Pradip Dharam

6. SGD mannual implemetation for Linear Regression

forked from Linear Regression by Pradip Dharam (+0/−0)

last run a minute ago · IPython Notebook HTML

using data from Scochastic Gradient Descent - · Private Make Public

Problem Statement:

Implement Stochastic Gradient Descent for Linear regression MANUALLY with given SG formulas and apply it to Boston home price dataset; and then compare the results with actual Linear Regression from sklearn. This will give inner work of SGD. We can implement SGD for any algorithm once it is tried for Linear Regression.

Optimization problem without regulatization term and Derivatives or Gradient Descent.

Gradient is a slope. Instead of n in GD in diagram, we need to take bunch of k random points, suppose k=10, then we get SGD (Stochastic Gradient Descent) yi is actual value, wTxi is predicted value in below slope or derivative formulae.

Solution

Importing lib's

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
```

Funtion which gives rates or derivatives or slopes

In [2]:
```
def get_slope(x_train, y_train, w, b, k=10):
    term_1 = np.subtract(y_train, np.matmul(x_train, w.T)) - b
    term_2 = -2 * x_train

    w = np.matmul(term_1 , term_2)
    b = np.average(-2 * term_1)
    #print("Size b : ", b.size)
    #Size of b is 10 here since 10 data points, b is intercept, 10 intercept for respective 10 da
ta points
    return w,b
```

SGD function for linear regression

In [3]:
```
def sgd_LR(x_train, y_train, w, b, iterations=100, rate_of_learning=0.01, k=10):

    for iteration in range(iterations):
        import random
        k_row_nums = random.sample(range(len(x_train)),k)
        #print("k_row_nums : ", k_row_nums)
        w_slope,b_slope = get_slope( x_train[k_row_nums], y_train[k_row_nums], w, b, k)

        w = np.subtract(w, rate_of_learning * w_slope)
        b = np.subtract(b, rate_of_learning * b_slope)
        rate_of_learning -= 0.00000001

    return w,b
```

Get the Boston home proces data ser from sklearn

In [4]:
```
from sklearn.datasets import load_boston
boston = load_boston()

import pandas as pd
bos = pd.DataFrame(boston.data)
#print(bos.head())
bos['PRICE'] = boston.target

X = bos.drop('PRICE', axis = 1)
Y = bos['PRICE']
```

Split the daat into tran and test

In [5]:
```
from sklearn.preprocessing import StandardScaler
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
```

Scale or standardize the data points and convert them to array

In [6]:
```
scale = StandardScaler()
scale.fit(x_train)
```

```
x_train = scale.transform(x_train)
x_test = scale.transform(x_test)

y_train=np.array(y_train)
y_test=np.array(y_test)
```

Initialize w and b

- **Initializing weight such that weights of all feature becomes zero**
- **B is scalar is only one float value. But in get_slope function, we get b as array of 10 integers since k=10 i.e. 10 data points. We need to take average of these 10 values as b. This is because in "Gradient Descent" we have by default 1 data point and we get only 1 b value b; here, for "Stochastic Gradient Descent", we use batch of 10 points i.e. batch SGD, so we get 10 b values. Ideally, per plain, we need to have only one intercept as b, so we take average of array b as final b value**

In [7]:
```
# Initializing weight such that weights of all feature becomes zero
w = np.array( [0.0 for i in range(len(x_train[0]))] )
# b is an intercept. Its a scalar not vector. Initializing b to zero
b = 0.0
```

Get w and b from manually developed SGD function

In [8]:
```
w,b = sgd_LR(x_train, y_train, w, b, 100000, 0.001, 20)
```

Predict the home prices for x_test using w and b got from manualy developed SGD function

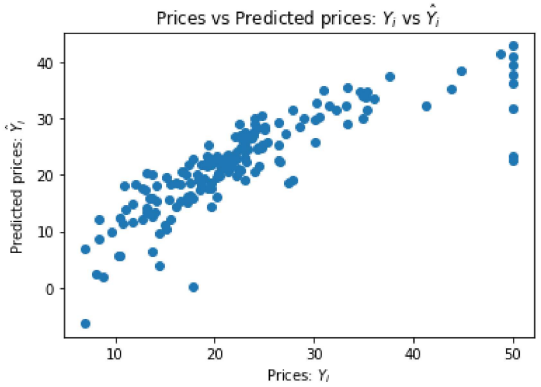
In [9]:
```
def y_predict(x_test, w, b):
    return np.matmul(x_test, w.T) + b

y_pred = y_predict(x_test, w, b)
y_delta_SGD_manual = y_test - y_pred;
```

Plotting Predicted vs Actual home prices

In [10]:
```
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()
```

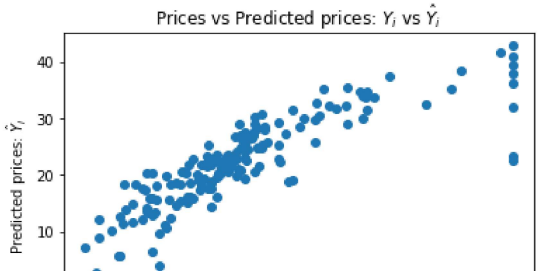


Linear Regression

In [11]:

```
# code source:https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regression-on-boston-housing-dataset-cd62a80775ef
lm = LinearRegression()
lm.fit(x_train, y_train)
y_pred = lm.predict(x_test)
y_delta_sklearn = y_test - 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()
```



Compare the weights of manually implemented SGD and sklearn Linear Regression

In [12]:
```
print("Weights generated by manual SGD : \n", w)
print("Weights generated by sklearn Linear Regression : \n", lm.coef_)
```

```
Weights generated by manual SGD :
 [-1.30429726  0.86322664 -0.16402212  0.20701467 -1.48975583  2.77694648
 -0.33523497 -2.74358216  3.01445002 -2.32157796 -2.12010284  1.05542991
 -3.33400682]
Weights generated by sklearn Linear Regression :
 [-1.31193031  0.86187745 -0.16719287  0.18957843 -1.48658584  2.79131565
 -0.32737703 -2.77204093  2.97567549 -2.2727549  -2.13375869  1.05842993
 -3.33495407]
```

Observation:

- **weights of manually implemented SGD and sklearn Linear Regression are almost similar**
- **Scatter plots of predicted vs actual home prices for both manually implemented SGD and sklearn Linear Regression are almost similar**

Plotting PDF's for delta y for both sklearn Linear Regression and manual SGD

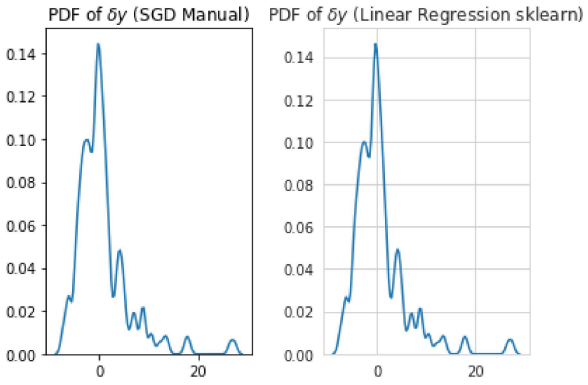
- **delta y is (y_test - y_predicted)**

In [13]:
```
import seaborn as sns;
import numpy as np;

plt.subplot(1,2,1)
sns.set_style('whitegrid')
sns.kdeplot(np.array(y_delta_SGD_manual), bw=0.5)
plt.title('PDF of $\delta{y}$ (SGD Manual)')
```

```
plt.subplot(1,2,2)
sns.set_style('whitegrid')
sns.kdeplot(np.array(y_delta_sklearn), bw=0.5)
plt.title('PDF of $\delta{y}$ (Linear Regression sklearn)')

plt.tight_layout()
plt.show()
print("delta y is (y_test - y_predicted)")
```




```
delta y is (y_test - y_predicted)
```

Observation

- **From above pdf, difference between actual home prices and predicted home prices by both the models is zero for most of the homes.**
- **This means that actual home prices and predicted home prices are almost identical in each model**
- **PDF's for both sklearn Linear Regression and manual SGD looks very similar**
- **HENCE, MANUALLY developed SGD code or function or model works WELL**

In [ ]:

Did you find this Kernel useful?
Show your appreciation with an upvote

0

Comments (0)

Sort by Select...

Click here to enter a comment...