“Data Processing Using Python” project—Linear Regression for the Boston house price prediction

Dazhuang@NJU

1. Background

Classification and regression belong to the two methods of supervised learning algorithms (the clustering belongs to the unsupervised learning algorithm) in the machine learning field. The supervised learning is to train a model through the existing training samples, and then input all the inputs and get corresponding output basing on the model. If the output is discrete, for example, it will rain today (1) or it will not rain (0) today, it is called classification. If the output is continuous, for example, input the attributes of a group of houses and then predict the price, it is called regression. This project will use the famous Boston house-price data to introduce simple linear regression. If the model prediction effect is not ideal, then further non-linear regression attempts can be made.

2. Dataset and Algorithm

(1) Data

The Boston house-price data is the same basic data set in machine learning feild as iris and minst. We can build experiments on the dataset and further crawl network data for similar research.

Obtaining Boston house-price data is similar to getting an iris data set:

```
from sklearn import datasets     # Using sklearn
boston = datasets.load_boston()
X = boston.data
y = boston.target
print(X.shape)
(506, 13)
```

There are 506 records in the Boston house-price data and 13 features/ attributes in every record about the house. The house price is recorded in the boston.target of MEDV feature. Please find the details below.

```
>>> print(boston.DESCR)
…
: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
```

(2) Linear Regression

The explanation of Linear Regression in Wiki is as follows:

"In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression."

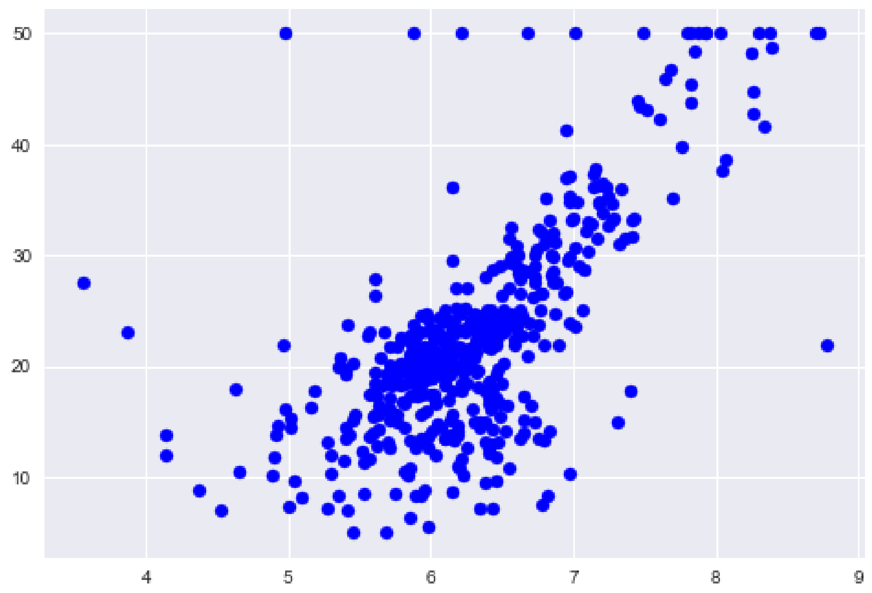
The original purpose of regression analysis was to estimate the parameters of the model in order to achieve the best fit to the data. Among the different criteria for determining the best fit, Linear least squares methods are very advantageous.

If there is only one independent variable and one dependent variable in the regression analysis model and the relationship between the two can be represented by a straight line, then it is called linear regression. If two or more independent variables are included, and the linear relationship between the variable and the independent variables is called the Multiple Linear Regression Analysis. The complete form of linear regression analysis is as follows:

$y = \alpha+\beta_1x_1+\beta_2x_2+\ldots+\beta_nx_n+\varepsilon$
x is the independent variable, y is the dependent variable, ε is the error variable

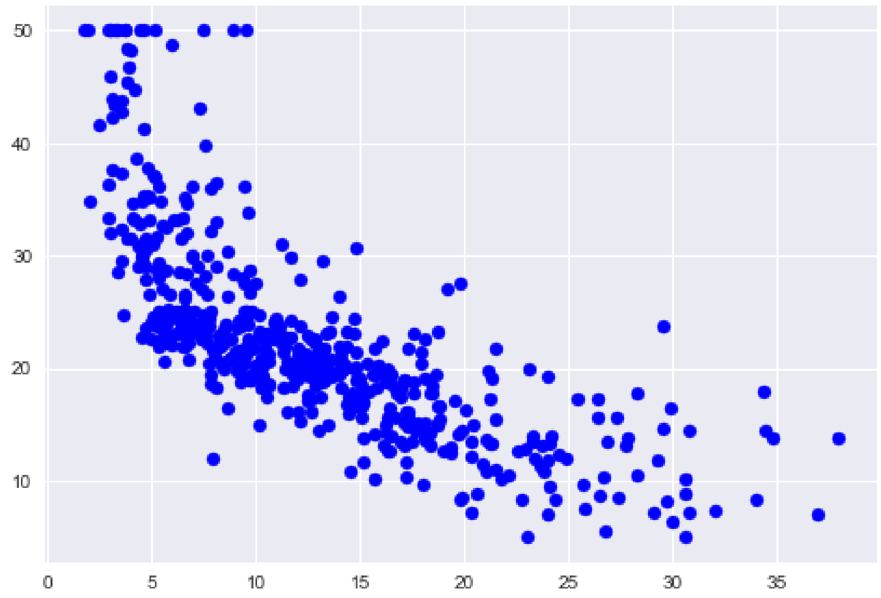
Is there a linear relationship between the features of the Boston house and the house price feature MEDV? Simply draw, for example, the scatter plot between the average room number per house and the house price MEDV, which is highly correlated with the price, is as follows:

```
>>> plt.scatter(X['RM'], y, color='blue')
```



The scatter plot between the features of LSTAT and MEDV is as follows:

```
>>> plt.scatter(X['LSTAT'], y, color='blue')
```



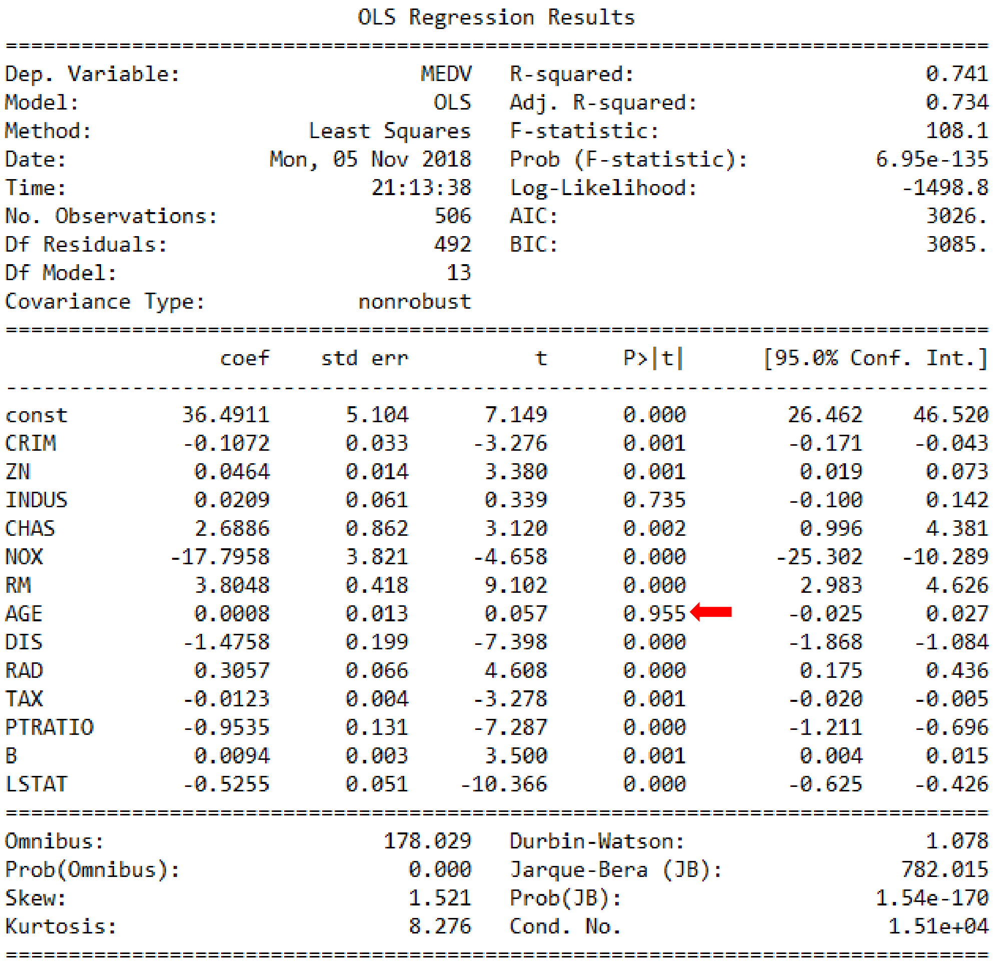
From the graph, there is a certain linear relationship, and we can try it with a linear regression model. There are 13 housing features in the Boston house-price data and which features can be selected as the independent variables of the regression equation? One way is to use the correlation analysis to view the attributes that are highly correlated with the house price feature MEDV. We can also use the SelectKBest() function in sklearn.feature_selection to specify the number of the best attributes to select the best attributes. We can also using T test to determine the significance of the independent variables to select features.

Taking the last method as an example, set a threshold for measuring the significance of the features, generally 0.05, put all features into the model for training and calculate the p value of each feature (if p value is smaller than the selected significance level 0.05 or lower as 0.01, then the null hypothesis that the coefficient should not be present in the model would be rejected, so select the feature), remove the feature with p value above the threshold from the training model. Then use the remaining features to perform a new round of fitting. If there are still features with a p value higher than the threshold, continue to remove until the conditions are met.

Use the Python statistical analysis module statsmodels to calculate the p-values of the features and train the model, predict the price basing on the model.

```
    boston = datasets.load_boston()
    # 13 features about the houses
    X = pd.DataFrame(boston.data, columns = boston.feature_names)
    y = pd.DataFrame(boston.target, columns = ['MEDV'])
    import statsmodels.api as sm
    # The linear regression model in statsmodels has no intercept term, and the following statement
adds a list of 1 to the training set
    X_add1 = sm.add_constant(X)
    # sm.OLS() is a normal least squares regression model, fit() is used for fitting
    model = sm.OLS(y, X_add1).fit()
    print (model.summary())
```

Output:

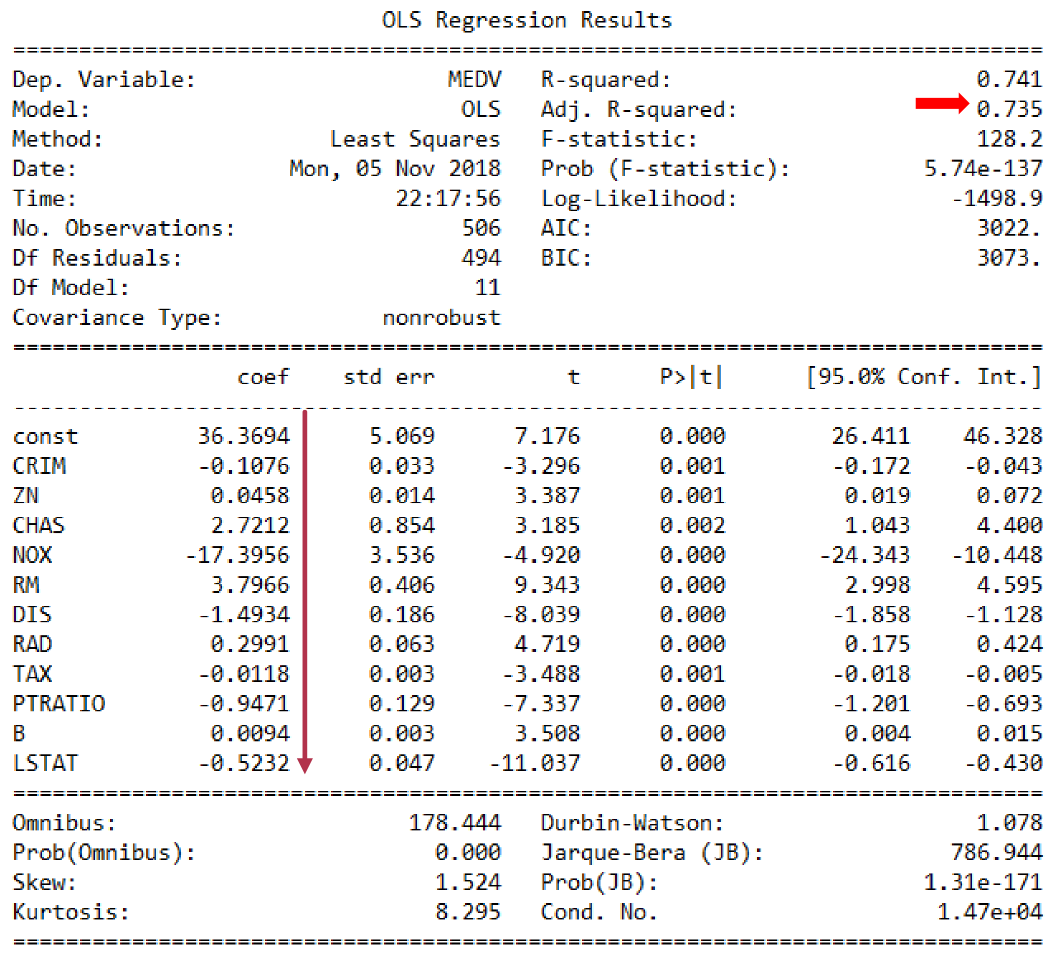


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.741
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     108.1
Date:                Mon, 05 Nov 2018   Prob (F-statistic):          6.95e-135
Time:                        21:13:38   Log-Likelihood:                -1498.8
No. Observations:                 506   AIC:                             3026.
Df Residuals:                     492   BIC:                             3085.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         36.4911      5.104      7.149      0.000        26.462    46.520
CRIM          -0.1072      0.033     -3.276      0.001        -0.171    -0.043
ZN             0.0464      0.014      3.380      0.001         0.019     0.073
INDUS          0.0209      0.061      0.339      0.735        -0.100     0.142
CHAS           2.6886      0.862      3.120      0.002         0.996     4.381
NOX          -17.7958      3.821     -4.658      0.000       -25.302   -10.289
RM             3.8048      0.418      9.102      0.000         2.983     4.626
AGE            0.0008      0.013      0.057      0.955        -0.025     0.027
DIS           -1.4758      0.199     -7.398      0.000        -1.868    -1.084
RAD            0.3057      0.066      4.608      0.000         0.175     0.436
TAX           -0.0123      0.004     -3.278      0.001        -0.020    -0.005
PTRATIO       -0.9535      0.131     -7.287      0.000        -1.211    -0.696
B              0.0094      0.003      3.500      0.001         0.004     0.015
LSTAT         -0.5255      0.051    -10.366      0.000        -0.625    -0.426
==============================================================================
Omnibus:                      178.029   Durbin-Watson:                   1.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              782.015
Skew:                           1.521   Prob(JB):                    1.54e-170
Kurtosis:                       8.276   Cond. No.                     1.51e+04
==============================================================================
```

It can be seen that the p value of the attribute AGE is much higher than the threshold value of 0.05, so the feature can be removed. The INDUS feature is also above the threshold after the continued training and the INDUS feature should be also removed. There are 11 features are left for training finally.

```
# remove two features
X.drop('AGE', axis = 1, inplace = True)
X.drop('INDUS', axis = 1, inplace = True)
# train again
X_add1 = sm.add_constant(X)
model = sm.OLS(y, X_add1).fit()
print(model.summary())
```

Output:



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.741
Model:                            OLS   Adj. R-squared:                  0.735
Method:                 Least Squares   F-statistic:                     128.2
Date:                Mon, 05 Nov 2018   Prob (F-statistic):          5.74e-137
Time:                        22:17:56   Log-Likelihood:                -1498.9
No. Observations:                 506   AIC:                             3022.
Df Residuals:                     494   BIC:                             3073.
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         36.3694      5.069      7.176      0.000        26.411    46.328
CRIM          -0.1076      0.033     -3.296      0.001        -0.172    -0.043
ZN             0.0458      0.014      3.387      0.001         0.019     0.072
CHAS           2.7212      0.854      3.185      0.002         1.043     4.400
NOX          -17.3956      3.536     -4.920      0.000       -24.343   -10.448
RM             3.7966      0.406      9.343      0.000         2.998     4.595
DIS           -1.4934      0.186     -8.039      0.000        -1.858    -1.128
RAD            0.2991      0.063      4.719      0.000         0.175     0.424
TAX           -0.0118      0.003     -3.488      0.001        -0.018    -0.005
PTRATIO       -0.9471      0.129     -7.337      0.000        -1.201    -0.693
B              0.0094      0.003      3.508      0.000         0.004     0.015
LSTAT         -0.5232      0.047    -11.037      0.000        -0.616    -0.430
==============================================================================
Omnibus:                      178.444   Durbin-Watson:                   1.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              786.944
Skew:                           1.524   Prob(JB):                    1.31e-171
Kurtosis:                       8.295   Cond. No.                     1.47e+04
==============================================================================
```

The coef column is the calculated regression coefficient, which can also be output with "print(model.params)". The regression equation is:

$$y=36.3694-0.1076x_1+0.0458x_2+2.7212x_3-17.3956x_4+3.7966x_5-1.4934x_6+0.2991x_7-0.0118x_8-0.9471x_9+0.0094x_{10}-0.5232x_{11}$$

Assume the test data is as follows:

```
X_test = np.array([[1, 0.006, 18.0, 0.0, 0.52, 6.6, 4.87, 1.0, 290.0,
15.2, 396.2, 5]])       # The first number 1 is the same constant term added
```

Calculate the prediction by substituting the test data into the equation. Simply use the model's predict() to obtain the prediction result.

```
    print(model.predict(X_test))
    [29.52160882]
```

The prediction result is 29.52160882.

In addition, the "Adj. R-squared" in the output is adjusted R^2, which is derived from R^2 but is more rigorous than R^2 and is more suitable for expressing the degree of fitting of the model. R^2 is also called coefficient of determination and it is the square of the Pearson correlation coefficient. R^2 can indicate the fitting degree to the actual data of the model. R^2 is between 0 and 1 and the fitting effect is better if it is closer to 1. The value of 0.735 indicates more than 70% of the data can be explained by the model and the fitting effect is not bad, but it can be improved. It is generally considered that it is good if the R^2 is more than 0.8. You can try to use the nonlinear model for further prediction.

Beginners can focus on the adjusted R^2 and the p-values of the T test. The follow-up can continue to understand the overall significance of the equations using the F test and the residual analysis. Interested learners can do further in-depth research.

3. Reference

(1) Linear Regression
Various statistical learning materials
(2) Linear Regression in sklearn
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
or other learning materials
(3) Linear Regression in statsmodel
http://www.statsmodels.org/dev/regression.html
or other learning materials