Introduction to Machine Learning

Program Assignment #3

Regression Problem

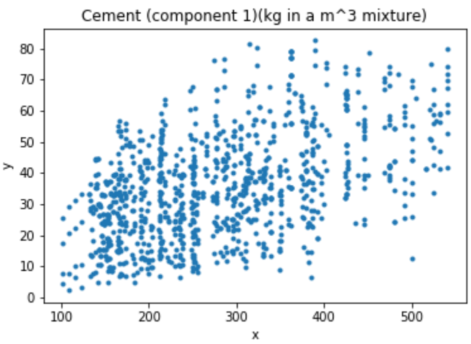
Team 32

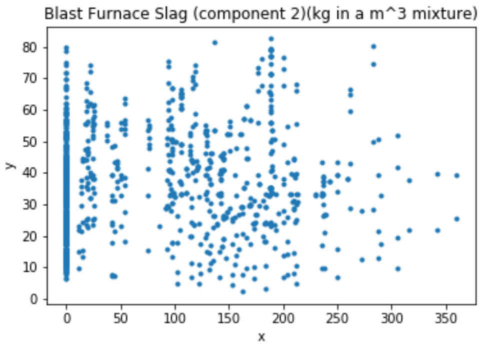
0416009 洪冠群
0416090 王湣晟
0416091 林彥岑
0416098 王于哲
0513406 陳凱文

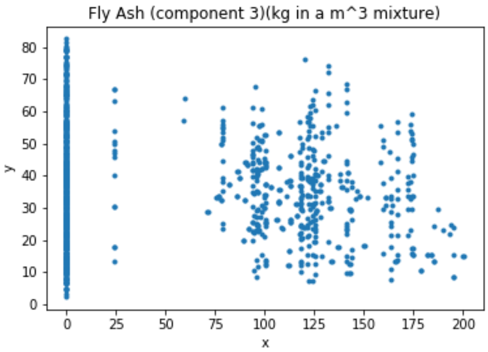
1. What environments the members are using

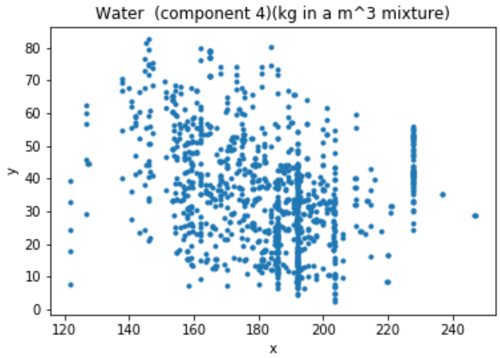
Python3.6 , Anaconda: Spyder and Jupyter

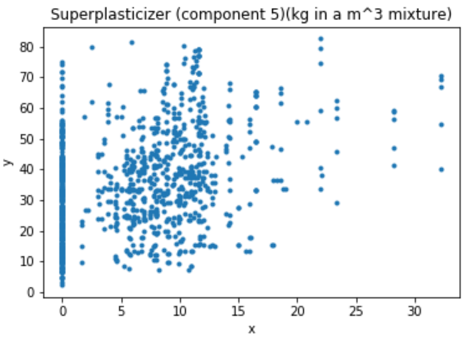
2. Visualization of all the features with the target

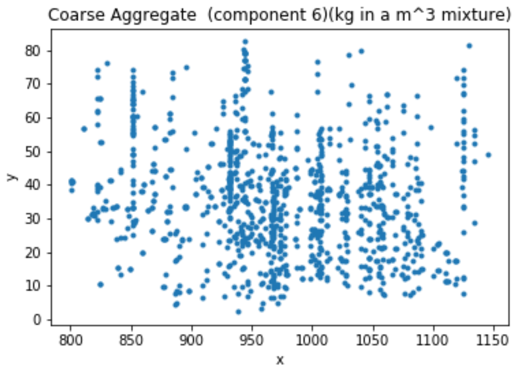


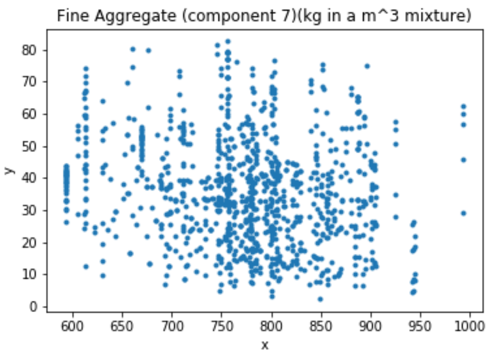


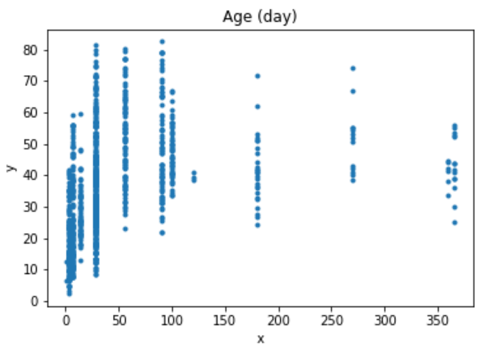






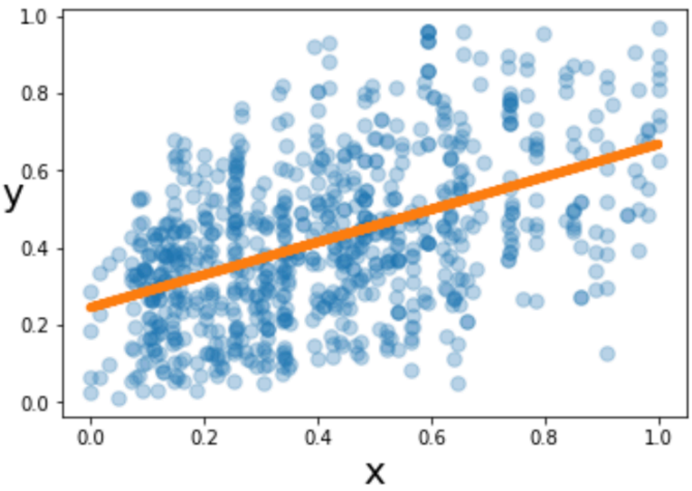






3. Problem 1 - Linear regression with single variable by built-in function

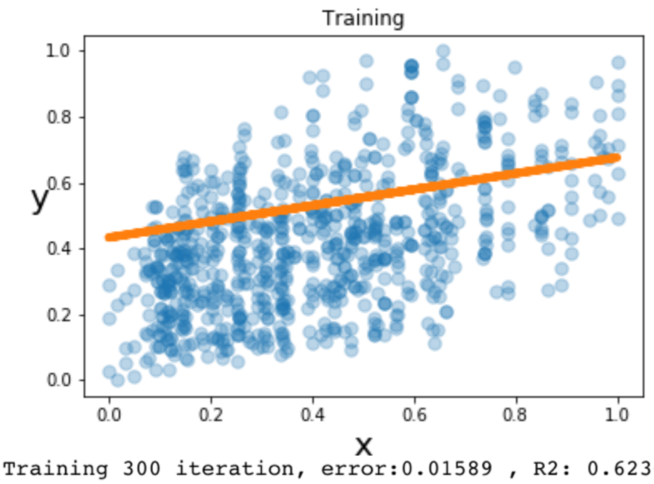
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

df = pd.read_csv("./Concrete_Data.csv")

"""
MSE = sum of [(y - x*w)**2] / length
"""
def MSE(X,coef,intercept,Y):
    return np.sum((Y-(X*coef+intercept))**2)/X.shape[0]

"""
error: MSE
shape: the number of data
R2 = 1 - SSE/SSTO
   = 1 - MSE*shape/ sum of [(y-y.mean)**2]
"""
def R2(error,shape,y):
    SSTO = np.sum((y - np.mean(y))**2)
    return 1 - error*shape/SSTO

y = df.iloc[:,8].values
column_name = list(df.columns.values)

for i in range(8):
    x = df.iloc[:,i].values
    plt.scatter(x, y,marker='.')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(column_name[i])
    plt.savefig(column_name[i])
    plt.show()

# use your eye to select x0
x = df.iloc[:,0].values
y = df.iloc[:,8].values # get target
x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))

x_train, x_test, y_train, y_test = train_test_split( x , y, test_size = 0.2)

from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train[:,np.newaxis],y_train)
error = MSE(x_train,model.coef_,model.intercept_,y_train)
R_squared = R2(error, x_train.shape[0], y_train)
print(model.coef_,model.intercept_)
print("Training\nerror:%.5f , R2: %.5f" % (error,R_squared))
error = MSE(x_test,model.coef_,model.intercept_,y_test)
R_squared = R2(error, x_test.shape[0], y_test)
print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))

plt.plot(x_train,y_train,'.',markersize=15,alpha=0.3)
plt.plot(x_train,model.intercept_+model.coef_*x_train,linewidth=5)
plt.xlabel('x',fontsize=20)
plt.ylabel('y',fontsize=20,rotation=0)
plt.savefig('scikit_learn')
```




```
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

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

df = pd.read_csv("./Concrete_Data.csv")

x = df.iloc[:,0].values # get x0 variable
y = df.iloc[:,8].values # get target
ones = np.ones((x.shape[0],1)) # constant
# normalization
# x = (x - min) / (max - min)
x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
# change dimension : 1D -> 2D
x = x[:,np.newaxis]
y = y[:,np.newaxis]
x_process = np.concatenate((ones,x), axis = 1 )
# seperate training and testing set
x_train, x_test, y_train, y_test = train_test_split( x_process,
                                                     y,
                                                     test_size = 0.2)
"""
MSE = sum of [(y - x*w)**2] / length
"""
def MSE(X,W,Y):
    return np.sum(np.dot((np.dot(X,W)-Y).T,
                        (np.dot(X,W)-Y)) /(2*X.shape[0]))

"""
error: MSE
shape: the number of data
R2 = 1 - SSE/SSTO
   = 1 - MSE*shape/ sum of [(y-y.mean)**2]
"""
def R2(error,shape,y):
    SSTO = np.sum((y - np.mean(y))**2)
    return 1 - error*shape/SSTO

lr = 0.5 # learning rate
iterations = 1000 # max loop size
iteration = 0 # loop number now
epsilon = 0.005 # minimum of error
w = np.random.randn(x_train.shape[1],1) # initial weight

# Training
while iteration < iterations:
    iteration += 1
    for var in range(x_train.shape[1]): # do GD to each variable
        gradient = np.dot((y_train - np.dot( x_train, w )).T,
                   x_train[:,var])/x_train.shape[0]
        w[var,0] = w[var,0] + lr*gradient
    error = MSE( x_train, w, y_train)
    R_squared = R2(error, x_train.shape[0], y_train)
    print("Training %d iteration, error:%.5f , R2: %.5f"
          % (iteration,error,R_squared))
    if error < epsilon: # the error is small enough, break
        break

# Testing
error = MSE( x_test, w, y_test)
R_squared = R2(error, x_test.shape[0], y_test)
print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))
print("weight: ",w[0][0],",\nbias: ",w[1][0])

plt.plot(x_train[:,1].flatten(),y_train,'.',markersize=15,alpha=0.3)
plt.plot(x_train[:,1].flatten(),
        x_train[:,1].flatten()*w[0][0]+w[1][0],linewidth=5)
plt.xlabel('x',fontsize=20)
plt.ylabel('y',fontsize=20,rotation=0)
plt.title('Training')
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，我們了解到了：

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
```

```python

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

df = pd.read_csv("./Concrete_Data.csv")

x = df.iloc[:,0:8].values # get all variable
y = df.iloc[:,8].values # get target
ones = np.ones((x.shape[0],1)) # constant
# normalization
# x = (x - min) / (max - min)
x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
# change y's dimension : 1D -> 2D
y = y[:,np.newaxis]
x_process = np.concatenate((ones,x), axis = 1 )
# seperate training and testing set
x_train, x_test, y_train, y_test = train_test_split( x_process,
                                                    y,
                                                    test_size = 0.2)


"""
MSE = sum of [(y - x*w)**2] / length
"""
def MSE(X,W,Y):
    return np.sum(np.dot((np.dot(X,W)-Y).T,
                        (np.dot(X,W)-Y)) /(2*X.shape[0]))

"""
error: MSE
shape: the number of data
R2 = 1 - SSE/SSTO
   = 1 - MSE*shape/ sum of [(y-y.mean)**2]
"""
def R2(error,shape,y):
    SSTO = np.sum((y - np.mean(y))**2)
    return 1 - error*shape/SSTO


lr = 0.5 # learning rate
iterations = 1000 # max loop size
iteration = 0 # loop number now
epison = 0.005 # minimum of error
w = np.random.randn(x_train.shape[1],1) # initial weight

# Training
while iteration < iterations:
    iteration += 1
    for var in range(x_train.shape[1]): # do GD to each variable
        gradient = np.dot((y_train - np.dot( x_train, w )).T,
                   x_train[:,var])/x_train.shape[0]
        w[var,0] = w[var,0] + lr*gradient
    error = MSE( x_train, w, y_train)
    R_squared = R2(error, x_train.shape[0], y_train)
    print("Training %d iteration, error:%.5f , R2: %.5f"
            % (iteration,error,R_squared))
    if error < epison: # the error is small enough, break
        break

# Testing
error = MSE( x_test, w, y_test)
R_squared = R2(error, x_test.shape[0], y_test)
print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))
```

7. Polynomial regression by your own gradient descent

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

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


df = pd.read_csv("./Concrete_Data.csv")


x = df.iloc[:,0:8].values # get all variable
y = df.iloc[:,8].values # get target
ones = np.ones((x.shape[0],1)) # constant
# normalization
# x = (x - min) / (max - min)
x = (x - np.amin(x, axis=0))/(np.amax(x, axis=0)-np.amin(x, axis=0))
y = (y - np.amin(y, axis=0))/(np.amax(y, axis=0)-np.amin(y, axis=0))
# change y's dimension : 1D -> 2D
y = y[:,np.newaxis]
# get the square of variable
x_square = x**2
# get the multiplication of variable
x_multi = [np.multiply(x[:,i],x[:,j])
               for i in range(0,x.shape[1]-1)
                    for j in range(i+1,x.shape[1])]
# concatenate all
x_process = np.concatenate((ones,x,x_square,
                            np.transpose(x_multi)),
                            axis = 1 )
# seperate training and testing set
x_train, x_test, y_train, y_test = train_test_split( x_process,
                                                    y,
                                                    test_size = 0.2)


"""
MSE = sum of [(y - x*w)**2] / length
"""
def MSE(X,W,Y):
    return np.sum(np.dot((np.dot(X,W)-Y).T,
                        (np.dot(X,W)-Y)) /(2*X.shape[0]))

"""
error: MSE
shape: the number of data
R2 = 1 - SSE/SSTO
   = 1 - MSE*shape/ sum of [(y-y.mean)**2]
"""
def R2(error,shape,y):
    SSTO = np.sum((y - np.mean(y))**2)
    return 1 - error*shape/SSTO


lr = 5 # learning rate
iterations = 1000 # max loop size
iteration = 0 # loop number now
epison = 0.005 # minimum of error
w = np.random.randn(x_train.shape[1],1) # initial weight

# Training
while iteration < iterations:
    iteration += 1
    for var in range(x_train.shape[1]): # do GD to each variable
        gradient = np.dot((y_train - np.dot( x_train, w )).T,
               x_train[:,var])/x_train.shape[0]
        w[var,0] = w[var,0] + lr*gradient
    error = MSE( x_train, w, y_train)
    R_squared = R2(error, x_train.shape[0], y_train)
    print("Traing %d iteration, error:%.5f , R2: %.5f"
          % (iteration,error,R_squared))
    if error < epison: # the error is small enough, break
        break

# Testing
error = MSE( x_test, w, y_test)
R_squared = R2(error, x_test.shape[0], y_test)
print("Testing\nerror:%.5f , R2: %.5f" % (error,R_squared))
```

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- SSE/SSTO = 1 – MSE* shape/ sum of [(y-y.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
```