**Linear Regression:**

**import** autograd.numpy **as** np

**from** autograd **import** grad

**import** matplotlib.pyplot **as** plt

In [2]:

*# Linear function returns continuous ranged value*

**def** linear(x,a,b):

**return** a**\***x**+**b

*# Function to calucate loss, return SSR*

**def** loss(x,y\_obs,a,b):

y\_model **=** linear(x,a,b)

**return** np**.**sum( (y\_model**-**y\_obs)**\*\***2 )

*# Create X and Y arrays, used for testing*

x **=** np**.**array( [0,1,2,3,4,5,6,7,8,9,10] )

y\_obs **=** np**.**array( [0,1.1,1.9,3.2,3.8,5.1,6.3,6.9,8.5,8.5,10.2] )

*# Assign weights for prediction randomly*

a,b **=** 2.0, 1.0

y\_model **=** linear(x,a,b)

print('First guess at a and b are ',a,b)

print('First loss function is ',loss(x,y\_obs,a,b))

*# Autograd used to get the derivative of the gradiant*

d\_by\_da **=** grad(loss,2)

d\_by\_db **=** grad(loss,3)

*# Learning rate is how much the algorithm with move*

*# Smaller value is more accurate but will take more attempts*

*# Larger is faster but may miss local min*

learning\_rate **=** 0.0001

*# Same as epochs, how many times to run the algorithm*

maximum\_number\_of\_iterations **=** 1000

ssr **=** []

*# Iterate through data and calulate the loss function with learning rate at current time*

**for** iter **in** range(maximum\_number\_of\_iterations):

a **-=** learning\_rate**\***d\_by\_da(x,y\_obs,a,b)

b **-=** learning\_rate**\***d\_by\_db(x,y\_obs,a,b)

y\_model **=** linear(x,a,b)

ssr**.**append(loss(x,y\_obs,a,b))

print('Best a and b are ',a,b)

print('Best loss function is ',loss(x,y\_obs,a,b))

*# Plot the graph*

plt**.**subplot(1,2,1)

plt**.**scatter(x,y\_obs)

plt**.**plot(x,y\_model)

plt**.**subplot(1,2,2)

plt**.**plot(ssr)

plt**.**show()

First guess at a and b are 2.0 1.0

First loss function is 500.15

Best a and b are 0.9390198950485866 0.47419724324276397

Best loss function is 1.3416095642628225

**Logistical Regression:**

**from** autograd **import** grad

**import** matplotlib.pyplot **as** plt

*# Sigmoid finction that returns output between 0 and 1 (probability)*

**def** logistic(x,a,b):

**return** 1**/**(1.0**+**np**.**exp(**-**a**\***x**+**b))

*# Loss function takes x and y\_obs arrays, compares to y\_model from logistic function to return SSR*

**def** loss(x,y\_obs,a,b):

y\_model **=** logistic(x,a,b)

**return** np**.**sum( (y\_model**-**y\_obs)**\*\***2 )

*# Arrays for logistic and loss functions*

x **=** np**.**array( [0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,2.50,2.75,3.00,3.25,3.50,4.00,4.25,4.50,4.75,5.00,5.50] )

y\_obs **=** np**.**array( [0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1] )

*# Assign weights for prediction randomly*

a,b **=** 5.0,3.5

y\_model **=** logistic(x,a,b)

print('First guess at a, b ',a,b)

print('First loss function is ',loss(x,y\_obs,a,b))

*# Autograd used to get the derivative of the gradiant*

d\_by\_da **=** grad(loss,2)

d\_by\_db **=** grad(loss,3)

*# Learning rate is how much the algorithm with move*

*# Smaller value is more accurate but will take more attempts*

*# Larger is faster but may miss local min*

learning\_rate **=** 0.001

*# Same as epochs, how many times to run the algorithm*

maximum\_number\_of\_iterations **=** 50000

ssr **=** []

*# Iterate through data and calulate the loss function with learning rate at current time*

**for** iter **in** range(maximum\_number\_of\_iterations):

a **-=** learning\_rate**\***d\_by\_da(x,y\_obs,a,b)

b **-=** learning\_rate**\***d\_by\_db(x,y\_obs,a,b)

y\_model **=** logistic(x,a,b)

ssr**.**append(loss(x,y\_obs,a,b))

print('Best a,b is ',a,b)

print('Best loss function is ',loss(x,y\_obs,a,b))

*# Plot the graph*

plt**.**subplot(1,2,1)

plt**.**scatter(x,y\_obs)

plt**.**plot(x,y\_model)

plt**.**subplot(1,2,2)

plt**.**plot(ssr)

plt**.**show()