LAB5

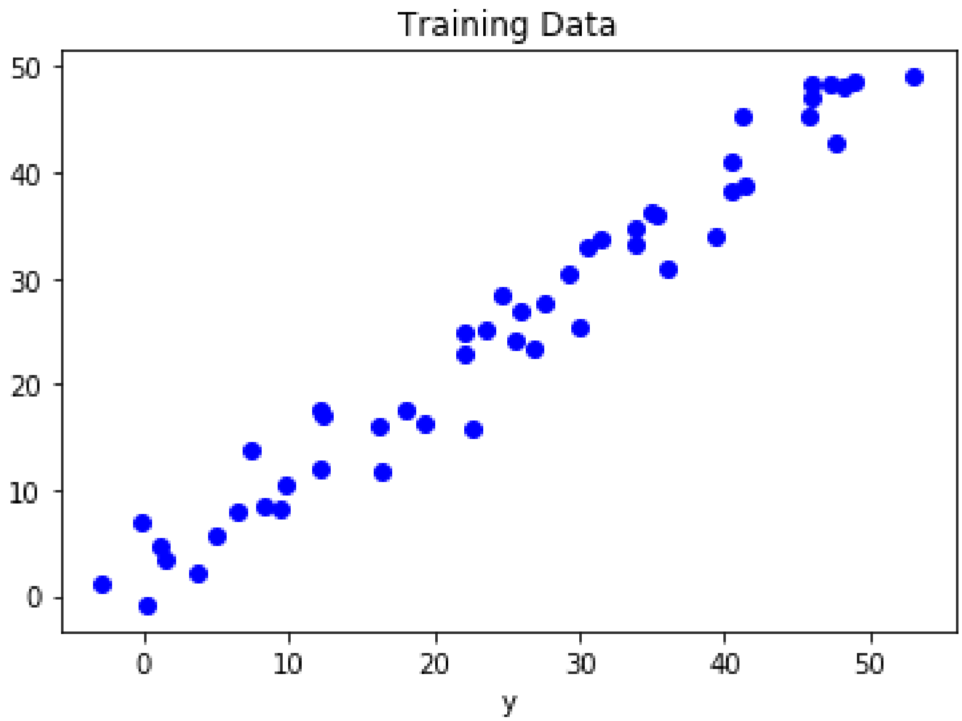
August 25, 2019

[0]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
```

[0]:
```
x = np.linspace(0,50,50)
y = np.linspace(0,50,50)
```

[0]:
```
x += np.random.uniform(-4, 4, 50)
y += np.random.uniform(-4, 4, 50)
n = len(x)
```

[11]:
```
plt.scatter(x, y, color='blue')
plt.xlabel('x')
plt.xlabel('y')
plt.title("Training Data")
plt.show()
```



[0]:
```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
```

[0]:
```
W = tf.Variable(np.random.randn(), name = "W")
b = tf.Variable(np.random.randn(), name = "b")
```

[0]:
```
learning_rate = 0.01
training_epochs = 1500
```

[15]:
```
y_pred = tf.add(tf.multiply(X, W), b)
cost = tf.div(tf.reduce_sum(tf.square(y_pred-Y)),(2 * n))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
init = tf.global_variables_initializer()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0825 10:37:18.480022 139839722747776 deprecation.py:323] From <ipython-
input-15-0315d83416be>:2: div (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
```

[16]:
```
with tf.Session() as sess:
  merged = tf.summary.merge_all()
  writer = tf.summary.FileWriter("logs", sess.graph)
  sess.run(init)
```

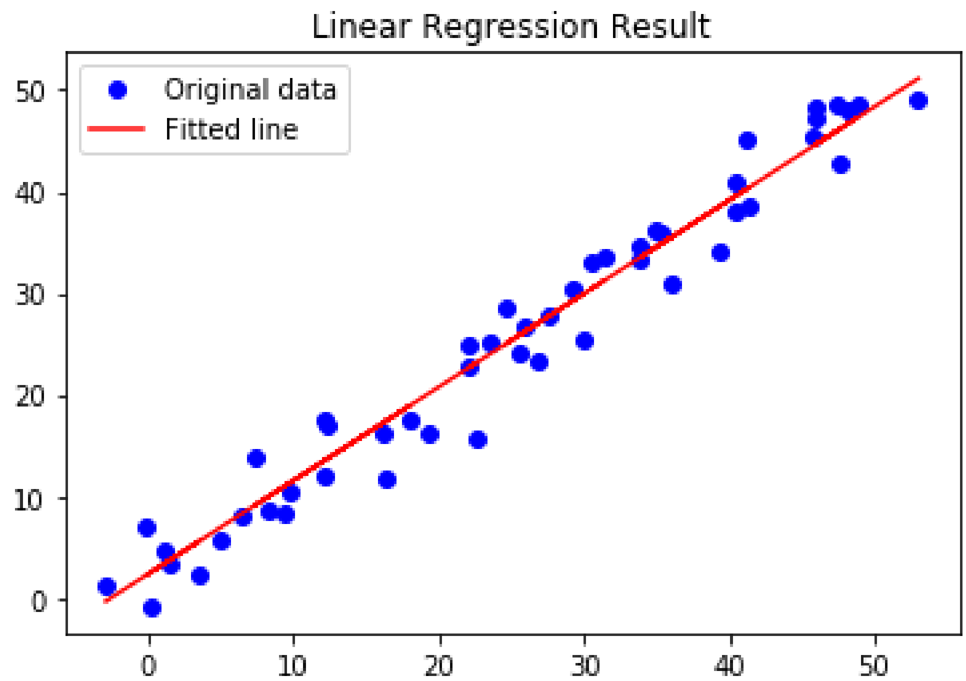
```
for epoch in range(training_epochs):
    for (_x, _y) in zip(x, y):
        sess.run(optimizer, feed_dict = {X : _x, Y : _y})
    if (epoch+1) % 50 == 0:
        c = sess.run(cost, feed_dict = {X : x, Y : y})
        print("Epoch", (epoch+1), ": cost =", c, "W =", sess.run(W), "b =", sess.
↪run(b))
training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
weight = sess.run(W)
bias = sess.run(b)
```

```
Epoch 50 : cost = 6.781017 W = 1.0048156 b = -1.9276061
Epoch 100 : cost = 6.135523 W = 0.9947624 b = -1.4226213
Epoch 150 : cost = 5.637338 W = 0.98585576 b = -0.97522396
Epoch 200 : cost = 5.25382 W = 0.97796464 b = -0.5788447
Epoch 250 : cost = 4.959443 W = 0.9709735 b = -0.22766782
Epoch 300 : cost = 4.73428 W = 0.9647795 b = 0.0834629
Epoch 350 : cost = 4.562768 W = 0.95929193 b = 0.3591139
Epoch 400 : cost = 4.432778 W = 0.9544301 b = 0.6033299
Epoch 450 : cost = 4.3348465 W = 0.9501227 b = 0.8196979
Epoch 500 : cost = 4.261615 W = 0.9463064 b = 1.0113928
Epoch 550 : cost = 4.2073536 W = 0.94292545 b = 1.1812264
Epoch 600 : cost = 4.1676173 W = 0.93992996 b = 1.3316915
Epoch 650 : cost = 4.1389556 W = 0.93727607 b = 1.4649987
Epoch 700 : cost = 4.1186976 W = 0.9349249 b = 1.5831044
Epoch 750 : cost = 4.104782 W = 0.9328418 b = 1.6877413
Epoch 800 : cost = 4.0956173 W = 0.93099624 b = 1.7804458
Epoch 850 : cost = 4.0899816 W = 0.92936116 b = 1.862578
Epoch 900 : cost = 4.0869384 W = 0.92791253 b = 1.9353461
Epoch 950 : cost = 4.085772 W = 0.92662907 b = 1.9998158
Epoch 1000 : cost = 4.08594 W = 0.9254919 b = 2.056933
Epoch 1050 : cost = 4.0870333 W = 0.9244846 b = 2.1075342
Epoch 1100 : cost = 4.0887394 W = 0.92359203 b = 2.1523685
Epoch 1150 : cost = 4.0908327 W = 0.9228014 b = 2.1920836
Epoch 1200 : cost = 4.0931444 W = 0.9221007 b = 2.227279
Epoch 1250 : cost = 4.09555 W = 0.9214801 b = 2.2584536
Epoch 1300 : cost = 4.097961 W = 0.92093027 b = 2.286073
Epoch 1350 : cost = 4.10032 W = 0.92044294 b = 2.3105524
Epoch 1400 : cost = 4.1025815 W = 0.9200113 b = 2.3322346
Epoch 1450 : cost = 4.1047196 W = 0.919629 b = 2.3514392
Epoch 1500 : cost = 4.106723 W = 0.9192901 b = 2.3684573
```

[17]:
```
predictions = weight * x + bias
print("Training cost =", training_cost, "Weight =", weight, "bias =", bias,␣
↪'\n')
```

```
Training cost = 4.106723 Weight = 0.9192901 bias = 2.3684573
```

[18]:
```python
plt.plot(x, y,'ro', color = 'blue', label ='Original data')
plt.plot(x, predictions, color = 'red', label ='Fitted line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
```



[19]:
```python
from sklearn.metrics import r2_score
R2 = r2_score(y, predictions, multioutput='variance_weighted')
print("R2 value:",R2)
```

```
R2 value: 0.9626758714904278
```

[20]:
```python
'''
Exercise 1:
Predict Y values for given X values.
'''
predictions1 = weight * np.array([3.987, 19.235, 23.098, 36.5, 22.765]) + bias
predictions1
```

[20]:
```
array([ 6.03366705, 20.05100288, 23.60222063, 35.92254689, 23.29609702])
```

[21]:
```python
'''
Exercise 2:
Change Stopping criterion to delta(J)<=0.000001
'''
```

```
with tf.Session() as sess:
  merged = tf.summary.merge_all()
  writer = tf.summary.FileWriter("logs", sess.graph)
  sess.run(init)
  prev = 0.0
  iters = 1
  while(iters):
    for (_x, _y) in zip(x, y):
      sess.run(optimizer, feed_dict = {X : _x, Y : _y})
    c = sess.run(cost, feed_dict = {X : x, Y : y})
    if iters % 50 == 0:
      print("Epoch", (iters), ": cost =", c, "W =", sess.run(W), "b =", sess.
↪run(b))
    if(abs(c-prev)<0.000001):
      break
    prev = c
    iters+=1
  training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
  weight = sess.run(W)
  bias = sess.run(b)
```

```
Epoch 50 : cost = 6.781017 W = 1.0048156 b = -1.9276061
Epoch 100 : cost = 6.135523 W = 0.9947624 b = -1.4226213
Epoch 150 : cost = 5.637338 W = 0.98585576 b = -0.97522396
Epoch 200 : cost = 5.25382 W = 0.97796464 b = -0.5788447
Epoch 250 : cost = 4.959443 W = 0.9709735 b = -0.22766782
Epoch 300 : cost = 4.73428 W = 0.9647795 b = 0.0834629
Epoch 350 : cost = 4.562768 W = 0.95929193 b = 0.3591139
Epoch 400 : cost = 4.432778 W = 0.9544301 b = 0.6033299
Epoch 450 : cost = 4.3348465 W = 0.9501227 b = 0.8196979
Epoch 500 : cost = 4.261615 W = 0.9463064 b = 1.0113928
Epoch 550 : cost = 4.2073536 W = 0.94292545 b = 1.1812264
Epoch 600 : cost = 4.1676173 W = 0.93992996 b = 1.3316915
Epoch 650 : cost = 4.1389556 W = 0.93727607 b = 1.4649987
Epoch 700 : cost = 4.1186976 W = 0.9349249 b = 1.5831044
Epoch 750 : cost = 4.104782 W = 0.9328418 b = 1.6877413
Epoch 800 : cost = 4.0956173 W = 0.93099624 b = 1.7804458
Epoch 850 : cost = 4.0899816 W = 0.92936116 b = 1.862578
Epoch 900 : cost = 4.0869384 W = 0.92791253 b = 1.9353461
Epoch 950 : cost = 4.085772 W = 0.92662907 b = 1.9998158
```

[22]:
```
'''
Exercise 3
Diffrent alpha values for 1500 epoches:
learning rate: 0.01 --> R2 value: 0.9618197718107743
learning rate: 0.05 --> R2 value: -0.9297278609667908
learning rate: 0.2 --> All values are Nan
```

```
learning rate: 0.5 --> All values are Nan
'''
```

[22]:
```
'\nExercise 3\nDiffrent alpha values for 1500 epoches:\nlearning rate: 0.01 -->
R2 value: 0.9618197718107743\nlearning rate: 0.05 --> R2 value:
-0.9297278609667908\nlearning rate: 0.2 --> All values are Nan\nlearning rate:
0.5 --> All values are Nan\n'
```

[23]:
```
boston = load_boston()
df = pd.DataFrame(
    data= np.c_[boston['data']],
    columns= boston['feature_names'])
df.insert(13,'target',boston['target'],True)
print(boston['DESCR'])
df.head()
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**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.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

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 regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

[23]:

	CRIM	ZN	INDUS	CHAS	NOX	...	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	...	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	...	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	...	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	...	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	...	222.0	18.7	396.90	5.33	36.2

[5 rows x 14 columns]

[24]:
```python
corr = df.corr()
corr
```

[24]:

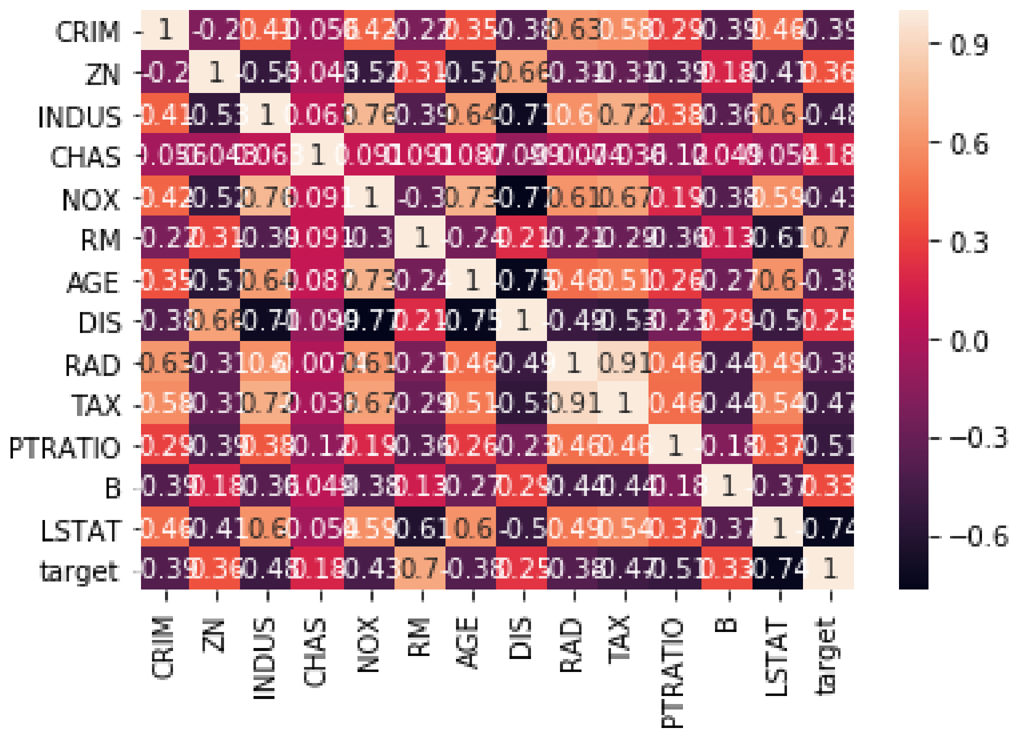
	CRIM	ZN	INDUS	...	B	LSTAT	target
CRIM	1.000000	-0.200469	0.406583	...	-0.385064	0.455621	-0.388305
ZN	-0.200469	1.000000	-0.533828	...	0.175520	-0.412995	0.360445
INDUS	0.406583	-0.533828	1.000000	...	-0.356977	0.603800	-0.483725
CHAS	-0.055892	-0.042697	0.062938	...	0.048788	-0.053929	0.175260
NOX	0.420972	-0.516604	0.763651	...	-0.380051	0.590879	-0.427321
RM	-0.219247	0.311991	-0.391676	...	0.128069	-0.613808	0.695360

```
AGE      0.352734 -0.569537  0.644779  ... -0.273534  0.602339 -0.376955
DIS     -0.379670  0.664408 -0.708027  ...  0.291512 -0.496996  0.249929
RAD      0.625505 -0.311948  0.595129  ... -0.444413  0.488676 -0.381626
TAX      0.582764 -0.314563  0.720760  ... -0.441808  0.543993 -0.468536
PTRATIO  0.289946 -0.391679  0.383248  ... -0.177383  0.374044 -0.507787
B       -0.385064  0.175520 -0.356977  ...  1.000000 -0.366087  0.333461
LSTAT    0.455621 -0.412995  0.603800  ... -0.366087  1.000000 -0.737663
target  -0.388305  0.360445 -0.483725  ...  0.333461 -0.737663  1.000000

[14 rows x 14 columns]
```

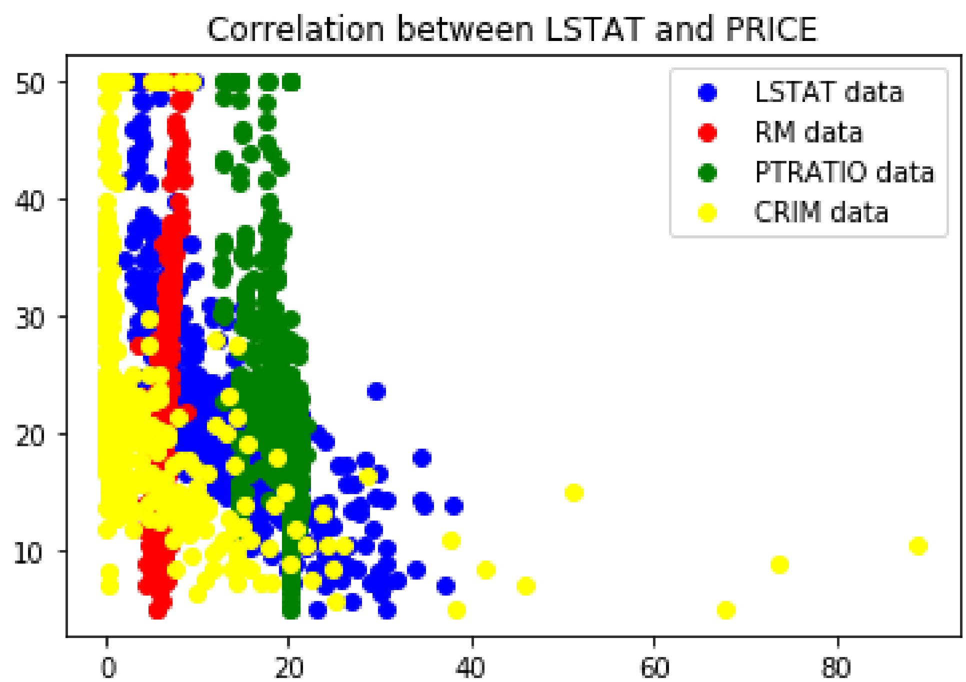
[25]:
```python
sns.heatmap(corr,annot=True)
```

[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ead68b0b8>
```



[26]:
```python
x1 =␣
↪df[['LSTAT','RM','PTRATIO','CRIM','ZN','INDUS','CHAS','NOX','AGE','DIS','RAD','TAX','B']]
y1 = df[['target']]
n1 = len(x1)
plt.plot(x1['LSTAT'], y1,'ro', color = 'blue', label ='LSTAT data')
plt.plot(x1['RM'], y1,'ro', color = 'red', label ='RM data')
plt.plot(x1['PTRATIO'], y1,'ro', color = 'green', label ='PTRATIO data')
plt.plot(x1['CRIM'], y1,'ro', color = 'yellow', label ='CRIM data')
plt.title('Correlation between LSTAT and PRICE')
```

```
plt.legend()
plt.show()
```



[0]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

[28]:
```
x_train,x_test,y_train,y_test = train_test_split(x1,y1,test_size=0.
 ↪2,random_state=5)
regr = LinearRegression()
regr.fit(x_train,y_train)
```

[28]: `LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)`

[30]:
```
y_predict = regr.predict(x_test)
r2 = r2_score(y_test,y_predict)
print("R2 score is:",format(r2))
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_predict)
print("Mean Square Error : ", mse)
print(y_test.head())
```

```
R2 score is: 0.7334492147453128
Mean Square Error :  20.869292183770405
     target
226    37.6
292    27.9
```

90	22.6
373	13.8
273	35.2