7.Stochastic Gradient Descent (SK Learn)

October 17, 2021

Linear Regression using Stochastic Gradient Descent with Sci-Kit (SK) Learn

[1]:
```
# Import necessary package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

0.0.1 Step 1: Load the dataset

[2]:
```
# Load dataset into pandas dataframe
dataset = pd.read_csv("E:\\MY LECTURES\\DATA SCIENCE\\3.
↪Programs\\dataset\\Advertising.csv")
# Change this location based on the location of dataset in your machine
```

[3]:
```
# Display the first five records
dataset.head()
```

[3]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

Advertising data comprises four features: TV, radio, newspaper, and sales. It explains the budget (in 1000$) spent on different mass media and the net outcome for every week.

sales for a product (output/dependent/target variable).

advertising budget for TV, radio, and newspaper media (input/independent/target variable).

Planning to perform regression on TV budget (X) as input and sales (Y) as output.

[4]:
```
# Dataset shape (number of rows and columns)
dataset.shape
```

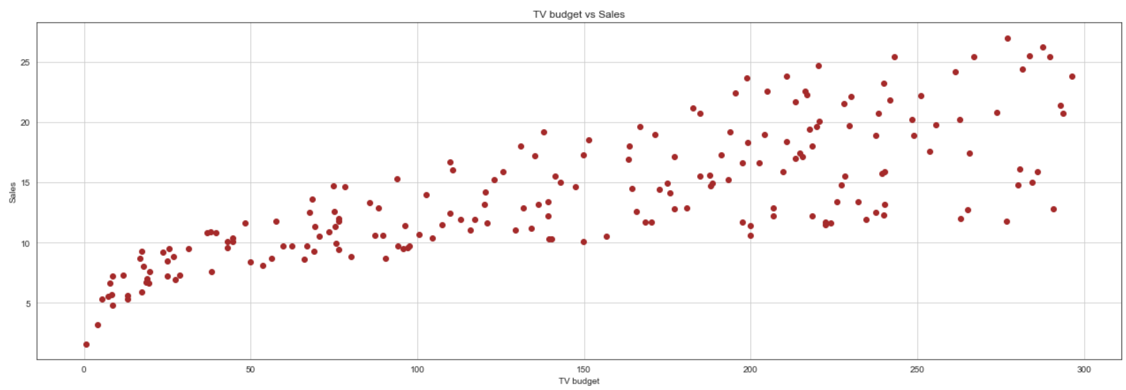
[4]:
```
(200, 4)
```

Row <=> record, tuple, instance, sample, observation, object, case, entity Column <=> attribute, variable, field, feature, characteristic, dimension

0.0.2 Step 2. EDA

Bivariate analysis

[5]:
```
# Scatter plot
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
plt.scatter(dataset.TV,dataset.sales,color="brown")
plt.grid(b=None)
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.show()
```



0.0.3 Step 3. Pre-process and extract the features

[6]:
```
# Normalize the data bring different scale values between 0 and 1
from sklearn import preprocessing
x = dataset.values                                  # returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)
df.head()
```

[6]:

	0	1	2	3
0	0.775786	0.762097	0.605981	0.807087
1	0.148123	0.792339	0.394019	0.346457
2	0.055800	0.925403	0.606860	0.303150
3	0.509976	0.832661	0.511873	0.665354

```
4  0.609063  0.217742  0.510994  0.444882
```

0.0.4 Step 4. Split the data for training and testing

[7]:
```
# Splitting dataset into training and testing set
from sklearn.model_selection import train_test_split
# x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 0)
# The above returns data as numpy array

# The following returns records in random as pandas.core.frame.DataFrame
X = df.iloc[:,0].values.reshape((-1, 1))     # Budget spent on TV
Y = df.iloc[:,3].values                      # Sales
x_train, x_test, y_train, y_test =  train_test_split(X,Y, test_size=0.2)
```

0.0.5 Step 5: Training phase (bulding the model) using Gradient Descent

Parameter initialization

[8]:
```
# np.random.seed(13)
# number of iterations (epochs)
epoch = 1000
# learning rate
learn_rate = 0.001
# batch_size
print("Number of records in traninig set : ",len(x_train),". This is the␣
 ↪maximum batch size.")
batch_size = 5
```

```
Number of records in traninig set :  160 . This is the maximum batch size.
```

Note: Batch size should be between 1 to number of records (n) in x_train. If batch size is

1 - stochastic gradient descent

2 to at least n-1 - mini batch stochastic gradient descent

n - batch gradient descent (simple gradient decent)

Gradient descent algorithm for 2 parameters

[9]:
```
import numpy as np
from sklearn import linear_model
SGDReg = linear_model.SGDRegressor(max_iter = epoch,tol = learn_rate)
# Stochastic gradient descent (batch size = 1 here)
SGDReg.fit(x_train, y_train)

# For mini batch gradient descent
```

```
# SGDReg.partial_fit(x_train, y_train)
```

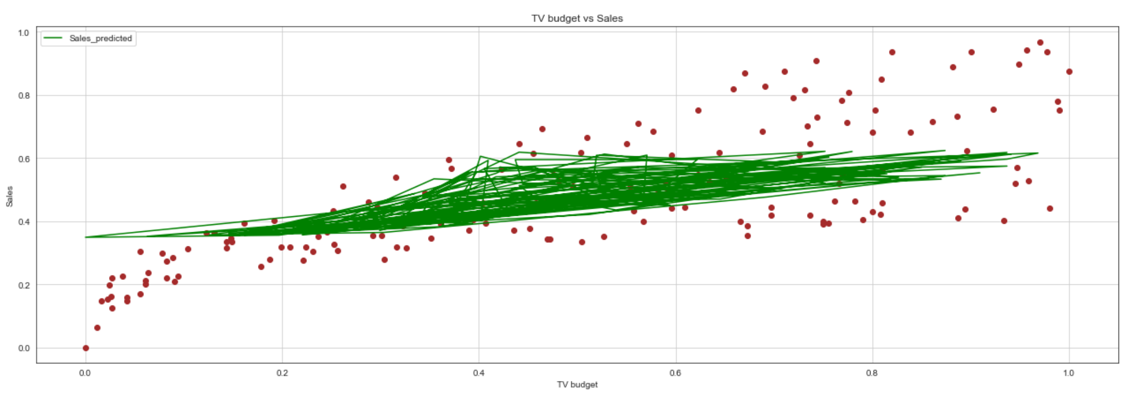
[9]: SGDRegressor()

[10]:
```
# perform the gradient descent search
m , b = SGDReg.intercept_, SGDReg.coef_
print("y = m x + b ==>  y = ",m," x + ",b)
```

```
y = m x + b ==>  y =  [0.34859146]  x +  [0.27509165]
```

Visualizing the model

[11]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
plt.scatter(x_train,y_train,color="brown")
y_train_pred = SGDReg.predict(x_train)
plt.grid(b=None)
plt.plot(y_train,y_train_pred,"g",label="Sales_predicted")
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```



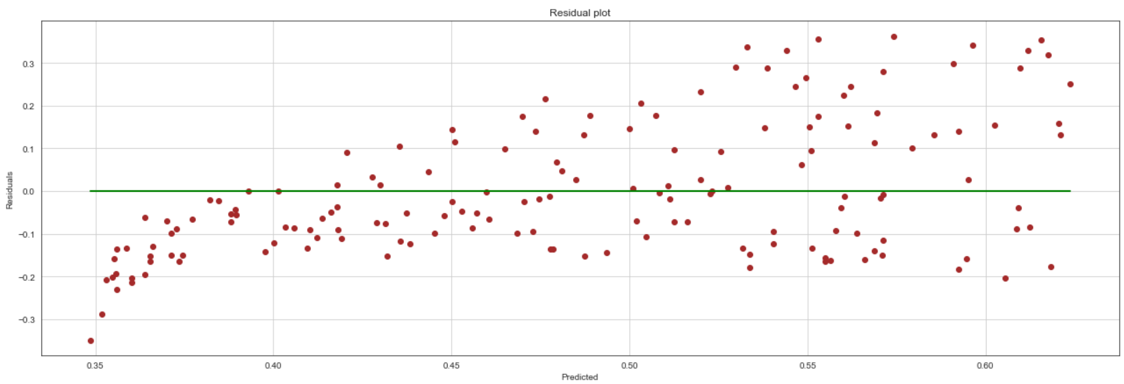
Plotting observed sale (x) and predicted sale (y) for training set

[12]:
```
# Predicting the Test set results
x = np.arange(len(y_train_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_train,"g-+",label="Sales_observed")
plt.plot(x,y_train_pred,"r-*",label="Sales_predicted")
plt.grid(b=None)
plt.xlabel("TV budget")
```

```
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```

Residual (Error) plot If the model has done good predictions, then the datapoints must be near around to horizontal line.

[13]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_train-y_train_pred
zeros = y_train-y_train
plt.scatter(y_train_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_train_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



0.0.6 Different Error calculations to asses the model for training set

1. Sum of Squared Error (SSE)

[14]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = y_train[i] - y_train_pred[i]
  squ_diff = diff**2
  sum = sum + squ_diff
Train_SSE = round(sum,2)
print("Sum of Squared Error (SSE) :",Train_SSE)
```

```
Sum of Squared Error (SSE) : 3.84
```

2. Mean Squared Error (MSE)m

[15]:
```
Train_MSE = round(Train_SSE/n,2)
print("Mean Squared Error (MSE) :",Train_MSE)
```

```
Mean Squared Error (MSE) : 0.02
```

3. Root Mean Squared Error (RMSE)

[16]:
```
Train_RMSE = round(np.sqrt(Train_MSE),2)
print("Root Mean Squared Error (RMSE) :",Train_RMSE)
```

```
Root Mean Squared Error (RMSE) : 0.14
```

4. Mean Absolute Error (MAE)

[17]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = y_train[i] - y_train_pred[i]
  sum = sum + np.abs(diff)
Train_MAE = round(sum/n,2)
print("Mean Absolute Error (MAE) :",Train_MAE)
```

```
Mean Absolute Error (MAE) : 0.13
```

5. Mean Absolute Percentage Error (MAPE)

[18]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
    if y_train[i] ==0:
        continue
    else:
```

```
        diff = (y_train[i] - y_train_pred[i])/y_train[i]
        sum = sum + np.abs(diff)
Train_MAPE = round(sum/n*100,2)
print("Mean Absolute Percentage Error (MAPE) :",Train_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 33.05
```

0.0.7 Calculating R-Squred value (goodness of model) using SSE

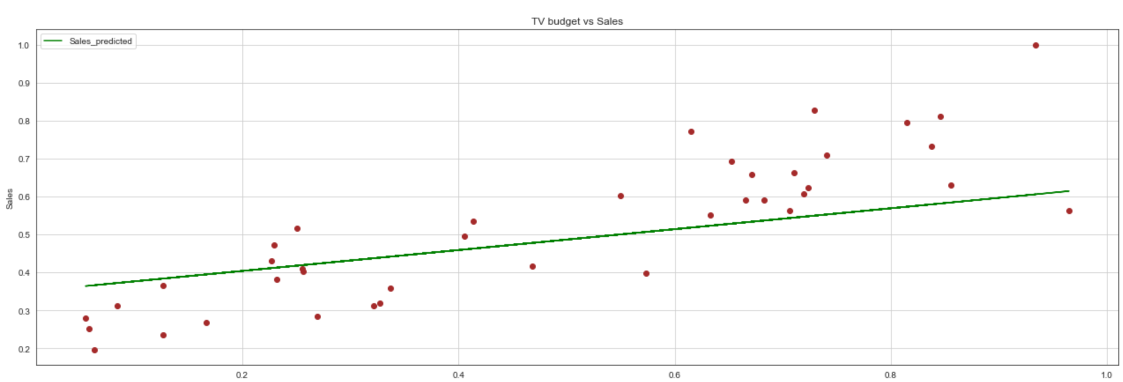
[19]:
```
from sklearn.metrics import r2_score
out = r2_score(y_train,y_train_pred)
Train_RS = round(out,2)*100
print("R-Squred value (goodness of model) for training set :",Train_RS,"%")
```

```
R-Squred value (goodness of model) for training set : 44.0 %
```

0.0.8 Step 6: Testing phase

[20]:
```
# Predicting the Test set results
y_test_pred = SGDReg.predict(x_test)
```

[21]:
```
# Predicting the Test set results
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
plt.scatter(x_test,y_test,color="brown")
plt.grid(b=None)
plt.plot(x_test,y_test_pred,"g",label="Sales_predicted")
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```

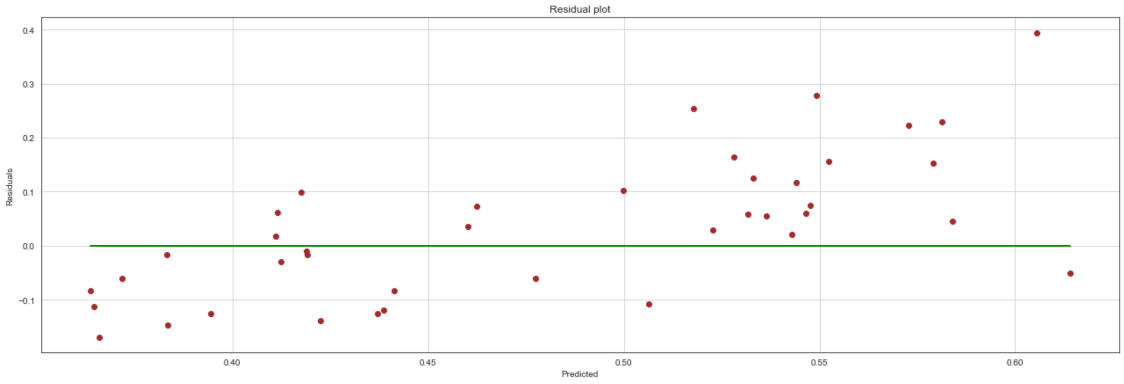


Plotting observed sale (x) and predicted sale (y) for test set

[22]:
```
# Predicting the Test set results
x = np.arange(len(y_test_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_test,"g-+",label="Sales_observed")
plt.plot(x,y_test_pred,"r-*",label="Sales_predicted")
plt.grid(b=None)
plt.xlabel("TV budget")
plt.ylabel("Sales")
plt.title("TV budget vs Sales")
plt.legend()
plt.show()
```

Residual (Error) plot If the model has done good predictions, then the datapoints must be near around to horizontal line.

[23]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_test-y_test_pred
zeros = y_test-y_test
plt.scatter(y_test_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_test_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



Storing the outcome in a file

[24]:
```
# Store the predicted value for sales in new column
dataset.rename(columns={'sales': 'observed_sales'}, inplace=True)
sales_data = dataset.iloc[:,0].values.reshape(-1, 1)
predicted_values = SGDReg.predict(sales_data)
dataset['predicted_sales'] = predicted_values
dataset.head()
```

[24]:

	TV	radio	newspaper	observed_sales	predicted_sales
0	230.1	37.8	69.2	22.1	63.647181
1	44.5	39.3	45.1	10.4	12.590170
2	17.2	45.9	69.3	9.3	5.080168
3	151.5	41.3	58.5	18.5	42.024977
4	180.8	10.8	58.4	12.9	50.085162

[25]:
```
# Write the above output input into new csv
# dataset.to_csv("8.Stochastic Gradient Descent.csv")
```

0.0.9 Different Error calculations to asses the model for the test set

1. Sum of Squared Error (SSE)

[26]:
```
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = y_test[i] - y_test_pred[i]
  squ_diff = diff**2
  sum = sum + squ_diff
Test_SSE = round(sum,2)
print("Sum of Squared Error (SSE) :",Test_SSE)
```

```
Sum of Squared Error (SSE) : 0.72
```

2. Mean Squared Error (MSE)m

[27]:
```
Test_MSE = round(Test_SSE/n,2)
print("Mean Squared Error (MSE) :",Test_MSE)
```

```
Mean Squared Error (MSE) : 0.02
```

3. Root Mean Squared Error (RMSE)

[28]:
```
Test_RMSE = round(np.sqrt(Test_MSE),2)
print("Root Mean Squared Error (RMSE) :",Test_RMSE)
```

```
Root Mean Squared Error (RMSE) : 0.14
```

4. Mean Absolute Error (MAE)

[29]:
```
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = y_test[i] - y_test_pred[i]
  sum = sum + np.abs(diff)
Test_MAE = round(sum/n,2)
print("Mean Absolute Error (MAE) :",Test_MAE)
```

```
Mean Absolute Error (MAE) : 0.11
```

5. Mean Absolute Percentage Error (MAPE)

[30]:
```
sum = 0
n = len(x_test)
for i in range (0,n):
    if y_test[i] == 0:
        continue
    else:
        diff = (y_test[i] - y_test_pred[i])/y_test[i]
    sum = sum + np.abs(diff)
Test_MAPE = round(sum/n*100,2)
print("Mean Absolute Percentage Error (MAPE) :",Test_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 22.68
```

0.0.10 Calculating R-Squred value (goodness of model) using SSE

[31]:
```
from sklearn.metrics import r2_score
out = r2_score(y_test,y_test_pred)
Test_RS = round(out,2)*100
print("R-Squred value (goodness of model) for training set :",Test_RS,"%")
```

```
R-Squred value (goodness of model) for training set : 50.0 %
```

0.0.11 Underfitting and overfitting observation

[32]:

```python
print("Error \t From training phase       From testing phase ")
print("=====================================================")
print("SSE   \t\t",Train_SSE,"\t\t\t", Test_SSE)
print("MSE   \t\t",Train_MSE,"\t\t\t", Test_MSE)
print("RMSE   \t\t",Train_RMSE,"\t\t\t", Test_RMSE)
print("MAE   \t\t",Train_MAE,"\t\t\t", Test_MAE)
print("RS   \t\t",Train_RS,"\t\t\t", Test_RS)
```

```
Error    From training phase       From testing phase
===================================================
SSE              3.84                        0.72
MSE              0.02                        0.02
RMSE             0.14                        0.14
MAE              0.13                        0.11
RS               44.0                        50.0
```