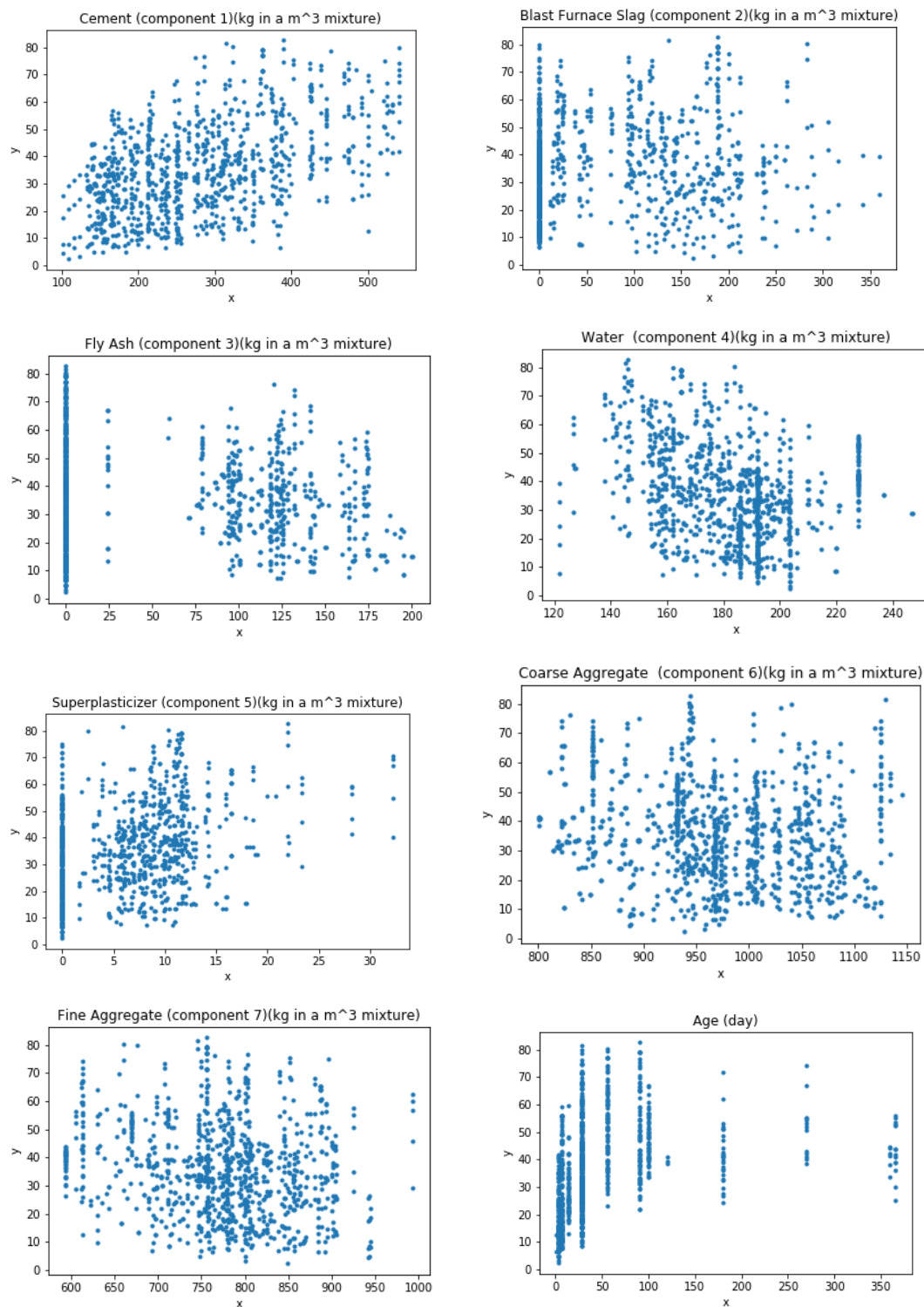


Introduction to Machine Learning
Program Assignment #3
Regression Problem

Team 32

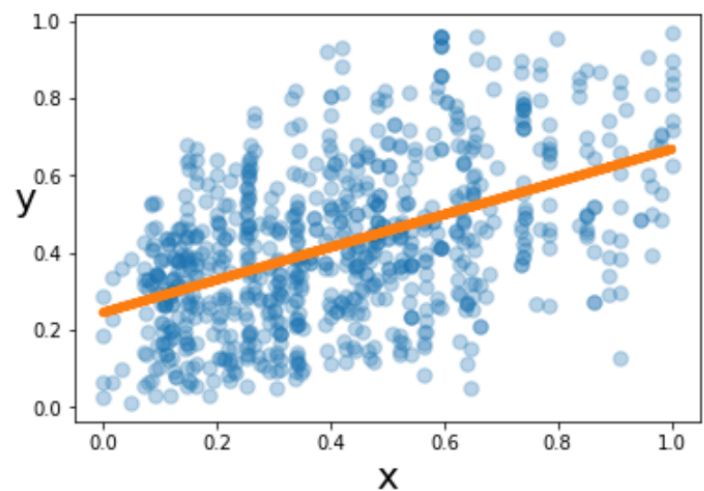
0416009 洪冠群
0416090 王洧晟
0416091 林彥岑
0416098 王于哲
0513406 陳凱文

1. What environments the members are using
Python3.6 , Anaconda: Spyder and Jupyter
2. Visualization of all the features with the target



3. Problem 1 - Linear regression with single variable by built-in function

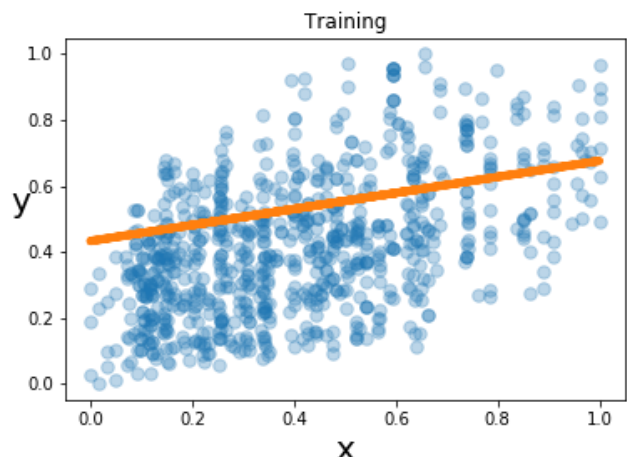
```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5
6 df = pd.read_csv("./Concrete_Data.csv")
7
8 """
9 MSE = sum of [(y - x*w)**2] / length
10 """
11 def MSE(X,coef,intercept,Y):
12     return np.sum((Y-(X*coef+intercept))**2)/X.shape[0]
13
14 """
15 error: MSE
16 shape: the number of data
17 R2 = 1 - SSE/SST0
18     = 1 - MSE*shape/ sum of [(y-y.mean)**2]
19 """
20 def R2(error,shape,y):
21     SST0 = np.sum((y - np.mean(y))**2)
22     return 1 - error*shape/SST0
23
24 y = df.iloc[:,8].values
25 column_name = list(df.columns.values)
26
27 for i in range(8):
28     x = df.iloc[:,i].values
29     plt.scatter(x, y,marker='.')
30     plt.xlabel('x')
31     plt.ylabel('y')
32     plt.title(column_name[i])
33     plt.savefig(column_name[i])
34     plt.show()
35
36 # use your eye to select x0
37 x = df.iloc[:,0].values
38 y = df.iloc[:,8].values # get target
39 x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
40 y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
41
42 x_train, x_test, y_train, y_test = train_test_split( x , y, test_size = 0.2)
43
44 from sklearn.linear_model import LinearRegression
45 model=LinearRegression()
46 model.fit(x_train[:,np.newaxis],y_train)
47 error = MSE(x_train,model.coef_,model.intercept_,y_train)
48 R_squared = R2(error, x_train.shape[0], y_train)
49 print(model.coef_,model.intercept_)
50 print("Training\nerror:%.5f , R2: %.5f" % (error,R_squared))
51 error = MSE(x_test,model.coef_,model.intercept_,y_test)
52 R_squared = R2(error, x_test.shape[0], y_test)
53 print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))
54
55 plt.plot(x_train,y_train, '.',markersize=15,alpha=0.3)
56 plt.plot(x_train,model.intercept_+model.coef_*x_train,linewidth=5)
57 plt.xlabel('x',fontsize=20)
58 plt.ylabel('y',fontsize=20,rotation=0)
59 plt.savefig('scikit_learn')
60
61
```



weight: [0.42204927] ,
bias: 0.2453236364077083
Training
error:0.03259 , R2: 0.23426
Testing
error:0.03242 , R2: 0.29558

4. Problem 2 - Linear regression with single variable by your own gradient descent

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5
6 df = pd.read_csv("./Concrete_Data.csv")
7
8 x = df.iloc[:,0].values # get x0 variable
9 y = df.iloc[:,1].values # get target
10 ones = np.ones((x.shape[0],1)) # constant
11 # normalization
12 # x = (x - min) / (max - min)
13 x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
14 y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
15 # change dimension : 1D -> 2D
16 x = x[:,np.newaxis]
17 y = y[:,np.newaxis]
18 x_process = np.concatenate((ones,x), axis = 1 )
19 # separate training and testing set
20 x_train, x_test, y_train, y_test = train_test_split( x_process,
21                                                    y,
22                                                    test_size = 0.2)
23
24 MSE = sum of [(y - x*w)**2] / length
25
26 def MSE(X,W,Y):
27     return np.sum(np.dot((np.dot(X,W)-Y).T,
28                           (np.dot(X,W)-Y)) / (2*X.shape[0]))
29
30
31 error: MSE
32 shape: the number of data
33 R2 = 1 - SSE/SST0
34     = 1 - MSE*shape/ sum of [(y-y.mean)**2]
35
36 def R2(error,shape,y):
37     SST0 = np.sum((y - np.mean(y))**2)
38     return 1 - error*shape/SST0
39
40 lr = 0.5 # learning rate
41 iterations = 1000 # max loop size
42 iteration = 0 # loop number now
43 epsilon = 0.005 # minimum of error
44 w = np.random.randn(x_train.shape[1],1) # initial weight
45
46 # Training
47 while iteration < iterations:
48     iteration += 1
49     for var in range(x_train.shape[1]): # do GD to each variable
50         gradient = np.dot((y_train - np.dot( x_train, w )).T,
51                           x_train[:,var])/x_train.shape[0]
52         w[var,0] = w[var,0] + lr*gradient
53     error = MSE( x_train, w, y_train)
54     R_squared = R2(error, x_train.shape[0], y_train)
55     print("Training %d iteration, error:%.5f , R2: %.5f"
56           % (iteration,error,R_squared))
57     if error < epsilon: # the error is small enough, break
58         break
59
60 # Testing
61 error = MSE( x_test, w, y_test)
62 R_squared = R2(error, x_test.shape[0], y_test)
63 print("Testing\error:%.5f , R2: %.5f" % (error,R_squared))
64 print("weight: ",w[0][0],",\nbias: ",w[1][0])
65
66 plt.plot(x_train[:,1].flatten(),y_train, '.',markersize=15,alpha=0.3)
67 plt.plot(x_train[:,1].flatten(),
68          x_train[:,1].flatten()*w[0][0]+w[1][0],linewidth=5)
69 plt.xlabel('x',fontsize=20)
70 plt.ylabel('y',fontsize=20,rotation=0)
71 plt.title('Training')
72 plt.savefig('GD')
```



Training 300 iteration, error:0.01589 , R2: 0.62366
Testing
error:0.01782 , R2: 0.62433
weight: 0.24327570151031047 ,
bias: 0.4326010997543369

5. Compare P1 & P2

Show what you got.

比較使用內建函式或用 gradient descent 這兩種方式以後，我們了解了：

1. Gradient descent 需要重複性的去做更新 weight 的動作，較耗時。
2. 在這次實驗中發現 Gradient descent 建構的 model 可以獲得較好的準確率。

6. Linear regression with multi-variable by your own gradient descent

Training 1000 iteration, error:0.00809 , R2: 0.80898

Testing

error:0.00999 , R2: 0.78686

```
1
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from sklearn.model_selection import train_test_split
6
7 df = pd.read_csv("./Concrete_Data.csv")
8
9 x = df.iloc[:,0:8].values # get all variable
10 y = df.iloc[:,8].values # get target
11 ones = np.ones((x.shape[0],1)) # constant
12 # normalization
13 # x = (x - min) / (max - min)
14 x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
15 y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
16 # change y's dimension : 1D -> 2D
17 y = y[:,np.newaxis]
18 x_process = np.concatenate((ones,x), axis = 1 )
19 # separate training and testing set
20 x_train, x_test, y_train, y_test = train_test_split( x_process,
21                                                    y,
22                                                    test_size = 0.2)
23
24
25 """
26 MSE = sum of [(y - x*w)**2] / length
27 """
28 def MSE(X,W,Y):
29     return np.sum(np.dot((np.dot(X,W)-Y).T,
30                           (np.dot(X,W)-Y)) / (2*X.shape[0]))
31
32 """
33 error: MSE
34 shape: the number of data
35 R2 = 1 - SSE/SST0
36     = 1 - MSE*shape/ sum of [(y-y.mean)**2]
37 """
38 def R2(error,shape,y):
39     SST0 = np.sum((y - np.mean(y))**2)
40     return 1 - error*shape/SST0
41
42
43 lr = 0.5 # learning rate
44 iterations = 1000 # max loop size
45 iteration = 0 # loop number now
46 epison = 0.005 # minimum of error
47 w = np.random.randn(x_train.shape[1],1) # initial weight
48
49 # Training
50 while iteration < iterations:
51     iteration += 1
52     for var in range(x_train.shape[1]): # do GD to each variable
53         gradient = np.dot((y_train - np.dot( x_train, w )).T,
54                           x_train[:,var])/x_train.shape[0]
55         w[var,0] = w[var,0] + lr*gradient
56     error = MSE( x_train, w, y_train)
57     R_squared = R2(error, x_train.shape[0], y_train)
58     print("Training %d iteration, error:%.5f , R2: %.5f"
59           % (iteration,error,R_squared))
60     if error < epison: # the error is small enough, break
61         break
62
63 # Testing
64 error = MSE( x_test, w, y_test)
65 R_squared = R2(error, x_test.shape[0], y_test)
66 print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))
67
68
```

7. Polynomial regression by your own gradient descent

Training 1000 iteration, error:0.00425 , R2: 0.89715

Testing

error:0.00640 , R2: 0.87399

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5
6
7 df = pd.read_csv("./Concrete_Data.csv")
8
9
10 x = df.iloc[:,0:8].values # get all variable
11 y = df.iloc[:,8].values # get target
12 ones = np.ones((x.shape[0],1)) # constant
13 # normalization
14 # x = (x - min) / (max - min)
15 x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
16 y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
17 # change y's dimension : 1D -> 2D
18 y = y[:,np.newaxis]
19 # get the square of variable
20 x_square = x**2
21 # get the multiplication of variable
22 x_multi = [np.multiply(x[:,i],x[:,j])
23            for i in range(0,x.shape[1]-1)
24            for j in range(i+1,x.shape[1])]
25 # concatenate all
26 x_process = np.concatenate((ones,x,x_square,
27                             np.transpose(x_multi)),
28                             axis = 1 )
29 # separate training and testing set
30 x_train, x_test, y_train, y_test = train_test_split( x_process,
31                                                     y,
32                                                     test_size = 0.2)
33
34
35 """
36 MSE = sum of [(y - x*w)**2] / length
37 """
38 def MSE(X,W,Y):
39     return np.sum(np.dot((np.dot(X,W)-Y).T,
40                           (np.dot(X,W)-Y)) / (2*X.shape[0]))
41
42
43 """
44 error: MSE
45 shape: the number of data
46 R2 = 1 - SSE/SST0
47     = 1 - MSE*shape/ sum of [(y-y.mean)**2]
48 """
49 def R2(error,shape,y):
50     SST0 = np.sum((y - np.mean(y))**2)
51     return 1 - error*shape/SST0
52
53 lr = 5 # learning rate
54 iterations = 1000 # max loop size
55 iteration = 0 # loop number now
56 epison = 0.005 # minimum of error
57 w = np.random.randn(x_train.shape[1],1) # initial weight
58
59 # Training
60 while iteration < iterations:
61     iteration += 1
62     for var in range(x_train.shape[1]): # do GD to each variable
63         gradient = np.dot((y_train - np.dot( x_train, w )).T,
64                           x_train[:,var])/x_train.shape[0]
65         w[var,0] = w[var,0] + lr*gradient
66     error = MSE( x_train, w, y_train)
67     R_squared = R2(error, x_train.shape[0], y_train)
68     print("Traing %d iteration, error:%.5f , R2: %.5f"
69           % (iteration,error,R_squared))
70     if error < epison: # the error is small enough, break
71         break
72
73 # Testing
74 error = MSE( x_test, w, y_test)
75 R_squared = R2(error, x_test.shape[0], y_test)
76 print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))
77
```

8. Answer the question(5%)

一. What is overfitting?

Overfitting就是指training時model的performance很好，但使用在testing set上時的效果不好，表示model對training set的資料過度擬合了，而非學習到廣泛適用的特性。

二. Stochastic gradient descent is also a kind of gradient descent, what is the benefit of using SGD?

可以有機會跳出local minimum而進到另一個local minimum，最後得到global minimum

三. Why the different initial value to GD model may cause different result?

因為 GD model的結果是local minimum而不是global minimum，因此不同的initial value可能會導向不同的local minimum，也就是不同的result

四. What is the bad learning rate? What problem will happen if we use it?

Bad learning rate是指learning rate太大或太小導致model無法收斂或收斂速度很慢。

五. After finishing this homework, what have you learned, what problems you encountered, and how the problems were solved?

經過這次作業我們學到了如何使用 Linear Regression，更熟悉了 Scikit learn 這個套件。了解到怎麼對 data 做 normalization，除此之外也實作了 Gradient Descent，因此對 MSE, R2 這些 Machine Learning 的符號定義有了充分的掌握。也因為這四個小題，發現從 Linear regression 發展到 Polynomial regression 其實只是 data 的次方數以及對應 weight 這個矩陣 column 數的差異，如果從第四個小題倒過來寫就會輕鬆很多。

我們遇到的第一個問題是 R2 的算法，從教授的投影片裡面比較難看出他推導出來的脈絡，後來上網找了一下才發現

$$R2 = 1 - \text{SSE} / \text{SSTO} = 1 - \text{MSE} * \text{shape} / \text{sum of } [(y - y.\text{mean})^2]$$

還有遇到一個問題就是 learning rate 的參數設定，但這個部分就是 try and error。

9. Bonus

我們的 Problem 4 就已經達到 Bonus 的目標了

```
Training 1000 iteration, error:0.00425 , R2: 0.89715
Testing
error:0.00640 , R2: 0.87399
```
