06 Implement SGD

February 28, 2020

1 Implementing SGD

```
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 prettytable import PrettyTable
        from sklearn.linear_model import SGDRegressor
        from sklearn import preprocessing
        from sklearn.metrics import mean_squared_error
```

```
In [22]: from sklearn.model_selection import train_test_split
         import sklearn

         from sklearn.datasets import load_boston # to load  datasets from sklearn
         import matplotlib.pyplot as plt

         import numpy as np
         import seaborn as sns

         from collections import Counter
         from sklearn.metrics import accuracy_score

         from sklearn.preprocessing import StandardScaler
         import pandas as pd
         import math
```

```
In [5]: boston = load_boston()
        # Shape of Boston datasets
        print(boston.data.shape)
```

```
(506, 13)
```

```
In [10]: print(boston.DESCR)
```

```
Boston House Prices dataset
===========================

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 Universi

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 regressi
problems.

References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources o
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on th
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)

In [11]:
```
boston.feature_names
```

Out[11]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [12]:
```
columnNames = boston.feature_names
columnNames
```

Out[12]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [19]:
```
boston.data
```

Out[19]:
```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

In [13]:
```
Data = pd.DataFrame(boston.data, columns = columnNames)
Data.head(2)
```

Out[13]:
```
      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

   PTRATIO      B  LSTAT
0     15.3  396.9   4.98
1     17.8  396.9   9.14
```

```
In [17]: Data_Labels = boston.target
         Data_Labels.shape
```

```
Out[17]: (506,)
```

```
In [15]: Data["PRICE"] = Data_Labels
         Data.head(2)
```

```
Out[15]:       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

            PTRATIO      B  LSTAT  PRICE
         0     15.3  396.9   4.98   24.0
         1     17.8  396.9   9.14   21.6
```

```
In [20]: Data.shape
```

```
Out[20]: (506, 14)
```

2 Train Test Split

```
In [23]: X_train, X_test, Y_train, Y_test = train_test_split(Data, Data["PRICE"],test_size = 0
         X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
         print(X_train.head(10))
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS   RAD    TAX  \
243  0.12757  30.0   4.93   0.0  0.428  6.393   7.8  7.0355   6.0  300.0
253  0.36894  22.0   5.86   0.0  0.431  8.259   8.4  8.9067   7.0  330.0
224  0.31533   0.0   6.20   0.0  0.504  8.266  78.3  2.8944   8.0  307.0
466  3.77498   0.0  18.10   0.0  0.655  5.952  84.7  2.8715  24.0  666.0
460  4.81213   0.0  18.10   0.0  0.713  6.701  90.0  2.5975  24.0  666.0
55   0.01311  90.0   1.22   0.0  0.403  7.249  21.9  8.6966   5.0  226.0
75   0.09512   0.0  12.83   0.0  0.437  6.286  45.0  4.5026   5.0  398.0
337  0.03041   0.0   5.19   0.0  0.515  5.895  59.6  5.6150   5.0  224.0
249  0.19073  22.0   5.86   0.0  0.431  6.718  17.5  7.8265   7.0  330.0
485  3.67367   0.0  18.10   0.0  0.583  6.312  51.9  3.9917  24.0  666.0

     PTRATIO       B  LSTAT  PRICE
243     16.6  374.71   5.19   23.7
253     19.1  396.90   3.54   42.8
224     17.4  385.05   4.14   44.8
466     20.2   22.01  17.15   19.0
460     20.2  255.23  16.42   16.4
55      17.9  395.93   4.81   35.4
75      18.7  383.23   8.94   21.4
337     20.2  394.81  10.56   18.5
249     19.1  393.74   6.56   26.2
485     20.2  388.62  10.58   21.2
```

```
In [27]: #First Standadize the data
         from sklearn import preprocessing
         scaler=preprocessing.StandardScaler()
         std_scale = scaler.fit(X_train[['CRIM', 'ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RA
         train_standardized= std_scale.transform(X_train[['CRIM', 'ZN','INDUS','CHAS','NOX','RM
         test_standardized= std_scale.transform(X_test[['CRIM','ZN','INDUS','CHAS','NOX','RM',
```

```
In [28]: X_train_standardized = pd.DataFrame(train_standardized, columns = columnNames)
         X_test_standardized = pd.DataFrame(test_standardized, columns = columnNames)
         Y_train=np.array(Y_train)
         Y_test=np.array(Y_test)
         print(Y_train)
         print(Y_train.shape)
         print(X_train_standardized.shape)
```

```
[23.7 42.8 44.8 19.  16.4 35.4 21.4 18.5 26.2 21.2 32.7 30.1 32.2 22.
 21.7 13.8 21.6 31.6 27.9 11.7 32.5 19.3 24.6 19.4 24.8 11.5 20.4 22.6
 14.4 19.5 19.3 28.7 31.   5.6 34.9 22.9 23.3 48.5 24.2 20.2  7.2 37.
 20.3 20.4 20.1 23.8 24.3  8.8 23.  21.9 19.3 50.  25.2 19.1 50.  19.8
 23.2 23.1 50.  50.   7.5 24.7 21.8 22.6 22.6 25.  28.4 15.  20.  20.
 18.4 50.  23.9 20.5 22.8 24.8 22.2 14.2 17.  12.6 34.7 25.1 14.9 30.8
 18.3 48.8 13.5 23.4 26.4 20.  14.5 10.5  8.4 22.9 23.1 23.3 18.9 23.
 35.2 10.5 23.3 19.5 13.2 22.2 17.6 28.2 17.8 23.9 28.1 14.3 13.4  5.
 13.8 14.  46.7 33.  24.8 16.1 22.1 33.1 16.7 18.2 10.9 20.1 16.8 21.5
 20.6 15.  20.2 23.8 13.4 19.2 12.5 25.   7.  13.8 19.4 15.4 14.6 23.1
 17.1 13.5 26.6 25.  14.9  9.7 17.5 23.2 18.5 23.2 31.6 24.1 20.6  8.3
 22.5 10.2  6.3 18.9 15.6  8.4  8.7 46.  19.5 15.  21.  20.8 13.3 32.
 12.1 28.  26.4 50.  21.6 16.1 18.3 23.1 43.1 23.  31.5 20.5 17.1 21.2
 12.7 14.5 18.6 12.  21.7 19.4 36.2 11.8 18.2 50.  50.  31.1 50.  42.3
 10.2 28.7 23.5 15.6 21.1 13.  27.  22.9 20.6 28.7 19.6 25.  22.2 50.
 34.9 20.3 14.1 15.2 22.5 18.5 13.8 20.1 38.7 20.3 30.5 13.3 23.6 37.2
 23.1 22.6 33.1 22.4 22.  14.1 22.2 20.4 25.  18.8 24.6 17.8 17.3 19.8
 22.6 13.1 15.6 14.8 17.8 20.6 33.4 26.7 21.4 23.9 18.  22.  50.  11.8
 13.6 28.5 37.3  7.2 24.5 19.2 17.4 20.8 14.6 31.5 43.8 31.2 27.1 18.9
 22.3 24.8 37.9 27.9 10.4 23.3 18.5 15.2 50.  16.2 25.  29.6 20.1 19.4
 29.9 17.5 24.4 22.9 19.7 19.4 13.1 29.8 22.  20.6 24.3 25.3 37.6 33.2
 20.  20.6 20.  24.7 17.4 19.   8.1 21.7 10.9 50.  20.4 19.1 21.7 21.8
 43.5 27.1 10.2 44.  23.6 24.4 19.7 39.8 16.1 32.9 17.1 24.1 35.1 21.2
 25.  27.5 20.3 16.7 19.6 23.9 18.7 17.2 32.  19.6 18.8 29.6 33.2 22.
 21.  19.6 17.8 19.3 13.9 36.1 19.8 36.2 34.6 23.7 50.  24.7 12.3 19.9
 33.3 20.5 13.9 20.1 20.8 23.8 21.2 21.4 25.  19.1 16.6 36.4 27.5 48.3
 16.  45.4 13.1 18.2  7.4 16.2 19.5 23.1 21.  32.4 11.  21.4 23.   5.
 20.7 29.8 29.1 21.7 14.3  9.6 19.6 13.8 16.5 22.5 23.7 22.   8.8 22.7
 33.8 14.5  8.5 21.9 11.9 12.8 36.5 22.7 15.2 19.1 21.9 22.8]
(404,)
(404, 13)
```

```
In [29]: X_train_standardized['PRICE']=Y_train
```

```
X_test_standardized['PRICE']=Y_test
```

In [30]:
```
print(X_train_standardized.shape)
print(X_test_standardized.shape)
```

```
(404, 14)
(102, 14)
```

3 Stochastic Gradient Descent

In [32]:
```
#have used the below code from github
```

In [31]:
```
# for references
#https://github.com/gauravtheP/Implementing-Stochastic-GradientDescent/blob/master/Li
#First step initilize the weights and b
#formulae of slope s=mx+b.
# mx is the weights*x1....weights_d*xd
# b is the intercept term
m = X_train.shape[0]
weight = np.random.randn(13)*np.sqrt(2/m) # defining initial random weight from norma
b = np.random.randn(1)*np.sqrt(2/m) # generating initial random y-intercept from norm
# initilize learing rate
learningRate = 0.2
print(m,weight,b,learningRate)
for i in range(2000): # running 2000 iterations
    Data_batch_10 = X_train_standardized.sample(n = 10) # taking 10 stochastic sample
    X_temp = Data_batch_10.drop("PRICE", axis = 1, inplace = False) # DROP the price
#X = pd.DataFrame(X_temp, columns = columnNames)
    X=X_temp
    Y = Data_batch_10["PRICE"]
    PartialGradient = np.empty(13)# in this we store the partial derivate with respec
    sum2 = 0
# Update the weights-------------------------------
# formula (w0=w1-lr*derivate)in every iteration
# step 1.
#First calculate the derivative
    for j in range(13): # as there are 13 dimensions in our dataset and dimensions of
        sum1 = 0
        for k in range(10):
            sum1 += -2 * X.iloc[k][j] * (Y.iloc[k] - np.dot(weight, X.iloc[k]) - b) #
        PartialGradient[j] = sum1
# step 2.
#multiply with learning rate
    PartialGradient *= learningRate
#step 3.
#Update the weights
    for l in range(13):
```

```
            weight[l] -= PartialGradient[l] # updating weights
    # Update the Intercepts or (b's)-------------------------------
        for m in range(10):
            sum2 += -2 * (Y.iloc[m]- np.dot(weight, X.iloc[m]) - b) # this is the derivat
        b = b - learningRate * sum2 #updating y-intercept 'b'
    # in every iteration u have to reduce the learing rate bro
        learningRate = 0.01 / pow(i+1, 0.25) #learning rate at every iteration
    # just add the regularization term to it
        weight = weight + 0.0001*np.dot(weight, weight) #adding l2 regularization
        b = b + 0.0001*np.dot(weight, weight) #adding L2 regularization
    print("Weight = "+str(weight))
    print("b = "+str(b))
```

```
404 [ 0.02381462  0.05560947  0.09227067  0.0094231   0.08282602 -0.09006225
 -0.04582119 -0.02474342  0.03889422  0.07088581  0.06431994  0.01697569
 -0.23645988] [-0.05827495] 0.2
Weight = [-0.77962561  1.46734951  0.57203129  1.10257263 -0.82167183  3.46647483
  0.40517137 -2.24246557  2.94903055 -2.19957943 -1.73340329  1.55374565
 -3.57664596]
b = [23.1528653]
```

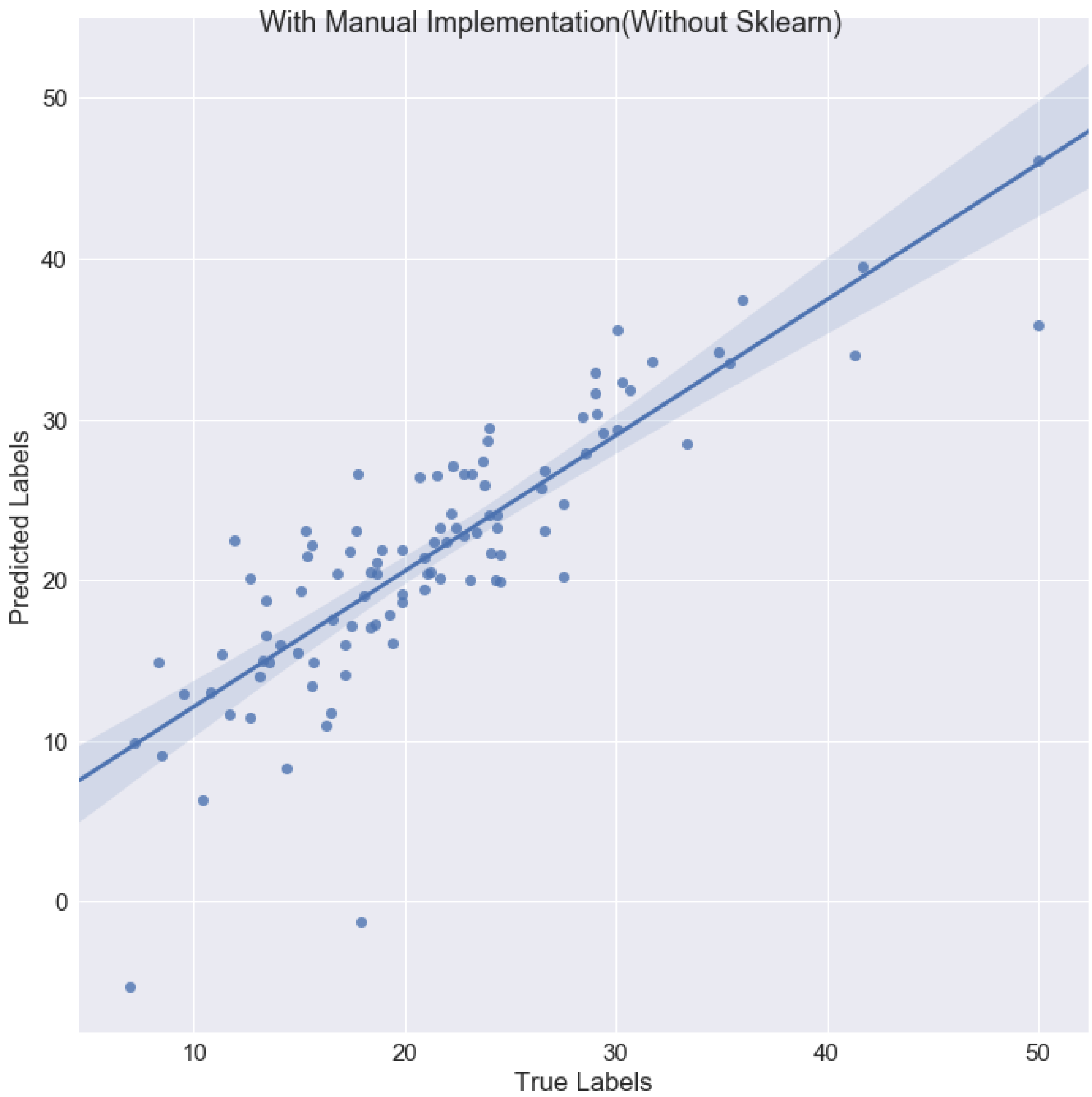
In [41]:
```
# time for testdata.. with our updated weights and coeffcients
import math
test_temp = X_test_standardized.drop("PRICE", axis = 1, inplace = False)
test_data = test_temp
test_labels = Y_test
y_predicted = []
for i in range(102):
    test_i = 0
    test_i = np.dot(weight, test_data.iloc[i]) + b[0] #making prediction by using opt
    y_predicted.append(test_i)
```

In [42]:
```
#Make the preditions
d1 = {'True Labels': Y_test, 'Predicted Labels': y_predicted}
df1 = pd.DataFrame(data = d1)
```

In [43]:
```
Mean_Sq_Error = mean_squared_error(Y_test, y_predicted)
print(Mean_Sq_Error)
```

```
19.430696876801047
```

In [44]:
```
import seaborn as sns
lm1 = sns.lmplot(x="True Labels", y="Predicted Labels", data = df1, size = 10)
fig1 = lm1.fig
fig1.suptitle("With Manual Implementation(Without Sklearn)", fontsize=18)
sns.set(font_scale = 1.5)
```



In [45]:
```
#Sklearn implementation
X_temp = X_train_standardized.drop("PRICE", axis = 1, inplace = False)
X=X_temp
Y = Y_train
X_test_temp = X_test_standardized.drop("PRICE", axis = 1, inplace = False)
X_te=X_test_temp
Y_te = Y_test
clf = SGDRegressor(shuffle = False, learning_rate= 'invscaling', max_iter = 2000)
clf.fit(X, Y)# fir train data
Y_pred = clf.predict(X_te)# predict test error
print("Weight = "+str(clf.coef_))
print("Y Intercept = "+str(clf.intercept_))
```

```
Weight = [-1.17297343  1.33565484  0.03873578  0.6983058  -1.52908145  2.43744545
 -0.02551971 -3.29220995  2.82299091 -2.19753749 -2.00640482  0.98792572
 -4.09702758]
Y Intercept = [22.80739133]
```

In [46]:
```python
d2 = {'True Labels': Y_te, 'Predicted Labels': Y_pred}
df2 = pd.DataFrame(data = d2)
df2
```

Out[46]:

	True Labels	Predicted Labels
0	23.2	26.961971
1	11.7	13.973693
2	21.5	24.903008
3	18.4	16.019471
4	21.2	20.579715
5	29.4	31.096605
6	34.9	34.723440
7	15.7	14.071281
8	30.7	31.618960
9	16.6	17.117845
10	19.4	16.610444
11	17.5	16.819046
12	23.7	28.307462
13	13.3	13.777073
14	50.0	41.876746
15	14.9	14.582086
16	35.4	34.247308
17	8.5	6.438918
18	22.3	27.041883
19	8.3	13.126012
20	21.4	20.714101
21	24.0	25.351100
22	13.1	14.044207
23	21.1	20.824257
24	13.6	13.499123
25	12.7	10.908501
26	27.5	18.833419
27	21.7	20.716709
28	26.5	25.710582
29	20.7	25.814126
..	...	...
72	23.4	23.916236
73	19.9	18.412606
74	22.8	24.373119
75	33.4	28.444640
76	16.5	10.390914
77	24.5	20.716349

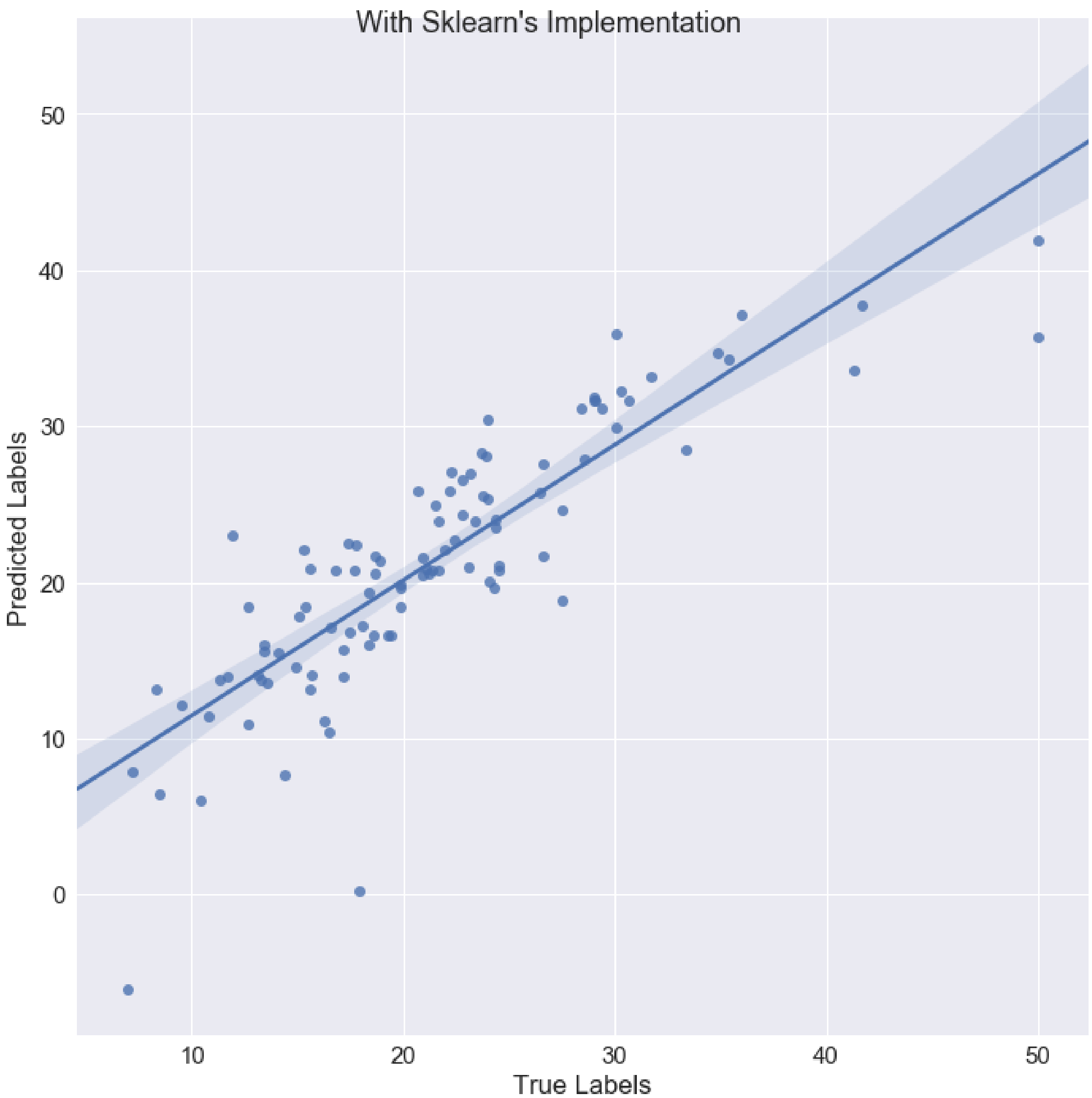
```
78          17.8         22.434002
79          24.4         23.468390
80          28.6         27.910439
81          19.3         16.588466
82          13.4         15.977304
83          14.4          7.674054
84          19.9         19.830509
85          28.4         31.093707
86          19.9         19.675946
87          23.9         28.043596
88          29.0         31.672917
89          29.1         31.602516
90          11.9         22.966558
91          17.2         15.639889
92          41.7         37.748548
93          14.1         15.471684
94          11.3         13.725981
95          15.6         13.141447
96          30.3         32.204866
97          13.4         15.625202
98          17.9          0.238130
99          23.1         20.996016
100         50.0         35.756602
101         24.4         23.983928

[102 rows x 2 columns]
```

In [47]:
```
Mean_Sq_Error = mean_squared_error(Y_te, Y_pred)
Mean_Sq_Error
```

Out[47]: 18.474840956762705

In [49]:
```
lm2 = sns.lmplot(x="True Labels", y="Predicted Labels", data = df2, size = 10)
fig2 = lm2.fig
# Add a title to the Figure
fig2.suptitle("With Sklearn's Implementation", fontsize=18)
sns.set(font_scale = 1.5)
```


With Sklearn's Implementation
Predicted Labels
True Labels
0
10
20
30
40
50
10
20
30
40
50