LinearRegression

January 2, 2019

1 SGDREGRESSOR VS IMPLEMENTED_SGD

1.1 About the dataset-

1.2 Boston House Prices dataset

1.3 Notes

Data Set Characteristics:

```
:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive

:Median Value (attribute 14) is usually the target

:Attribute Information (in order):
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    - NOX      nitric oxides concentration (parts per 10 million)
    - RM       average number of rooms per dwelling
    - AGE      proportion of owner-occupied units built prior to 1940
    - DIS      weighted distances to five Boston employment centres
    - RAD      index of accessibility to radial highways
    - TAX      full-value property-tax rate per $10,000
    - PTRATIO  pupil-teacher ratio by town
    - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    - LSTAT    % lower status of the population
    - MEDV     Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.
```

This is a copy of UCI ML housing dataset. http://archive.ics.uci.edu/ml/datasets/Housing

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression problems.

References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
- many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)

1.3.1 DataSet: Boston Housing dataset

```
In [1194]: from sklearn.datasets import load_boston
           boston = load_boston()

In [1195]: boston.target.shape

Out[1195]: (506,)

In [1196]: print(boston.data.shape)

(506, 13)
```

1.4 #columns in dataset

```
In [1197]: print(boston.feature_names)

['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']


In [1198]: import pandas as pd
           bos = pd.DataFrame(boston.data)

In [1200]: X = bos
           Y = pd.DataFrame(boston.target)

In [1203]: import sklearn.cross_validation
           X_train, X_test, Y_train, Y_test = sklearn.cross_validation.train_test_split(X, Y, t
           print(X_train.shape)
           print(X_test.shape)
           print(Y_train.shape)
           print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339, 1)
(167, 1)
```

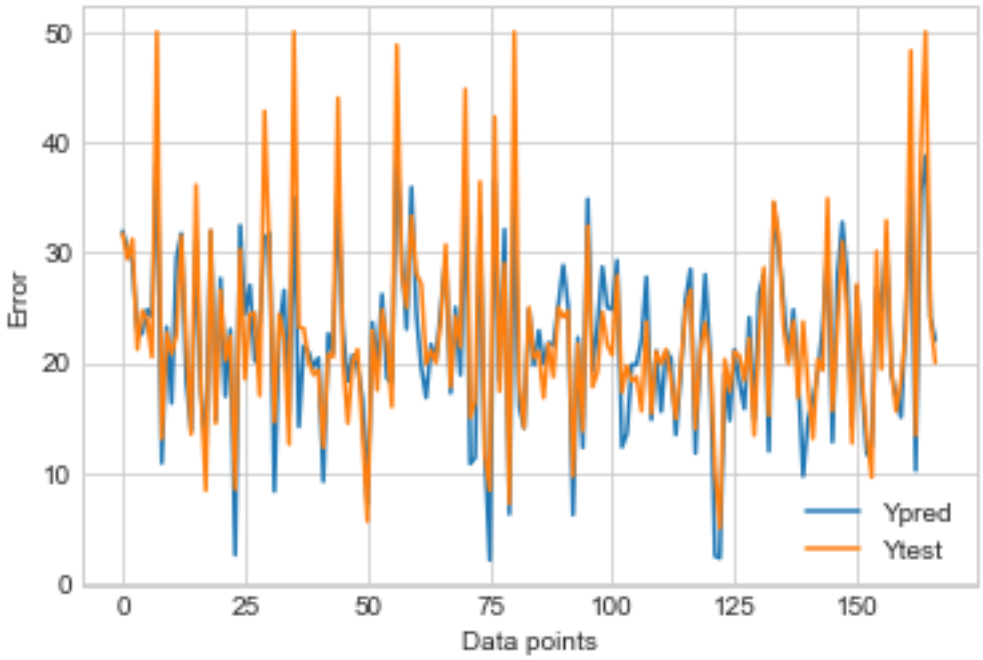
In [1204]:
```
#standardizing Xtest and Xtrain
from sklearn.preprocessing import StandardScaler
X_test= StandardScaler().fit_transform(X_test)
```

In [1205]:
```
from sklearn.preprocessing import StandardScaler
X_train= StandardScaler().fit_transform(X_train)
```

In [1206]:
```
# code source:https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regr
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
lm = SGDRegressor(eta0=0.01,learning_rate='constant',shuffle=False,max_iter=5000,to
lm.partial_fit(X_train, Y_train)

Y_pred = lm.predict(X_test)
```

```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\sklearn\utils\validati
  y = column_or_1d(y, warn=True)
```

In [1207]:
```
import numpy as np
plt.plot(Y_pred)
plt.plot(np.array(Y_test[0]))# plotting by columns
plt.xlabel("Data points")
plt.ylabel("Error")
plt.legend(('Ypred','Ytest'))
plt.show()
```

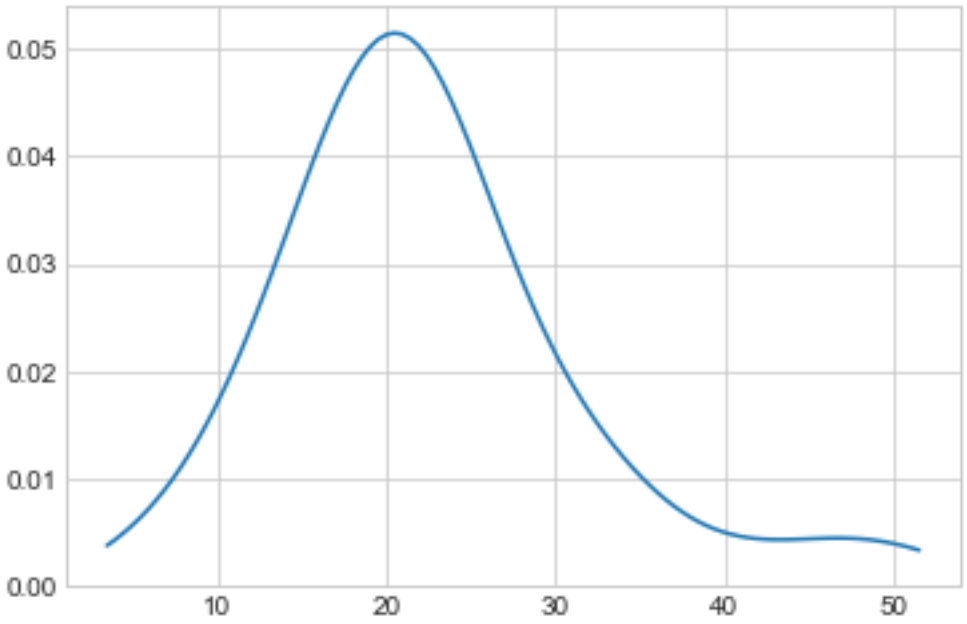



```
In [1208]: print("Actual Y's: \n {}".format(np.array(Y_test[0])))

Actual Y's:
 [31.7 29.4 31.2 21.2 24.7 23.9 20.5 50.  13.1 22.7 20.8 22.5 31.5 21.8
 13.5 36.1 17.5  8.4 32.  14.5 26.6 20.2 22.4  8.5 30.3 18.5 24.4 24.5
 17.  42.8 29.8 14.6 24.4 22.  12.6 50.  23.2 23.1 20.1 18.9 19.3 12.3
 20.8 20.5 44.  20.3 14.5 20.1 21.2 13.8  5.6 22.9 17.5 24.8 22.  16.
 48.8 28.2 25.  33.3 28.1 27.1 19.9 21.1 20.  23.8 30.7 17.8 24.2 21.4
 44.8 15.  16.5 36.4 11.7  8.4 42.3 17.4 29.   7.2 50.  18.1 14.1 25.
 20.4 21.  16.8 21.7 18.7 25.  24.1 24.7  9.7 21.7 13.8 32.4 17.8 19.2
 24.6 21.6 20.7 27.9 17.3 19.7 18.3 18.7 15.6 23.7 15.4 21.  19.9 21.2
 18.6 14.9 19.6 24.8 26.6 14.  21.1 23.6 20.4 10.2  5.  20.3 17.4 21.
 20.6 18.4 22.2 13.4 22.6 28.6 15.2 34.6 31.5 23.3 19.9 23.8 16.8 23.7
 17.8 13.1 20.3 19.3 34.9 15.6 23.3 31.  24.5 12.7 27.1 18.7 13.5  9.6
 30.1 19.4 32.9 19.6 15.6 18.4 23.  48.3 13.4 38.7 50.  24.1 20. ]
```

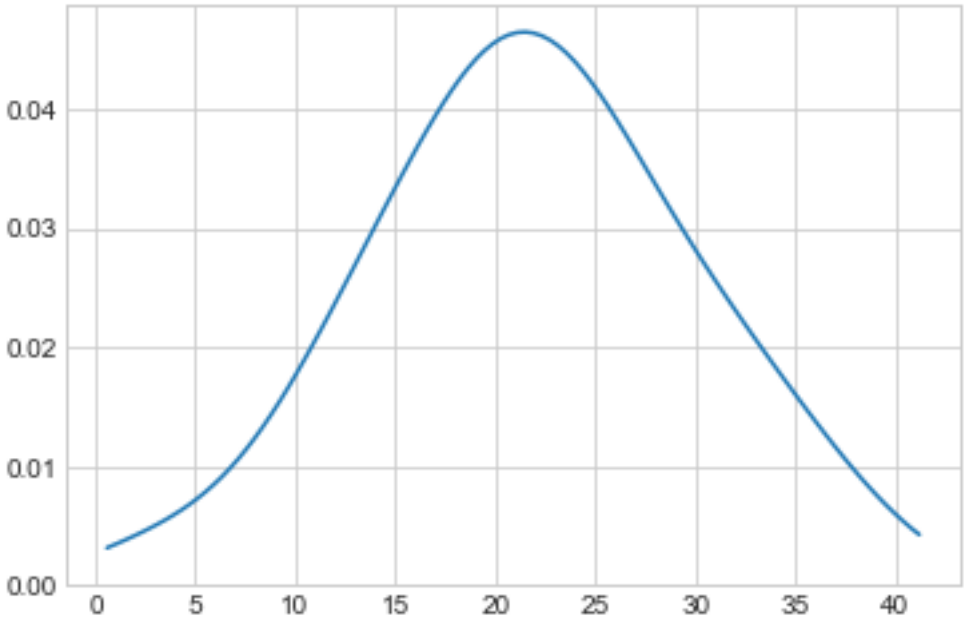
```
In [1209]: print("Predicted Y's: \n {}".format(np.array(Y_pred)))

Predicted Y's:
 [31.93177828 30.25250687 29.0947738  22.89772604 22.59926014 24.81852673
 24.17420754 39.66862998 10.88409381 23.22787569 16.31345145 29.46986572
 31.71606776 18.29513005 13.82594264 30.94916409 17.21985237 11.98324433
 31.93027855 14.77003876 27.64495539 16.94242611 23.023784    2.54898506
 32.4544021  24.3984297  27.01616544 20.13815384 23.58045048 31.31529822
 31.74652526  8.32360996 23.00945164 26.56565183 15.21821704 34.94116086
```

```
 14.1894343  21.5035387  20.94169013 19.61280998 20.4266217   9.22780061
 22.62848252 20.86052237 35.28705133 22.94930677 18.31908662 20.68239831
 19.62117321 16.69211698  7.20503749 23.63533489 19.70217029 26.19252198
 18.58819333 18.15362967 39.37696554 29.8765317  23.0378004  35.90729427
 24.41678619 19.59185713 16.82511888 21.66338354 20.23542388 24.99870173
 29.56964812 17.24402814 25.01849611 18.87761572 36.24839662 10.82148109
 11.39545002 33.41278198 11.82000995  2.08720101 36.42122131 18.374965
 32.08523296  6.26331404 34.76400123 16.03121951 14.07216782 24.85520042
 19.76562891 22.90249916 19.90433171 21.80677553 21.57684562 25.24175157
 28.82842795 25.08690259  6.202476   22.27942526 12.29354811 34.82434276
 19.23559121 23.30887097 28.6875801  25.00823628 24.82908198 29.27041148
 12.34542258 13.57157978 19.81183755 19.84029179 22.11029335 27.75659736
 14.8237627  21.07982097 15.57484247 21.02142094 20.39405885 13.45974767
 18.94132498 25.74059838 28.48369665 11.7909054  21.19904806 27.98889385
 20.47772194  2.46111156  2.26355076 19.06470977 14.75102604 21.18066773
 18.20105049 15.81364282 24.10414535 14.75569613 26.25812502 28.03407004
 11.96856989 34.38458608 30.14833083 25.55374967 20.12707808 24.80953347
 18.63779842  9.69856313 14.90366498 16.56116303 19.13974728 23.15713454
 30.44309679 12.7984194  27.92377018 32.74540369 27.82718763 14.28225613
 26.88902518 17.93249412 11.65784292 12.49147203 26.74300022 24.57869221
 30.99948361 19.07070232 16.79107067 15.04327824 24.33790303 35.63876473
 10.22088724 35.19214425 38.72830696 24.09650298 22.03961759]
```

```
In [1210]: import numpy as np
           delta_y = np.array(Y_test[0]) - np.array(Y_pred)

In [1211]: mse=0
           for i in delta_y:
               mse+=i*i
           print("MSE from SGDRegressor: {}".format(mse/delta_y.shape[0]))

MSE from SGDRegressor: 17.010850768557532

In [1212]: import seaborn as sns

In [1213]: sns.set_style('whitegrid')
           sns.kdeplot(np.array(Y_test[0]), bw=0.5)
           plt.show()
```



In [1214]:
```
sns.set_style('whitegrid')
sns.kdeplot(Y_pred, bw=0.5)
plt.show()
```



In [1215]:
```
print("SGDRegressor weights: \n{}".format(lm.coef_))
```

```
SGDRegressor weights:
[-1.16060011  0.4312397  -0.70667846  0.71829374 -0.99024065  2.25379444
 -0.06380014 -1.71951966  0.51487848 -0.65662334 -1.91688281  0.58323157
 -3.85489385]
```

```
In [1216]: lm.intercept_
```

```
Out[1216]: array([21.78751453])
```

2 Implemented_SGD

```
In [1274]: def del2(w,b):
               """This function calcuate (X^t(Y-(X.W+b)))"""
               del_2=np.zeros((1,13))
               #abc=X.sample(n=50)
               Y_ = np.array(Y_train[0])
               X_ = np.array(X_train)
               a=np.dot(np.array(X_),w)
               b=np.array(a+b)
               #print(a)
               #print(b)
               del_2[0]=np.dot(np.transpose(-np.array(X_)),(np.array(Y_)-b))
               return del_2[0]/X_train.shape[0]
```

```
In [1275]: #b0 = b0 - alpha* 2* avg of (y-(w0*x+b0))
           def delb(b):
               """This function will calculate (-(Y-(X.W+b)))"""
               #del_1=0
               Y_ = np.array(Y_train[0])
               X_ = np.array(X_train)
               b=np.sum(-(np.array(Y_)-np.array(np.dot(np.array(X_),w)+b)))
               #print(del_1/X_train.shape[0])
               return b/X_train.shape[0]
```

```
In [1297]: w=np.random.rand(1,13)

           w[0]
```

```
Out[1297]: array([0.53853973, 0.08071966, 0.59446326, 0.49770104, 0.98197586,
                  0.41413791, 0.54624655, 0.8088257 , 0.53140931, 0.83771029,
                  0.41807082, 0.02875459, 0.51661372])
```

```
In [1298]: #b=w[0][0]
           b=0
```

```
In [1299]: w=w[0]
```

```
In [1300]: from tqdm import tqdm
```

In [1301]:
```
#    Now we are calculating best w
count=0
ab=[]
r=0.001
ith=0
#b=2*r*np.linalg.norm(delb(b))
for i in tqdm(range(1,5000)):
    if i==1:

        ab.append(w-(2*r*del2(w,b)))
        b=b-2*r*delb(b)

    else:

        ab.append(ab[i-2]-(2*r*del2(ab[i-2],b)))
        b=b-2*r*delb(b)
        if (np.linalg.norm(ab[i-1] - ab[i-2]))< 0.001:
            print(i)
            ith=i-1
            break
```

```
 40%|                                   | 2013/4999 [00:00<00:00, 10038.25it/s]

2336
```

In [1302]:
```
W_=ab[ith]
W_.shape
```

Out[1302]: (13,)

In [1303]:
```
print("W* = {}".format(W_))
```

```
W* = [-0.57117119  0.44226734 -0.65803754  1.17408943 -1.00150116  2.7207344
  0.23478806 -1.87753641  1.01009135 -0.26862149 -1.89540966  0.66616235
 -3.9142166 ]
```

In [1315]:
```
print("Intercept term from ImplementedSGD: ",b)
```

```
Intercept term from ImplementedSGD:  22.327641873703747
```

In [1305]:
```
Y_=np.dot(X_test,W_)+b
```

In [1306]:
```
print("Predicted Y from ImplementedSGD: {}".format(Y_))
```

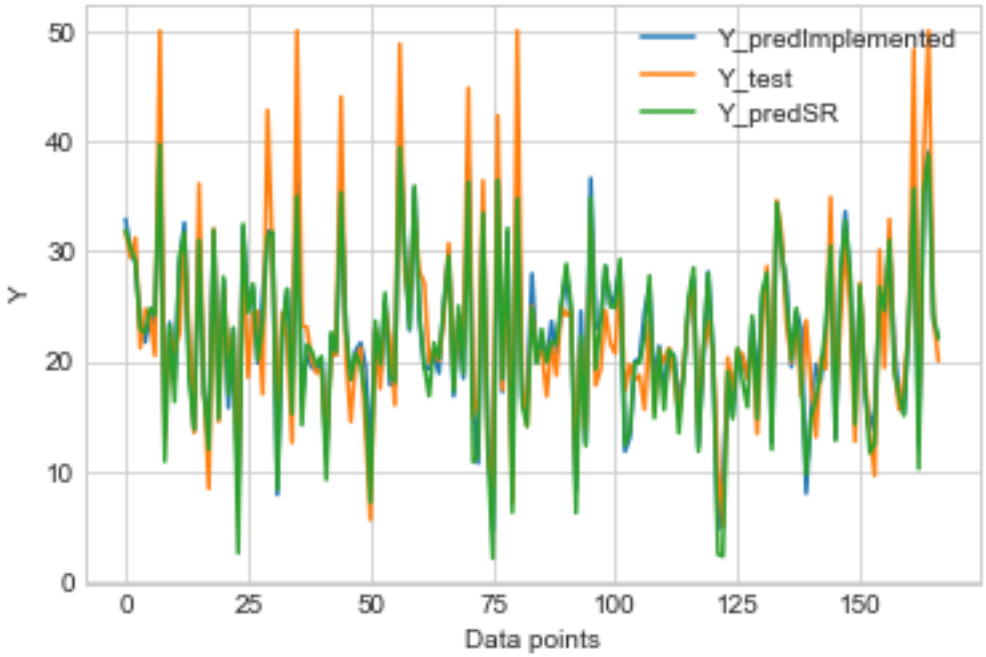
```
Predicted Y from ImplementedSGD: [32.89381297 29.91524946 28.91556265 23.12850535 21.7238574  2
 23.36800662 40.05357892 14.95129106 23.50588283 18.24329406 28.65146524
 32.56751444 19.75251188 13.68309692 31.08060472 16.90717039 15.00255001
 31.74725453 17.37127652 27.28836307 15.75093446 22.90106562  7.06967165
 32.07867029 24.85410641 26.62029257 19.80684775 26.11004003 31.84245549
 31.53252939  7.86344805 22.55161292 26.51805967 18.19650296 36.44372323
 16.51826817 20.6749109  19.47511282 19.13527882 20.24117804 11.55964259
 22.35527964 20.62646585 35.18694563 22.65245446 18.31081365 20.953624
 21.65948865 19.54638705 11.08326682 22.8456181  18.77490194 25.49521329
 17.8491156  17.87999015 40.96368869 30.09097301 22.78813918 35.71012267
 23.81329619 19.37895775 19.28552822 20.60628665 18.90156042 25.21552964
 30.02896277 16.82740696 24.17938908 18.43803663 37.90928075 16.25894663
 10.72276154 33.49912128 13.64968854  4.62841238 36.41975354 17.19079684
 31.6711007   9.62566699 35.20539807 16.68212816 16.4653003  27.93317206
 19.9594338  22.06252821 20.01573325 23.60643007 20.79743619 24.81625011
 27.97393859 23.79033484  8.38245602 24.53495548 12.59361783 36.60845263
 22.92218144 23.54900475 28.58511356 24.98711225 26.09118495 28.64307082
 11.79901435 13.07196419 19.97714081 20.22846275 24.4799929  26.84611904
 18.00436528 21.35820304 17.12156849 20.80913313 20.52669011 16.37763229
 18.90200417 24.99025238 27.56876362 12.22582483 20.05825223 28.09979037
 19.94702954  4.70834965  9.04769412 18.56985986 15.04780709 20.71781767
 20.09471795 18.51594754 23.58423077 17.49011428 25.68745408 27.55804546
 12.61190145 34.19573079 29.94109391 27.05712398 19.46963316 24.51106253
 22.93939263  7.99043165 15.43795819 19.67135028 18.19067191 22.32135659
 30.08268399 12.93328569 27.57617508 33.5812205  27.05583506 17.2664373
 26.5089756  17.20428198 13.75298358 15.61723988 25.67435374 23.44764421
 30.73038662 21.53702548 16.79728939 15.26112042 23.66625701 37.14501545
 12.41356271 36.18220903 39.01235713 23.49536415 22.14597559]


In [1307]: Y=np.array(Y_test[0])

In [1308]: print("Actual Y : {}".format(Y))

Actual Y : [31.7 29.4 31.2 21.2 24.7 23.9 20.5 50.  13.1 22.7 20.8 22.5 31.5 21.8
 13.5 36.1 17.5  8.4 32.  14.5 26.6 20.2 22.4  8.5 30.3 18.5 24.4 24.5
 17.  42.8 29.8 14.6 24.4 22.  12.6 50.  23.2 23.1 20.1 18.9 19.3 12.3
 20.8 20.5 44.  20.3 14.5 20.1 21.2 13.8  5.6 22.9 17.5 24.8 22.  16.
 48.8 28.2 25.  33.3 28.1 27.1 19.9 21.1 20.  23.8 30.7 17.8 24.2 21.4
 44.8 15.  16.5 36.4 11.7  8.4 42.3 17.4 29.   7.2 50.  18.1 14.1 25.
 20.4 21.  16.8 21.7 18.7 25.  24.1 24.7  9.7 21.7 13.8 32.4 17.8 19.2
 24.6 21.6 20.7 27.9 17.3 19.7 18.3 18.7 15.6 23.7 15.4 21.  19.9 21.2
 18.6 14.9 19.6 24.8 26.6 14.  21.1 23.6 20.4 10.2  5.  20.3 17.4 21.
 20.6 18.4 22.2 13.4 22.6 28.6 15.2 34.6 31.5 23.3 19.9 23.8 16.8 23.7
 17.8 13.1 20.3 19.3 34.9 15.6 23.3 31.  24.5 12.7 27.1 18.7 13.5  9.6
 30.1 19.4 32.9 19.6 15.6 18.4 23.  48.3 13.4 38.7 50.  24.1 20. ]


In [1309]: dela=Y-Y_
```

```
In [1310]: sqerror=0
           for i in dela:
               sqerror+=i**2
           print("MSE for ImplementedSGD: {}".format(sqerror/dela.shape[0]))
```

```
MSE for ImplementedSGD: 17.43963608653024
```

2.1 Comparing Y_predImplemented and Y_test and Y_predSGD

```
In [1311]: # Shape is now: (10, 80)
           plt.plot(Y_) # plotting by columns
           plt.plot(Y)
           plt.plot(Y_pred)
           plt.xlabel("Data points")
           plt.ylabel("Y")
           plt.legend(('Y_predImplemented','Y_test','Y_predSR'))
           plt.show()
```

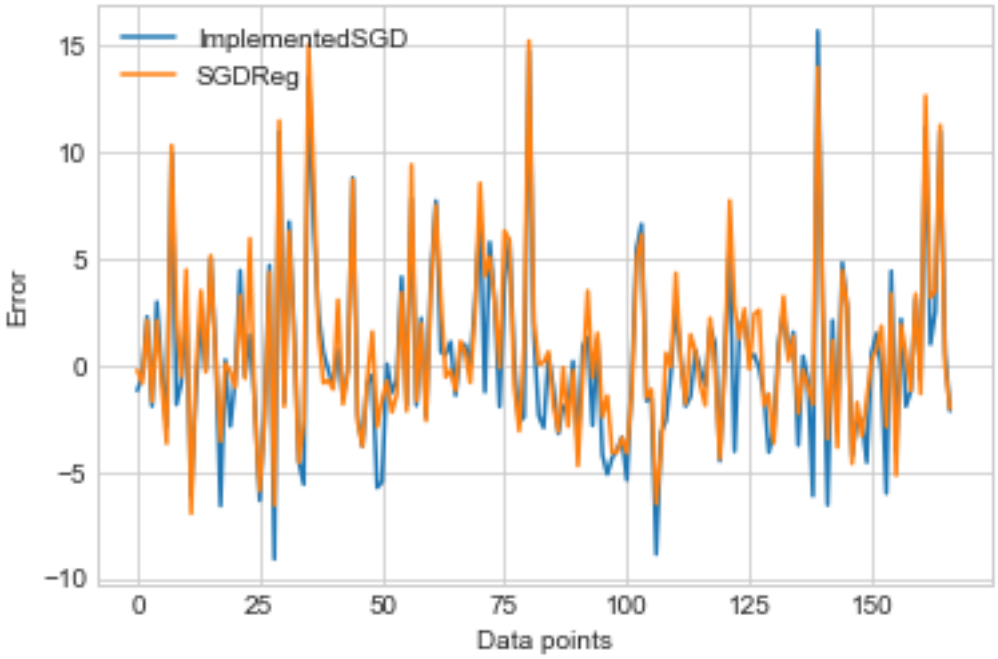


2.1.1 Observation: We can clearly see that Implemented SGD predicting better than SGDRegressor.

2.2 Comparing (Y-Y_hat) with implemented SGD vs SGDRegressor

```
In [1313]: plt.plot(dela)
           plt.plot(delta_y)# plotting by columns
```

```
plt.xlabel("Data points")
plt.ylabel("Error")
plt.legend(('ImplementedSGD','SGDReg'))
plt.show()
```

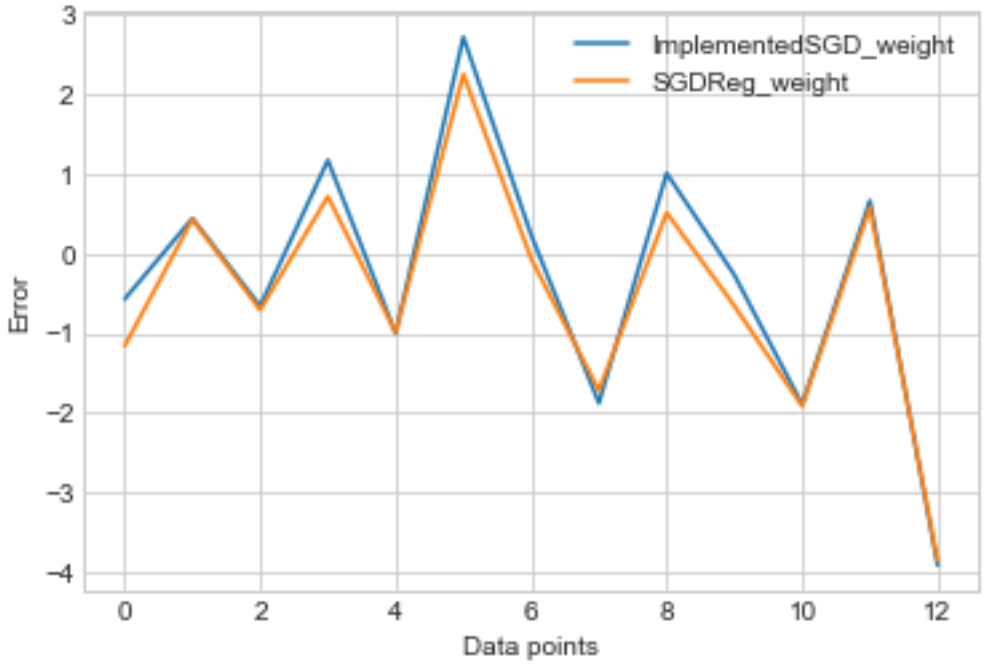


2.3 Comparing Weights get from Implemented SGD vs SGDRegressor

```
In [1314]: plt.plot(np.array(W_))
           print("Weight from ImplementedSGD")
           print(np.array(W_))
           print("Weight from SGDregressor")
           print(lm.coef_)
           plt.plot(lm.coef_)# plotting by columns
           plt.xlabel("Data points")
           plt.ylabel("Error")
           plt.legend(('ImplementedSGD_weight','SGDReg_weight'))
           plt.show()
```

```
Weight from ImplementedSGD
[-0.57117119  0.44226734 -0.65803754  1.17408943 -1.00150116  2.7207344
  0.23478806 -1.87753641  1.01009135 -0.26862149 -1.89540966  0.66616235
 -3.9142166 ]
Weight from SGDregressor
[-1.16060011  0.4312397  -0.70667846  0.71829374 -0.99024065  2.25379444
 -0.06380014 -1.71951966  0.51487848 -0.65662334 -1.91688281  0.58323157
```

```
-3.85489385]
```



3 Result:

```
M_SQ-loss from ImplemetedSGD= 17.43963608653024
M_SQ-loss from SGDRegressor= 17.010850768557532

Intercept Term from ImplementedSGD= 22.309236273545338
Intercept Term from SGDRegressor= 21.78751453
```