Machine Learning Program Assignment #3

Team ID: 8

Team members:

0516037 朱俐瑄 0516212 吳子涵 0516218 軒轅照雯 0516239 王盈筑

1. What environments the members are using:

朱俐瑄:

Environment: Linux (Ubuntu)

Programming Language: Python 2.7

王盈筑:

Environment: Windows10

Programming Language: Python 3.7

吳子涵:

Environment: macOS

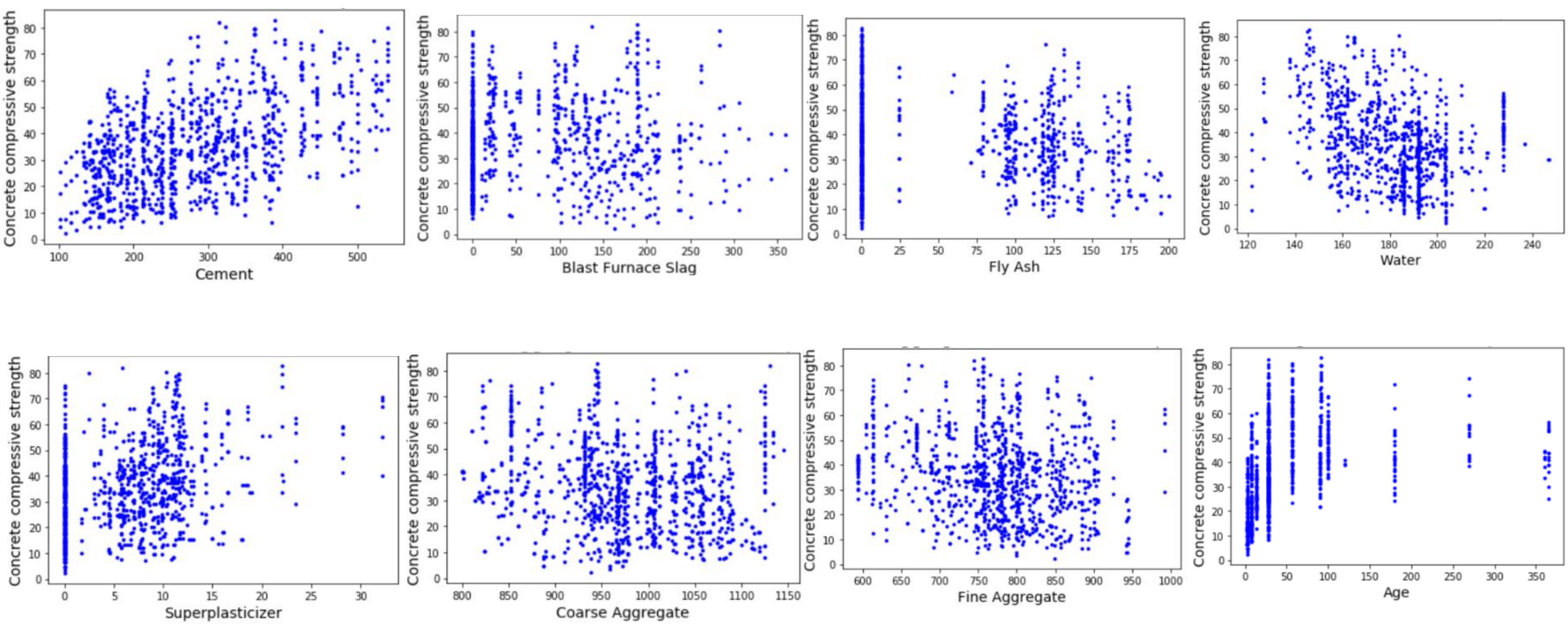
Programming Language: Python 3

軒轅照雯:

Environment: macOS

Programming Language: Python 2.7

2. Visualization of all the features with the target:



Scatter Plot of [each feature] with Concrete compressive strength

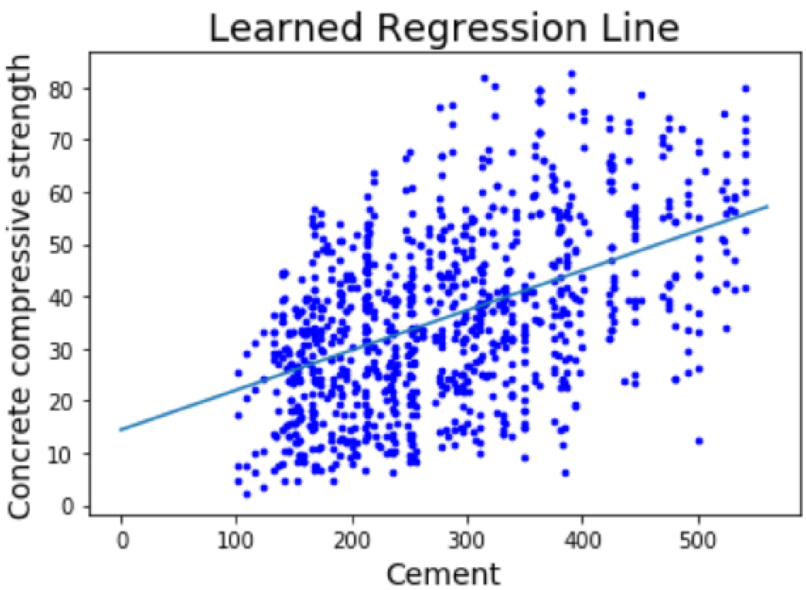
Code:

```
data=df.values[:,8]
data_ = data[:, np.newaxis] #將Concrete_compressive_strength的資料轉成2D模式

#'Cement (component 1)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Cement and Concrete compressive strength', fontsize=18)
plt.xlabel('Cement', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,0]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Blast Furnace Slag (component 2)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Blast Furnace Slag and Concrete compressive strength', fontsize=18)
plt.xlabel('Blast Furnace Slag', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,1]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Fly Ash (component 3)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Fly Ash and Concrete compressive strength', fontsize=18)
plt.xlabel('Fly Ash', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,2]
data1_=data1[:, np.newaxis]
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Water  (component 4)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Water and Concrete compressive strength', fontsize=18)
plt.xlabel('Water', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,3]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Superplasticizer (component 5)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Superplasticizer and Concrete compressive strength', fontsize=18)
plt.xlabel('Superplasticizer', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,4]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Coarse Aggregate  (component 6)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Coarse Aggregate and Concrete compressive strength', fontsize=18)
plt.xlabel('Coarse Aggregate', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,5]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Fine Aggregate (component 7)(kg in a m^3 mixture)'
plt.title('Scatter Plot of Fine Aggregate and Concrete compressive strength', fontsize=18)
plt.xlabel('Fine Aggregate', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,6]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()

#'Age (day)'
plt.title('Scatter Plot of Age and Concrete compressive strength', fontsize=18)
plt.xlabel('Age', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
data1=df.values[:,7]
data1_=data1[:, np.newaxis]
#print(data_)
#print(data1_)
plt.scatter(data1_, data_, s=7, c='b')
plt.show()
```

3. The code, graph, r2_score, weight and bias for problem 1:

```
from sklearn.model_selection import train_test_split
from sklearn import linear_model
#train, test = train_test_split(df, test_size=0.2)
y=data_
data1=df.values[:,0] #Cement
data1_=data1[:, np.newaxis]
X=data1_
#print(y)
#print(data1_)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
#X_train=X_train.ravel()
#X_test=X_test.ravel()
y_train=y_train.ravel()
y_test=y_test.ravel()
lm=linear_model.LinearRegression()
model=lm.fit(X_train,y_train)
weight=lm.coef_
bias=lm.intercept_
#accuracy=lm.score(X_test,y_test)
print("weight:", weight)
print("bias:", bias)
#print("accuracy (r2_score):", accuracy)
#
from sklearn.metrics import r2_score
y_predict=lm.predict(X_test)
score2=r2_score(y_test, y_predict)
#print(score2)
print("accuracy (r2_score):", score2)
#
#plot Learned Regression Line
plt.title('Learned Regression Line', fontsize=18)
plt.xlabel('Cement', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
plt.scatter(data1_, data_, s=7, c='b')
x=np.arange(0, 560, 0.01) #get values between 0 and 560 with 0.01 step and set to x
y=weight*x+bias #get x values from y
plt.plot(x, y)
plt.show()
```

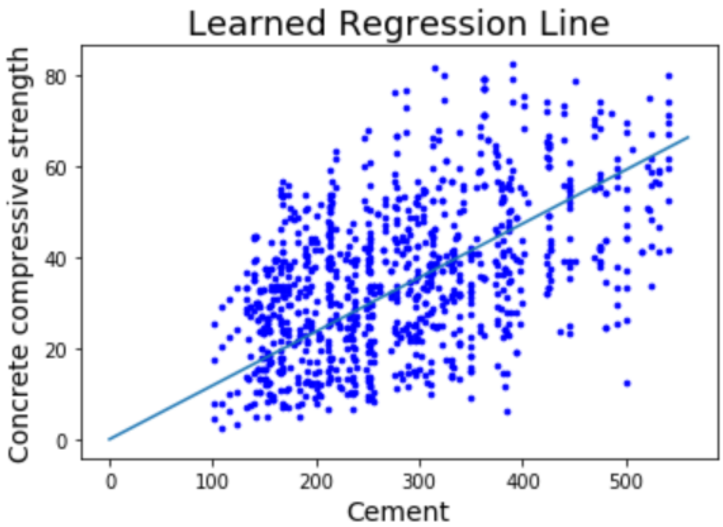
```
weight: [0.07621133]
bias: 14.417421400734831
accuracy (r2_score): 0.3448206041466938
```



4. The code, graph, r2_score, weight and bias for problem 2:

```
y=data_
data1=df.values[:,0] #Cement
data1_=data1[:, np.newaxis]
X=data1_
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
y_train=y_train.ravel()
y_test=y_test.ravel()
#'''
###
def predict(x_, weight, bias):
    return weight*x_+bias

#def cost_function(cement, ccs, weight, bias):
#    num=len(cement)
#    total_error=0.0
#    for i in range(num):
#        total_error+=(ccs[i]-(weight*cement[i]+bias))**2
#    return total_error/num

def r2_score_(x_, y_, weight, bias):
    num=len(x_)
    rss=0.0
    tss=0.0
    y_mean=np.mean(y_)
    for i in range(num):
        #rss=sum of squares of difference between actual values(yi) and predicted values(yi^)
        rss+=(y_[i]-(weight*x_[i]+bias))**2
        #tss=sum of squares of difference between actual values (yi) and mean value (Before applying Regression)
        tss+=(y_[i]-y_mean)**2
    return 1-(rss/tss)

def update_weights(x_, y_, weight, bias, learning_rate):
    weight_dev=0
    bias_dev=0
    num=len(x_)

    for i in range(num):
        # Calculate partial derivatives
        # -2x(y - (mx + b))
        weight_dev += -2*x_[i] * (y_[i] - (weight*x_[i] + bias))
        # -2(y - (mx + b))
        bias_dev += -2*(y_[i] - (weight*x_[i] + bias))

    #if j>0 :
    #    learning_rate=((learning_rate*10)/(10+j))
    weight -= (weight_dev/num)*learning_rate
    bias -= (bias_dev/num)*learning_rate

    return weight, bias, weight_dev, bias_dev

def train(x_, y_, weight, bias, learning_rate, iters):
    cost_history=[]

    #weight,bias,weight_dev,bias_dev=update_weights(x_, y_, weight, bias, learning_rate)
    #i=0

    for i in range(iters):
    #while abs(weight_dev)<0.1 and abs(bias_dev)<0.1:
        weight,bias,weight_dev,bias_dev=update_weights(x_, y_, weight, bias, learning_rate)

        #cost=cost_function(cement, ccs, weight, bias)
        cost=r2_score_(x_, y_, weight, bias)
        cost_history.append(cost)

        #print("iter: ", i, " cost: ", cost)

    return weight, bias, cost_history
import random
weight=random.uniform(-0.2, 0.2)
bias=random.uniform(-0.2, 0.2)
print("initial weight:", weight)
print("initial bias:", bias)
learning_rate=0.00000018
iters=100
w, b, c = train(X_train, y_train, weight, bias, learning_rate, iters)
print("weight after iterations:", w)
print("bias after iterations:", b)
accuracy = r2_score_(X_test, y_test, w, b)
print("accuracy (r2_score):", accuracy)
#
#plot Learned Regression Line
plt.title('Learned Regression Line', fontsize=18)
plt.xlabel('Cement', fontsize=14)
plt.ylabel('Concrete compressive strength', fontsize=14)
plt.scatter(data1_, data_, s=7, c='b')
x=np.arange(0, 560, 0.01) #get values between 0 and 560 with 0.01 step and set to x
y=w*x+b #get x values from y
plt.plot(x, y)
plt.show()
```

```
initial weight: 0.050271425869025776
initial bias: -0.05420252151718383
weight after iterations: [0.11879898]
bias after iterations: [-0.05392768]
accuracy (r2_score): [0.23832058]
```



5. Compare Problem1 and Problem2, show what you got:

Initial weight and initial bias would different gradient descent, and different gradient descent could change the accuracy of linear regression. Also, the operation of iteration and the formula used inside would affect as well.

6. The code, MSE, and the r2_score for problem 3:

```
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
from sklearn.model_selection import train_test_split


df=pd.read_csv('Concrete_Data.csv')
y=df['Concrete compressive strength(MPa, megapascals) ']
y=y.values.ravel()
df=df.drop('Concrete compressive strength(MPa, megapascals) ', axis=1)
df=df.drop('Blast Furnace Slag (component 2)(kg in a m^3 mixture)', axis=1)
df=df.drop('Fly Ash (component 3)(kg in a m^3 mixture)', axis=1)
#df=df.drop('Fine Aggregate (component 7)(kg in a m^3 mixture)', axis=1)
df=df.drop('Water  (component 4)(kg in a m^3 mixture)', axis=1)
df=df.drop('Superplasticizer (component 5)(kg in a m^3 mixture)', axis=1)
#df=df.drop('Cement (component 1)(kg in a m^3 mixture)', axis=1)
df=df.drop('Age (day)', axis=1)
df=df.drop('Coarse Aggregate  (component 6)(kg in a m^3 mixture)', axis=1)
x_train, x_test, y_train, y_test = train_test_split(df, y, test_size=0.2)
x_train.insert(2, column='1', value=1)
x_test.insert(2, column='1', value=1)
x_train=x_train.values
x_test=x_test.values
a = 0.0000000001  #learning rate
n = y_train.shape[0]
######################each iteration only update wj###################
print("for each iteration only update wj: ")
#w= np.zeros((3,1))
w=np.random.rand(3,1)
w =w
```

```python
deltaw=np.zeros((3, 1))
#print(x_train)
k = 0
while(k < 1000):
    for j in range(3):
        deltaw[j]=0
        for i in range(n):
            y=np.sum(np.matmul(x_train[i], w))
            deltaw[j]=deltaw[j]+x_train[i][j]*(y_train[i]-y)

        w[j]=w[j]+a*deltaw[j]
    k=k+1

#######training analyze#########
error_1=0
pred=np.zeros(n)
for i in range(n):
    pred[i]=np.sum(np.matmul(x_train[i], w))

print("mean square error of training: ", mean_squared_error(y_train, pred))
print("R2 of training: ", r2_score(y_train,pred))
#######testing analyze#########
error_2=0
m=y_test.shape[0]
pred_=np.zeros(m)
for i in range(m):
    pred_[i]=np.sum(np.matmul(x_test[i], w))
print("mean square error of testing: ", mean_squared_error(y_test, pred_))
print("R2 of testing: ", r2_score(y_test,pred_))
print("\n")

######################each iteration update w####################
print("for each iteration update w: ")
#w_= np.zeros((3,1))
#w_=np.random.rand(3,1)
k_=0
while(k_ < 1000):
    deltaw_=np.zeros((3, 1))
    for j in range(3):
        for i in range(n):
            y_=np.sum(np.matmul(x_train[i], w_))
            deltaw_[j]=deltaw_[j]+x_train[i][j]*(y_train[i]-y_)
    w_=w_+a*deltaw_
    k_=k_+1

#######training analyze#########
error_3=0
pred=np.zeros(n)
for i in range(n):
    pred[i]=np.sum(np.matmul(x_train[i], w_))

print("mean square error of training: ", mean_squared_error(y_train, pred))
print("R2 of training: ", r2_score(y_train, pred))
#######testing analyze#########
error_4=0
pred_=np.zeros(m)
mean_4=np.sum(y_test)/m
for i in range(m):
    pred_[i]=np.sum(np.matmul(x_test[i], w_))

print("mean square error of testing: ", mean_squared_error(y_test, pred_))
print("R2 of testing: ", r2_score(y_test, pred_))
```

```
for each iteration only update wj:
mean square error of training:  218.532197843157
R2 of training:  0.21395292075775985
mean square error of testing:  213.23388795551932
R2 of testing:  0.24143965138149837


for each iteration update w:
mean square error of training:  215.62477932466018
R2 of training:  0.2244107290677254
mean square error of testing:  207.70178020635376
R2 of testing:  0.26111962637533004
```

7. Compare the performance between two different update method:

The method that update w each iteration performs better than which only update w_j. Cause the linear model: $y = b + \sum_{j=1}^{D} w_j x_j = b + w^T X$ shows that w is more important than w_j.

8. The code, MSE, and the r2_score for problem 4:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('Concrete_Data.csv')
xx = df.iloc[:,0:8].values
yy = df.iloc[:,8].values

standard_x = (xx - np.amin(xx, axis=0))/(np.amax(xx, axis=0)-np.amin(xx, axis=0))
standard_y = (yy - np.amin(yy, axis=0))/(np.amax(yy, axis=0)-np.amin(yy, axis=0))

tmp1=standard_x**2
for i in range(7):
    for j in range(i+1, 8):
        tmp2=standard_x[:,i]*standard_x[:,j]
        tmp1=np.concatenate((tmp1,tmp2[:,np.newaxis]),axis=1)

x=np.concatenate((np.ones((standard_x.shape[0],1)),standard_x,tmp1),axis=1)
y=standard_y[:,np.newaxis]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 100)
```

```
def MSE (X, Y, W):
    return np.sum((Y - np.dot(X, W))**2)/(2*X.shape[0])
    #return np.sum(np.dot((np.dot(X,W)-Y).T,(np.dot(X,W)-Y)) /(2*X.shape[0]))

def R2 (mse, num, y):
    return 1-((mse*num)/(np.sum((y - np.mean(y))**2)))
    #SSTO = np.sum((y - np.mean(y))**2)
    #return 1 - mse*num/SSTO
learning_rate=5
iterations=1000
epsilon=0.005

theta = np.random.randn(45,1)

x_train = np.array(x_train, dtype=np.float128)
print("training:")
for i in range(iterations):
    for j in range(45):
        predict = np.dot(x_train, theta)
        gradients = np.dot((y_train - predict).T, x_train[:,j])/x_train.shape[0]
        theta[j,0] += (learning_rate*gradients)

    err = MSE(x_train, y_train, theta)
    r2 = R2(err, x_train.shape[0], y_train)
    #print("iteration %d: mse= %.6f, r2= %.6f" % (i,err,r2))
    if err < epsilon:
        break

print("mse= %.6f, r2= %.6f" % (err,r2))
err = MSE(x_test, y_test, theta)
r2 = R2(err, x_test.shape[0], y_test)
print("\ntesting:\nmse= %.6f, r2= %.6f" % (err,r2))
```

```
training:
mse= 0.004998, r2= 0.882250

testing:
mse= 0.005576, r2= 0.880169
```

9. Answer the question:

I. What is overfitting?

Overfitting is the result of an overly complex model with too many parameters.
It occurs when a model tries to predict a trend in data that is too noisy.
A model that is overfitted is inaccurate because the trend does not reflect the reality of the data.

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

On large datasets, SGD can converge faster than batch training because it performs updates more frequently.

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

Cause the gradient descent algorithm starts tuning initial value with the goal of finding smaller value for formula. It may converge at the local minimums or global minimum depends on where initial value is.

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

Bad learning rate in neural network has two kinds.
One is the learning rate is too small that the gradient descent is slow, another is the learning rate is too large that the gradient descent overshoots the minimum, that is, the gradient descent may fail to converge or even diverge.

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

10. Bonus:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('Concrete_Data.csv')
xx = df.iloc[:,0:8].values
yy = df.iloc[:,8].values

standard_x = (xx - np.amin(xx, axis=0))/(np.amax(xx, axis=0)-np.amin(xx, axis=0))
standard_y = (yy - np.amin(yy, axis=0))/(np.amax(yy, axis=0)-np.amin(yy, axis=0))

tmp1=standard_x**2
for i in range(7):
    for j in range(i+1, 8):
        tmp2=standard_x[:,i]*standard_x[:,j]
        tmp1=np.concatenate((tmp1,tmp2[:,np.newaxis]),axis=1)

x=np.concatenate((np.ones((standard_x.shape[0],1)),standard_x,tmp1),axis=1)
y=standard_y[:,np.newaxis]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 100)
```

```python
def MSE (X, Y, W):
    return np.sum((Y - np.dot(X, W))**2)/(2*X.shape[0])
    #return np.sum(np.dot((np.dot(X,W)-Y).T,(np.dot(X,W)-Y)) /(2*X.shape[0]))

def R2 (mse, num, y):
    return 1-((mse*num)/(np.sum((y - np.mean(y))**2)))
    #SSTO = np.sum((y - np.mean(y))**2)
    #return 1 - mse*num/SSTO
```

```python
learning_rate=5
iterations=1000
epsilon=0.005

theta = np.random.randn(45,1)

x_train = np.array(x_train, dtype=np.float128)
print("training:")
for i in range(iterations):
    for j in range(45):
        predict = np.dot(x_train, theta)
        gradients = np.dot((y_train - predict).T, x_train[:,j])/x_train.shape[0]
        theta[j,0] += (learning_rate*gradients)

    err = MSE(x_train, y_train, theta)
    r2 = R2(err, x_train.shape[0], y_train)
    #print("iteration %d: mse= %.6f, r2= %.6f" % (i,err,r2))
    if err < epsilon:
        break

print("mse= %.6f, r2= %.6f" % (err,r2))
err = MSE(x_test, y_test, theta)
r2 = R2(err, x_test.shape[0], y_test)
print("\ntesting:\nmse= %.6f, r2= %.6f" % (err,r2))
```

```
training:
mse= 0.004998, r2= 0.882250

testing:
mse= 0.005576, r2= 0.880169
```