main-lstsq-sgd

November 18, 2021

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab, mlab

from IPython.display import display
from IPython.core.pylabtools import figsize, getfigs

figsize(15, 10)
```

1 General functions

[2]:
```
def zero_mean_normalize(df: pd.DataFrame, columns: list):
    """
    Zero mean normalize the given columns in the given dataframe.
    """
    for column in columns:
        df[column] -= df[column].mean()
        df[column] /= df[column].std()


def min_max_normalize(df: pd.DataFrame, columns: list):
    """
    Min-max normalize the given columns in the given dataframe.
    """
    for column in columns:
        df[column] -= df[column].min()
        df[column] /= (df[column].max() - df[column].min())


def mean_squared_error(y_true: np.ndarray, y_pred: np.ndarray) -> float:
    return np.mean((y_true - y_pred)**2)
```

1.1 Least Square algorithm

[3]:
```python
def calculate_theta(X: np.ndarray, y: np.ndarray) -> tuple[float, np.ndarray]:
    """
    calculate theta and theta_0 for the given X and y.
    return result as a tuple.
    """
    theta = (np.linalg.inv(X.T@X)@X.T)@y
    y_intercept = np.mean(y_train - (X_train@theta))
    return y_intercept, theta
```

1.2 Batch Gradient Descent

refrence: Batch Gradient Descent and Linear Regression

[4]:
```python
def batch_gradient_descend(X: np.ndarray, y: np.ndarray,theta: np.ndarray, bias:
 ↪ float, alpha: float,
                           iterations: int) -> np.ndarray:
    """
    Batch gradient descend algorithm.
    Return theta, bias and costs.
    """
    m_samples, n_features = X.shape
    costs = []
    for i in range(iterations):
        y_predicted = X@theta+bias
        theta -= alpha*(X.T@(y_predicted-y))
        bias -= alpha*(y_predicted-y).sum()
        costs.append(mean_squared_error(y, y_predicted))
    return theta, bias, costs
```

1.3 Stochastic Gradient Descent

[5]:
```python
def stochastic_gradient_descent(X: np.ndarray, y: np.ndarray,
                                theta: np.ndarray, bias: float, alpha: float,
                                num_iterations: int, random_state=None):
    """
    Stochastic gradient descent for linear regression.
    Return theta, bias and costs.
    """
    costs = []*num_iterations
    if random_state:
        np.random.seed(random_state)
    for i in range(num_iterations):
        j = np.random.randint(0, len(X))
        y_predicted = X[j]@theta+bias
        theta -= alpha*(X[j].T@(y_predicted-y[j]))
        bias -= alpha*(y_predicted-y[j])
```

```
        costs.append(mean_squared_error(y, y_predicted))
    return theta, bias, costs
```

2 Home Work Tasks

2.0.1 Read Data

[6]:
```
train = pd.read_csv("Data-Train.csv")
zero_mean_normalize(train, ["x"])

test = pd.read_csv("Data-Test.csv")
zero_mean_normalize(test, ["x"])
```

[7]:
```
X_train = np.array([train["x"]], copy=True).T
y_train = np.array([train["y"]], copy=True).T

X_test = np.array([test["x"]], copy=True).T
y_test = np.array([test["y"]], copy=True).T
```

2.1 Least Square (closed form)

2.1.1 Train plot

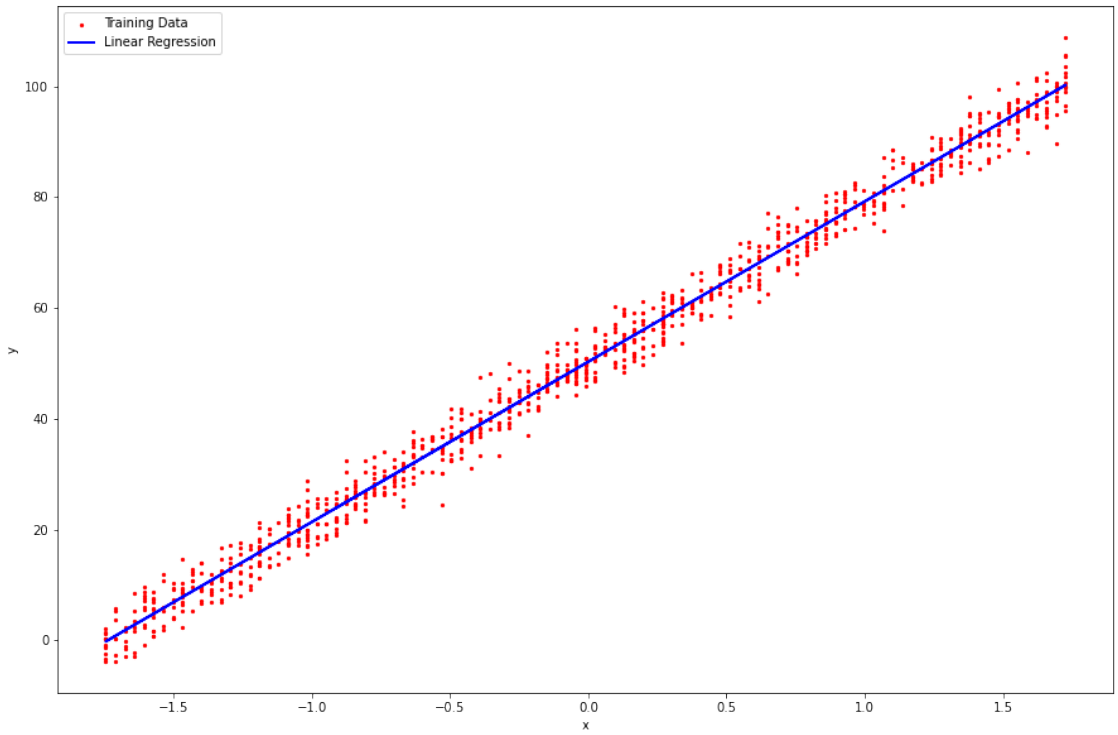
Scatter plot for samples and the fitted line are drawn in red and blue colors respectively. *slope*, *y-intercept* and *MSE* for train data are printed above the figure

[8]:
```
y_intercept, theta = calculate_theta(X_train, y_train)
y_predicted = X_train@theta+y_intercept
train["y_predicted"] = y_predicted
mse_train = mean_squared_error(y_train, y_predicted)

ax = train.plot.scatter(x="x", y="y", s=5, c="red", label="Training Data")
train.plot.line(x="x", y="y_predicted", lw=2, c="b",
                label="Linear Regression", ax=ax);

print("slope:", theta[0][0])
print("y-intercept:", y_intercept)
print("MSE (train):", mse_train)
```

```
slope: 28.937856243162933
y-intercept: 50.299641248803006
MSE (train): 8.328012371573907
```



2.1.2 Test plot

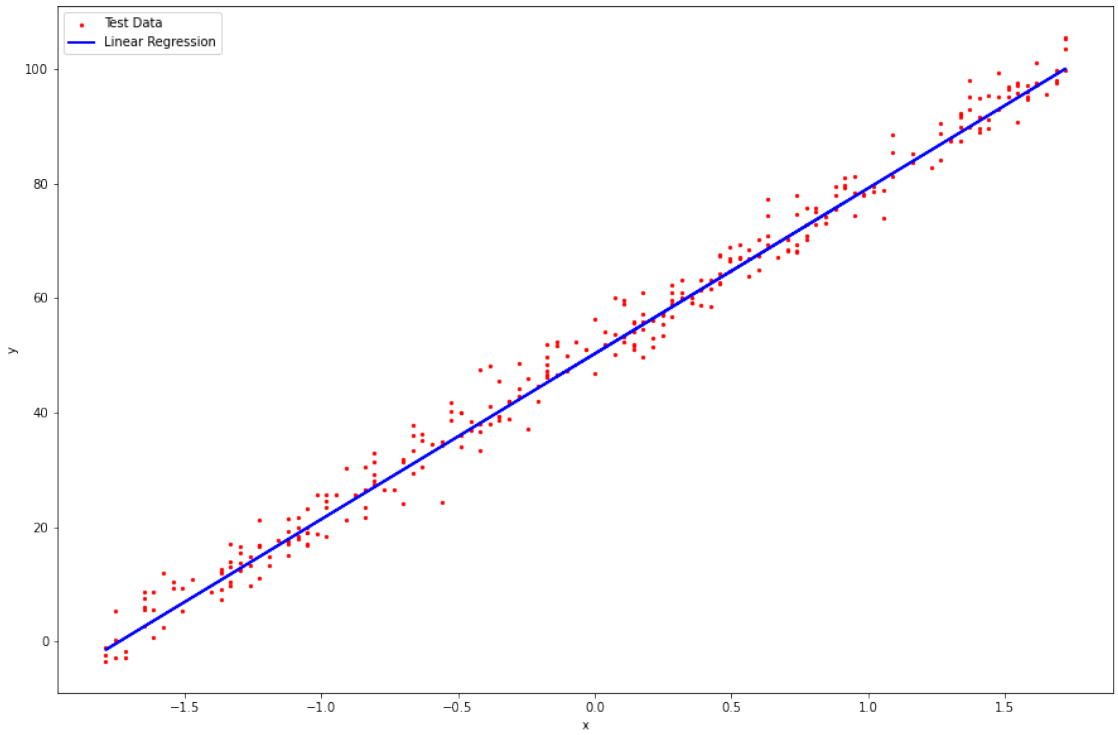
Scatter plot for samples and the fitted line are drawn in red and blue colors respectively. *MSE* for test data is printed above the figure

[9]:
```
y_predicted_test = X_test@theta+y_intercept
test["y_predicted"] = y_predicted_test
mse_test = mean_squared_error(y_test, y_predicted_test)

ax = test.plot.scatter(x="x", y="y", s=5, c="red", label="Test Data")
test.plot(x="x", y="y_predicted", lw=2, c="b",
          label="Linear Regression", ax=ax);

print("MSE (test):", mse_test)
```

```
MSE (test): 9.984675357528035
```



2.2 Batch Gradient Descent

2.2.1 Train Plot

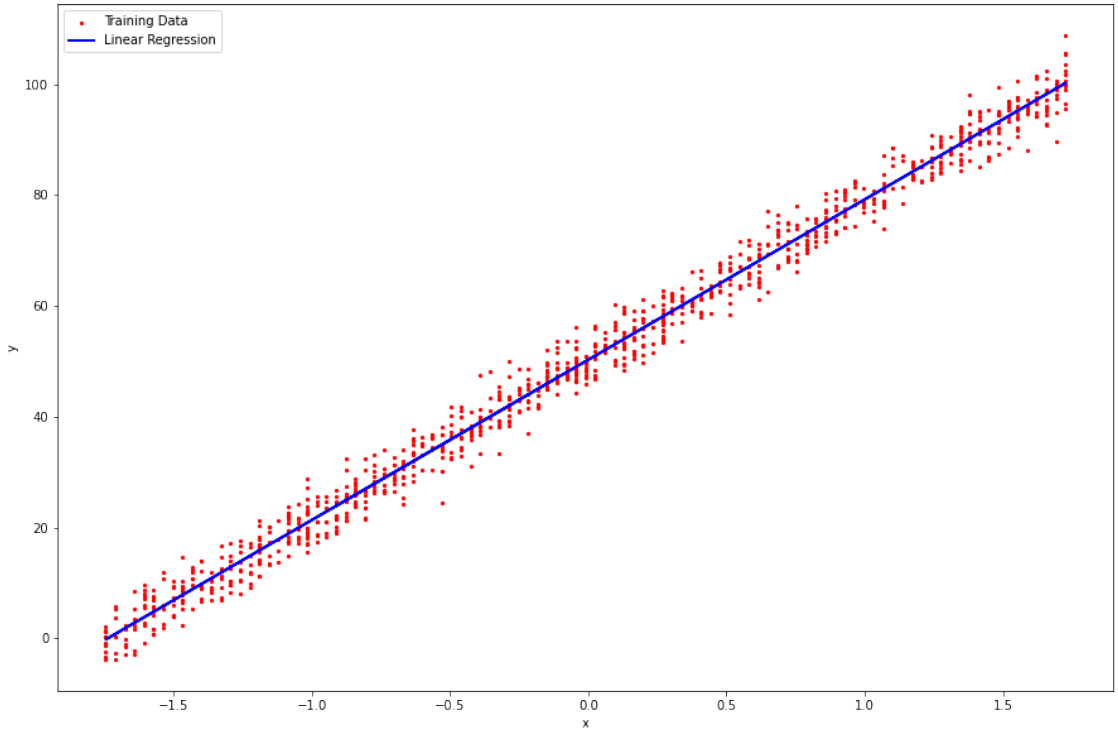
Scatter plot for samples and the fitted line are drawn in red and blue colors respectively. *slope*, *y-intercept* and *MSE* for train data are printed above the figure

[10]:
```
theta = np.array([[1.0]])
bias = 0
alpha = 0.0001
iterations = 100
theta, bias, costs = batch_gradient_descend(
    X_train, y_train, theta, bias, alpha, iterations)

ax = train.plot.scatter(x="x", y="y", s=5, c="red", label="Training Data")
train.plot.line(x="x", y="y_predicted", lw=2, c="b", label="Linear Regression",␣
↪ax=ax);

print("slope:", theta[0][0])
print("intercept:", bias)
print("MSE (train):", costs[-1])
```

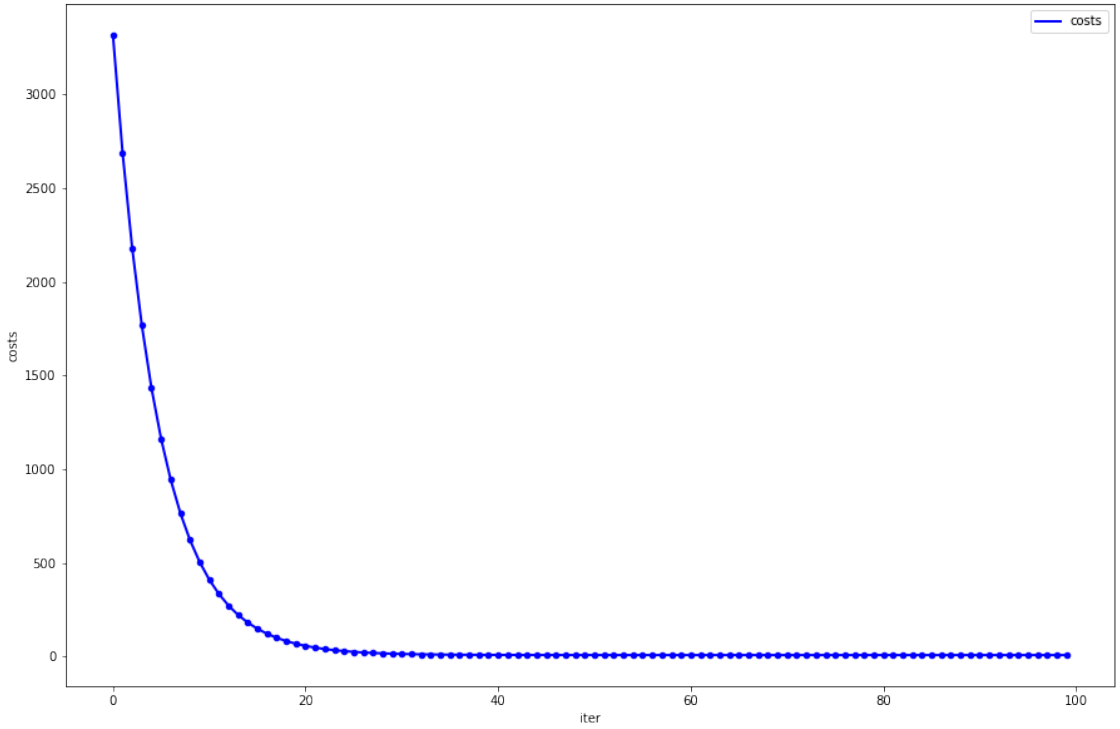
```
slope: 28.937105883899484
intercept: 50.29830521996789
MSE (train): 8.328015269504846
```



2.2.2 Cost (MSE) plot for each iteration

using the list of cost for iterations received from the previous step, figure below is created

[11]:
```
df = pd.DataFrame({"iter": [i for i in range(len(costs))], "costs": costs})
ax = df.plot(x="iter", y="costs", lw=2, c="b")
df.plot.scatter(x="iter", y="costs", s=20, c="b", ax=ax);
```



2.2.3 Test Plot

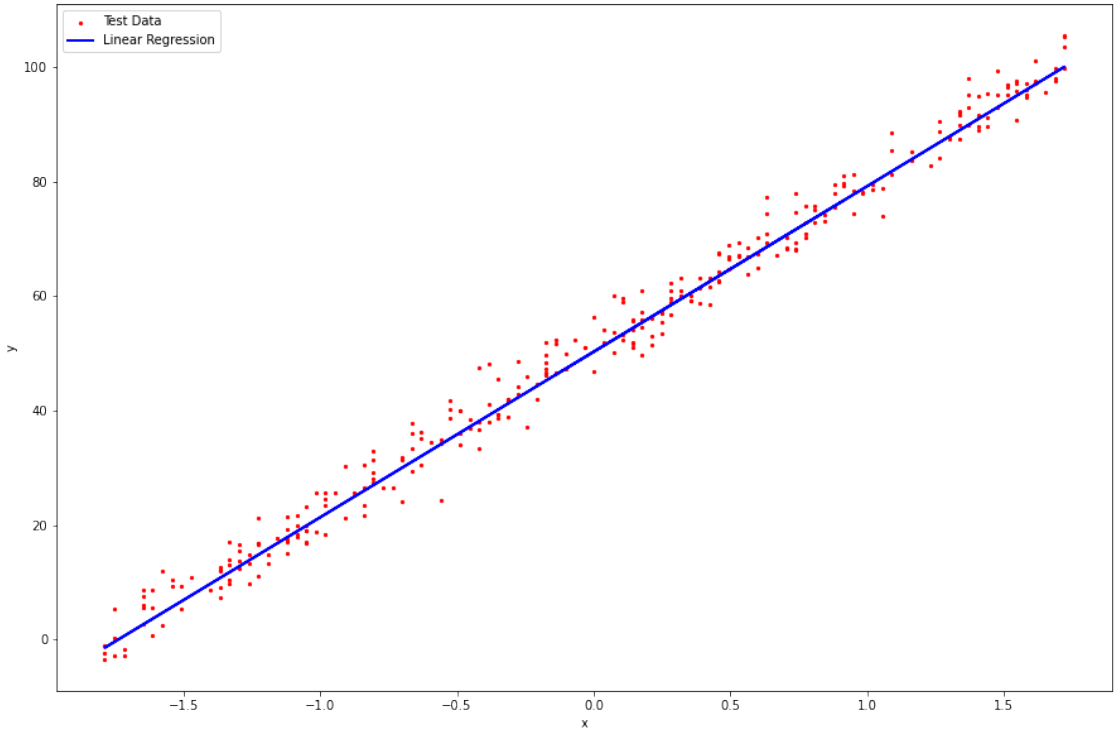
Scatter plot for samples and the fitted line are drawn in red and blue colors respectively. *MSE* for test data is printed above the figure

[12]:
```
y_predicted_test = X_test@theta+bias
test["y_predicted"] = y_predicted_test

ax = test.plot.scatter(x="x", y="y", s=5, c="red", label="Test Data")
test.plot(x="x", y="y_predicted", lw=2, c="b",
          label="Linear Regression", ax=ax);

print("MSE (tets):", mean_squared_error(y_test, X_test@theta+bias))
```

```
MSE (tets): 9.98705969065643
```


Test Data
Linear Regression
100
80
60
40
20
0
y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x