Assignment06 - Implement SGD

June 12, 2019

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
from random import seed
from random import randrange
from csv import reader
from math import sqrt
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
import sys
import random
from prettytable import PrettyTable
```

In [2]:
```
X = load_boston().data
Y = load_boston().target
```

In [3]:
```
scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

In [4]:
```
clf = SGDRegressor()
clf.fit(X, Y)
skmse = mean_squared_error(Y, clf.predict(X))
print(skmse)
```

```
22.685385712722375
```

In [5]:
```
y_pred_sk = clf.predict(X).tolist()
```

In [6]:
```
a = clf.coef_.tolist()
```

In [7]:
```
prettytable_data = []
prettytable_data.append(a)
```

In [8]:
```
class CustomSGDRegressor:

    def __init__(self, learning_rate = 0.001, iterations = 100000):
        self.max_iters = iterations
        self.weights = None
        self.alpha = learning_rate

    def fit(self, X, Y):
        # appending the features with ones of shape [n,1] to include the W0
        X = np.append(X, np.ones((X.shape[0], 1)), axis=1)
        self.weights = np.ones((1, X.shape[1]))

        mse_or = mean_squared_error(Y, self._predict(X))

        for iteration in range(self.max_iters):
            self.weights = self.weights - self.alpha * self._gradient(X[iteration%X.sha
            mse = mean_squared_error(Y, self._predict(X))
            # printing at every 100th interation
            if iteration % 2000 == 0:
                print("iteration : {0}, MSE : {1}".format(iteration, mse))

            if mse_or - mse < 0.01 :
                print('Converged !! \niteration : {0}, MSE : {1}\n'.format(iteration, m
                mse_or = mse
                break
            elif mse_or - mse < 0: # reducing the alpha value by 10 when the model ove
                self.alpha = self.alpha/10

    def _gradient(self, x, y):
        return -2 * x * (y - np.dot(self.weights, x))

    def _predict(self, X):
        return np.array([np.dot(self.weights, X[i]) for i in range(X.shape[0])])

    def predict(self, X):
        X = np.append(X, np.ones((X.shape[0], 1)), axis=1)
        return np.array([np.dot(self.weights, X[i]) for i in range(X.shape[0])])
```

In [9]:
```
clf = CustomSGDRegressor()
```

In [10]:
```
clf.fit(X, Y)
```

```
iteration : 0, MSE : 595.4584748485299
iteration : 2000, MSE : 28.212488120504464
iteration : 4000, MSE : 23.243975664228152
iteration : 6000, MSE : 22.732775743702348
iteration : 8000, MSE : 22.66915618056451
iteration : 10000, MSE : 23.44950674101431
```

```
iteration : 12000, MSE : 22.484530834031222
iteration : 14000, MSE : 22.431096281526347
iteration : 16000, MSE : 22.430580790338944
iteration : 18000, MSE : 22.96131515494494
iteration : 20000, MSE : 23.465865506555872
iteration : 22000, MSE : 22.69886027787685
iteration : 24000, MSE : 22.758531667619458
iteration : 26000, MSE : 22.232552909753498
iteration : 28000, MSE : 22.33665359185766
iteration : 30000, MSE : 22.549217900147823
iteration : 32000, MSE : 22.365185497138025
iteration : 34000, MSE : 22.342564676553284
iteration : 36000, MSE : 22.32850424471496
iteration : 38000, MSE : 22.35233318555242
iteration : 40000, MSE : 22.40794453544401
iteration : 42000, MSE : 22.33837588587905
iteration : 44000, MSE : 22.368043707298597
iteration : 46000, MSE : 22.40290362569195
iteration : 48000, MSE : 22.297849655615828
iteration : 50000, MSE : 22.845965754403796
iteration : 52000, MSE : 23.45692450413233
iteration : 54000, MSE : 22.52072808258266
iteration : 56000, MSE : 22.369027861138722
iteration : 58000, MSE : 22.373910756621093
iteration : 60000, MSE : 22.861208365901646
iteration : 62000, MSE : 23.727412625501735
iteration : 64000, MSE : 22.603900712039408
iteration : 66000, MSE : 22.69307600884999
iteration : 68000, MSE : 22.269635263722883
iteration : 70000, MSE : 22.320856028617165
iteration : 72000, MSE : 22.731973309588074
iteration : 74000, MSE : 22.362421741549763
iteration : 76000, MSE : 22.32772407973608
iteration : 78000, MSE : 22.324227973691194
iteration : 80000, MSE : 22.333759186821197
iteration : 82000, MSE : 22.405361835881447
iteration : 84000, MSE : 22.36812241326722
iteration : 86000, MSE : 22.37126633034529
iteration : 88000, MSE : 22.414804056054074
iteration : 90000, MSE : 22.276357077725027
iteration : 92000, MSE : 23.600223008889675
iteration : 94000, MSE : 23.18186636960017
iteration : 96000, MSE : 22.259499607734536
iteration : 98000, MSE : 22.371582292974036
```

```
In [11]: y_pred = clf.predict(X).tolist() # storing the predicted values by my custom SGD clas
```

0.0.1 Comparing coefficients of CustomSGD vs Scikit Learn SGD

```
In [12]: tmp = clf.weights.reshape((14,)) # reshaping weights as per clf.coef_ of scikit-learn
```

```
In [13]: b = [tmp[i] for i in range(tmp.shape[0]-1)] # converting to list
```

```
In [14]: coefficients = pd.DataFrame({'scikit SGD coef':a, 'custom SGD coef':b})
```

```
In [15]: coefficients
```

```
Out[15]:     scikit SGD coef  custom SGD coef
         0          -0.729991        -1.127545
         1           0.628754         0.805467
         2          -0.490527         0.137264
         3           0.819534         0.522197
         4          -0.978776        -1.956269
         5           3.197023         3.071332
         6          -0.237259         0.153272
         7          -2.350684        -2.997373
         8           0.944963         2.339893
         9          -0.526823        -1.830586
         10         -1.864794        -1.930334
         11          0.873629         0.847150
         12         -3.496772        -3.993627
```

0.0.2 Comparing MSE of Scikit learn vs Custom implementations of SGD

```
In [16]: print("Custom MSE : {0}".format(mean_squared_error(Y, clf.predict(X))))
         print("Scikit Learn MSE : {0}".format(skmse))
```

```
Custom MSE : 22.371672778607884
Scikit Learn MSE : 22.685385712722375
```

0.0.3 Comparing predictions Scikit Learn vs Custom implementations of SGD

```
In [17]: predicted = pd.DataFrame({'scikit predictions':y_pred_sk, 'custom predictions':y_pred
```

```
In [18]: predicted
```

```
Out[18]:     scikit predictions    custom predictions
         0            30.689994  [30.086010625671886]
         1            24.731081   [25.16947638451916]
         2            30.826738   [31.23820275712525]
         3            29.269687   [29.17357858650119]
         4            28.724617  [28.549671987583494]
         5            25.456474  [25.483391927904016]
         6            22.845453  [22.501519161649387]
         7            19.603330  [19.049532370580526]
         8            11.571377  [10.377271682918527]
```

9	19.112299	[18.394247531689658]
10	19.423510	[18.59322281910813]
11	21.548068	[21.166005900328816]
12	20.983535	[20.08481191962663]
13	19.990633	[19.625106269293877]
14	19.600393	[19.464520683219344]
15	19.630004	[19.261541465756054]
16	21.125667	[20.475822273847456]
17	17.259636	[16.85071349805859]
18	16.246214	[15.683500408382823]
19	18.387099	[18.2352680899443]
20	12.472608	[12.047130477543226]
21	17.818268	[17.647166064653856]
22	16.255588	[15.738954512704696]
23	13.984427	[13.536791086816766]
24	15.972603	[15.589989698608889]
25	13.520339	[13.069951156943098]
26	15.753368	[15.36350833546766]
27	15.177160	[14.630152351049412]
28	20.161621	[19.908553003137776]
29	21.586424	[21.317744033669662]
..	...	...
476	19.752735	[20.271947167657366]
477	10.211928	[10.166713177737288]
478	18.220347	[18.679602032416405]
479	20.837091	[21.38695243665316]
480	22.154861	[23.173208977570756]
481	25.929557	[27.242436695900484]
482	27.617205	[28.928440153485983]
483	19.795276	[20.51598907967164]
484	18.658844	[18.890108968514305]
485	21.698116	[22.03410333202888]
486	18.769819	[19.264782113807794]
487	20.242399	[20.813130277068982]
488	13.429567	[11.904122216230832]
489	9.884842	[8.06617177080757]
490	5.303989	[3.126619254444783]
491	15.686005	[14.138874621356452]
492	17.922996	[16.415553156050777]
493	20.751945	[20.2663022996026]
494	21.062126	[20.263993958373486]
495	17.555142	[16.20946220301471]
496	14.197792	[13.261977096144152]
497	19.404631	[18.826893167265656]
498	21.595412	[21.135983069100845]
499	18.412752	[18.00194377483939]
500	20.720924	[20.33513262865832]
501	23.876243	[23.945258490763347]

```
502          22.209006  [22.567480862920892]
503          27.806063  [28.485603872640866]
504          26.295129  [26.855829327025322]
505          22.109455  [22.557598177270215]

[506 rows x 2 columns]
```