Implementing linear regression with Python

Let's now go ahead and try to make a simple linear regression model and see what are the issues that we face and how can they be resolved to make the model more robust. We will use the advertising data from the lab folder.

The following two methods implement linear regression in Python:

- The `ols` method and the `statsmodel.formula.api` library
- The `scikit-learn` package

Let's implement a simple linear regression using the first method and then build upon a multiple-linear regression model. We will then also look at how the second method is used to do the same.

Linear regression using the statsmodel library

Let's first import the `Advertising` data, as shown:

```
import pandas as pd
advert=pd.read_csv('C:/FILE_PATH/Advertising.csv')
advert.head()
```

This dataset contains data about the advertising budget spent on TV, Radio, and Newspapers, for a particular product and the resulting sales. We will expect a positive correlation between such advertising costs and sales. We have already know that there is a good correlation between TV advertising costs and sales. Let's see whether it is present or not. If yes, how does the relationship look like and to do that we write the following code:

```
import statsmodels.formula.api as smf
model1=smf.ols(formula='Sales~TV',data=advert).fit()
model1.params
```

In this code snippet, we have assumed a linear relationship between advertising costs on TV and sales. We have also created a best fit using the least sum of square method. This snippet will output the values for model parameters that is a and β. The following is the output:

```
Intercept    7.032594
TV           0.047537
dtype: float64
```

In the notation that we have been using, a is the intercept and β is the slope. Thus:

$$a = 7.03 \, and \, \beta = 0.047$$

The equation for the model will be:

$$Sales = 7.032 + 0.047 * TV$$

The equation implies that an increase of 100 units in advertising costs will increase the sale by four units.

If you remember, we learnt that the values of these parameters are estimates and there will be a p-value associated to these. If the p-values are very small, then it can be accepted that these parameters have a non-zero value and are statistically significant in the model. Let's have a look at the p-values for these parameters:

```
model1.pvalues
```

```
Intercept    1.406300e-35
TV           1.467390e-42
dtype: float64
```

As it can be seen, the p-values are very small; hence, the parameters are significant.

Let's also check another important indicator of the model efficacy and that is R^2. As we saw earlier, there is a ready-made method for doing this. This can be done by typing the following code line:

```
model1.rsquared
```

The value comes out to be 0.61.

If we want the entire important model parameters at one go, we can take a look at the model summary by writing this snippet:

```
model1.summary()
```

The result is as follows:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.612
Model:	OLS	**Adj. R-squared:**	0.610
Method:	Least Squares	**F-statistic:**	312.1
Date:	Mon, 07 Sep 2015	**Prob (F-statistic):**	1.47e-42
Time:	01:30:20	**Log-Likelihood:**	-519.05
No. Observations:	200	**AIC:**	1042.
Df Residuals:	198	**BIC:**	1049.
Df Model:	1		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	7.0326	0.458	15.360	0.000	6.130 7.935
TV	0.0475	0.003	17.668	0.000	0.042 0.053

Model 1 Summary

As we can see, the F-statistic for this model is very high and the associated p-value is negligible, suggesting that the parameter estimates for this model were all significant and non-zero.

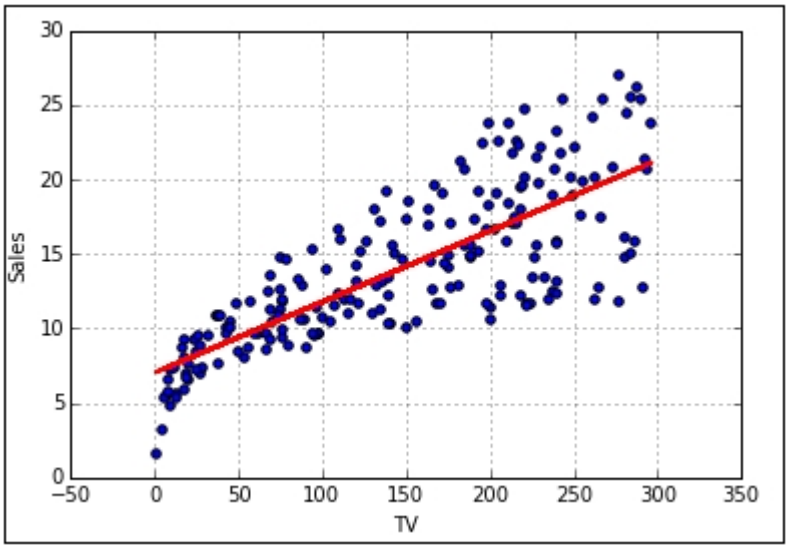
Let's now predict the values of sales based on the equation we just derived. This can be done using the following snippet:

```
sales_pred=model1.predict(pd.DataFrame(advert['TV']))
sales_pred
```

This equation basically calculates the predicted sales value for each row based on the model equation using TV costs. One can plot `sales_pred` against the TV advertising costs to find the line of best fit. Let's do that:

```
import matplotlib.pyplot as plt
%matplotlib inline
advert.plot(kind='scatter', x='TV', y='Sales')
plt.plot(pd.DataFrame(advert['TV']),sales_pred,c='red',linewidth=2)
```

We get the following plot as the output. The red line is the line of best fit (obtained from the model). The blue dots represent the actual data present:



Line of best fit (obtained from the model) and the scatter plot of actual data

Now, let's calculate the RSE term for our prediction using the following code snippet:

```
advert['sales_pred']=0.047537*advert['TV']+7.03
advert['RSE']=(advert['Sales']-advert['sales_pred'])**2
RSEd=advert.sum()['RSE']
RSE=np.sqrt(RSEd/198)
salesmean=np.mean(advert['Sales'])
error=RSE/salesmean
RSE,salesmean,error
```

The output consists of three numbers, first of which is *RSE=3.25*, second is `salesmean` (mean of actual sales) = 14.02 and error is their ratio, which is equal to 0.23. Thus, on an average this model will have 23%, even if the coefficients are correctly predicted. This is a significant amount of errors and we would like to bring it down by some means. Also, the R^2 value of 0.61 can be improved upon. One thing we can try is to add more columns in the model, as predictors and see whether it improves the result or not.

Multiple linear regression

When linear regression involves more than one predictor variable, then it is called multiple linear regression. The nature of the model remains the same (linear), except that there might be separate slope (β) coefficients associated with each of the predictor variables. The model will be represented, as follows:

$$y \bmod el = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 \ldots\ldots + \beta_n X_n$$

Each β_i will be estimated using the same least sum of squares method; hence, would have a p-value associated with the estimation. The smaller the p-value associated with a variable, the more the significance of that variable to the model. The variables with large p-values should be eliminated from the model as they aren't good predictors of the output variable.

While the multiple regression gives us with the possibility of using more variables as predictors; hence, it increases the efficiency of the model. It also increases the complexity of the process of model building, as the selection of the variables to be kept and discarded in the model becomes tedious.

With this simple dataset of three predictor variables, there can be seven possible models. They are as follows:

- Model 1: Sales~TV
- Model 2: Sales~Newspaper
- Model 3: Sales~Radio
- Model 4: Sales~TV+Radio
- Model 5: Sales~TV+Newspaper
- Model 6: Sales~Newspaper+Radio
- Model 7: Sales~TV+Radio+Newspaper

For a model with *p* possible predictor variables, there can be 2^p-1 possible models; hence, as the number of predictors increases, the selection becomes tedious.

It would have been a tedious task to choose from so many possible models. Thankfully, there are a few guidelines to filter some of these and then navigate towards the most efficient one. The following are the guidelines:

- Keep the variables with low p-values and eliminate the ones with high p-values
- Inclusion of a variable to the model should ideally increase the value of R^2 (although it is not a very reliable indicator of the same and looking at the adjusted R^2 is preferred. The concept of adjusted R^2 and why it is a better indicator than R^2 will be explained later in this chapter).

Based on these guidelines, there are two kinds of approaches to select the predictor variables to go in the final model:

- **Forward selection**: In this approach, we start with a null model without any predictor and then start adding predictor variables one by one. The variable whose addition results into a model with the lowest residual sum of squares will be added first to the model. If the p-value for the variable is small enough and the value of the adjusted R^2 goes up; the predictor variable is included in the model. Otherwise, it is not included in the model.
- **Backward selection**: In this approach, one starts with a model that has all the possible predictor variables in the model and discards some of them. If the p-value of a predictor variable is large and the value of the adjusted R^2 goes up, the predictor variable is discarded from the model. Otherwise, it remains a part of the model.

Many of the statistical programs, including the Python, give us an option to select from the two preceding approaches while implementing a linear regression. The statistical program then implements the linear regression using the selected approach.

For now, let us manually add a few variables and see how it changes the model parameters and efficacy, so that we can get a better glimpse of what goes on behind the curtain when these approaches are implemented by the statistical program.

We have already seen one model assuming a linear relationship between sales and TV advertising costs. We can ignore the other models consisting of single variables (that is newspaper and radio, as they have a small correlation compared to TV). Let us now try to add more variables to the model we already have and see how the parameters and efficacy change.

Let us try adding the newspaper variable to the model using the following code snippet:

```
import statsmodels.formula.api as smf
model2=smf.ols(formula='Sales~TV+Newspaper',data=advert).fit()
model2.params
```

The following are the results:

```
Intercept    5.774948
TV           0.046901
Newspaper    0.044219
dtype: float64
```

```
Intercept    3.145860e-22
TV           5.507584e-44
Newspaper    2.217084e-05
dtype: float64
```

Model 2 coefficients and p-values

The p-values for the coefficients are very small, suggesting that all the estimates are significant. The equation for this model will be:

$$Sales = 5.77 + 0.046 * TV + 0.04 * Newspaper$$

The values of R^2 and adjusted R^2 are 0.646 and 0.642, which is just a minor improvement from the value obtained in the earlier model.

The values can be predicted using the following snippet:

```
sales_pred=model2.predict(advert[['TV','Newspaper']])
sales_pred
```

To calculate the RSE, we modify the snippet a little:

```
import numpy as np
advert['sales_pred']=5.77 + 0.046*advert['TV'] + 0.04*advert['Newspaper']
advert['RSE']=(advert['Sales']-advert['sales_pred'])**2
RSEd=advert.sum()['RSE']
RSE=np.sqrt(RSEd/197)
salesmean=np.mean(advert['Sales'])
error=RSE/salesmean
RSE,salesmean,error
```

The value of RSE comes out to be 3.12 (22%), not very different from the model with only TV. The number 197 comes from the (n-p-1) term in the formula for RSE, where *n=200*, *p=2* for the current model. The following table is the model summary:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.646
Model:	OLS	**Adj. R-squared:**	0.642
Method:	Least Squares	**F-statistic:**	179.6
Date:	Wed, 09 Sep 2015	**Prob (F-statistic):**	3.95e-45
Time:	19:11:41	**Log-Likelihood:**	-509.89
No. Observations:	200	**AIC:**	1026.
Df Residuals:	197	**BIC:**	1036.
Df Model:	2		

Model 2 Summary

Although as the F-statistic decreases, the associated p-value also decreases. But, it is just a marginal improvement to the model, as we can see in the adj. R^2 value. So, adding newspaper didn't improve the model significantly.

Let's try adding radio to the model instead of the newspaper. Radio had the second best correlation with the `Sales` variable in the correlation matrix we created earlier in the chapter. Thus, one expects some significant improvement in the model upon its addition to the model. Let's see if that happens or not:

```
import statsmodels.formula.api as smf
model3=smf.ols(formula='Sales~TV+Radio',data=advert).fit()
model3.params
```

The output parameters and the associated p-values of this model are, as follows:

```
Intercept    2.921100      Intercept    4.565557e-19
TV           0.045755      TV           5.436980e-82
Radio        0.187994      Radio        9.776972e-59
dtype: float64             dtype: float64
```

Model 2 coefficients and p-values

The model can be represented as the following:

$$Sales = 2.92 + 0.045 * TV + 0.18 * Radio$$

The values can be predicted based on the preceding model using the following snippet:

```
sales_pred=model3.predict(advert[['TV','Radio']])
sales_pred
```

The model summary looks something similar to the following screenshot:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.896
Method:	Least Squares	**F-statistic:**	859.6
Date:	Tue, 08 Sep 2015	**Prob (F-statistic):**	4.83e-98
Time:	00:15:23	**Log-Likelihood:**	-386.20
No. Observations:	200	**AIC:**	778.4
Df Residuals:	197	**BIC:**	788.3
Df Model:	2		

Model 3 summary

One thing to observe here is that the R^2 value has improved considerably due to the addition of radio to the model. Also, the F-statistic has increased significantly from the last model indicating a very efficient model.

The RSE can be calculated using the same method described previously. The value for this model comes out to be 1.71 (around 12%),which is much better than the 23% and 22% in the previous model.

Thus, we can conclude that radio is a great addition to the model and TV and radio advertising costs have been able to describe the sales very well and this model itself is a very efficient model. But, can we improve it a bit further by combining all three predictor variables?

The last thing that we should try is, all the predictor variables together by using the following code:

```
import statsmodels.formula.api as smf
model4=smf.ols(formula='Sales~TV+Radio+Newspaper',data=advert).fit()
model4.params
```

The estimates of the coefficients and the associated p-values for this model will be as follows:

```
Intercept    2.938889
TV           0.045765
Radio        0.188530
Newspaper   -0.001037
dtype: float64
```

```
Intercept    1.267295e-17
TV           1.509960e-81
Radio        1.505339e-54
Newspaper    8.599151e-01
dtype: float64
```

Model 4 coefficients and p-values

The p-values for the coefficients are very small, suggesting that all the estimates are significant. The equation for this model will be:

$$Sales = 2.93 + 0.045 * TV + 0.18 * Radio - 0.01 * Newspaper$$

The values of sales can be predicted using the following snippet:

```
sales_pred=model4.predict(advert[['TV','Radio','Newspaper']])
sales_pred
```

The summary of the model is shown in the following table:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.896
Method:	Least Squares	**F-statistic:**	570.3
Date:	Thu, 10 Sep 2015	**Prob (F-statistic):**	1.58e-96
Time:	22:05:18	**Log-Likelihood:**	-386.18
No. Observations:	200	**AIC:**	780.4
Df Residuals:	196	**BIC:**	793.6
Df Model:	3		

Model 4 summary

The most striking feature of this model is that the estimate of the coefficients is very similar to that in the previous model. The intercept, coefficient for TV, and the coefficient for Radio are more or less the same. The values of R^2 and *adj-*R^2 are also similar to the previous model.

The value of RSE can be calculated in a similar way, as before. The value comes out to 2.57 (18%), which is more than the previous model.

Other things to note about this model are the following:

- There is a small negative coefficient for the newspaper. When we considered only TV and newspaper in the model, the coefficient of the newspaper was significantly positive. Something affected the coefficient of the newspaper when it became a part of the model in presence of TV and radio.
- For this model, the F-statistic has decreased considerably to 570.3 from 859.6 in the previous model. This suggests that the partial benefit of adding newspaper to the model containing TV and radio is negative.
- The value of RSE increases on addition of newspaper to the model.

All these point in the direction that the model actually became a little less efficient on addition of newspaper to the previous model. What is the reason? Multi-collinearity is the reason for the sub-optimal performance of the model when newspaper was added to the final model. Multi-collinearity alludes to the correlation between the predictor variables of the model. Go back to the correlation matrix that we created for this dataset and you will find that there is a significant correlation of 0.35 between radio and newspaper. This means that the expense on Newspaper is related to that on the Radio. This relationship between the predictor variable increases the variability of the co-efficient estimates of the related predictor variables.

Model validation

Any predictive model needs to be validated to see how it is performing on different sets of data, whether the accuracy of the model is constant over all the sources of similar data or not. This checks the problem of over-fitting, wherein the model fits very well on one set of data but doesn't fit that well on another dataset. One common method is to validate a model train-test split of the dataset. Another method is k-fold cross validation, about which we will learn more in the later chapter.

Training and testing data split

Ideally, this step should be done right at the onset of the modelling process so that there are no sampling biases in the model; in other words, the model should perform well even for a dataset that has the same predictor variables, but their means and variances are very different from what the model has been built upon. This can happen because the dataset on which the model is built (training) and the one on which it is applied (testing) can come from different sources. A more robust way to do this is a process called the k-fold cross validation, we will cover cross validation in a future lab.

Let's see how we can split the available dataset in the training and testing dataset and apply the model to the testing dataset to get other results:

```
import numpy as np
a=np.random.randn(len(advert))
check=a<0.8
training=advert[check]
testing=advert[~check]
```

NOTE: The above sampling technique is naïve and doesn’t consider sampling etc. You might remember from the previous lab that Scikitlearn has a very helpful function train_test_split which automates the process of selecting a training and testing set.
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

The ratio of split between training and testing datasets is 80:20; in other words, 160 rows of the advert dataset will be in training and 40 rows in testing.

Let's create a model on training the data and test the model performance on the testing data. Let us create the only model that works best (we have found it already), the one with TV and radio variables, as predictor variables:

```
import statsmodels.formula.api as smf
model5=smf.ols(formula='Sales~TV+Radio',data=training).fit()
model5.summary()
```

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.878
Model:	OLS	**Adj. R-squared:**	0.876
Method:	Least Squares	**F-statistic:**	529.7
Date:	Sat, 12 Sep 2015	**Prob (F-statistic):**	6.44e-68
Time:	12:21:12	**Log-Likelihood:**	-293.70
No. Observations:	150	**AIC:**	593.4
Df Residuals:	147	**BIC:**	602.4
Df Model:	2		

	coef	**std err**	**t**	**P>|t|**	**[95.0% Conf. Int.]**
Intercept	2.8606	0.361	7.917	0.000	2.147 3.575
TV	0.0468	0.002	27.901	0.000	0.043 0.050
Radio	0.1793	0.010	18.437	0.000	0.160 0.199

Model 5 coefficients and p-values

Most of the model parameters, such as intercept, coefficient estimates, and R^2 are very similar. The difference in F-statistics can be attributed to a smaller dataset. The smaller the dataset, the larger the value of SSD and the smaller the value of the (n-p-1) term in F-statistic formula; both contribute towards the decrease in the F-statistic value.

The model can be written, as follows:

$$Sales \sim 2.86 + 0.04 * TV + 0.17 * Radio$$

Let us now predict the sales values for the testing dataset:

```
sales_pred=model5.predict(training[['TV','Radio']])
sales_pred
```

The value of RSE for this prediction on the testing dataset can be calculated using the following snippet:

```
import numpy as np
testing['sales_pred']=2.86 + 0.04*testing['TV'] + 0.17*testing['Radio']
testing['RSE']=(testing['Sales']-testing['sales_pred'])**2
RSEd=testing.sum()['RSE']
RSE=np.sqrt(RSEd/51)
```

```
salesmean=np.mean(testing['Sales'])
error=RSE/salesmean
RSE,salesmean,error
```

The value of RSE comes out to be 2.54 over a sales mean (in the testing data) of 14.80 amounting to an error of 17%.

We can see that the model doesn't generalize very well on the testing dataset, as the RSE for the same model is different in the two cases. It implies some degree of over fitting when we tried to build the model based on the entire dataset. The RSE with the training-testing split, albeit a bit more, is more reliable and replicable.

Summary of models

We have tried four models previously. Let us summarize the major results from each of the models, at one place:

Name	Definition	R2/Adj-R2	F-statistic	F-statistic (p-value)	RSE
Model 1	Sales ~ TV	0.612/0.610	312.1	1.47e-42	3.25 (23%)
Model 2	Sales ~ TV+Newspaper	0.646/0.642	179.6	3.95e-45	3.12(22%)
Model 3	Sales ~ TV+Radio	0.897/0.896	859.6	4.83e-98	1.71(12%)
Model 4	Sales ~ TV+Radio+Newspaper	0.897/0.896	570.3	1.58e-96	1.80(13%)

Guide for selection of variables

To summarize, for a good linear model, the predictor variables should be chosen based on the following criteria:

- **R^2**: R^2 will always increase when you add a new predictor variable to the model. However, it is not a very reliable check of the increased efficiency of the model. Rather, for an efficient model, we should check the adjusted-R^2. This should increase on adding a new predictor variable.
- **p-values**: The lower the p-value for the estimate of the predictor variable, the better it is to add the predictor variable to the model.
- **F-statistic**: The value of the F-statistic for the model should increase after the addition of a new predictor variable for a predictor variable to be an efficient addition to the model. The increase in the F-statistic is a proxy to the improvement in the model brought upon solely by the addition of that particular variable. Alternatively, the p-value associated with the F-statistic should decrease on the addition of a new predictor variable.
- **RSE**: The value of RSE for the new model should decrease on the addition of the new predictor variable.
- **VIF**: To take care of the issues arising due to multi-collinearity one needs to eliminate the variables with large values of VIF.

Linear regression with scikit-learn

Let's now re-implement the linear regression model using the `scikit-learn` package. This method is more elegant as it has more in-built methods to perform the regular processes associated with regression. For example, you might remember from the last lab that there is a separate method for splitting the dataset into training and testing datasets:

```
from sklearn.linear_model import LinearRegression
from sklearn.cross_validation import train_test_split
feature_cols = ['TV', 'Radio']
X = advert[feature_cols]
Y = advert['Sales']
trainX,testX,trainY,testY = train_test_split(X,Y, test_size = 0.2)
lm = LinearRegression()
lm.fit(trainX, trainY)
```

We split the advert dataset into train and test dataset and built the model on TV and radio variables from the test dataset. The following are the parameters of the model:

```
print lm.intercept_
print lm.coef_
```

The result is as follows: Intercept – 2.918, TV coefficient – 0.04, Radio coefficient – 0.186

A better way to look at the coefficients is to use the `zip` method to write the variable name and coefficient together. The required snippet and the output are mentioned in the following code:

```
zip(feature_cols, lm.coef_)
[('TV', 0.045706061219705982), ('Radio', 0.18667738715568111)]
```

The value of R^2 is calculated by typing the following code:

```
lm.score(trainX, trainY)
```

The value comes out to be around 0.89, very close to the value obtained by the method used earlier.

The model can be used to predict the value of sales using TV and radio variables from the test dataset, as follows:

```
lm.predict(testX)
```

Feature selection with scikit-learn

As stated before, many of the statistical tools and packages have in-built methods to conduct a variable selection process (forward selection and backward selection). If it is done manually, it will consume a lot of time and selecting the most important variables will be a tedious task compromising the efficiency of the model.

One advantage of using the `scikit-learn` package for regression in Python is that it has this particular method for feature selection. This works more or less like backward selection (not

exactly) and is called **Recursive Feature Elimination** (**RFE**). One can specify the number of variables they want in the final model.

The model is first run with all the variables and certain weights are assigned to all the variables. In the subsequent iterations, the variables with the smallest weights are pruned from the list of variables till the desired number of variables is left.

Let us see how one can do a feature selection in `scikit-learn`:

```
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
feature_cols = ['TV', 'Radio','Newspaper']
X = advert[feature_cols]
Y = advert['Sales']
estimator = SVR(kernel="linear")
selector = RFE(estimator,2,step=1)
selector = selector.fit(X, Y)
```

We use the methods named RFE and SVR in-built in `scikit-learn`. We indicate that we want to estimate a linear model and the number of desired variables in the model to be two.

To get the list of selected variables, one can write the following code snippet:

```
selector.support_
```

It results in an array mentioning whether the variables in *X* have been selected for the model or not. **True** means that the variable has been selected, while **False** means otherwise. In this case, the result is as follows:

```
array([ True,  True, False], dtype=bool)
```

Result of feature selection process

In our case, *X* consists of three variables: TV, radio, and newspaper. The preceding array suggests that TV and radio have been selected for the model, while the newspaper hasn't been selected. This concurs with the variable selection we had done manually.

This method also returns a ranking, as described in the following example:

```
selector.ranking_
```

```
array([1, 1, 2])
```

Result of feature selection process

All the selected variables will have a ranking of **1** while the subsequent ones will be ranked in descending order of their significance. A variable with rank **2** will be more significant to the model than the one with a rank of **3** and so on.