Lab 2 - Linear Regression in Python

February 24, 2016

This lab on Linear Regression is a python adaptation of p. 109-119 of “Introduction to Statistical Learning with Applications in R” by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. Written by R. Jordan Crouser at Smith College for SDS293: Machine Learning (Spring 2016).

1 3.6.1 Importing Libraries

In [2]: # Tells matplotlib to display images inline instead of a new window
   %matplotlib inline

   import numpy as np
   import pandas as pd
   import statsmodels.api as sm

We’ll start by importing the data from Boston.csv into a pandas dataframe:

In [105]: df = pd.read_csv('Boston.csv', index_col=0)
df.head()

Out[105]:
<table>
<thead>
<tr>
<th>crim</th>
<th>zn</th>
<th>indus</th>
<th>chas</th>
<th>nox</th>
<th>rm</th>
<th>age</th>
<th>dis</th>
<th>rad</th>
<th>tax</th>
<th>ptratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00632</td>
<td>18</td>
<td>2.31</td>
<td>0</td>
<td>0.538</td>
<td>6.575</td>
<td>65.2</td>
<td>4.0900</td>
<td>1</td>
<td>296</td>
<td>15.3</td>
</tr>
<tr>
<td>0.02731</td>
<td>0</td>
<td>7.07</td>
<td>0</td>
<td>0.469</td>
<td>6.421</td>
<td>78.9</td>
<td>4.9671</td>
<td>2</td>
<td>242</td>
<td>17.8</td>
</tr>
<tr>
<td>0.02729</td>
<td>0</td>
<td>7.07</td>
<td>0</td>
<td>0.469</td>
<td>7.185</td>
<td>61.1</td>
<td>4.9671</td>
<td>2</td>
<td>242</td>
<td>17.8</td>
</tr>
<tr>
<td>0.03237</td>
<td>0</td>
<td>2.18</td>
<td>0</td>
<td>0.458</td>
<td>6.998</td>
<td>45.8</td>
<td>6.0622</td>
<td>3</td>
<td>222</td>
<td>18.7</td>
</tr>
<tr>
<td>0.06905</td>
<td>0</td>
<td>2.18</td>
<td>0</td>
<td>0.458</td>
<td>7.147</td>
<td>54.2</td>
<td>6.0622</td>
<td>3</td>
<td>222</td>
<td>18.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>black</th>
<th>lstat</th>
<th>medv</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>396.90</td>
<td>4.98</td>
</tr>
<tr>
<td>2</td>
<td>396.90</td>
<td>9.14</td>
</tr>
<tr>
<td>3</td>
<td>392.83</td>
<td>4.03</td>
</tr>
<tr>
<td>4</td>
<td>394.63</td>
<td>2.94</td>
</tr>
<tr>
<td>5</td>
<td>396.90</td>
<td>5.33</td>
</tr>
</tbody>
</table>

2 3.6.2 Simple Linear Regression

Now let’s fit a simple linear model (OLS - for “ordinary least squares” method) with medv as the response and lstat as the predictor:

In [106]: lm = sm.OLS.from_formula('medv ~ lstat', df)
   result = lm.fit()

To get detailed information about the model, we can print the results of a call to the .summary() method:

In [107]: print result.summary()
### OLS Regression Results

<table>
<thead>
<tr>
<th>Dep. Variable:</th>
<th>medv</th>
<th>R-squared:</th>
<th>0.544</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model:</td>
<td>OLS</td>
<td>Adj. R-squared:</td>
<td>0.543</td>
</tr>
<tr>
<td>Method:</td>
<td>Least Squares</td>
<td>F-statistic:</td>
<td>601.6</td>
</tr>
<tr>
<td>Date:</td>
<td>Wed, 03 Feb 2016</td>
<td>Prob (F-statistic):</td>
<td>5.08e-88</td>
</tr>
<tr>
<td>Time:</td>
<td>21:36:56</td>
<td>Log-Likelihood:</td>
<td>-1641.5</td>
</tr>
<tr>
<td>Df Residuals:</td>
<td>504</td>
<td>BIC:</td>
<td>3295.</td>
</tr>
<tr>
<td>Df Model:</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covariance Type:</td>
<td>nonrobust</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| coef     | std err | t    | P>|t| | [95.0% Conf. Int.] |
|----------|---------|------|------|---------------------|
| Intercept| 34.5538 | 0.563| 61.415| 0.000  | 33.448  | 35.659  |
| lstat    | -0.9500| 0.039| -24.528| 0.000  | -1.026  | -0.874  |

| Omnibus: | 137.043 | Durbin-Watson: | 0.892 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 291.373 |
| Skew:    | 1.453  | Prob(JB):      | 5.36e-64 |
| Kurtosis:| 5.319  | Cond. No.      | 29.7 |

**Warnings:**

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Want individual attributes? You can access them independently like this:

```python
In [108]: result.rsquared, result.fvalue, result.params.Intercept, result.params.lstat
```

```python
Out[108]: (0.5441462975864797, 601.6178711098953, 34.553840879383074, -0.9500493537579906)
```

For a complete list of attributes and methods of a `RegressionResults` object, see: http://statsmodels.sourceforge.net/devel/generated/statsmodels.regression.linear_model.RegressionResults.html?highlight=regressionresults

Now let’s try making some predictions using this model. First, we’ll set up a dataframe containing the `lstat` values for which we want to predict a response:

```python
In [109]: new = pd.DataFrame([[1, 5], [1, 10], [1, 15]], columns=['Intercept', 'lstat'])
```

Now we just call the `.predict()` method:

```python
In [110]: result.predict(new)
```

```python
Out[110]: array([ 29.80359411, 25.05334734, 20.30310057])
```

Technically those are the right prediction values, but maybe it would be good to have the confidence intervals along with them. Let’s write a little helper function to get that and package it all up:

```python
In [111]: def predict(res, new):
    # Get the predicted values
    fit = pd.DataFrame(res.predict(new), columns=['fit'])
```

---

2
# Get the confidence interval for the model (and rename the columns to something a bit more useful)

```python
ci = res.conf_int().rename(columns={0: 'lower', 1: 'upper'})
```

# Now a little bit of matrix multiplication to get the confidence intervals for the predictions

```python
ci = ci.T.dot(new.T).T
```

# And finally wrap up the confidence intervals with the predicted values

```python
return pd.concat([fit, ci], axis=1)
```

In [112]: predict(result, new)

Out[112]:
<table>
<thead>
<tr>
<th></th>
<th>fit</th>
<th>lower</th>
<th>upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>29.803594</td>
<td>28.317716</td>
<td>31.289472</td>
</tr>
<tr>
<td>1</td>
<td>25.053347</td>
<td>23.186975</td>
<td>26.919720</td>
</tr>
<tr>
<td>2</td>
<td>20.303101</td>
<td>18.056234</td>
<td>22.549967</td>
</tr>
</tbody>
</table>

Seaborn is a Python visualization library based on matplotlib that provides a high-level interface for drawing attractive statistical graphics.

In [113]: import seaborn as sns

We will now plot medv and lstat along with the least squares regression line using the `regplot()` function. We can define the color of the fit line using `line_kws` ("line keywords"):

In [114]: sns.regplot('lstat', 'medv', df, line_kws = {"color":"r"}, ci=None)

Out[114]: <matplotlib.axes._subplots.AxesSubplot at 0x10aa0d3d0>

We can also plot the residuals against the fitted values:
In [115]: fitted_values = pd.Series(result.fittedvalues, name="Fitted Values")
residuals = pd.Series(result.resid, name="Residuals")
sns.regplot(fitted_values, residuals, fit_reg=False)

Out[115]: <matplotlib.axes._subplots.AxesSubplot at 0x10b5888d0>

Perhaps we want normalized residuals instead?

In [116]: s_residuals = pd.Series(result.resid_pearson, name="S. Residuals")
sns.regplot(fitted_values, s_residuals, fit_reg=False)

Out[116]: <matplotlib.axes._subplots.AxesSubplot at 0x10a843b10>
We can also look for points with high leverage:

```python
In [130]: from statsmodels.stats.outliers_influence import OLSInfluence
   ...: leverage = pd.Series(OLSInfluence(result).influence, name = "Leverage")
   ...: sns.regplot(leverage, s_residuals, fit_reg=False)
```

Out[130]: <matplotlib.axes._subplots.AxesSubplot at 0x10bc653d0>
In order to fit a multiple linear regression model using least squares, we again use the \texttt{from\_formula()} function. The syntax \texttt{from\_formula(y \sim x_1 + x_2 + x_3)} is used to fit a model with three predictors, $x_1$, $x_2$, and $x_3$. The \texttt{summary()} function now outputs the regression coefficients for all the predictors.

\begin{verbatim}
In [131]: model = sm.OLS.from_formula('medv ~ lstat + age', df)
result = model.fit()
print result.summary()
\end{verbatim}

\begin{verbatim}
OLS Regression Results
==============================================================================
Dep. Variable: medv R-squared: 0.551
Model: OLS Adj. R-squared: 0.549
Method: Least Squares F-statistic: 309.0
Date: Wed, 03 Feb 2016 Prob (F-statistic): 2.98e-88
No. Observations: 506 AIC: 3281.0
Df Residuals: 503 BIC: 3294.2
Df Model: 2
Covariance Type: nonrobust
==============================================================================
coef std err t P>|t| [95.0% Conf. Int.]
Intercept 33.2228 0.731 45.458 0.000 31.787 34.659
lstat -1.0321 0.048 -21.416 0.000 -1.127 -0.937
age 0.0345 0.012 2.826 0.005 0.011 0.059
\end{verbatim}
The Boston data set contains 13 variables, and so it would be cumbersome to have to type all of these in order to perform a regression using all of the predictors. Instead, we can use the following short-hand:

```python
In [132]: # All columns (except medv, which is our response)
    model = sm.OLS.from_formula('medv ~ ' + '+'.join(df.columns.difference(['medv'])), df)
    result = model.fit()
    print result.summary()
```

### OLS Regression Results

| coef | std err | t | P>|t| | [95.0% Conf. Int.] |
|------|---------|---|-----|-------------------------------|
| Intercept | 36.4595 | 5.103 | 7.144 | 0.000 | 26.432 | 46.487 |
| age | 0.0007 | 0.013 | 0.052 | 0.958 | -0.025 | 0.027 |
| black | 0.0093 | 0.003 | 3.467 | 0.001 | 0.004 | 0.015 |
| chas | 2.6867 | 0.862 | 3.118 | 0.002 | 0.994 | 4.380 |
| crim | -0.1080 | 0.033 | -3.287 | 0.001 | -0.173 | -0.043 |
| dis | -1.4756 | 0.199 | -7.398 | 0.000 | -1.867 | -1.084 |
| indus | 0.0206 | 0.061 | 0.334 | 0.738 | -0.100 | 0.141 |
| lstat | -0.5248 | 0.051 | -10.347 | 0.000 | -0.624 | -0.425 |
| nox | -17.7666 | 3.820 | -4.651 | 0.000 | -25.272 | -10.262 |
| ptratio | -0.9527 | 0.131 | -7.283 | 0.000 | -1.210 | -0.696 |
| rad | 3.8099 | 0.418 | 9.116 | 0.000 | 2.989 | 4.631 |
| tax | -0.0123 | 0.004 | -3.280 | 0.001 | -0.020 | -0.005 |
| zn | 0.0464 | 0.014 | 3.382 | 0.001 | 0.019 | 0.073 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
The condition number is large, 1.51e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Note that we used the syntax `df.columns.difference(['medv'])` to exclude the response variable above. We can use the same approach to perform a regression using just a subset of the predictors? For example, in the above regression output, age and indus have a high p-value. So we may wish to run a regression excluding these predictors:

```
In [133]: # All columns (except medv)
model = sm.OLS.from_formula('medv ~ ' + '+'.join(df.columns.difference(['medv', 'age', 'indus'])), df)
result = model.fit()
print result.summary()
```

### 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:       Wed, 03 Feb 2016   Prob (F-statistic): 5.54e-137
Time:        21:38:18   Log-Likelihood: -1498.9
Df Residuals:    494  BIC: 3072.
Df Model:         11
Covariance Type:  nonrobust
```

```
coef  std err     t    P>|t|     [95.0% Conf. Int.]
-------------------------------------------------------------------
Intercept 36.3411  5.067    7.171  0.000     26.385   46.298
black  0.0093   0.003    3.475  0.001      0.004    0.015
chas  2.7187   0.854    3.183  0.002      1.040    4.397
crim -0.1084  0.033   -3.307  0.001     -0.173   -0.044
dis -1.4927  0.186   -8.037  0.000     -1.858   -1.128
lstat -0.5226  0.047  -11.019  0.000     -0.616   -0.429
nox -17.3760  3.535  -4.915  0.000     -24.322  -10.430
prratio -0.9465  0.129  -7.334  0.000     -1.200  -0.693
rad  0.2996  0.063    4.726  0.000       0.175    0.424
rm  3.8016  0.406    9.356  0.000       3.003    4.600
tax -0.0118  0.003  -3.493  0.001     -0.018   -0.005
zn   0.0458  0.014    3.390  0.001       0.019    0.072
```

```
Omnibus: 178.430  Durbin-Watson: 1.078
Prob(Omnibus): 0.000  Jarque-Bera (JB): 787.785
Skew: 1.523  Prob(JB): 8.60e-172
Kurtosis: 8.300  Cond. No. 1.47e+04
```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.47e+04. This might indicate that there are strong multicollinearity or other numerical problems.

### 3.6.4 Interaction Terms

It is easy to include interaction terms in a linear model using the `sm.formula()` function. The syntax `lstat : black` tells Python to include an interaction term between `lstat` and `black`. The syntax `lstat * age`
simultaneously includes lstat, age, and the interaction term lstat \times age as predictors; it is a shorthand for lstat + age + lstat:age.

In [134]: print sm.OLS.from_formula('medv ~ lstat*age', df).fit().summary()

OLS Regression Results
==============================================================================
Dep. Variable: medv R-squared: 0.556
Model: OLS Adj. R-squared: 0.553
Method: Least Squares F-statistic: 209.3
Date: Wed, 03 Feb 2016 Prob (F-statistic): 4.86e-88
Time: 21:38:20 Log-Likelihood: -1635.0
Df Residuals: 502 BIC: 3295.
Df Model: 3
Covariance Type: nonrobust
==============================================================================
coef std err t P>|t| [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept 36.0885 1.470 24.553 0.000 33.201 38.976
lstat -1.3921 0.167 -8.313 0.000 -1.721 -1.063
age -0.0007 0.020 -0.036 0.971 -0.040 0.038
lstat:age 0.0042 0.002 2.244 0.025 0.001 0.008
==============================================================================
Omnibus: 135.601 Durbin-Watson: 0.965
Prob(Omnibus): 0.000 Jarque-Bera (JB): 296.955
Skew: 1.417 Prob(JB): 3.29e-65
Kurtosis: 5.461 Cond. No. 6.88e+03
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.88e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

5.3.6.5 Non-linear Transformations of the Predictors

The .from_formula() function can also accommodate non-linear transformations of the predictors. For instance, given a predictor X, we can create a predictor X^2 using np.square(X). We now perform a regression of medv onto lstat and lstat^2.

In [135]: lm.fit2 = sm.OLS.from_formula('medv ~ lstat + np.square(lstat)', df).fit()
print lm.fit2.summary()

OLS Regression Results
==============================================================================
Dep. Variable: medv R-squared: 0.641
Model: OLS Adj. R-squared: 0.639
Method: Least Squares F-statistic: 448.5
Date: Wed, 03 Feb 2016 Prob (F-statistic): 1.56e-112
Df Residuals: 503 BIC: 3181.
Df Model: 3
Covariance Type: nonrobust
==============================================================================
coef std err t P>|t| [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept 36.0885 1.470 24.553 0.000 33.201 38.976
lstat -1.3921 0.167 -8.313 0.000 -1.721 -1.063
age -0.0007 0.020 -0.036 0.971 -0.040 0.038
lstat:age 0.0042 0.002 2.244 0.025 0.001 0.008
==============================================================================
Omnibus: 135.601 Durbin-Watson: 0.965
Prob(Omnibus): 0.000 Jarque-Bera (JB): 296.955
Skew: 1.417 Prob(JB): 3.29e-65
Kurtosis: 5.461 Cond. No. 6.88e+03
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.88e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
Covariance Type: nonrobust

|        | coef   | std err | t     | P>|t| | [95.0% Conf. Int.] |
|--------|--------|---------|-------|------|-------------------|
| Intercept | 42.8620 | 0.872   | 49.149| 0.000 | 41.149 - 44.575   |
| lstat  | -2.3328 | 0.124   | -18.843| 0.000 | -2.576 - 2.090    |
| np.square(lstat) | 0.0435 | 0.004   | 11.628| 0.000 | 0.036 - 0.051     |

Omnibus: 107.006 Durbin-Watson: 0.921
Prob(Omnibus): 0.000 Jarque-Bera (JB): 228.388
Skew: 1.128 Prob(JB): 2.55e-50
Kurtosis: 5.397 Cond. No. 1.13e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.13e+03. This might indicate that there are strong multicollinearity or other numerical problems.

The near-zero p-value associated with the quadratic term suggests that it leads to an improved model. We use the `anova_lm()` function to further quantify the extent to which the quadratic fit is superior to the linear fit.

```python
In [136]: lm.fit = sm.OLS.from_formula('medv ~ lstat', df).fit()
print sm.stats.anova_lm(lm.fit, lm.fit2)
```

```
| df_resid | ssr | df_diff | ss_diff | F       | Pr(>|F|) |
|----------|-----|---------|---------|---------|---------|
| 0        | 504 | 19472.381418 | 0 | NaN | NaN |
| 1        | 503 | 15347.243158 | 1 | 4125.13826 | 135.199822 | 7.630116e-28 |
```

Here Model 0 represents the linear submodel containing only one predictor, `lstat`, while Model 1 corresponds to the larger quadratic model that has two predictors, `lstat` and `lstat2`. The `anova_lm()` function performs a hypothesis test comparing the two models. The null hypothesis is that the two models fit the data equally well, and the alternative hypothesis is that the full model is superior.

The F-statistic is 135 and the associated p-value is virtually zero. This provides very clear evidence that the model containing the predictors `lstat` and `lstat2` is far superior to the model that only contains the predictor `lstat`. This is not surprising, since earlier we saw evidence for non-linearity in the relationship between `medv` and `lstat`.

If we type:

```python
In [137]: fitted_values = pd.Series(lm.fit2.fittedvalues, name="Fitted Values")
residuals = pd.Series(lm.fit2.resid, name="S. Residuals")
sns.regplot(fitted_values, s_residuals, fit_reg=False)
```

Out[137]: <matplotlib.axes._subplots.AxesSubplot at 0x10bf09d50>
then we see that when the lstat2 term is included in the model, there is little discernible pattern in the residuals.

In order to create a cubic fit, we can include a predictor of the form \( \text{np.power}(x, 3) \). However, this approach can start to get cumbersome for higher order polynomials. A better approach involves using list comprehension inside a `join()`. For example, the following command produces a fifth-order polynomial fit:

```python
In [138]: sm.OLS.from_formula('medv ~ ' + '+'.join([f'np.power(lstat, {i})' for i in range(1,6)]), df).fit().summary()
```

```
Out[138]: <class 'statsmodels.iolib.summary.Summary'>
```

```
OLS Regression Results
==============================================================================
Dep. Variable: medv R-squared: 0.682
Model: OLS Adj. R-squared: 0.679
Method: Least Squares F-statistic: 214.2
Date: Wed, 03 Feb 2016 Prob (F-statistic): 8.73e-122
Time: 21:38:31 Log-Likelihood: -1550.6
Df Residuals: 500 BIC: 3139.
Df Model: 5
Covariance Type: nonrobust
```

```
| coef      | std err | t    | P>|t| | [95.0% Conf. Int.] |
|-----------|---------|------|------|-------------------|
| Intercept | 67.6997 | 3.604| 18.783| 0.000 | 60.618 | 74.781 |
| np.power(lstat, 1) | -11.9911 | 1.526| -7.859| 0.000 | -14.989 | -8.994 |
| np.power(lstat, 2) | 1.2728 | 0.223| 5.703| 0.000 | 0.834 | 1.711 |
| np.power(lstat, 3) | -0.0683 | 0.014| -4.747| 0.000 | -0.097 | -0.040 |
| np.power(lstat, 4) | 0.0017 | 0.000| 4.143| 0.000 | 0.001 | 0.003 |
```

```
Of course, we are in no way restricted to using polynomial transformations of the predictors. Here we try a log transformation.

```
In [139]: sm.OLS.from_formula('medv ~ np.log(rm)', df).fit().summary()
```

```
Out[139]: <class 'statsmodels.iolib.summary.Summary'>
```

```
6 3.6.6 Qualitative Predictors

We will now examine the Carseats data, which is part of the ISLR library. We will attempt to predict Sales (child car seat sales) in 400 locations based on a number of predictors.
```

```
In [3]: df2 = pd.read_csv('Carseats.csv')
df2.head()
```

```
```
The `Carseats` data includes qualitative predictors such as `Shelveloc`, an indicator of the quality of the shelving location—that is, the space within a store in which the car seat is displayed—at each location. The predictor `Shelveloc` takes on three possible values, `Bad`, `Medium`, and `Good`.

Given a qualitative variable such as `Shelveloc`, Python generates dummy variables automatically. Below we fit a multiple regression model that includes some interaction terms.

```python
In [5]: sm.OLS.from_formula('Sales ~ Income:Advertising+Price:Age + ' + '+'.join(df2.columns.difference(['Sales'])))

Out[5]: <class 'statsmodels.iolib.summary.Summary'>
```
Skew: 0.129  Prob(JB): 0.564
Kurtosis: 3.050  Cond. No. 1.31e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.31e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

To learn how to set other coding schemes (or contrasts), see:
http://statsmodels.sourceforge.net/devel/examples/notebooks/generated/contrasts.html

In [ ]: