Thursday, 18 January 2018

Problem Set 1

Machine Learning AT70.9022

- Estimate a linear regression model using the normal equations

For estimating the linear regression model using the normal equation, first we need to get the dataset. The data set we are using for this model is 'weight.txt' which has 80 elements. By using height and age, weight will be predicted.

To find $\boldsymbol{\theta}$, this formula is used $\boldsymbol{\theta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

Once we have $\boldsymbol{\theta}$, we can calculate

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2.$$

Which will give us the cost we get by using the hypothesis.

The python implementation of this experiment is given below:

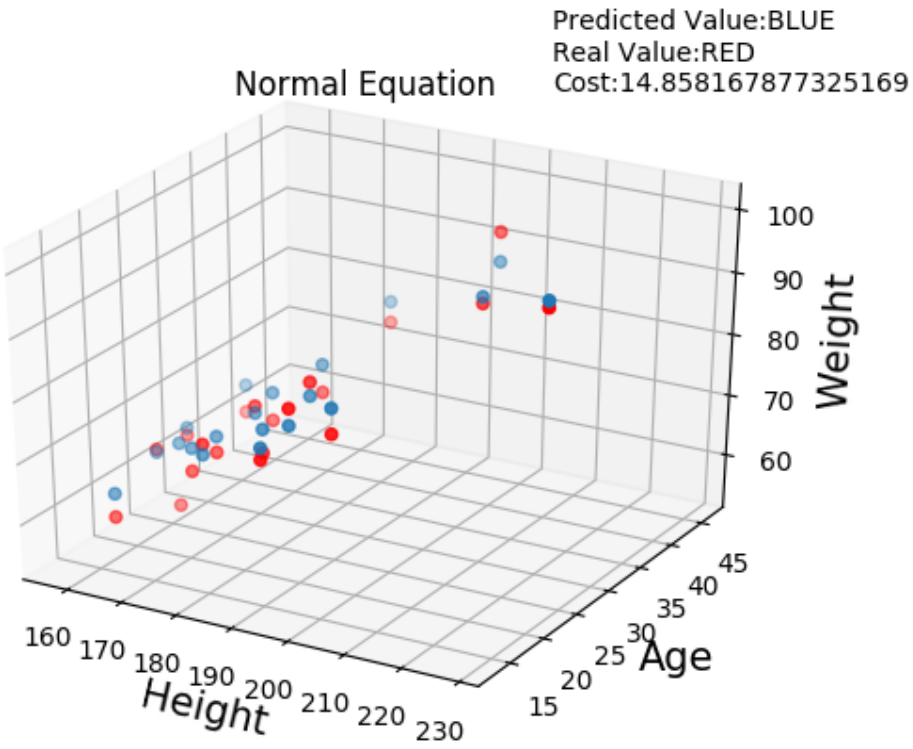
```
import matplotlib.pyplot as plt
import numpy
from mpl_toolkits.mplot3d import Axes3D

#3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

fig.subplots_adjust(top=0.85)
ax.set_title('Normal Equation')
```

```
#importing dataset

datain = numpy.loadtxt('weight.txt')


x0 = numpy.matrix(numpy.ones(100)).transpose()

X = numpy.concatenate((x0,datain[:,0:2]),1)

Y = datain[:,2:]


Xtrain = X[:-20,:]

Xtest = X[-20:,:]

Ytrain = Y[:-20,:]

Ytest = Y[-20:,:]


# Finding the theta

theta_opt = numpy.linalg.inv(Xtrain.transpose() * Xtrain) * Xtrain.transpose() * Ytrain


# plotting actual values and predicted value

ax.scatter(Xtest[:,1],Xtest[:,2], Xtest* theta_opt)

ax.scatter(Xtest[:,1],Xtest[:,2], Ytest,c='red')


#The Cost

cost_function = ((Xtest * theta_opt - Ytest).transpose() * (Xtest * theta_opt - Ytest))/
len(Xtrain)


ax.text2D(0.70, 0.95, "Predicted Value:BLUE\nReal
Value:RED\nCost:"+str(cost_function[0,0]), transform=ax.transAxes)

ax.set_xlabel('Height', fontsize=15, rotation = 0)

ax.set_ylabel('Age', fontsize=15, rotation = 0)

ax.set_zlabel('Weight', fontsize=15, rotation = 0)
```

```
plt.show()
```

Observation:

After calculating the θ we test and see the observed and predicted values in the graph given below:



We can see that the cost we are getting is very small 14.8 approximate. This seems to be a very good cost. Therefore we can call this as a good hypothesis.

- Implement batch gradient descent.

For implementing the batch gradient descent we again take the dataset. First we take the initial theta as [0,1,1]. We use below formula to calculate the new **θ** :

$$\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} - \alpha \nabla_J(\boldsymbol{\theta}^{(n)}).$$

After 10 iterations we use the **θ** to predict the 'weights'. And in the end we find the cost we get after using the hypothesis.

The python implementation of this experiment is given below:

```
import matplotlib.pyplot as plt

import numpy

from mpl_toolkits.mplot3d import Axes3D


#3D

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')


fig.subplots_adjust(top=0.85)

ax.set_title('Batch gradient')


# importing data

datain = numpy.loadtxt('weight.txt')


x0 = numpy.matrix(numpy.ones(100)).transpose()
```

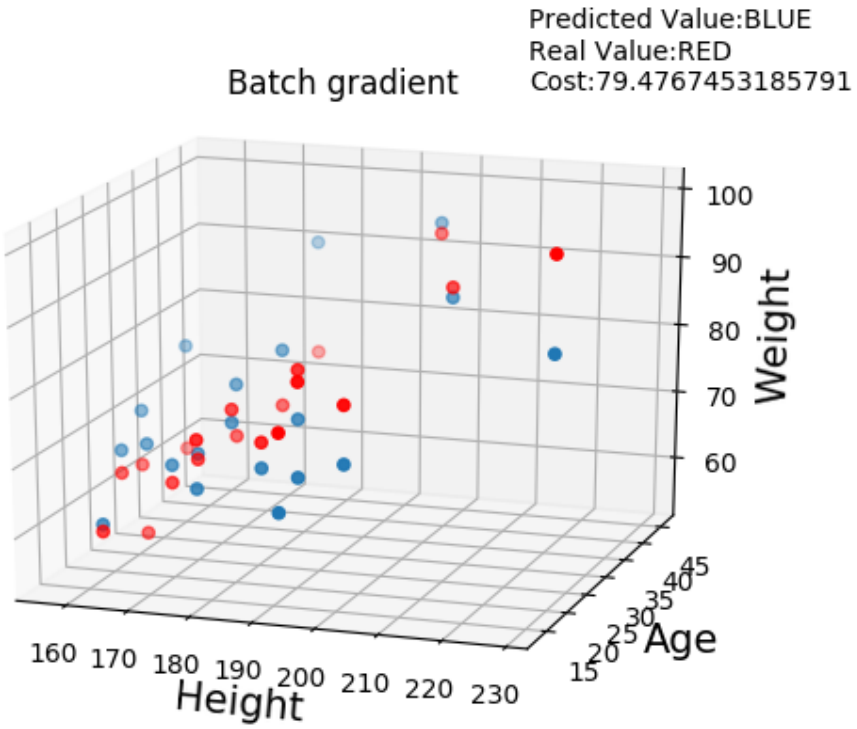
```
X = numpy.concatenate((x0,datain[:,0:2]),1)

Y = datain[:,2:]


Xtrain = X[:-20,:]

Xtest = X[-20:,:]

Ytrain = Y[:-20,:]

Ytest = Y[-20:,:]


# initial theta values

theta_cur = numpy.matrix([0, 1, 1]).transpose()


storing_residual = numpy.empty([10, 1])


for iter in range(0,10):

    # calculating the residual

    residual = (Xtrain * theta_cur - Ytrain).transpose() * (Xtrain * theta_cur - Ytrain)


    # recording

    storing_residual[iter,0] = residual


    # delta values

    delta_intercept = numpy.sum(Xtrain * theta_cur - Ytrain)

    delta_height = numpy.sum((Xtrain * theta_cur - Ytrain).transpose()*Xtrain[:,1])
```

```
    delta_age = numpy.sum((Xtrain * theta_cur - Ytrain).transpose()*Xtrain[:,2])


    # gradient
    gradient = numpy.matrix([delta_intercept, delta_height, delta_age]).transpose()


    # new theta value
    theta_cur = theta_cur - 0.00001/len(Xtrain) * gradient


cost_function = ((Xtest * theta_cur - Ytest).transpose() * (Xtest * theta_cur - Ytest))/
len(Xtest)


# Plotting 3d graph for actual values and predicted values
ax.scatter(Xtest[:,1],Xtest[:,2], Xtest* theta_cur)
ax.scatter(Xtest[:,1],Xtest[:,2], Ytest,c='red')
ax.text2D(0.70, 0.95, "Predicted Value:BLUE\nReal
Value:RED\nCost:"+str(cost_function[0,0]), transform=ax.transAxes)
ax.set_xlabel('Height', fontsize=15, rotation = 0)
ax.set_ylabel('Age', fontsize=15, rotation = 0)
ax.set_zlabel('Weight', fontsize=15, rotation = 0)



plt.show()


plt.title('epochs vs. J($\dot{\Theta}$)')
```

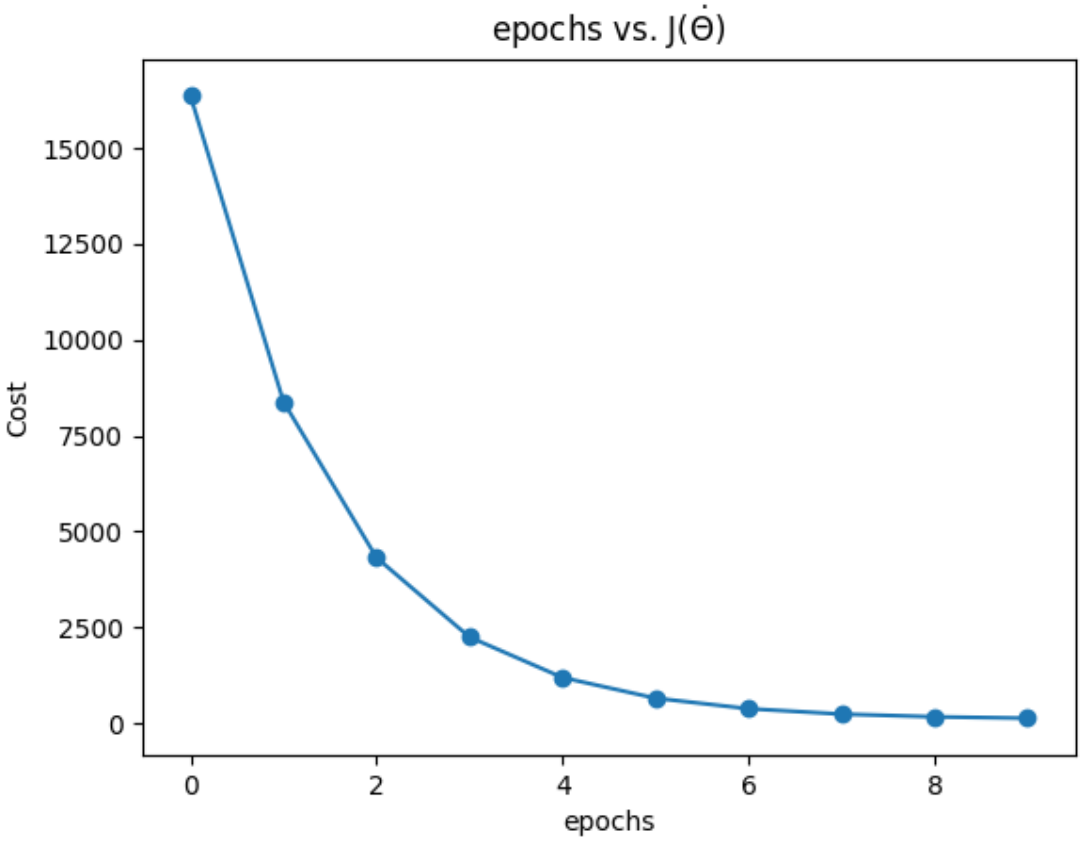
```
plt.plot(range(0,10), storing_residual/len(Xtrain))

plt.scatter(range(0,10), storing_residual/len(Xtrain))

plt.xlabel('epochs')

plt.ylabel('Cost')

plt.show()
```

Observation:

After calculating the **θ** we test and see the observed and predicted values in the graph given below:

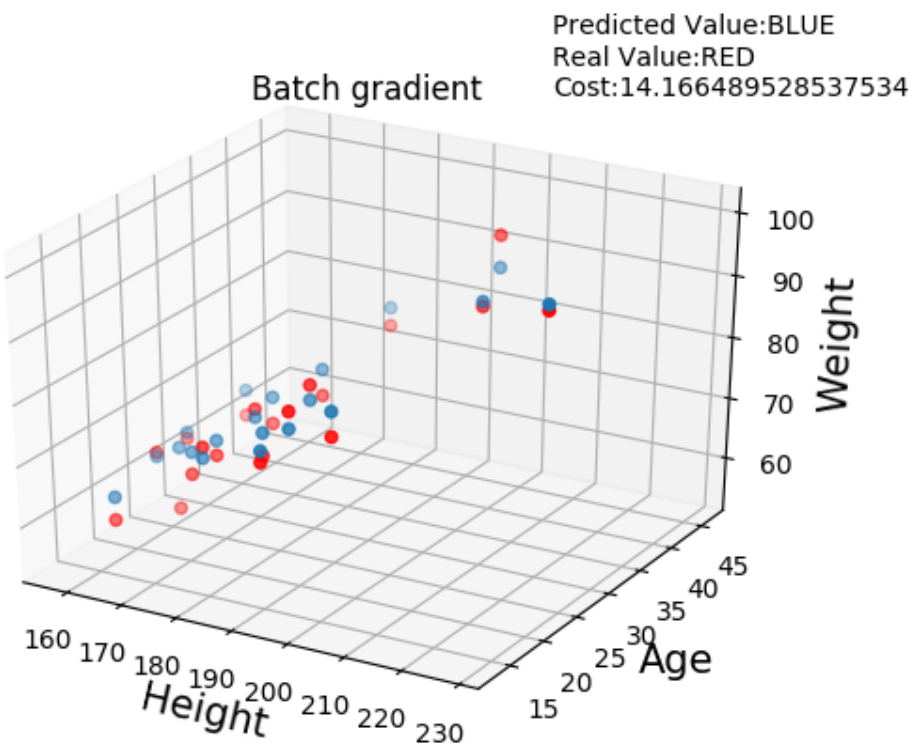


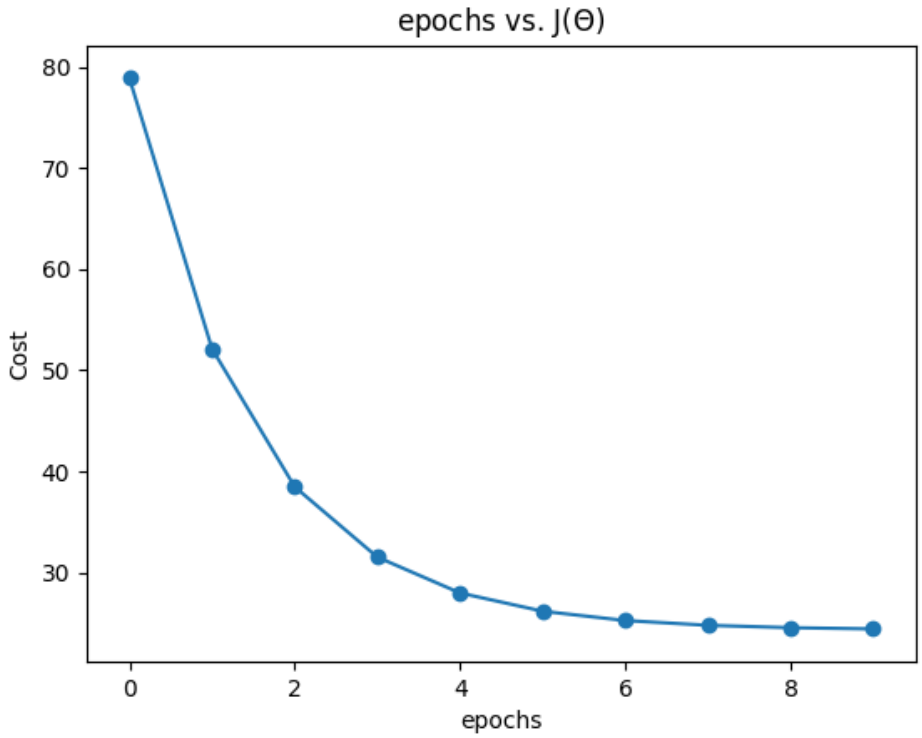
We can see that the cost doesn't seem to be as good as the Normal equation ones. It might get better if we play around with the initial value and the learning rate.



The cost seems to go down really fast in just 10 iterations. But not close to the cost as in normal equation.

After playing around with the initial values of $\boldsymbol{\theta}$, these are the significant changes that we can see.


Predicted Value:BLUE
Real Value:RED
Cost:14.166489528537534
Batch gradient
Weight
100
90
80
70
60
Age
45
40
35
30
25
20
15
Height
160
170
180
190
200
210
220
230


epochs vs. J(Θ)
Cost
80
70
60
50
40
30
epochs
0
2
4
6
8

- Implement stochastic gradient descent.

For implementing the stochastic gradient descent we again take the dataset. First we take the initial theta as [0,1,1]. We use below formula to calculate the new $\boldsymbol{\theta}$:

$\boldsymbol{\theta} \leftarrow$ some initial guess
While not converged
    For $i \in 1..m$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})\mathbf{x}^{(i)}$

Here rather than calculating the gradient with the whole dataset we just take one elements from the dataset and then find the gradient and get the new $\boldsymbol{\theta}$ value. In the end we will find the cost using our test values.

The python implementation of this experiment is given below:

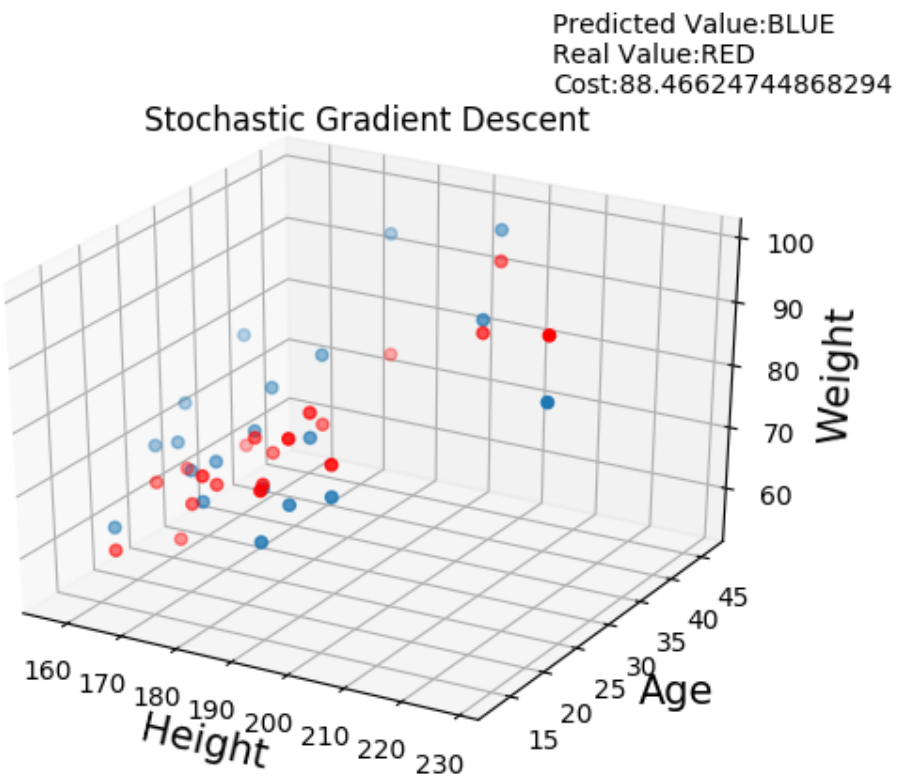
```
 import matplotlib.pyplot as plt
import numpy
from mpl_toolkits.mplot3d import Axes3D

# 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

fig.subplots_adjust(top=0.85)
ax.set_title('Stochastic Gradient Descent')

# Importing data
datain = numpy.loadtxt('weight.txt')
```

```
x0 = numpy.matrix(numpy.ones(100)).transpose()
X = numpy.concatenate((x0,datain[:,0:2]),1)
Y = datain[:,2:]

Xtrain = X[:-20,:]
Xtest = X[-20:,:]
Ytrain = Y[:-20,:]
Ytest = Y[-20:,:]

# initial values for thetas
theta_cur = numpy.matrix([-28.0, 0.5, 0.2]).transpose()

# Storing squared residuals
storing_residual = numpy.empty([10, 1])

for iter in range(0,10):
    # getting the squared residuals
    residual = (Xtrain * theta_cur - Ytrain).transpose() * (Xtrain * theta_cur - Ytrain)

    # recording
    storing_residual[iter,0] = residual

    # iteration to find gradient after every single dataset element
    for x in range(0,len(Xtrain)):
        delta_intercept = (Xtrain[x,:] * theta_cur - Ytrain[x])[0,0]
        delta_height = ((Xtrain[x,:] * theta_cur - Ytrain[x]) * Xtrain[x,1])[0,0]
        delta_age = ((Xtrain[x,:] * theta_cur - Ytrain[x]) * Xtrain[x,2])[0,0]
```

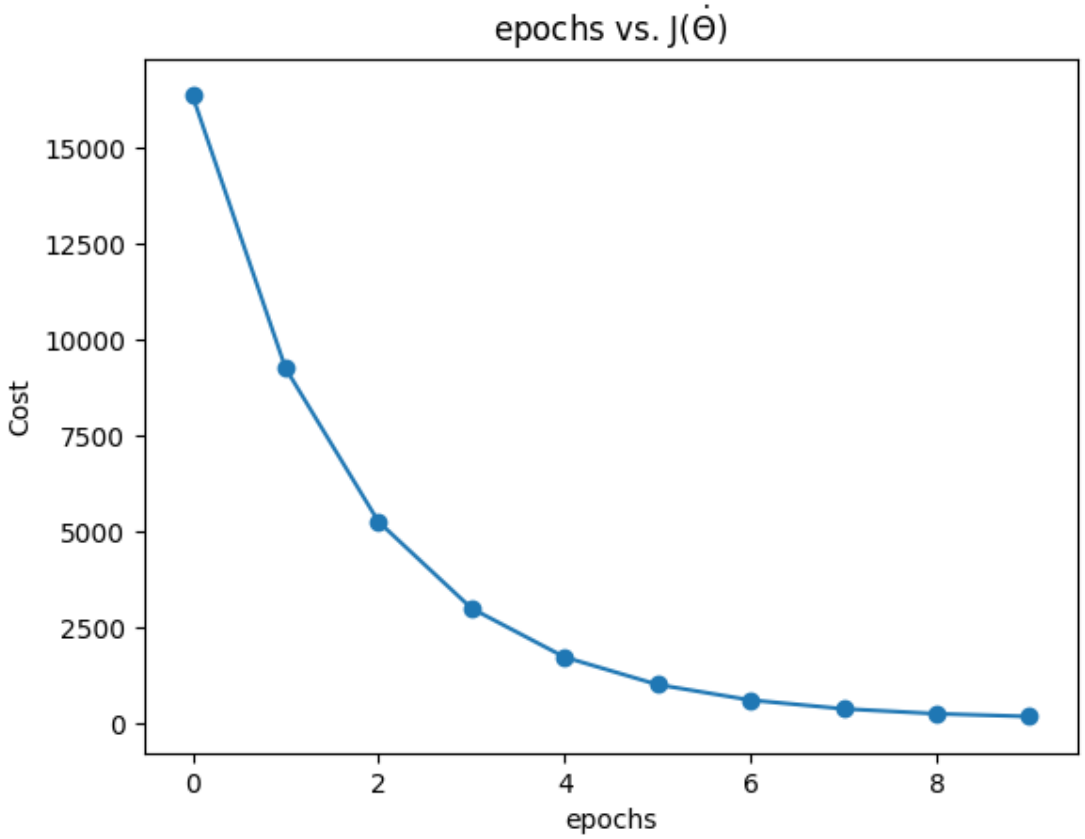
```
        gradient = numpy.matrix([delta_intercept, delta_height, delta_age]).transpose()
        theta_cur = theta_cur - 0.00001/len(Xtrain) * gradient

    cost_function = ((Xtest * theta_cur - Ytest).transpose() * (Xtest * theta_cur - Ytest))/
len(Xtest)

# Plotting 3d graph for actual values and predicted values
ax.scatter(Xtest[:,1],Xtest[:,2], Xtest* theta_cur)
ax.scatter(Xtest[:,1],Xtest[:,2], Ytest,c='red')
ax.text2D(0.70, 1, "Predicted Value:BLUE\nReal
Value:RED\nCost:"+str(cost_function[0,0]), transform=ax.transAxes)
ax.set_xlabel('Height', fontsize=15, rotation = 0)
ax.set_ylabel('Age', fontsize=15, rotation = 0)
ax.set_zlabel('Weight', fontsize=15, rotation = 0)


plt.show()

plt.title('epochs vs. J($\dot{\Theta}$)')

plt.plot(range(0,10), storing_residual/len(Xtrain))
plt.scatter(range(0,10), storing_residual/len(Xtrain))
plt.xlabel('epochs')
plt.ylabel('Cost')
plt.show()
```

Observation:

After calculating the **θ** we test and see the observed and predicted values in the graph given below:

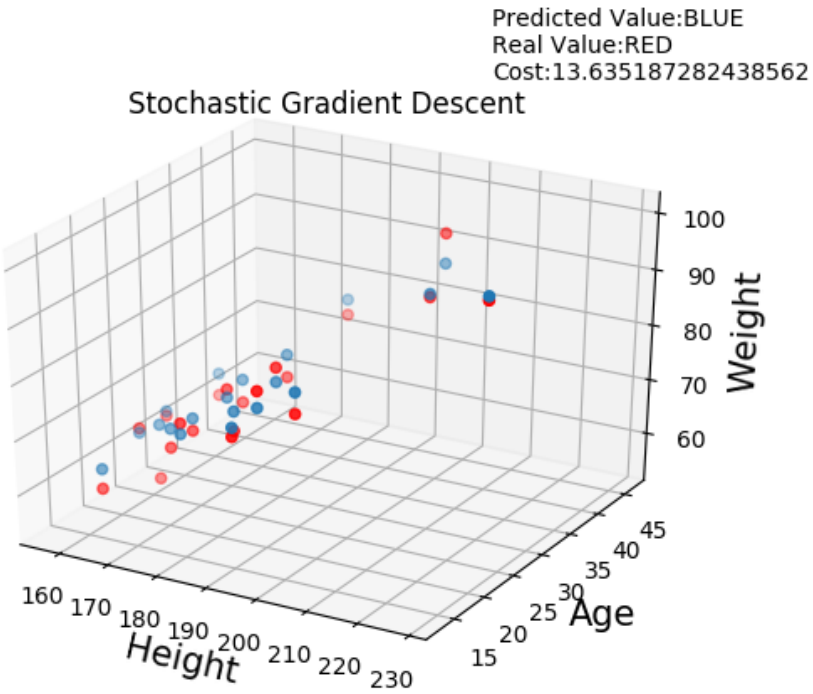


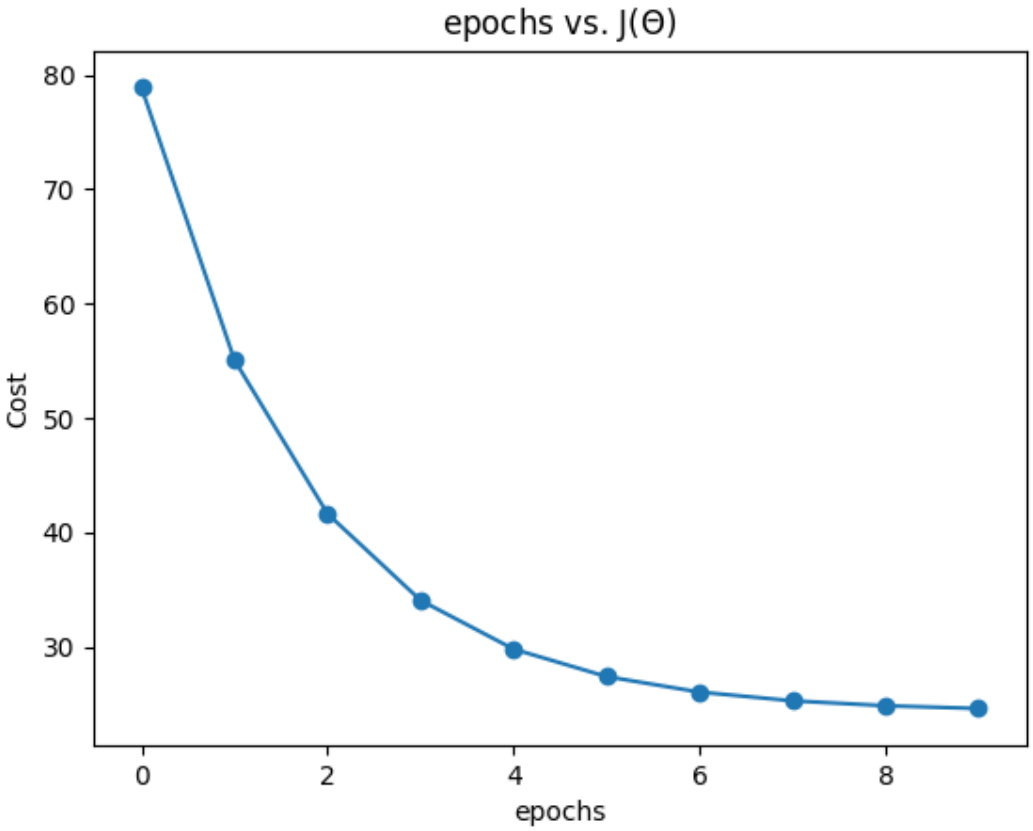
We can see that the cost isn't even better than the batch gradient stochastic. It might get better if we play around with the initial value and the learning rate.



The graph shows that the cost comes down fast but maybe it started oscillating around the minimum because of which we did not get a better cost.

After playing around with the initial values of $\boldsymbol{\theta}$, these are the significant changes that we can see.


Predicted Value:BLUE
Real Value:RED
Cost:13.635187282438562
Stochastic Gradient Descent
Weight
Height
Age
160
170
180
190
200
210
220
230
15
20
25
30
35
40
45
60
70
80
90
100


epochs vs. J(Θ)
Cost
epochs
30
40
50
60
70
80
0
2
4
6
8