Machine Learning - simple linear regression with python

August 11, 2021

[1]:
```python
# Linear Equation y = a_0 + a_1 * x
```

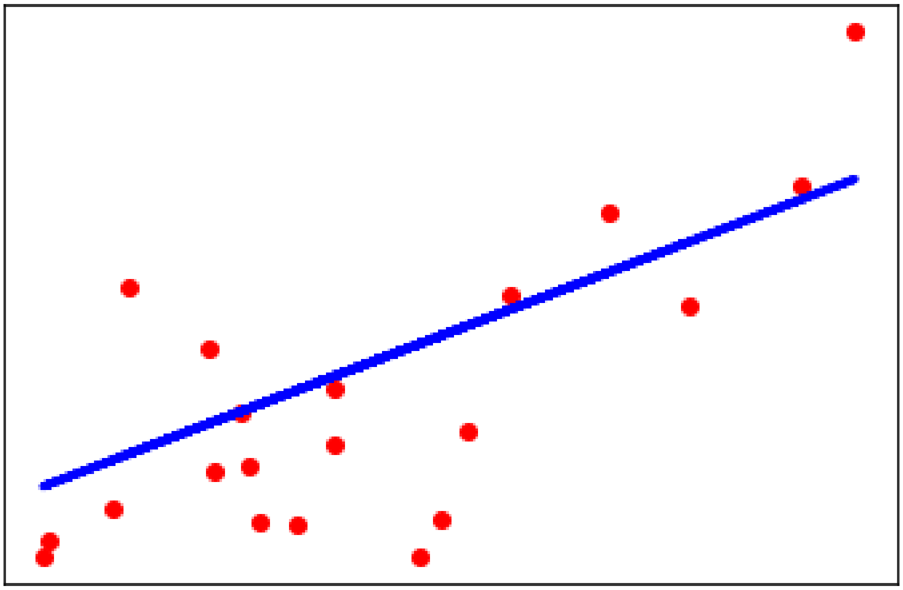
[2]:
```python
# 1 Basic Theory: Ordinary Least Squares (scipy.linalg.lstsq) Linear Regression

import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)

print('score is', reg.score(X, y))

print('coef is', reg.coef_)

print('intercept is', reg.intercept_)

print('predict is', reg.predict(np.array([[3, 5]])))
```

```
score is 1.0
coef is [1. 2.]
intercept is 3.0000000000000018
predict is [16.]
```

[30]:
```python
# 2 Basic Theory: Ordinary Least Squares (scipy.linalg.lstsq) Linear Regression

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
```

```python
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='r')
plt.plot(diabetes_X_test, diabetes_y_pred, color='b', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

```
Coefficients: 
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



[4]:
```
# Create a pandas DataFrame for the counts data set.
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/tvelichkovt/PyTorch/main/
↪nyc_bb_bicyclist_counts.csv", header=0, infer_datetime_format=True,␣
↪parse_dates=[0], index_col=[0])

df.head()
```

[4]:

Date	HIGH_T	LOW_T	PRECIP	BB_COUNT
2017-04-01	46.0	37.0	0.00	606
2017-04-02	62.1	41.0	0.00	2021
2017-04-03	63.0	50.0	0.03	2470
2017-04-04	51.1	46.0	1.18	723
2017-04-05	63.0	46.0	0.00	2807

[5]:
```
df.describe()
```

[5]:

	HIGH_T	LOW_T	PRECIP	BB_COUNT
count	214.000000	214.000000	214.000000	214.000000
mean	74.201869	62.027103	0.132430	2680.042056
std	10.390443	9.305792	0.394004	854.710864
min	46.000000	37.000000	0.000000	151.000000
25%	66.900000	55.225000	0.000000	2298.000000
50%	75.900000	64.000000	0.000000	2857.000000
75%	82.000000	70.000000	0.037500	3285.000000

```
max     93.900000   78.100000    3.030000  4960.000000
```

[6]:
```
import pandas as pd
from patsy import dmatrices
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

[7]:
```
# Add a few derived regression variables.
ds = df.index.to_series()

#before
print('before is: ', df.head().index[1])

#after
print('after is: ', ds.dt.month[1:2])
```

```
before is:  2017-04-02 00:00:00
after is:  Date
2017-04-02    4
Name: Date, dtype: int64
```

[8]:
```
df['MONTH'] = ds.dt.month
df['DAY_OF_WEEK'] = ds.dt.dayofweek
df['DAY'] = ds.dt.day

df.to_csv("nyc_bb_bicyclist_counts_output.csv")

df.head()
```

[8]:

Date	HIGH_T	LOW_T	PRECIP	BB_COUNT	MONTH	DAY_OF_WEEK	DAY
2017-04-01	46.0	37.0	0.00	606	4	5	1
2017-04-02	62.1	41.0	0.00	2021	4	6	2
2017-04-03	63.0	50.0	0.03	2470	4	0	3
2017-04-04	51.1	46.0	1.18	723	4	1	4
2017-04-05	63.0	46.0	0.00	2807	4	2	5

[9]:
```
# Create the training and testing data sets.
mask = np.random.rand(len(df)) < 0.8
df_train = df[mask]
df_test = df[~mask]

df_train.to_csv("nyc_bb_bicyclist_counts_output_train.csv")
df_test.to_csv("nyc_bb_bicyclist_counts_output_test.csv")
```

```
print('Training data set length='+str(len(df_train)))
print('Testing data set length='+str(len(df_test)))
```

```
Training data set length=184
Testing data set length=30
```

[10]:
```
# Setup the regression expression in patsy notation. We are telling patsy that␣
↪BB_COUNT is our dependent variable and
# it depends on the regression variables: DAY, DAY_OF_WEEK, MONTH, HIGH_T,␣
↪LOW_T and PRECIP.
expr = """BB_COUNT ~ DAY  + DAY_OF_WEEK + MONTH + HIGH_T + LOW_T + PRECIP"""

expr
```

[10]:
```
'BB_COUNT ~ DAY  + DAY_OF_WEEK + MONTH + HIGH_T + LOW_T + PRECIP'
```

[11]:
```
# Set up the X and y matrices
y_train, X_train = dmatrices(expr, df_train, return_type='dataframe')
y_test, X_test = dmatrices(expr, df_test, return_type='dataframe')
```

[12]:
```
# Using the statsmodels GLM class, train the Poisson regression model on the␣
↪training data set.
poisson_training_results = sm.GLM(y_train, X_train, family=sm.families.
↪Poisson()).fit()
```

[13]:
```
# Print the training summary.
print(poisson_training_results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:               BB_COUNT   No. Observations:                  184
Model:                            GLM   Df Residuals:                      177
Model Family:                 Poisson   Df Model:                            6
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -13341.
Date:                Tue, 10 Aug 2021   Deviance:                       24905.
Time:                        07:18:16   Pearson chi2:                 2.47e+04
No. Iterations:                     5
Covariance Type:            nonrobust
===============================================================================
                coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept     6.9769      0.012    578.315      0.000       6.953       7.000
DAY           0.0012      0.000      7.089      0.000       0.001       0.001
DAY_OF_WEEK  -0.0180      0.001    -25.360      0.000      -0.019      -0.017
MONTH         0.0181      0.001     23.871      0.000       0.017       0.020
HIGH_T        0.0248      0.000     76.862      0.000       0.024       0.025
```

```
LOW_T           -0.0156      0.000    -43.273      0.000      -0.016      -0.015
PRECIP          -0.7524      0.008    -96.031      0.000      -0.768      -0.737
================================================================================
```

[14]:
```python
# Make some predictions on the test data set.
poisson_predictions = poisson_training_results.get_prediction(X_test)
```

[15]:
```python
# .summary_frame() returns a pandas DataFrame
predictions_summary_frame = poisson_predictions.summary_frame()
print(predictions_summary_frame.head())
```

```
                   mean    mean_se  mean_ci_lower  mean_ci_upper
Date
2017-04-05  2607.539028  10.797762    2586.461453    2628.788368
2017-04-17  2694.046390  10.213303    2674.102870    2714.138648
2017-04-25  1029.791080   7.646093    1014.913528    1044.886722
2017-05-03  2307.691014   9.049759    2290.021803    2325.496556
2017-05-04  2447.816585   8.428800    2431.352062    2464.392603
```

[16]:
```python
predicted_counts = predictions_summary_frame['mean']
actual_counts = y_test['BB_COUNT']

output = pd.concat([(predicted_counts), (actual_counts)], axis=1, join='inner',␣
 ↪ignore_index=False, keys=None,
          levels=None, names=None, verify_integrity=False, copy=True)

output['diff'] = output['mean'] - output['BB_COUNT']

output.head()
```

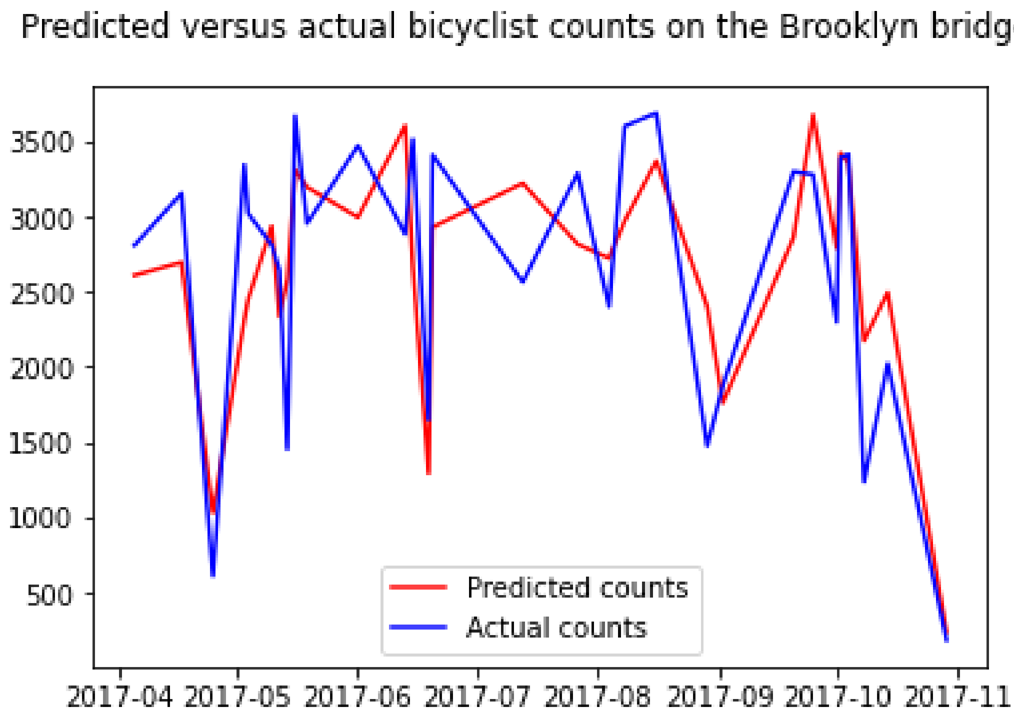
[16]:
```
                   mean  BB_COUNT         diff
Date
2017-04-05  2607.539028    2807.0  -199.460972
2017-04-17  2694.046390    3152.0  -457.953610
2017-04-25  1029.791080     611.0   418.791080
2017-05-03  2307.691014    3342.0 -1034.308986
2017-05-04  2447.816585    3019.0  -571.183415
```

[17]:
```python
percentage_err = "{:.2%}".format(output['diff'].abs().sum()/output['mean'].
 ↪sum())
percentage_err
```

[17]: '17.76%'

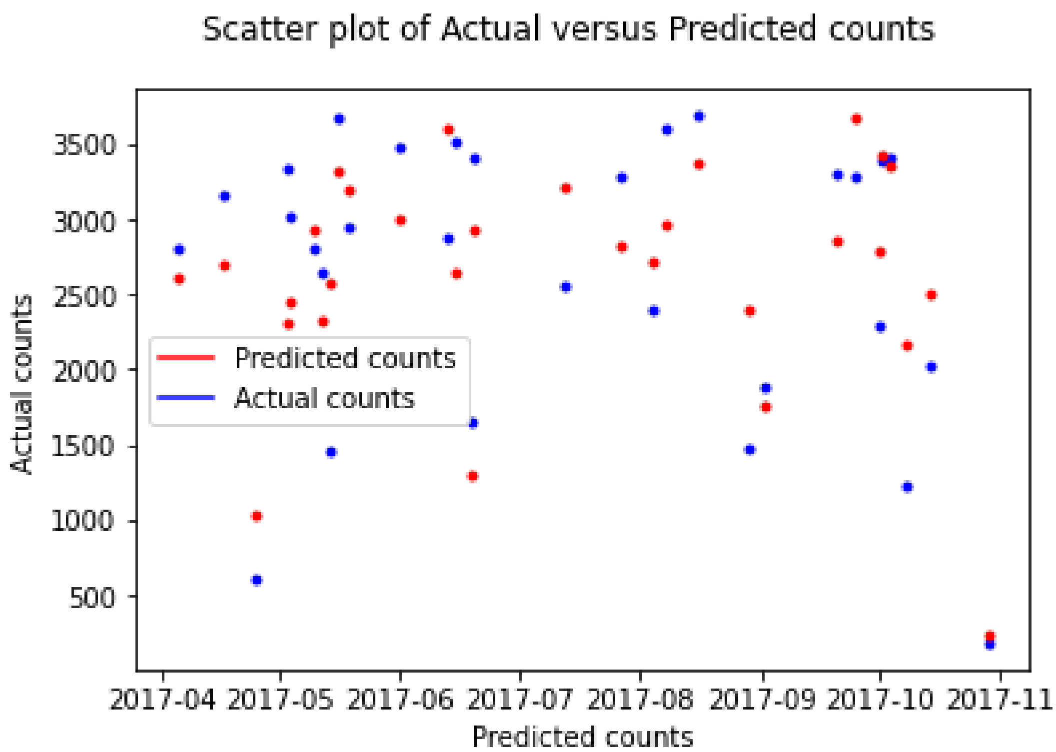
[27]:
```python
# Plot the predicted counts versus the actual counts for the test data.
fig = plt.figure()
fig.suptitle('Predicted versus actual bicyclist counts on the Brooklyn bridge')
```

```
predicted, = plt.plot(X_test.index, predicted_counts, 'r', label='Predicted␣
 ↪counts')
actual, = plt.plot(X_test.index, actual_counts, 'b', label='Actual counts')
plt.legend(handles=[predicted, actual])
plt.show()
```



[29]:
```
# Show scatter plot of Actual versus Predicted counts
plt.clf()
fig = plt.figure()
fig.suptitle('Scatter plot of Actual versus Predicted counts')
plt.scatter(X_test.index, actual_counts, label='original data',  marker='.',␣
 ↪color='b')
plt.scatter(X_test.index, predicted_counts, label='predicted data',  marker='.
 ↪', color='r')
plt.legend(handles=[predicted, actual])
plt.xlabel('Predicted counts')
plt.ylabel('Actual counts')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```


Scatter plot of Actual versus Predicted counts
Actual counts
Predicted counts
Predicted counts
Actual counts
3500
3000
2500
2000
1500
1000
500
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11