Implement SGD

June 18, 2019

```
In [ ]: ## reference link : https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-r

In [23]: # Imported necessary libraries
         from sklearn.datasets import load_boston
         from sklearn.model_selection import train_test_split
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.metrics import mean_squared_error
         from sklearn import preprocessing
         from sklearn.linear_model import SGDRegressor

         # Data loaded
         bostan = load_boston()

         # Data shape
         bostan.data.shape

Out[23]: (506, 13)

In [24]: # Feature name
         bostan.feature_names

Out[24]: array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
                'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')

In [25]: # This is y value i.e. target
         bostan.target.shape

Out[25]: (506,)

In [26]: # Convert it into pandas dataframe
         data = pd.DataFrame(bostan.data, columns = bostan.feature_names)
         data.head()

Out[26]:       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
         0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
         1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
```

```
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

In [27]:
```python
# Statistical summary
data.describe()
```

Out[27]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT
count  506.000000
mean    12.653063
std      7.141062
min      1.730000
25%      6.950000
50%     11.360000
75%     16.955000
max     37.970000
```

In [28]:
```python
#standardize for fast convergence to minima
data = (data - data.mean())/data.std()
data.head()
```

Out[28]:
```
       CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
0 -0.419367  0.284548 -1.286636 -0.272329 -0.144075  0.413263 -0.119895
```

```
1 -0.416927 -0.487240 -0.592794 -0.272329 -0.739530  0.194082  0.366803
2 -0.416929 -0.487240 -0.592794 -0.272329 -0.739530  1.281446 -0.265549
3 -0.416338 -0.487240 -1.305586 -0.272329 -0.834458  1.015298 -0.809088
4 -0.412074 -0.487240 -1.305586 -0.272329 -0.834458  1.227362 -0.510674

        DIS       RAD       TAX   PTRATIO         B     LSTAT
0  0.140075 -0.981871 -0.665949 -1.457558  0.440616 -1.074499
1  0.556609 -0.867024 -0.986353 -0.302794  0.440616 -0.491953
2  0.556609 -0.867024 -0.986353 -0.302794  0.396035 -1.207532
3  1.076671 -0.752178 -1.105022  0.112920  0.415751 -1.360171
4  1.076671 -0.752178 -1.105022  0.112920  0.440616 -1.025487
```

In [29]:
```python
# MEDV(median value is usually target), change it to price
data["PRICE"] = bostan.target
data.head()
```

Out[29]:
```
       CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
0 -0.419367  0.284548 -1.286636 -0.272329 -0.144075  0.413263 -0.119895
1 -0.416927 -0.487240 -0.592794 -0.272329 -0.739530  0.194082  0.366803
2 -0.416929 -0.487240 -0.592794 -0.272329 -0.739530  1.281446 -0.265549
3 -0.416338 -0.487240 -1.305586 -0.272329 -0.834458  1.015298 -0.809088
4 -0.412074 -0.487240 -1.305586 -0.272329 -0.834458  1.227362 -0.510674

        DIS       RAD       TAX   PTRATIO         B     LSTAT  PRICE
0  0.140075 -0.981871 -0.665949 -1.457558  0.440616 -1.074499   24.0
1  0.556609 -0.867024 -0.986353 -0.302794  0.440616 -0.491953   21.6
2  0.556609 -0.867024 -0.986353 -0.302794  0.396035 -1.207532   34.7
3  1.076671 -0.752178 -1.105022  0.112920  0.415751 -1.360171   33.4
4  1.076671 -0.752178 -1.105022  0.112920  0.440616 -1.025487   36.2
```

In [30]:
```python
# Target and features
Y = data["PRICE"]
X = data.drop("PRICE", axis = 1)
```

In [31]:
```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)

x_train["PRICE"] = y_train
```

```
(354, 13) (152, 13) (354,) (152,)


/home/pranay/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  """
```

0.1 Custome SGD Implementation

```
In [32]: def gradient_decent(w0, b0, points, x_test, y_test, learning_rate):
             iterations = 1000
             gradient_m = 0
             gradient_b = 0
             cost_train = []
             cost_test = []
             for j in range(1, iterations):

                 # Random Train
                 train_sample = points.sample(160)
                 y = np.asmatrix(train_sample["PRICE"])
                 x = np.asmatrix(train_sample.drop("PRICE", axis = 1))

                 for i in range(len(x)):
                     dot_prod = np.dot(x[i] , w0) + b0
                     substract_term =y[:,i] - dot_prod
                     gradient_m += np.dot(-2*x[i].T , substract_term)
                     gradient_b += -2*(substract_term)
         #               print(substract_term.shape)

                 w1 = w0 - learning_rate * gradient_m
                 b1 = b0 - learning_rate * gradient_b

                 if (w0==w1).all():
                     break
                 else:
                     w0 = w1
                     b0 = b1
                     learning_rate = learning_rate/2

             return w0, b0

         def mse_metric(b, m, features, target):
             totalError = 0
             for i in range(0, len(features)):
                 x = features
                 y = target
                 totalError += (y[:,i] - (np.dot(x[i] , m) + b)) ** 2
             return totalError / len(x)

In [33]: learning_rate = 0.001
         w0_random = np.random.rand(13)
         w0 = np.asmatrix(w0_random).T
         b0 = np.random.rand()

         optimal_w, optimal_b = gradient_decent(w0, b0, x_train, x_test, y_test, learning_rate
         print("Coefficient: {} \n y_intercept: {}".format(optimal_w, optimal_b))
```

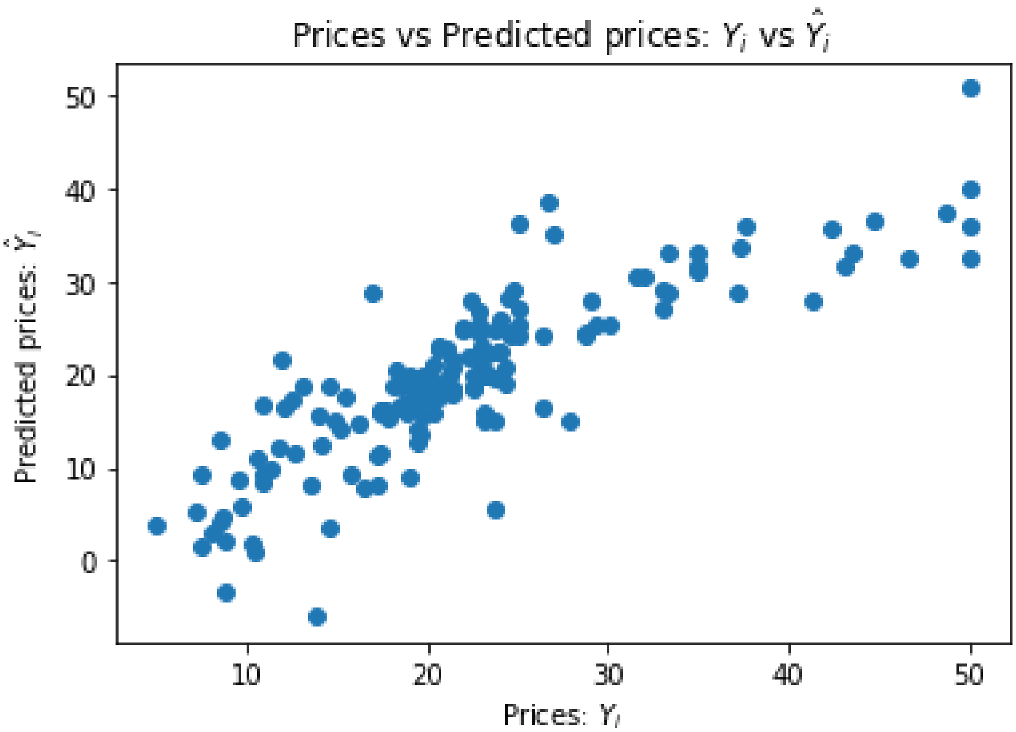
```
Coefficient: [[-0.68292829]
 [ 1.13355708]
 [-0.13956617]
 [ 3.11930798]
 [-0.24116852]
 [ 4.15839783]
 [ 0.28884544]
 [-0.66286099]
 [ 0.16888711]
 [-0.34984621]
 [ 0.06535574]
 [ 1.62262885]
 [-3.5635532 ]]
 y_intercept: [[20.65043429]]
```

```
In [34]: # Implemented SGD
         # The mean squared error
         mse_error = mse_metric(optimal_b, optimal_w, np.asmatrix(x_test), np.asmatrix(y_test))
         print("Custom SGD Mean squared error: %.2f" % mse_error)
```

```
Custom SGD Mean squared error: 34.08
```

```
In [35]: y_pred= [(np.dot(np.asmatrix(x_test), optimal_w) + optimal_b)]

         # manual_error=plot_(y_pred)
         plt.scatter(y_test, y_pred)
         plt.xlabel("Prices: $Y_i$")
         plt.ylabel("Predicted prices: $\hat{Y}_i$")
         plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
         plt.show()
```



0.2 SKLearn SGD Implementation

In [36]:
```
X = load_boston().data
Y = load_boston().target

df=pd.DataFrame(X)
df.head()

# standardise data
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)
df=pd.DataFrame(X)

df['price']=Y
df.head()
```

Out[36]:

	0	1	2	3	4	5	6 \
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180

```
            7         8         9        10        11        12  price
0  0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562   24.0
1  0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439   21.6
2  0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727   34.7
3  1.077737 -0.752922 -1.106115  0.113032  0.416163 -1.361517   33.4
4  1.077737 -0.752922 -1.106115  0.113032  0.441052 -1.026501   36.2
```

In [37]:
```python
# standardise data
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)
df=pd.DataFrame(X)

df['price']=Y
df.head()
```

Out[37]:
```
          0         1         2         3         4         5         6  \
0 -0.419782  0.284830 -1.287909 -0.272599 -0.144217  0.413672 -0.120013
1 -0.417339 -0.487722 -0.593381 -0.272599 -0.740262  0.194274  0.367166
2 -0.417342 -0.487722 -0.593381 -0.272599 -0.740262  1.282714 -0.265812
3 -0.416750 -0.487722 -1.306878 -0.272599 -0.835284  1.016303 -0.809889
4 -0.412482 -0.487722 -1.306878 -0.272599 -0.835284  1.228577 -0.511180

            7         8         9        10        11        12  price
0  0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562   24.0
1  0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439   21.6
2  0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727   34.7
3  1.077737 -0.752922 -1.106115  0.113032  0.416163 -1.361517   33.4
4  1.077737 -0.752922 -1.106115  0.113032  0.441052 -1.026501   36.2
```

In [38]:
```python
# Split data into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_sta
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339,)
(167,)
```

In [39]:
```python
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score
clf = SGDRegressor()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)

print("Coefficients: \n", clf.coef_)
print("Y_intercept", clf.intercept_)
```

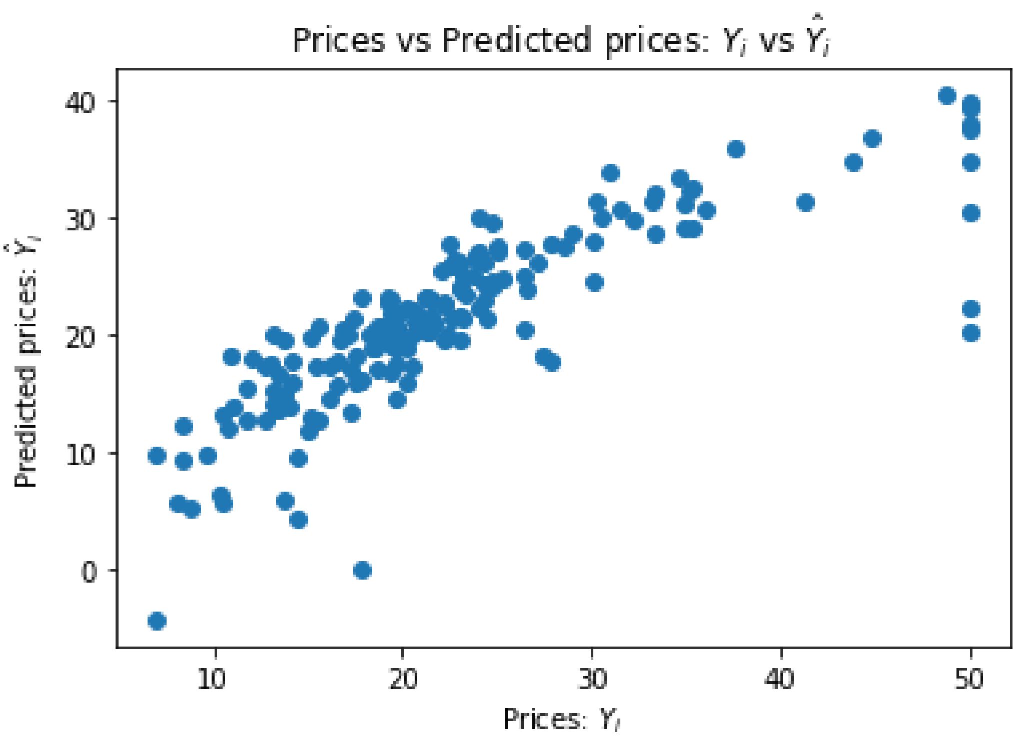
```
Coefficients:
 [-0.9266483   0.19881195 -0.6338864   0.27567329 -0.45419415  3.06297705
 -0.40894034 -1.82317767  1.04266262 -0.41628519 -1.98692311  0.97787009
 -3.06916569]
Y_intercept [21.75067408]
```

```
/home/pranay/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gradient.py
  FutureWarning)
```

In [40]:
```
# Sklearn SGD
# The mean squared error
print("SKLearn Mean squared error: %.2f" % mean_squared_error(Y_test, Y_pred))
```

```
SKLearn Mean squared error: 30.59
```

In [41]:
```
plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```



```
In [45]: import pandas as pd
         from prettytable import PrettyTable

         x = PrettyTable()
         x.field_names =  ['Metric','Sklearn','Custokm SGD']
         x.add_row(["MSE vales ", 30.59,34.08])


         print('\n')
         print(x)
```

```
+------------+---------+-------------+
|   Metric   | Sklearn | Custokm SGD |
+------------+---------+-------------+
| MSE vales  |  30.59  |    34.08    |
+------------+---------+-------------+
```

0.2.1 final weight from Custom SGD

```
In [43]: print("Coefficient: {} \n y_intercept: {}".format(optimal_w, optimal_b))
```

```
Coefficient: [[-0.68292829]
 [ 1.13355708]
 [-0.13956617]
 [ 3.11930798]
 [-0.24116852]
 [ 4.15839783]
 [ 0.28884544]
 [-0.66286099]
 [ 0.16888711]
 [-0.34984621]
 [ 0.06535574]
 [ 1.62262885]
 [-3.5635532 ]]
 y_intercept: [[20.65043429]]
```

0.2.2 final weight from SGD Sklearn

```
In [44]: print("Coefficients: \n", clf.coef_)
         print("Y_intercept", clf.intercept_)
```

```
Coefficients:
 [-0.9266483   0.19881195 -0.6338864   0.27567329 -0.45419415  3.06297705
 -0.40894034 -1.82317767  1.04266262 -0.41628519 -1.98692311  0.97787009
 -3.06916569]
```

```
Y_intercept [21.75067408]
```