圖)

* **結果**

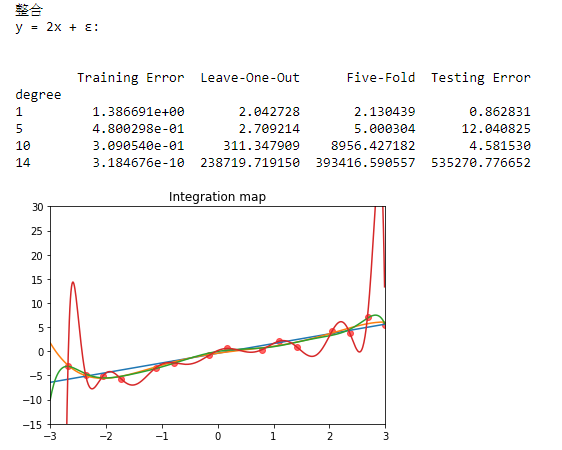
(a)



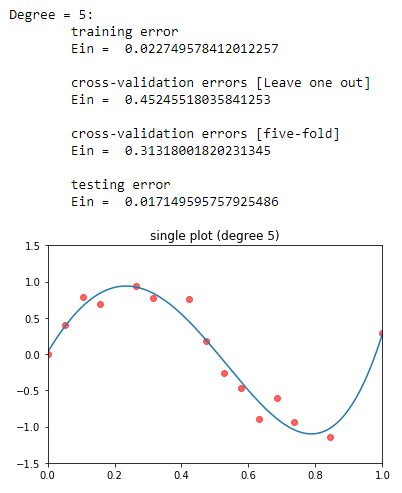
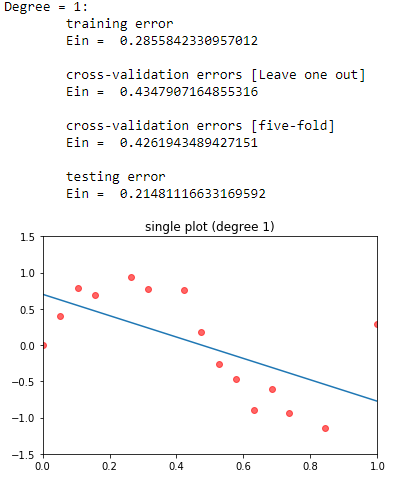
(b)

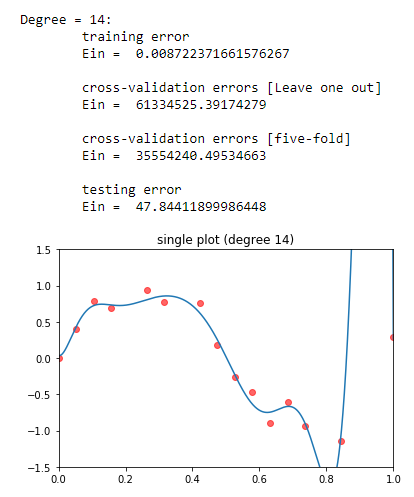
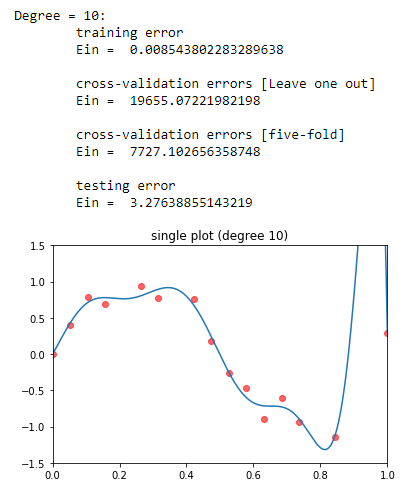


(a)(b) 整合後的圖與各個Ein:

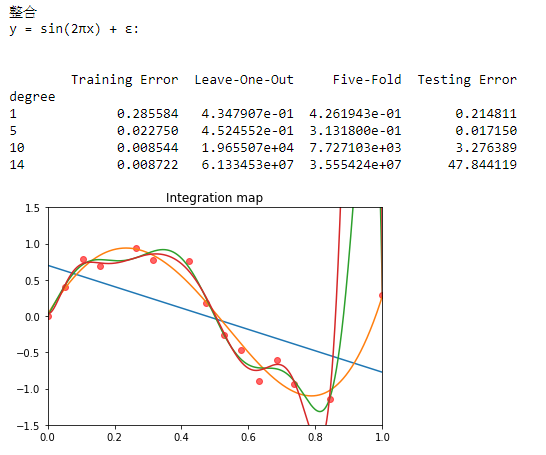


(c)

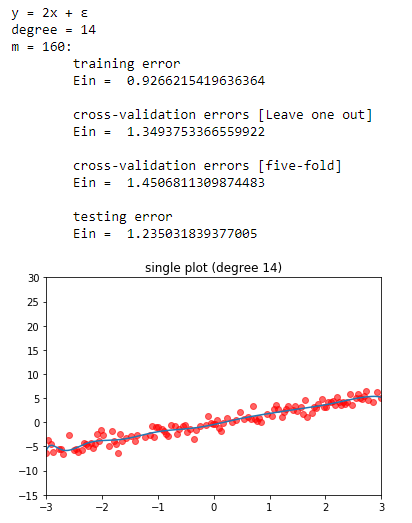
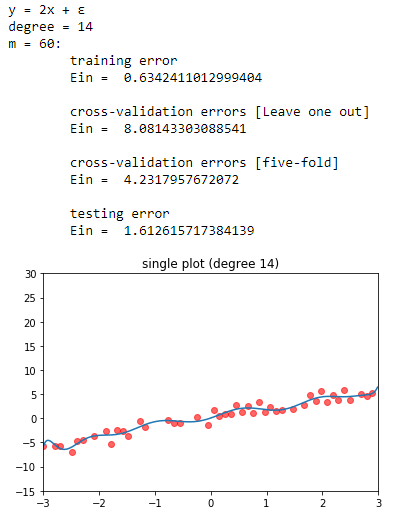


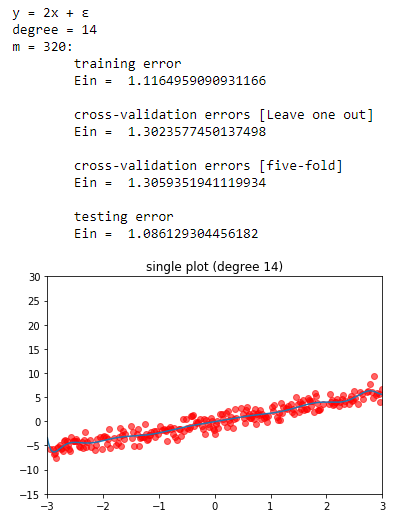


整合的圖與Ein:

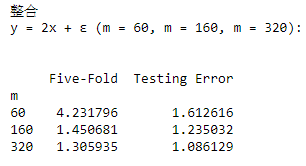


(d)

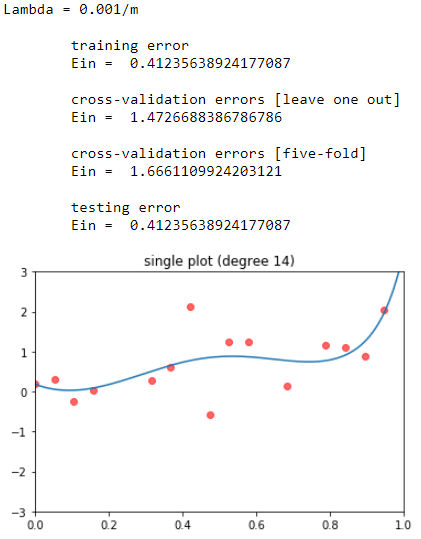
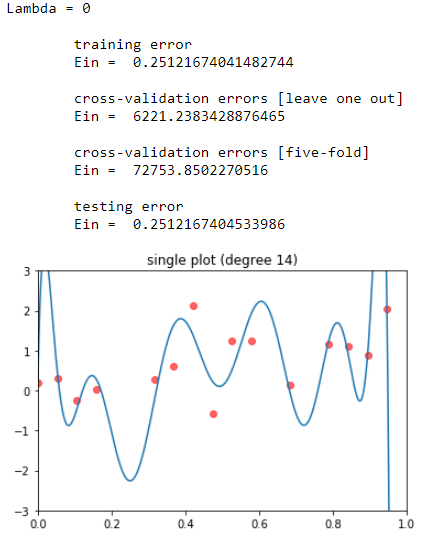


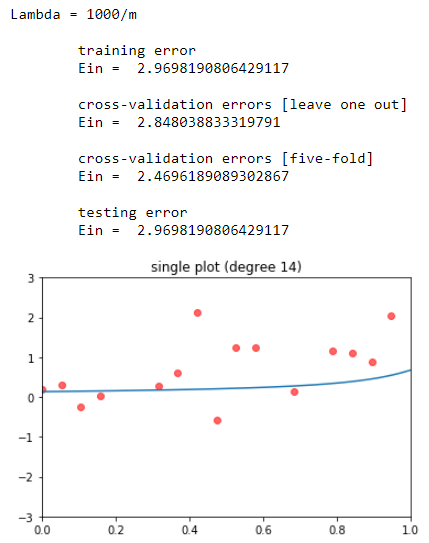
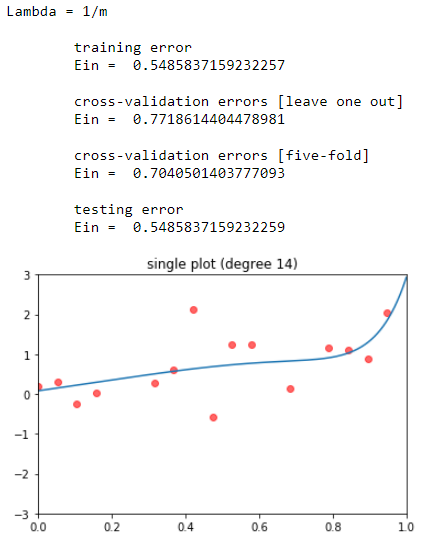


整理的Five-Fold 和Testing Error的Ein:

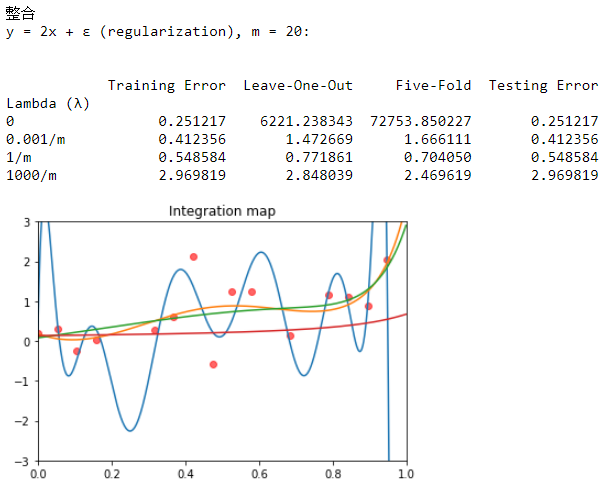


(e)





整理各個用Lambda做regularization的Ein:



* **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)

1. 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。
2. 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。