What is Backward Elimination?

Backward elimination is a feature selection technique while building a machine learning model. It is used to remove those features that do not have a significant effect on the dependent variable or prediction of output. There are various ways to build a model in Machine Learning, which are:

1. All-in
2. Backward Elimination
3. Forward Selection
4. Bidirectional Elimination
5. Score Comparison

Above are the possible methods for building the model in Machine learning, but we will only use here the Backward Elimination process as it is the fastest method.

Steps of Backward Elimination

Below are some main steps which are used to apply backward elimination process:

Step-1: Firstly, We need to select a significance level to stay in the model. (SL=0.05)

Step-2: Fit the complete model with all possible predictors/independent variables.

Step-3: Choose the predictor which has the highest P-value, such that.

a. If P-value >SL, go to step 4.
b. Else Finish, and Our model is ready.

Step-4: Remove that predictor.

Step-5: Rebuild and fit the model with the remaining variables.

Need for Backward Elimination: An optimal Multiple Linear Regression model:

In the previous chapter, we discussed and successfully created our Multiple Linear Regression model, where we took **4 independent variables (R&D spend, Administration spend, Marketing spend, and state (dummy variables)) and one dependent variable (Profit)**. But that model is not optimal, as we have included all the independent variables and do not know which independent model is most affecting and which one is the least affecting for the prediction.

Unnecessary features increase the complexity of the model. Hence it is good to have only the most significant features and keep our model simple to get the better result.

So, in order to optimize the performance of the model, we will use the Backward Elimination method. This process is used to optimize the performance of the MLR model as it will only include the most affecting feature and remove the least affecting feature. Let's start to apply it to our MLR model.

Steps for Backward Elimination method:

We will use the same model which we build in the previous chapter of MLR. Below is the complete code for it:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('50_CompList.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 4].values

#Catgorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
labelencoder_x= LabelEncoder()
x[:, 3]= labelencoder_x.fit_transform(x[:,3])
onehotencoder= OneHotEncoder(categorical_features= [3])
x= onehotencoder.fit_transform(x).toarray()

#Avoiding the dummy variable trap:
x = x[:, 1:]



# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)

#Predicting the Test set result;
y_pred= regressor.predict(x_test)

#Checking the score
print('Train Score: ', regressor.score(x_train, y_train))
print('Test Score: ', regressor.score(x_test, y_test))
```

From the above code, we got training and test set result as:

```
Train Score:  0.9501847627493607
Test Score:  0.9347068473282446
```

The difference between both scores is 0.0154.

> Note: On the basis of this score, we will estimate the effect of features on our model after using the Backward elimination process.

Step: 1- Preparation of Backward Elimination:

- **Importing the library:** Firstly, we need to import the **statsmodels.formula.api** library, which is used for the estimation of various statistical models such as OLS(Ordinary Least Square). Below is the code for it:

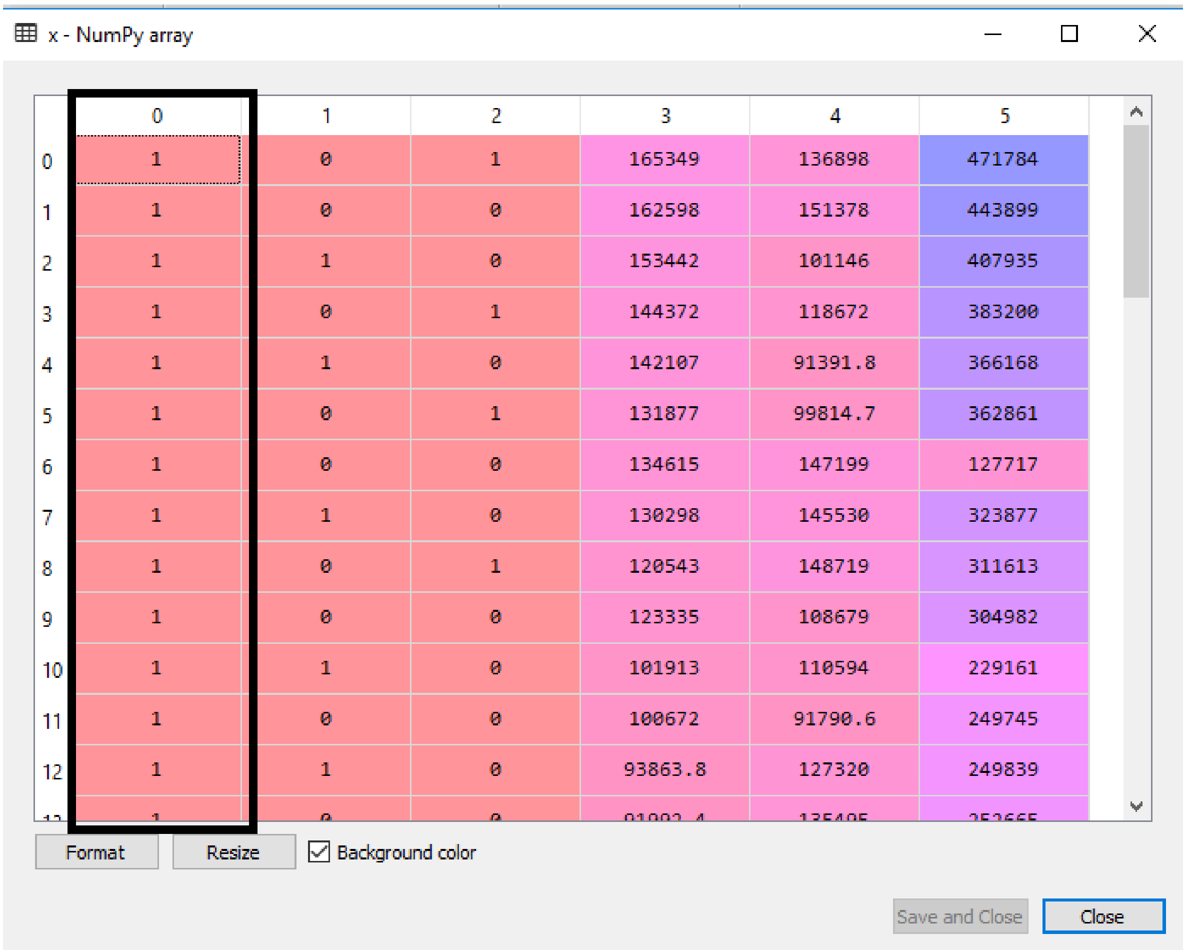
```
import statsmodels.api as smf
```

- **Adding a column in matrix of features:** As we can check in our MLR equation (a), there is one constant term b_0, but this term is not present in our matrix of features, so we need to add it manually. We will add a column having values $x_0 = 1$ associated with the constant term b_0.
  To add this, we will use **append** function of **Numpy** library (nm which we have already imported into our code), and will assign a value of 1. Below is the code for it.

```
x = nm.append(arr = nm.ones((50,1)).astype(int), values=x, axis=1)
```

Here we have used axis =1, as we wanted to add a column. For adding a row, we can use axis =0.

Output: By executing the above line of code, a new column will be added into our matrix of features, which will have all values equal to 1. We can check it by clicking on the x dataset under the variable explorer option.



	0	1	2	3	4	5
0	1	0	1	165349	136898	471784
1	1	0	0	162598	151378	443899
2	1	1	0	153442	101146	407935
3	1	0	1	144372	118672	383200
4	1	1	0	142107	91391.8	366168
5	1	0	1	131877	99814.7	362861
6	1	0	0	134615	147199	127717
7	1	1	0	130298	145530	323877
8	1	0	1	120543	148719	311613
9	1	0	0	123335	108679	304982
10	1	1	0	101913	110594	229161
11	1	0	0	100672	91790.6	249745
12	1	1	0	93863.8	127320	249839

As we can see in the above output image, the first column is added successfully, which corresponds to the constant term of the MLR equation.

Step: 2:

- Now, we are actually going to apply a backward elimination process. Firstly we will create a new feature vector **x_opt**, which will only contain a set of independent features that are significantly affecting the dependent variable.
- Next, as per the Backward Elimination process, we need to choose a significant level(0.5), and then need to fit the model with all possible predictors. So for fitting the model, we will create a **regressor_OLS** object of new class **OLS** of **statsmodels** library. Then we will fit it by using the **fit()** method.
- Next we need **p-value** to compare with SL value, so for this we will use **summary()** method to get the summary table of all the values. Below is the code for it:

```
x_opt=x [:, [0,1,2,3,4,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

Output: By executing the above lines of code, we will get a summary table. Consider the below image:

```
Out[10]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.945
Method:                 Least Squares   F-statistic:                     169.9
Date:                Mon, 14 Oct 2019   Prob (F-statistic):           1.34e-27
Time:                        17:49:58   Log-Likelihood:                -525.38
No. Observations:                  50   AIC:                             1063.
Df Residuals:                      44   BIC:                             1074.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.013e+04   6884.820      7.281      0.000    3.62e+04     6.4e+04
x1           198.7888   3371.007      0.059      0.953   -6595.030    6992.607
x2           -41.8870   3256.039     -0.013      0.990   -6604.003    6520.229
x3             0.8060      0.046     17.369      0.000       0.712       0.900
x4            -0.0270      0.052     -0.517      0.608      -0.132       0.078
x5             0.0270      0.017      1.574      0.123      -0.008       0.062
==============================================================================
Omnibus:                       14.782   Durbin-Watson:                   1.283
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.266
Skew:                          -0.948   Prob(JB):                     2.41e-05
Kurtosis:                       5.572   Cond. No.                     1.45e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.45e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

In the above image, we can clearly see the p-values of all the variables. Here **x1, x2 are dummy variables, x3 is R&D spend, x4 is Administration spend, and x5 is Marketing spend**.

From the table, we will choose the highest p-value, which is for x1=0.953 Now, we have the highest p-value which is greater than the SL value, so will remove the x1 variable (dummy variable) from the table and will refit the model. Below is the code for it:

```
x_opt=x[:, [0,2,3,4,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

Output:

```
Out[11]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.946
Method:                 Least Squares   F-statistic:                     217.2
Date:                Mon, 14 Oct 2019   Prob (F-statistic):           8.50e-29
Time:                        18:03:48   Log-Likelihood:                -525.38
No. Observations:                  50   AIC:                             1061.
Df Residuals:                      45   BIC:                             1070.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.018e+04   6747.623      7.437      0.000    3.66e+04    6.38e+04
x1          -136.5042   2801.719     -0.049      0.961   -5779.456    5506.447
x2             0.8059      0.046     17.571      0.000       0.714       0.898
x3            -0.0269      0.052     -0.521      0.605      -0.131       0.077
x4             0.0271      0.017      1.625      0.111      -0.007       0.061
==============================================================================
Omnibus:                       14.892   Durbin-Watson:                   1.284
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.665
Skew:                          -0.949   Prob(JB):                     1.97e-05
Kurtosis:                       5.608   Cond. No.                     1.43e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.43e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

As we can see in the output image, now five variables remain. In these variables, the highest p-value is 0.961. So we will remove it in the next iteration.

- Now the next highest value is 0.961 for x1 variable, which is another dummy variable. So we will remove it and refit the model. Below is the code for it:

```
x_opt= x[:, [0,3,4,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

Output:

```
Out[12]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                     296.0
Date:                Mon, 14 Oct 2019   Prob (F-statistic):           4.53e-30
Time:                        18:08:41   Log-Likelihood:                -525.39
No. Observations:                  50   AIC:                             1059.
Df Residuals:                      46   BIC:                             1066.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.012e+04   6572.353      7.626      0.000    3.69e+04    6.34e+04
x1             0.8057      0.045     17.846      0.000       0.715       0.897
x2            -0.0268      0.051     -0.526      0.602      -0.130       0.076
x3             0.0272      0.016      1.655      0.105      -0.006       0.060
==============================================================================
Omnibus:                       14.838   Durbin-Watson:                   1.282
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.442
Skew:                          -0.949   Prob(JB):                     2.21e-05
Kurtosis:                       5.586   Cond. No.                     1.40e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
[2] The condition number is large, 1.4e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

In the above output image, we can see the dummy variable(x2) has been removed. And the next highest value is .602, which is still greater than .5, so we need to remove it.

- Now we will remove the Admin spend which is having .602 p-value and again refit the model.

```
x_opt=x[:, [0,3,5]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

Output:

```
Out[13]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.950
Model:                            OLS   Adj. R-squared:                  0.948
Method:                 Least Squares   F-statistic:                     450.8
Date:                Mon, 14 Oct 2019   Prob (F-statistic):           2.16e-31
Time:                        18:13:46   Log-Likelihood:                -525.54
No. Observations:                  50   AIC:                             1057.
Df Residuals:                      47   BIC:                             1063.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4.698e+04   2689.933     17.464      0.000    4.16e+04    5.24e+04
x1             0.7966      0.041     19.266      0.000       0.713       0.880
x2             0.0299      0.016      1.927      0.060      -0.001       0.061
==============================================================================
Omnibus:                       14.677   Durbin-Watson:                   1.257
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.161
Skew:                          -0.939   Prob(JB):                     2.54e-05
Kurtosis:                       5.575   Cond. No.                     5.32e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.32e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

As we can see in the above output image, the variable (Admin spend) has been removed. But still, there is one variable left, which is **marketing spend** as it has a high p-value **(0.60)**. So we need to remove it.

- Finally, we will remove one more variable, which has .60 p-value for marketing spend, which is more than a significant level.
  Below is the code for it:

```
x_opt=x[:, [0,3]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

Output:

```
Out[14]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.947
Model:                            OLS   Adj. R-squared:                  0.945
Method:                 Least Squares   F-statistic:                     849.8
Date:                Mon, 14 Oct 2019   Prob (F-statistic):           3.50e-32
Time:                        18:16:40   Log-Likelihood:                -527.44
No. Observations:                  50   AIC:                             1059.
Df Residuals:                      48   BIC:                             1063.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4.903e+04   2537.897     19.320      0.000    4.39e+04    5.41e+04
x1             0.8543      0.029     29.151      0.000       0.795       0.913
==============================================================================
Omnibus:                       13.727   Durbin-Watson:                   1.116
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               18.536
Skew:                          -0.911   Prob(JB):                     9.44e-05
Kurtosis:                       5.361   Cond. No.                     1.65e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified
[2] The condition number is large, 1.65e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

As we can see in the above output image, only two variables are left. So only the **R&D independent variable** is a significant variable for the prediction. So we can now predict efficiently using this variable.

Estimating the performance:

In the previous topic, we have calculated the train and test score of the model when we have used all the features variables. Now we will check the score with only one feature variable (R&D spend). Our dataset now looks like:

data_set - DataFrame

Index	R&D Spend	Profit
0	165349	192262
1	162598	191792
2	153442	191050
3	144372	182902
4	142107	166188
5	131877	156991
6	134615	156123
7	130298	155753
8	120543	152212
9	123335	149760
10	101913	146122
11	100672	144259
12	93863.8	141586

Format Resize Background Column min; Save and Close Close

Below is the code for Building Multiple Linear Regression model by only using R&D spend:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('50_CompList1.csv')

#Extracting Independent and dependent Variable
x_BE= data_set.iloc[:, :-1].values
y_BE= data_set.iloc[:, 1].values



# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_BE_train, x_BE_test, y_BE_train, y_BE_test= train_test_split(x_BE, y_BE, test_size= 0.2, random_state=0)

#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(nm.array(x_BE_train).reshape(-1,1), y_BE_train)

#Predicting the Test set result;
y_pred= regressor.predict(x_BE_test)

#Cheking the score
print('Train Score: ', regressor.score(x_BE_train, y_BE_train))
print('Test Score: ', regressor.score(x_BE_test, y_BE_test))
```

Output:

After executing the above code, we will get the Training and test scores as:

```
Train Score:  0.9449589778363044
Test Score:  0.9464587607787219
```

As we can see, the training score is 94% accurate, and the test score is also 94% accurate. The difference between both scores is **.00149**. This score is very much close to the previous score, i.e., **0.0154**, where we have included all the variables.

We got this result by using one independent variable (R&D spend) only instead of four variables. Hence, now, our model is simple and accurate.

Python Design Patterns

Preparation

Aptitude
Aptitude

Logical Reasoning
Reasoning

Verbal Ability
Verbal Ability

Interview Questions
Interview Questions

Company Interview Questions
Company Questions

Trending Technologies

Artificial Intelligence
Artificial Intelligence

AWS Tutorial
AWS

Selenium tutorial
Selenium

Cloud Computing
Cloud Computing

Hadoop tutorial
Hadoop

ReactJS Tutorial
ReactJS

Data Science Tutorial
Data Science

Angular 7 Tutorial
Angular 7

Blockchain Tutorial
Blockchain

Git Tutorial
Git

Machine Learning Tutorial
Machine Learning

DevOps Tutorial
DevOps

B.Tech / MCA

DBMS tutorial
DBMS

Data Structures tutorial

DAA tutorial
DAA

Operating System

Computer Network tutorial