Implementation of SGDRegressor from Scratch:

In [ ]:
```
#Using the Boston data set from the Sklearn Library :
```

In [29]:
```
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.cross_validation import train_test_split
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

boston = load_boston()
```

In [30]:
```
#Loading the data from the boston dataset:
data = boston.data #featured data
target = boston.target #variable values


boston_df = pd.DataFrame(data)
X = boston_df
y = target
```

In [31]:
```
# Standardize the data (mean=0, std=1) using training data
X_scaler = StandardScaler().fit(X)
standardized_X = X_scaler.transform(X)
```

Implementing the SGD Model :

In [32]:
```
# The below function will compute the cost for each point:
def  cal_cost(theta,X,y):
    m = len(y)

    predictions = X.dot(theta)

    cost = (1/2*m) * np.sum(np.square(predictions-y))

    return cost
```

In [227]:
```
#The below method will compute the optimal weights and cost :
def stochastic_gradient_descent(X,y,theta,learning_rate=0.01,iterations=10):

    m = len(y) #length of the data set
    cost_value = np.zeros(iterations)


    for it in range(iterations):
        cost =0.0

        for i in range(m):
            rand_ind = np.random.randint(0,m)
            X_i = X[rand_ind,:].reshape(1,X.shape[1])
            y_i = y[rand_ind].reshape(1,1)
            prediction = np.dot(X_i,theta)

            theta = theta -(2/m)*learning_rate*( X_i.T.dot((prediction - y_i)))
            cost += cal_cost(theta,X_i,y_i)


        cost_value[it]  = cost

    return theta, cost_value
```

In [228]:
```
lr =0.2 #learning_rate
n_iter = 100 #no. of iterations

theta = np.random.randn(14,1)

X_b = np.c_[np.ones((len(standardized_X),1)),standardized_X] #adding the bias weig

theta_updated,cost_history = stochastic_gradient_descent(X_b,y,theta,lr,n_iter) #
```

In [229]:
```
print('Intercept Term(bias term) :  {:0.3f}\n'.format(theta_updated[0][0]))

print('*'*100)
print('Predicted Weights(without bias term :)')
weight_vector = theta_updated[1:]
print(weight_vector)
```

```
Intercept Term(bias term) :  22.555

****************************************************************************************************
Predicted Weights(without bias term :)
[[-0.9638304 ]
 [ 0.9149475 ]
 [-0.01600848]
 [ 0.56743312]
 [-2.08737184]
 [ 2.80564371]
 [ 0.1813708 ]
 [-3.09703528]
 [ 2.45499635]
 [-1.75823064]
 [-2.11377119]
 [ 0.80181488]
 [-3.9207029 ]]
```

In [230]:
```
y_predicted = X_b.dot(theta_updated)

y_predicted = y_predicted.ravel()
error = y -y_predicted
mse_sgd = np.mean(error**2)
print('Mean Squared Error using the predicted weights :' , mse_sgd)
```

```
Mean Squared Error using the predicted weights : 21.991174425443887
```

Sklearn's SGDRegressor :

In [231]:
```
#Computing the intercept and weight coffecients using the sklearn library:

lm = SGDRegressor(n_iter = 100 , penalty=None , loss='squared_loss' )
lm.fit(standardized_X , y)
```

Out[231]:
```
SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', max_iter=None, n_iter=100, penalty=None,
       power_t=0.25, random_state=None, shuffle=True, tol=None, verbose=0,
       warm_start=False)
```

In [232]:
```
#Intercept Calculation:
print('Intercept term :' , lm.intercept_ )
```

```
Intercept term : [22.53564532]
```

In [233]:
```
#Coffecient Calculation :
print('Weight vector :\n' , lm.coef_.reshape(-1 , 1))
```

```
Weight vector :
 [[-0.91026934]
 [ 1.05890443]
 [ 0.11355234]
 [ 0.67833332]
 [-2.04260332]
 [ 2.65719259]
 [ 0.02343965]
 [-3.12238977]
 [ 2.55149977]
 [-1.9320931 ]
 [-2.05428794]
 [ 0.84663064]
 [-3.73052709]]
```

In [234]:
```
#Predicting the target values for standardised data :

y_predict = lm.predict(standardized_X)
y_predict.shape
```

Out[234]:
```
(506,)
```

In [235]:
```
#mean square error

mse_sklearn = np.mean((y - y_predict)**2)
print('MSE:' , mse_sklearn)
```

```
MSE: 21.90851533259455
```

Comparing our SGD model to Sklearn's SGDRegressor :

Bias Term (Intercept Term) :

In [236]:
```
intercept = np.hstack([theta_updated[0][0] , lm.intercept_])
intercept
```

Out[236]:
```
array([22.55490109, 22.53564532])
```

Weight Vector :

In [237]:
```
weight = np.hstack([weight_vector , lm.coef_.reshape(-1 ,1)])
weight
```

Out[237]:
```
array([[-0.9638304 , -0.91026934],
       [ 0.9149475 ,  1.05890443],
       [-0.01600848,  0.11355234],
       [ 0.56743312,  0.67833332],
       [-2.08737184, -2.04260332],
       [ 2.80564371,  2.65719259],
       [ 0.1813708 ,  0.02343965],
       [-3.09703528, -3.12238977],
       [ 2.45499635,  2.55149977],
       [-1.75823064, -1.9320931 ],
       [-2.11377119, -2.05428794],
       [ 0.80181488,  0.84663064],
       [-3.9207029 , -3.73052709]])
```

MSE Comparison:

In [238]:
```
mse_array = np.hstack([mse_sgd , mse_sklearn])
mse_array
```

Out[238]:
```
array([21.99117443, 21.90851533])
```

CONCLUSION:

1) We got same approximately same values for the weight vector , intercept term and Mean Squared Error values in case of SGD from scratch and Sklearn SGD model.

2) So our model is working fine with learning rate(eat0) = 0.2 and n_iter = 100.

In [ ]:
```

```