LinearRegression

December 29, 2018

1 SGDREGRESSOR VS IMPLEMENTED_SGD

1.0.1 DataSet: Boston Housing dataset

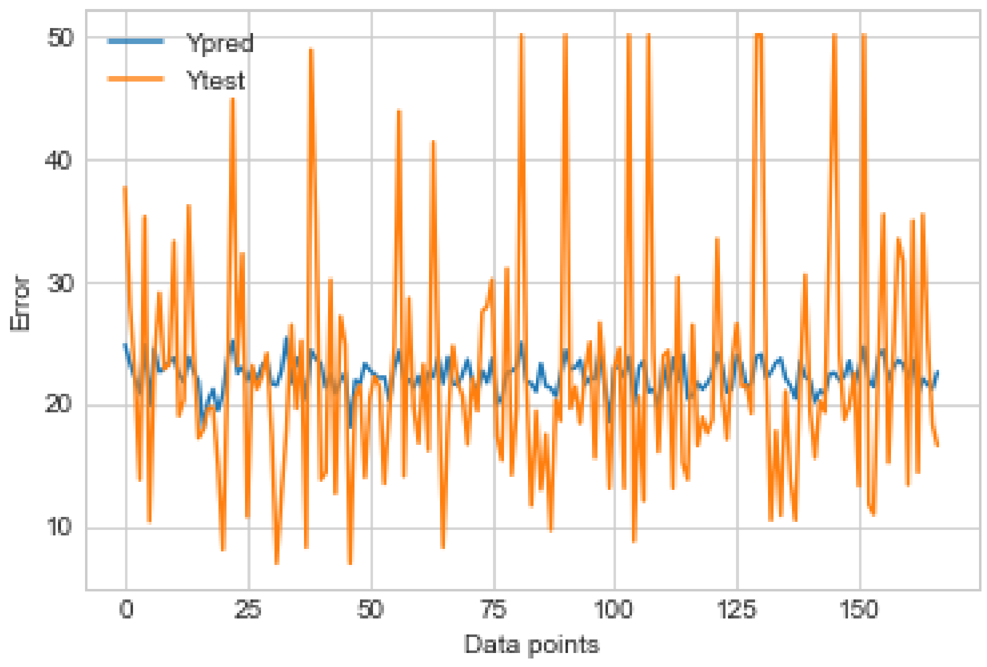
```
In [1259]: from sklearn.datasets import load_boston
           boston = load_boston()

In [1260]: boston.target.shape

Out[1260]: (506,)

In [1261]: print(boston.data.shape)

(506, 13)
```

1.1 #columns in dataset

```
In [1262]: print(boston.feature_names)

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

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

In [1264]: X = bos
           Y = pd.DataFrame(boston.target)

In [1265]: from sklearn.preprocessing import StandardScaler
           standardized_data = StandardScaler(with_mean=False).fit_transform(X)
           print(standardized_data.shape)

(506, 13)

In [1266]: X=pd.DataFrame(standardized_data)
In [1268]: X['Price']=Y

In [1270]: import sklearn.cross_validation
           X_train, X_test, Y_train, Y_test = sklearn.cross_validation.train_test_split(X.drop
           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 [ ]: # code source:https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regress
        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=0.0001,learning_rate='constant',shuffle=False,max_iter=25000,tol
        lm.partial_fit(X_train, Y_train)

        Y_pred = lm.predict(X_test)

In [1272]: 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 [1318]: print("Actual Y's: \n {}".format(np.array(Y_test[0])))

Actual Y's:
 [37.6 27.9 22.6 13.8 35.2 10.4 23.9 29.  22.8 23.2 33.2 19.  20.3 36.1
 24.4 17.2 17.9 19.6 19.7 15.   8.1 23.  44.8 23.1 32.2 10.8 23.1 21.2
 22.2 24.1 17.3  7.  12.7 17.8 26.4 19.6 25.1  8.3 48.8 34.9 13.8 14.4
 30.1 12.7 27.1 24.8  7.  20.5 21.5 14.  20.4 22.2 21.4 13.5 19.4 24.7
 43.8 14.1 28.6 19.7 16.8 23.2 16.2 41.3 22.7  8.3 18.4 24.7 21.7 20.6
 16.7 22.1 19.4 27.5 27.9 30.1 17.4 15.4 31.  14.2 19.6 50.  21.7 11.7
 19.4 13.  17.5  9.7 20.3 18.6 50.  19.6 21.4 18.4 22.6 25.  15.6 26.6
 22.4 13.1 23.  24.5 13.1 50.   8.8 20.6 12.1 50.  24.1 16.1 23.9 24.3
 13.1 30.3 15.2 13.8 26.4 16.6 18.9 17.6 18.7 33.4 20.7 17.1 23.4 26.5
 21.4 21.5 19.2 50.  50.  23.  10.5 17.8 10.9 21.  13.8 10.5 22.2 30.5
 19.4 15.6 20.2 19.3 34.6 50.  24.  18.7 19.8 22.5 13.3 50.  11.8 11.
 23.7 35.4 15.2 24.4 33.4 31.6 13.4 34.9 14.4 35.4 25.3 18.3 16.6]
```

```
In [1317]: print("Predicted Y's: \n {}".format(np.array(Y_pred)))

Predicted Y's:
 [24.81250199 23.27626783 22.03635716 20.8307045  24.80934552 19.85850698
 24.45441751 22.58717016 22.86767678 23.39546948 23.72556713 22.36963781
 21.78077926 23.77633643 22.65203726 21.84306532 17.95151249 20.24638391
 21.15350392 19.45105004 20.77988622 23.74871018 25.08948699 22.47568296
 22.99349803 21.87954708 22.80598495 21.89681755 23.09897931 23.51505984
 21.64046843 21.49128924 22.69395377 25.34928079 21.68475441 23.7148544
 22.70306782 20.34990431 24.33095167 23.65497768 23.25809181 21.29111808
 22.43379707 20.81960085 21.81149625 22.53887937 18.14258061 21.89981735
 21.62325851 23.24509558 22.72724538 22.35017335 22.08567872 22.15456314
 20.27421433 22.88107321 24.27062632 21.95722711 21.95665    21.28928768
 22.25500187 20.99027677 22.52905231 22.1519009  23.74000081 21.64003096
 23.7715496  21.65335298 21.50443114 22.40416648 23.58104698 21.86716405
 20.92832365 22.68738645 21.72609673 23.65611301 19.94990965 20.54242471
 22.51879087 22.61885046 23.00546958 24.94270689 21.81836931 21.63987955
 20.98674283 23.23853903 21.42751211 21.26506674 20.59753841 22.36886622
 24.3427314  22.81032275 22.92245855 23.54494079 21.52718938 22.13784375
 21.9653229  24.33606296 21.48028987 18.51593581 22.94131121 22.97638159
 22.14067413 23.80232928 19.90081934 22.93804683 23.47319688 20.93696472
 21.19233663 19.51263702 23.20554472 21.21347101 23.71009999 21.87714676
 23.99127132 20.38269957 20.68317653 21.67636624 21.16382803 21.68802682
 22.40945482 24.13470386 23.08390504 20.91368684 21.25848635 23.94034952
 23.20221368 21.27564578 21.70442221 23.82314988 23.99138264 22.06491079
 22.57961832 23.31293126 23.66474425 22.03827857 21.43817312 20.43738183
 23.44422637 22.16432571 21.89130786 20.13855782 21.06368168 20.81546917
 22.3980499  22.48529794 21.90391553 22.18737847 23.4691258  21.38701668
 22.77925803 24.64161426 21.88553284 21.39248128 23.83075067 24.36645593
 21.86648983 22.96859979 23.551837   23.19440936 22.0375861  23.55999522
 20.65112579 22.0581271  21.52674517 21.22006166 22.55891141]
```

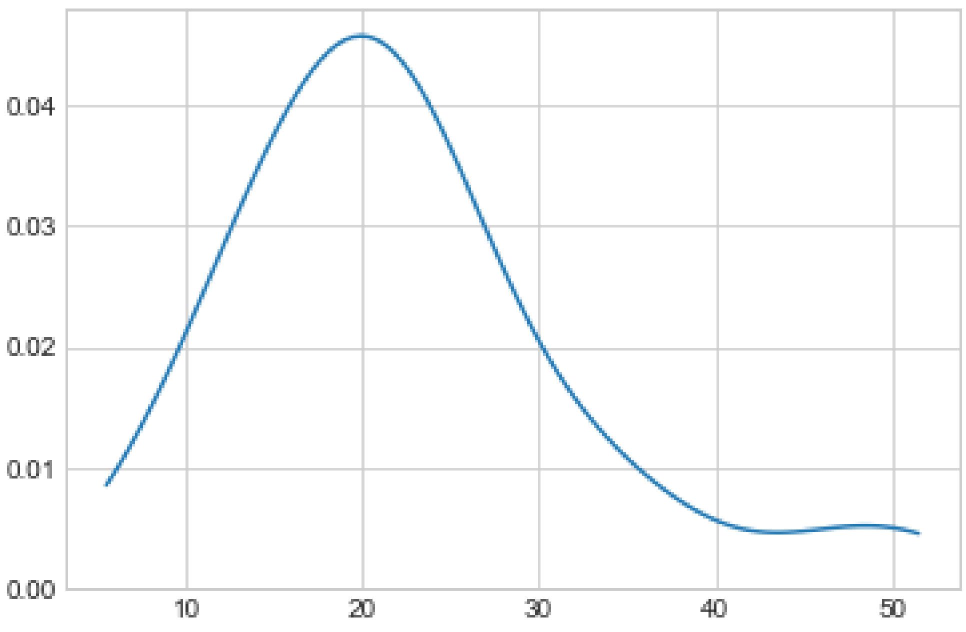
```
In [ ]: import numpy as np
        delta_y = np.array(Y_test[0]) - np.array(Y_pred)
```

```
In [1319]: mse=0
           for i in delta_y:
               mse+=i*i
           print("MSE from SGDRegressor: {}".format(mse/delta_y.shape[0]))
```

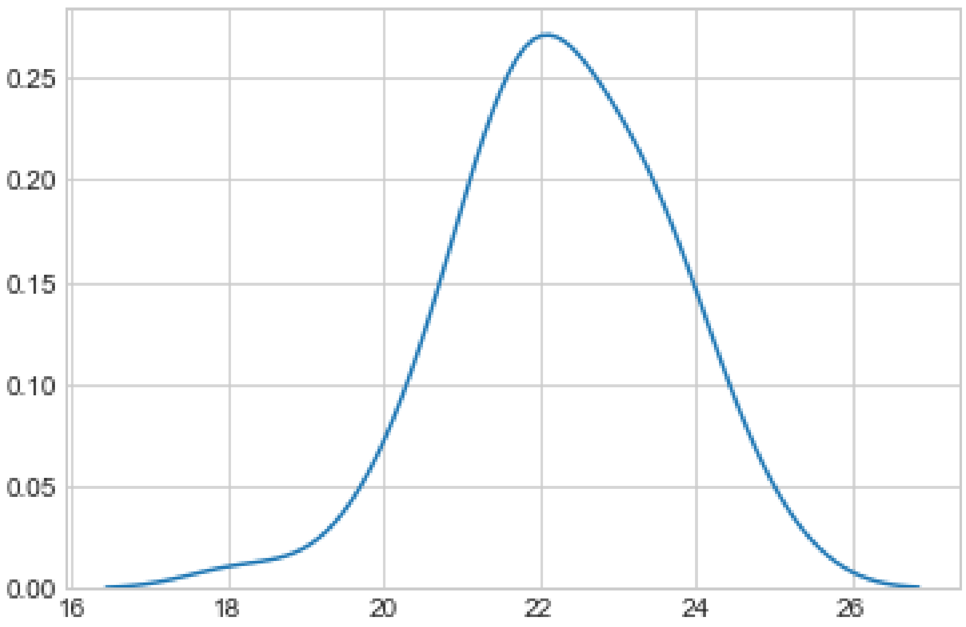
```
MSE from SGDRegressor: 82.52820483081915
```

```
In [1277]: import seaborn as sns
```

```
In [1278]: sns.set_style('whitegrid')
           sns.kdeplot(np.array(Y_test[0]), bw=0.5)
           plt.show()
```



```
In [1279]: sns.set_style('whitegrid')
           sns.kdeplot(Y_pred, bw=0.5)
           plt.show()
```



```
In [1313]: print("SGDRegressor weights: \n{}".format(lm.coef_))
```

```
SGDRegressor weights:
[-0.09383638  0.15998861  0.0144585   0.05195328  0.38743359  1.18389834
  0.1423731   0.29346169 -0.02230794  0.10160212  0.76481751  0.53107774
 -0.03547905]
```

2 Implemented_SGD

```
In [1282]: def del2(w):
               """This function calcuate (X^t(Y-(X.W)))"""
               del_2=np.zeros((1,13))
               #abc=X.sample(n=50)
               Y_ = X['Price']
               X_ = X.drop({'Price'},axis=1)

               del_2[0]=np.dot(np.transpose(-np.array(X_)),(np.array(Y_)-np.dot(np.array(X_),w)

               return del_2[0]/50

In [1283]: w=np.zeros((1,13))

           w=w[0]

In [1285]: w.shape

Out[1285]: (13,)

In [1286]: del2(w)

Out[1286]: array([  -59.54415344,  -144.73147906,  -325.56483848,   -78.45735686,
                 -1052.91359663, -2106.33157052,  -521.02008571,  -434.61929986,
                  -214.84963297,  -509.32885824, -1898.61510236,  -922.77125521,
                  -335.85402724])

In [1287]: from tqdm import tqdm

In [1288]: X.shape

Out[1288]: (506, 14)

In [ ]: #    Now we are calculating best w
        count=0
        ab=[]
        r=0.0001
        ith=0
        for i in tqdm(range(1,25000)):
            if i==1:
                ab.append(w-(2*r*del2(w)))
            else:
                ab.append(ab[i-2]-(2*r*del2(ab[i-2])))
                if (np.linalg.norm(ab[i-1] - ab[i-2]))< 0.0001:
                    print(i)
                    ith=i-1
                    break
```

```
In [1291]: W_=ab[ith]
           W_.shape
```

```
Out[1291]: (13,)
```

```
In [1320]: print("W* = {}".format(W_))
```

```
W* = [-0.74974642  1.04576643 -0.19546251  0.74951276 -0.34717126  4.14724664
 -0.19785074 -2.02486254  1.04405522 -1.04790464 -0.87955881  1.3482447
 -2.99775842]
```

```
In [1294]: a=np.array(X_test)
```

```
In [1295]: a.shape
```

```
Out[1295]: (167, 13)
```

```
In [1296]: Y_=np.dot(a,W_)
```

```
In [1323]: print("Predicted Y from ImplementedSGD: {}".format(Y_))
```

```
Predicted Y from ImplementedSGD: [38.46267008 31.33617456 26.321094    5.44094887 37.5359039
 29.80453603 30.88974173 26.96010339 22.45114307 33.05803542 21.47867981
 22.58801712 33.47461433 27.74138741 15.89984229 -2.30015233 17.97553012
 14.04611207 11.49122114  5.12835014 22.34080774 39.71191285 24.47970446
 32.11373009 12.30999234 25.03823928 22.96233892 23.50127795 23.36902329
 15.65770607  9.86774016 18.15792101 26.28963947 27.27796279 21.7577271
 29.56571812  8.77497708 42.47661627 34.5521272  20.87937194  5.47409032
 28.27983063 11.83168356 25.76254725 30.18352972 -5.90133579 19.43293953
 22.3140102  16.47081552 21.26445133 20.04611905 23.23161552 14.88337405
 17.46858126 25.88042504 36.67511373 16.18492861 27.18864429 20.13940601
 20.91838357 23.73539659 16.43337868 31.60770825 24.20983136 12.61040835
 21.89531594 23.92545277 20.64104818 20.24209289 21.17803426 25.73520896
 15.93346541 19.66361879 17.99063431 26.89141268 19.60410209 15.82555235
 34.15588076 18.38207409 21.67455355 43.65069777 22.86253339 15.63667284
 23.30591519 18.71152316 18.01437188  9.33936037 17.38417533 18.7378044
 37.12794015 18.854417   21.37559028 20.27980088 24.19312753 27.00626091
 14.23265916 26.07498913 21.81728234 12.0741341  22.38392519 22.2009664
 15.04590135 44.60407495  5.8849936  23.28787944 19.67027128 19.24446554
 27.29452658 15.52204788 28.54549277 22.26080396 21.64291727 31.717903
 21.55563227 13.75890012 19.88757355 17.52544951 18.28886216 15.63285691
 21.47459162 35.47247046 23.00998312 19.1586538  22.64981379 27.33595573
 21.04064671 23.16847696 22.70617639 40.79415371 40.92964363 28.28633727
 14.28482149 18.75341823 20.126631   21.51039263 14.1942637   5.12953523
 24.54072693 29.6997433  22.32458522 20.66526725 15.79778811 20.38703667
 34.08975726 24.63864789 29.41049116 17.59996672 23.80873216 27.39383562
 15.76642363 34.65692131 13.86855956 13.89029927 26.6167862  32.84578932
 13.32119706 25.02983027 29.7229109  31.74134591 16.94386273 31.64310309
  8.56711518 32.39674872 24.19915328 19.36162211 16.6765486 ]
```

```
In [1298]: Y=np.array(Y_test[0])
```

```
In [1324]: print("Actual Y : {}".format(Y))
```

```
Actual Y : [37.6 27.9 22.6 13.8 35.2 10.4 23.9 29.  22.8 23.2 33.2 19.  20.3 36.1
 24.4 17.2 17.9 19.6 19.7 15.   8.1 23.  44.8 23.1 32.2 10.8 23.1 21.2
 22.2 24.1 17.3  7.  12.7 17.8 26.4 19.6 25.1  8.3 48.8 34.9 13.8 14.4
 30.1 12.7 27.1 24.8  7.  20.5 21.5 14.  20.4 22.2 21.4 13.5 19.4 24.7
 43.8 14.1 28.6 19.7 16.8 23.2 16.2 41.3 22.7  8.3 18.4 24.7 21.7 20.6
 16.7 22.1 19.4 27.5 27.9 30.1 17.4 15.4 31.  14.2 19.6 50.  21.7 11.7
 19.4 13.  17.5  9.7 20.3 18.6 50.  19.6 21.4 18.4 22.6 25.  15.6 26.6
 22.4 13.1 23.  24.5 13.1 50.   8.8 20.6 12.1 50.  24.1 16.1 23.9 24.3
 13.1 30.3 15.2 13.8 26.4 16.6 18.9 17.6 18.7 33.4 20.7 17.1 23.4 26.5
 21.4 21.5 19.2 50.  50.  23.  10.5 17.8 10.9 21.  13.8 10.5 22.2 30.5
 19.4 15.6 20.2 19.3 34.6 50.  24.  18.7 19.8 22.5 13.3 50.  11.8 11.
 23.7 35.4 15.2 24.4 33.4 31.6 13.4 34.9 14.4 35.4 25.3 18.3 16.6]
```

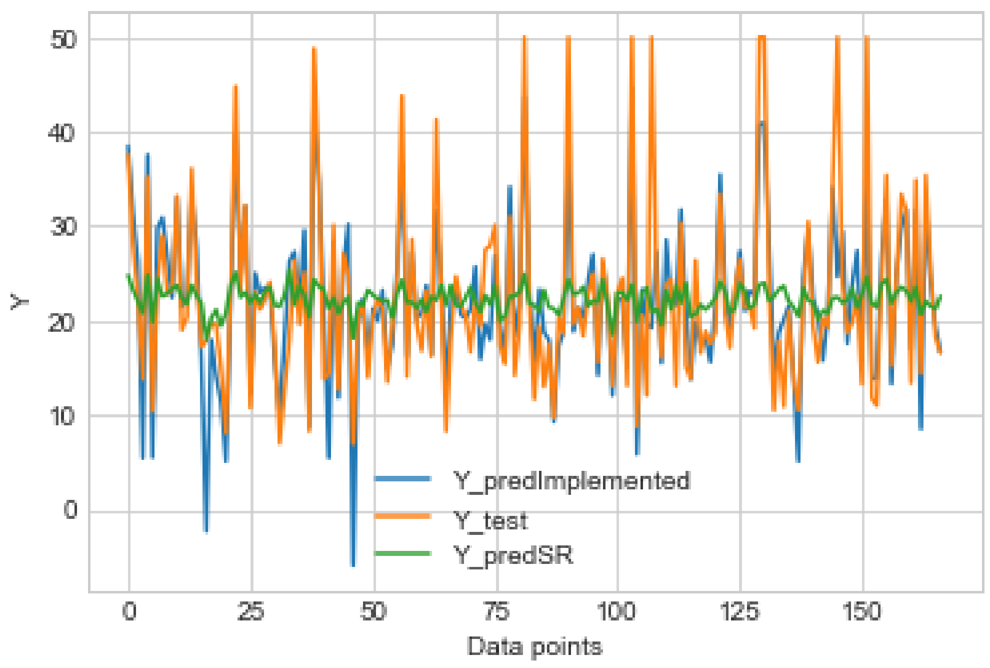
```
In [1300]: dela=Y-Y_
```

```
In [1308]: sqerror=0
           for i in dela:
               sqerror+=i**2
           print("MSE for ImplementedSGD: {}".format(sqerror/dela.shape[0]))
```

```
MSE for ImplementedSGD: 28.432717892291812
```

2.1 Comparing Y_predImplemented and Y_test and Y_predSGD

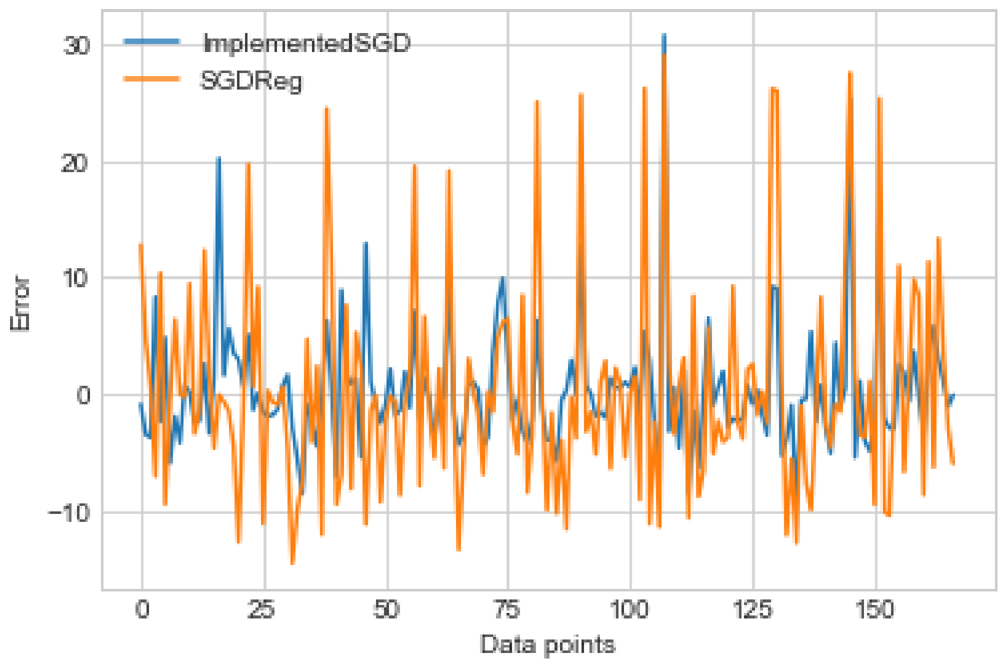
```
In [1311]: # Shape is now: (10, 80)
           plt.plot(Y_) # plotting by columns
           plt.plot(Y)
           plt.plot(Y_pred)
           plt.xlabel("Data points")
           plt.ylabel("Y")
           plt.legend(('Y_predImplemented','Y_test','Y_predSR'))
           plt.show()
```



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

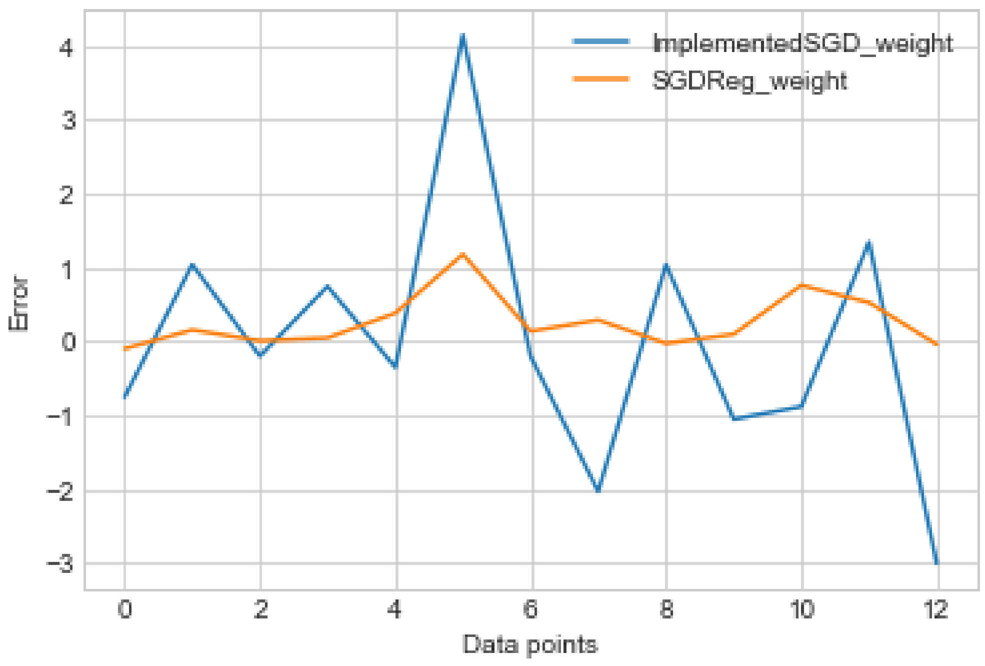
```
In [1304]: plt.plot(dela)
           plt.plot(delta_y)# plotting by columns
           plt.xlabel("Data points")
           plt.ylabel("Error")
           plt.legend(('ImplementedSGD','SGDReg'))
           plt.show()
```



2.3 Comparing Weights get from Implemented SGD vs SGDRegressor

```
In [1305]: 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
[-0.74974642  1.04576643 -0.19546251  0.74951276 -0.34717126  4.14724664
 -0.19785074 -2.02486254  1.04405522 -1.04790464 -0.87955881  1.3482447
 -2.99775842]
Weight from SGDregressor
[-0.09383638  0.15998861  0.0144585   0.05195328  0.38743359  1.18389834
  0.1423731   0.29346169 -0.02230794  0.10160212  0.76481751  0.53107774
 -0.03547905]
```



3 Result:

```
M_SQ-loss from ImplemetedSGD= 28.432717892291812
M_SQ-loss from SGDRegressor= 83.52820483081915
```