Python Tutorial 9

March 27, 2022

This tutorial is for Dr. Xin Tong's DSO 530 class at the University of Southern California in Spring 2022. It aims to give you some supplementary code to implement *Subset Selection* in Python.

1 Linear Model Subset Selection

[1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import itertools
import time
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

1.1 Best Subset Selection

Here we apply the best subset selection approach to the Hitters data. We wish to predict a baseball player's Salary on the basis of various statistics associated with performance in the previous year. Let's take a quick look:

[2]:
```
df = pd.read_csv('Hitters.csv')
df.head()
```

[2]:
```
              Player  AtBat  Hits  HmRun  Runs  RBI  Walks  Years  CAtBat  \
0      -Andy Allanson    293    66      1    30   29     14      1     293
1         -Alan Ashby    315    81      7    24   38     39     14    3449
2        -Alvin Davis    479   130     18    66   72     76      3    1624
3       -Andre Dawson    496   141     20    65   78     37     11    5628
4   -Andres Galarraga    321    87     10    39   42     30      2     396

   CHits  …  CRuns  CRBI  CWalks  League Division PutOuts  Assists  Errors  \
0     66  …     30    29      14       A        E     446       33      20
1    835  …    321   414     375       N        W     632       43      10
2    457  …    224   266     263       A        W     880       82      14
3   1575  …    828   838     354       N        E     200       11       3
4    101  …     48    46      33       N        E     805       40       4

   Salary  NewLeague
```

```
0     NaN          A
1   475.0          N
2   480.0          A
3   500.0          N
4    91.5          N

[5 rows x 21 columns]
```

First, we note that the `Salary` variable is missing for some of the players. The `isnull()` function can be used to identify the missing observations. It returns a vector of the same length as the input vector, with a `TRUE` value for any missing element, and a `FALSE` value for a non-missing element. The `sum()` function can then be used to count the missing elements:

[3]:
```python
print(df["Salary"].isnull().sum())
```

```
59
```

We see that `Salary` is missing for 59 players. The `dropna()` function removes all of the rows that have missing values in any variable:

[4]:
```python
# Drop the player names as they are not a reasonable potential predictor
df = df.drop('Player', axis=1)

# Print the dimensions of the Hitters data (322 rows x 20 columns)(Players'␣
↪names not included)
print("before dropna():",df.shape)

# Drop any rows the contain missing values. Note that this is not necessarily␣
↪the recommended practice for a given problem.
df = df.dropna()

# Print the dimensions of the modified Hitters data (263 rows x 20 columns)
print("after dropna():",df.shape)

# One last check: should return 0
print("check the number of missing salary after dropna():",df["Salary"].
↪isnull().sum())
```

```
before dropna(): (322, 20)
after dropna(): (263, 20)
check the number of missing salary after dropna(): 0
```

Here, we use `pd.get_dummies` function to transform the original categorical variable `League`, `Division` and `NewLeague` into the usable "1/0" format.

[5]:
```python
df[['League', 'Division', 'NewLeague']].head()
```

[5]:
```
  League Division NewLeague
1      N        W         N
```

```
2       A          W           A
3       N          E           N
4       N          E           N
5       A          W           A
```

[6]:
```
dummies = pd.get_dummies(df[['League', 'Division', 'NewLeague']])
```

[7]:
```
dummies.head()
```

[7]:

	League_A	League_N	Division_E	Division_W	NewLeague_A	NewLeague_N
1	0	1	0	1	0	1
2	1	0	0	1	1	0
3	0	1	1	0	0	1
4	0	1	1	0	0	1
5	1	0	0	1	1	0

Note that for every categorical variable with K categories, we only need K-1 dummies to represent it.

[8]:
```
y = df.Salary

# Drop the column with the dependent variable (Salary), and columns for which␣
↪we created dummy variables
X_ = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1)

# Define the feature set X.
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
## Note that alternatively, you can use drop_first=True in .get_dummies to␣
↪directly get K-1 dummies
## for a categorical variable with K categories.
```

We can perform *best subset selection* by identifying the best model that contains a given number of predictors, where **best** is quantified as having the smallest RSS. We'll define a helper function to output the best set of variables for each model size:

[9]:
```
def processSubset(feature_set):
    # Fit model on feature_set and calculate RSS
    X1 = sm.add_constant(X[list(feature_set)])
    model = sm.OLS(y,X1)
    regr = model.fit()
    RSS = ((regr.predict(X1) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}
```

Here, we calculate the RSS along with the process of model building.

[10]:
```
def getBest(k):

    tic = time.time()
```

```
    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the smallest RSS
    best_model = models.loc[models['RSS'].idxmin]
    # idxmin() function returns index of first occurrence of minimum.

    toc = time.time()
    print("Processed ", models.shape[0], "models on", k, "predictors in",␣
↪(toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the␣
↪model
    return best_model
```

Note that getting the smallest RSS is the same as getting the highest R^2.

This returns a *DataFrame* containing the best model that we generated, along with the RSS.

What function `itertools.combinations(iterable, r)` does is to return `r` length subsequences of elements from the input `iterable`. For example:

[11]:
```
print(list(itertools.combinations('12345',2)))
```

```
[('1', '2'), ('1', '3'), ('1', '4'), ('1', '5'), ('2', '3'), ('2', '4'), ('2',
'5'), ('3', '4'), ('3', '5'), ('4', '5')]
```

[12]:
```
print(list(itertools.combinations('12345',3)))
```

```
[('1', '2', '3'), ('1', '2', '4'), ('1', '2', '5'), ('1', '3', '4'), ('1', '3',
'5'), ('1', '4', '5'), ('2', '3', '4'), ('2', '3', '5'), ('2', '4', '5'), ('3',
'4', '5')]
```

Now we want to call that function for each number of predictors k:

[13]:
```
# Could take quite awhile to complete...

models = pd.DataFrame(columns=["RSS", "model"])

tic = time.time()
for i in range(0,8):
    models.loc[i] = getBest(i)
```

```python
toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
Processed  1 models on 0 predictors in 0.02093195915222168 seconds.
Processed  19 models on 1 predictors in 0.05792999267578125 seconds.
Processed  171 models on 2 predictors in 0.37050867080688477 seconds.
Processed  969 models on 3 predictors in 2.239161729812622 seconds.
Processed  3876 models on 4 predictors in 10.45738697052002 seconds.
Processed  11628 models on 5 predictors in 29.638815879821777 seconds.
Processed  27132 models on 6 predictors in 80.33709907531738 seconds.
Processed  50388 models on 7 predictors in 158.28794288635254 seconds.
Total elapsed time: 282.6905539035797 seconds.
```

Now we have one big *DataFrame* that contains the best models of sizes 0 to 7. Note that to save computation time, we did not look at models of sizes 8 to 19.

[14]:
```python
models
```

[14]:

	RSS	model
0	5.331911e+07	<statsmodels.regression.linear_model.Regressio…
1	3.617968e+07	<statsmodels.regression.linear_model.Regressio…
2	3.064656e+07	<statsmodels.regression.linear_model.Regressio…
3	2.924930e+07	<statsmodels.regression.linear_model.Regressio…
4	2.797085e+07	<statsmodels.regression.linear_model.Regressio…
5	2.714990e+07	<statsmodels.regression.linear_model.Regressio…
6	2.619490e+07	<statsmodels.regression.linear_model.Regressio…
7	2.590655e+07	<statsmodels.regression.linear_model.Regressio…

If we want to access the details of each model, we can get a full rundown of a single model using the `summary()` function:

[15]:
```python
print(models.loc[2, "model"].summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.425
Model:                            OLS   Adj. R-squared:                  0.421
Method:                 Least Squares   F-statistic:                     96.17
Date:                Sun, 27 Mar 2022   Prob (F-statistic):           5.43e-32
Time:                        17:21:38   Log-Likelihood:                -1907.2
No. Observations:                 263   AIC:                             3820.
Df Residuals:                     260   BIC:                             3831.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -47.9559     55.982     -0.857      0.392    -158.193      62.281
```

```
Hits            3.3008      0.482      6.851      0.000       2.352       4.250
CRBI            0.6899      0.067     10.262      0.000       0.558       0.822
==============================================================================
Omnibus:                      117.673   Durbin-Watson:                   1.927
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              700.819
Skew:                           1.704   Prob(JB):                    6.59e-153
Kurtosis:                      10.235   Cond. No.                     1.24e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.24e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

This output indicates that the best two-variable model contains only `Hits` and `CRBI`.

Here, as a digression, we recall a key difference between `df.loc()` and `df.iloc()`:

`loc()` gets rows (or columns) with particular `labels` from the index.

`iloc()` gets rows (or columns) at particular `positions` in the index (so it only takes integers).

For example:

[16]:
```python
data = pd.Series(np.arange(10), index=[20, 21, 22, 23, 24, 1, 2, 3, 4, 5])
data
```

[16]:
```
20    0
21    1
22    2
23    3
24    4
1     5
2     6
3     7
4     8
5     9
dtype: int64
```

[17]:
```python
data.loc[:3]
```

[17]:
```
20    0
21    1
22    2
23    3
24    4
1     5
2     6
3     7
```

```
dtype: int64
```

[18]:
```
data.iloc[:3]
```

[18]:
```
20    0
21    1
22    2
dtype: int64
```

You can use the functions we defined above to explore as many variables as are desired.

[19]:
```
print(getBest(19)["model"].summary())
```

```
Processed  1 models on 19 predictors in 0.007917642593383789 seconds.
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.546
Model:                            OLS   Adj. R-squared:                  0.511
Method:                 Least Squares   F-statistic:                     15.39
Date:                Sun, 27 Mar 2022   Prob (F-statistic):           7.84e-32
Time:                        17:21:38   Log-Likelihood:                -1876.2
No. Observations:                 263   AIC:                             3792.
Df Residuals:                     243   BIC:                             3864.
Df Model:                          19
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const         163.1036     90.779      1.797      0.074     -15.710     341.917
AtBat          -1.9799      0.634     -3.123      0.002      -3.229      -0.731
Hits            7.5008      2.378      3.155      0.002       2.818      12.184
HmRun           4.3309      6.201      0.698      0.486      -7.885      16.546
Runs           -2.3762      2.981     -0.797      0.426      -8.248       3.495
RBI            -1.0450      2.601     -0.402      0.688      -6.168       4.078
Walks           6.2313      1.829      3.408      0.001       2.630       9.833
Years          -3.4891     12.412     -0.281      0.779     -27.938      20.960
CAtBat         -0.1713      0.135     -1.267      0.206      -0.438       0.095
CHits           0.1340      0.675      0.199      0.843      -1.195       1.463
CHmRun         -0.1729      1.617     -0.107      0.915      -3.358       3.013
CRuns           1.4543      0.750      1.938      0.054      -0.024       2.933
CRBI            0.8077      0.693      1.166      0.245      -0.557       2.172
CWalks         -0.8116      0.328     -2.474      0.014      -1.458      -0.165
PutOuts         0.2819      0.077      3.640      0.000       0.129       0.434
Assists         0.3711      0.221      1.678      0.095      -0.065       0.807
Errors         -3.3608      4.392     -0.765      0.445     -12.011       5.290
League_N       62.5994     79.261      0.790      0.430     -93.528     218.727
Division_W   -116.8492     40.367     -2.895      0.004    -196.363     -37.335
NewLeague_N   -24.7623     79.003     -0.313      0.754    -180.380     130.855
```

```
==============================================================================
Omnibus:                       87.414   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              452.923
Skew:                           1.236   Prob(JB):                     4.46e-99
Kurtosis:                       8.934   Cond. No.                     2.09e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.09e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In addition to using the `summary` function to print to the screen, we can access just the parts we need using the model's attributes. For example, if we want the R^2 value:

[20]:
```
models.loc[2, "model"].rsquared
```

[20]:
```
0.42522374646677885
```

In addition to the verbose output, we get when we print the summary to the screen, fitting the `OLS` also produced many other useful statistics such as adjusted-R^2, AIC, and BIC. We can examine these to try to select the best overall model across different model sizes. Let's start by looking at R^2 across all our models:

[21]:
```
# Gets the second element from each row ('model') and pulls out its R rsquared␣
↪attribute
models.apply(lambda row: row[1].rsquared, axis=1)
```

[21]:
```
0    0.000000
1    0.321450
2    0.425224
3    0.451429
4    0.475407
5    0.490804
6    0.508715
7    0.514123
dtype: float64
```

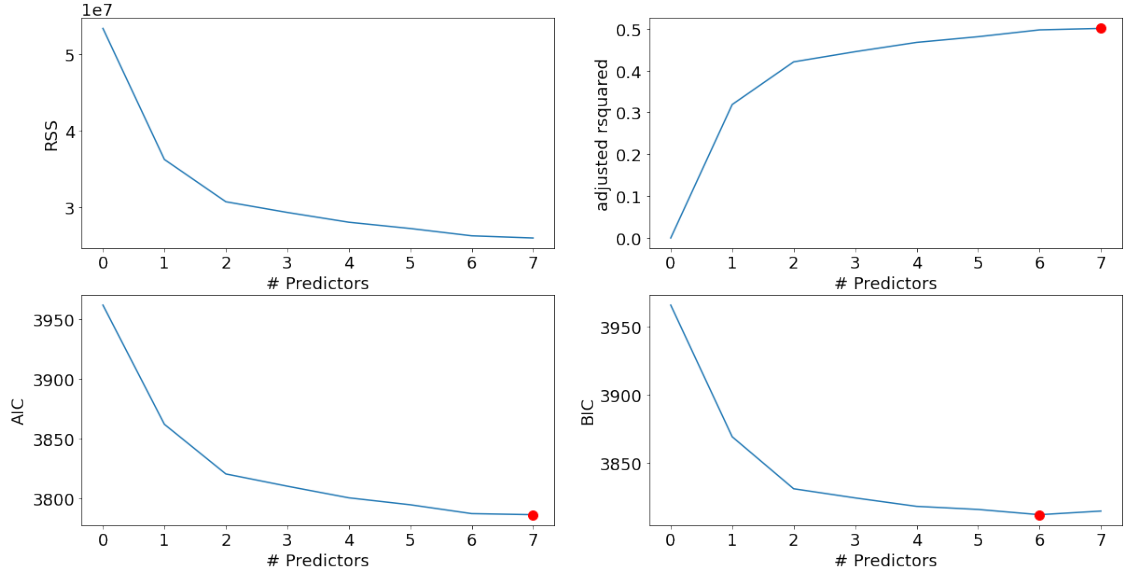
As expected, the R^2 statistic increases monotonically as more variables are included.

Plotting RSS, adjusted-R^2, AIC, and BIC for all of the models at once will help us decide which model to select.:

[22]:
```
plt.figure(figsize=(20,10))
plt.rcParams.update({'font.size': 18, 'lines.markersize': 10})

# Set up a 2x2 grid so we can look at 4 plots at once
plt.subplot(2, 2, 1)
```

```
# We will now plot a curve to show the relationship between the number of␣
↪predictors and the RSS
plt.plot(models["RSS"])
plt.xlabel('# Predictors')
plt.ylabel('RSS')

# We will now plot a red dot to indicate the model with the largest adjusted␣
↪R^2 statistic.
# The idxmax() function can be used to identify the location of the maximum␣
↪point of a vector

rsquared_adj = models.apply(lambda row: row[1].rsquared_adj, axis=1)

plt.subplot(2, 2, 2)
plt.plot(rsquared_adj)
plt.plot(rsquared_adj.idxmax(), rsquared_adj.max(), "or")
plt.xlabel('# Predictors')
plt.ylabel('adjusted rsquared')

# We'll do the same for AIC and BIC, this time looking for the models with the␣
↪SMALLEST statistic
aic = models.apply(lambda row: row[1].aic, axis=1)

plt.subplot(2, 2, 3)
plt.plot(aic)
plt.plot(aic.idxmin(), aic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('AIC')

bic = models.apply(lambda row: row[1].bic, axis=1)

plt.subplot(2, 2, 4)
plt.plot(bic)
plt.plot(bic.idxmin(), bic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('BIC')
```

[22]: Text(0, 0.5, 'BIC')



Recall that in the second step of our selection process, we narrowed the field down to just one model of each size $k(\leq p)$. According to BIC, the best performer is the model with 6 variables. According to AIC and adjusted-R^2 something a bit more complex might be better.

1.2 Forward Stepwise Selection (optional)

We can also use a similar approach to perform forward stepwise or backward stepwise selection, using a slight modification of the functions we defined above:

[23]:
```
def forward(predictors):

    # Pull out predictors we still need to process
    remaining_predictors = [p for p in X.columns if p not in predictors]

    tic = time.time()

    results = []

    for p in remaining_predictors:
        results.append(processSubset(predictors+[p]))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].idxmin]

    toc = time.time()
```

```
    print("Processed ", models.shape[0], "models on", len(predictors)+1,␣
↪"predictors in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the␣
↪model
    return best_model
```

Exccept for the null model, forward stepwise regression can be implemented as below.

[24]:
```
models2 = pd.DataFrame(columns=["RSS", "model"])

tic = time.time()
predictors = []

for i in range(1,len(X.columns)+1):
    models2.loc[i] = forward(predictors)
    predictors = models2.loc[i]["model"].model.exog_names.copy()
    predictors.remove('const')

toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
Processed  19 models on 1 predictors in 0.04671430587768555 seconds.
Processed  18 models on 2 predictors in 0.039489030838012695 seconds.
Processed  17 models on 3 predictors in 0.03871488571166992 seconds.
Processed  16 models on 4 predictors in 0.03974795341491699 seconds.
Processed  15 models on 5 predictors in 0.04239082336425781 seconds.
Processed  14 models on 6 predictors in 0.04398989677429199 seconds.
Processed  13 models on 7 predictors in 0.041918039321899414 seconds.
Processed  12 models on 8 predictors in 0.0393519401550293 seconds.
Processed  11 models on 9 predictors in 0.03719592094421387 seconds.
Processed  10 models on 10 predictors in 0.03577899932861328 seconds.
Processed  9 models on 11 predictors in 0.033032894134521484 seconds.
Processed  8 models on 12 predictors in 0.03284788131713867 seconds.
Processed  7 models on 13 predictors in 0.029610157012939453 seconds.
Processed  6 models on 14 predictors in 0.02554774284362793 seconds.
Processed  5 models on 15 predictors in 0.02216792106628418 seconds.
Processed  4 models on 16 predictors in 0.018649816513061523 seconds.
Processed  3 models on 17 predictors in 0.015812158584594727 seconds.
Processed  2 models on 18 predictors in 0.01139521598815918 seconds.
Processed  1 models on 19 predictors in 0.006227970123291016 seconds.
Total elapsed time: 0.6431300640106201 seconds.
```

Clearly, forward stepwise selection runs faster than best subset selection.

Let's see how the forward stepwise selection models stack up against best subset selection for models with 7 predictors:

```
[25]: print("Best Subset Selection:\n", models.loc[7, "model"].summary())
print("\n\nForward Stepwise Selection:\n", models2.loc[7, "model"].summary())
```

```
Best Subset Selection:
                             OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.514
Model:                            OLS   Adj. R-squared:                  0.501
Method:                 Least Squares   F-statistic:                     38.55
Date:                Sun, 27 Mar 2022   Prob (F-statistic):           1.19e-36
Time:                        17:21:39   Log-Likelihood:                -1885.1
No. Observations:                 263   AIC:                             3786.
Df Residuals:                     255   BIC:                             3815.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         79.4509     63.794      1.245      0.214     -46.179     205.081
Hits           1.2834      0.584      2.196      0.029       0.133       2.434
Walks          3.2274      1.203      2.684      0.008       0.859       5.596
CAtBat        -0.3752      0.097     -3.866      0.000      -0.566      -0.184
CHits          1.4957      0.334      4.480      0.000       0.838       2.153
CHmRun         1.4421      0.422      3.420      0.001       0.612       2.272
PutOuts        0.2367      0.075      3.161      0.002       0.089       0.384
Division_W  -129.9866     39.499     -3.291      0.001    -207.773     -52.200
==============================================================================
Omnibus:                      104.575   Durbin-Watson:                   2.022
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              681.266
Skew:                           1.440   Prob(JB):                    1.16e-148
Kurtosis:                      10.340   Cond. No.                     1.23e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.23e+04. This might indicate that there are
strong multicollinearity or other numerical problems.


Forward Stepwise Selection:
                             OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.513
Model:                            OLS   Adj. R-squared:                  0.500
Method:                 Least Squares   F-statistic:                     38.41
Date:                Sun, 27 Mar 2022   Prob (F-statistic):           1.50e-36
```

```
Time:                        17:21:39   Log-Likelihood:                -1885.4
No. Observations:                 263   AIC:                             3787.
Df Residuals:                     255   BIC:                             3815.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        109.7873     65.908      1.666      0.097     -20.006     239.580
CRBI           0.8538      0.151      5.640      0.000       0.556       1.152
Hits           7.4499      1.661      4.485      0.000       4.179      10.721
PutOuts        0.2533      0.075      3.382      0.001       0.106       0.401
Division_W  -127.1224     39.807     -3.193      0.002    -205.515     -48.730
AtBat         -1.9589      0.529     -3.701      0.000      -3.001      -0.917
Walks          4.9131      1.443      3.405      0.001       2.072       7.755
CWalks        -0.3053      0.199     -1.538      0.125      -0.696       0.086
==============================================================================
Omnibus:                      102.245   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              609.074
Skew:                           1.434   Prob(JB):                    5.51e-133
Kurtosis:                       9.882   Cond. No.                     2.58e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.58e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The results above indicate that the best seven-variable models are different between the best subset selection and forward stepwise selection.

1.3 Backward Stepwise Selection (optional)

We only need a minor change to implement backward stepwise selection: loop through the predictors in reverse.

[26]:
```python
def backward(predictors):

    tic = time.time()

    results = []

    for combo in itertools.combinations(predictors, len(predictors)-1):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
```

```
    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].idxmin]

    toc = time.time()
    print("Processed ", models.shape[0], "models on", len(predictors)-1,␣
↪"predictors in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the␣
↪model
    return best_model
```

[27]:
```
models3 = pd.DataFrame(columns=["RSS", "model"], index = range(1,len(X.
↪columns)))

tic = time.time()
predictors = X.columns

models3.loc[len(predictors)] = getBest(len(predictors))

while(len(predictors) > 1):
    models3.loc[len(predictors)-1] = backward(predictors)
    predictors = models3.loc[len(predictors)-1]["model"].model.exog_names.copy()
    predictors.remove('const')

toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
Processed  1 models on 19 predictors in 0.006092786788940430 seconds.
Processed  19 models on 18 predictors in 0.07275176048278809 seconds.
Processed  18 models on 17 predictors in 0.06374907493591309 seconds.
Processed  17 models on 16 predictors in 0.05979299545288086 seconds.
Processed  16 models on 15 predictors in 0.05682206153869629 seconds.
Processed  15 models on 14 predictors in 0.0506129264831543 seconds.
Processed  14 models on 13 predictors in 0.046527862548828125 seconds.
Processed  13 models on 12 predictors in 0.04227089881896973 seconds.
Processed  12 models on 11 predictors in 0.03786587715148926 seconds.
Processed  11 models on 10 predictors in 0.034002065658569336 seconds.
Processed  10 models on 9 predictors in 0.03050708770751953 seconds.
Processed  9 models on 8 predictors in 0.027508974075317383 seconds.
Processed  8 models on 7 predictors in 0.02334117889404297 seconds.
Processed  7 models on 6 predictors in 0.019824981689453125 seconds.
Processed  6 models on 5 predictors in 0.016820907592773438 seconds.
Processed  5 models on 4 predictors in 0.014647960662841797 seconds.
Processed  4 models on 3 predictors in 0.012042999267578125 seconds.
Processed  3 models on 2 predictors in 0.008568763732910156 seconds.
Processed  2 models on 1 predictors in 0.005591869354248047 seconds.
```

```
Total elapsed time: 0.649940013885498 seconds.
```

For this data, the best 7-variable models identified by *forward stepwise selection*, *backward stepwise selection*, and *best subset selection* are different.

[28]:
```
print("Best Subset Selection:\n",models.loc[7, "model"].params)
print("\nForward Stepwise Selection:\n",models2.loc[7, "model"].params)
print("\nBackward Stepwise Selection:\n",models3.loc[7, "model"].params)
```

```
Best Subset Selection:
 const          79.450947
Hits            1.283351
Walks           3.227426
CAtBat         -0.375235
CHits           1.495707
CHmRun          1.442054
PutOuts         0.236681
Division_W   -129.986643
dtype: float64

Forward Stepwise Selection:
 const         109.787306
CRBI            0.853762
Hits            7.449877
PutOuts         0.253340
Division_W   -127.122393
AtBat          -1.958885
Walks           4.913140
CWalks         -0.305307
dtype: float64

Backward Stepwise Selection:
 const         105.648749
AtBat          -1.976284
Hits            6.757491
Walks           6.055869
CRuns           1.129309
CWalks         -0.716335
PutOuts         0.302885
Division_W   -116.169217
dtype: float64
```