01-Classification

October 30, 2017

```
In [16]: from datascience import *
         import matplotlib.pyplot as plt
         import numpy as np
         %matplotlib inline
         plt.style.use('ggplot')
```

1 Intro to Regression

A popular classification model is logistic regression. This is what Underwood and Sellers use in their article to classify whether a text was reviewed or randomly selected from HathiTrust. Today we'll look at the difference between regression and classification tasks, and how we can use a logistic regression model to classify text like Underwood and Sellers. We won't have time to go through their full code, but if you're interested I've provided a walk-through in the second notebook.

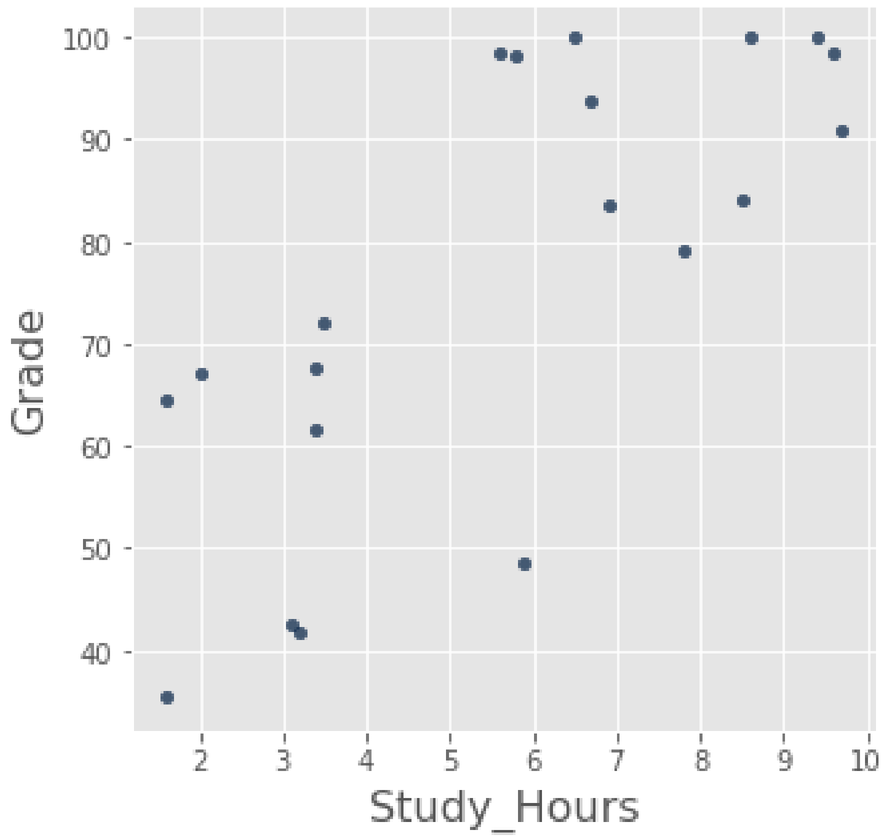
To explore the regression model let's first create some dummy data:

```
In [17]: demo_tb = Table()
         demo_tb['Study_Hours'] = [2.0, 6.9, 1.6, 7.8, 3.1, 5.8, 3.4, 8.5, 6.7, 1.6, 8.6, 3.4,
         demo_tb['Grade'] = [67.0, 83.6, 35.4, 79.2, 42.4, 98.2, 67.6, 84.0, 93.8, 64.4, 100.0
         demo_tb['Pass'] = [0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1]
         demo_tb.show()

<IPython.core.display.HTML object>
```

1.1 Intuiting the Linear Regression Model

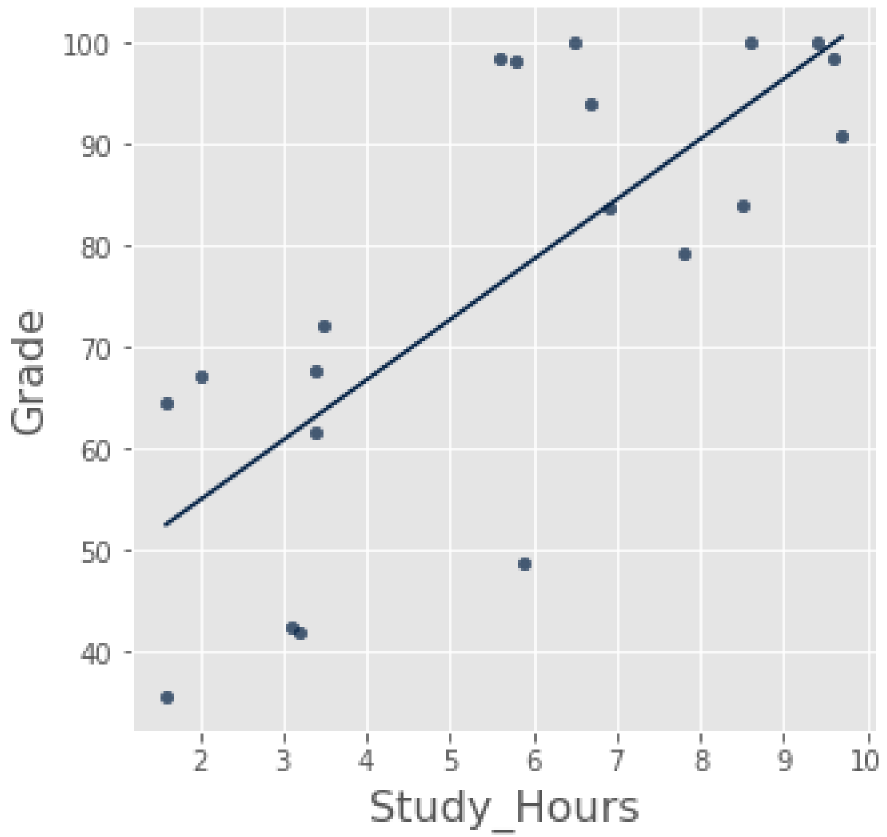
You may have encountered linear regression in previous coursework of yours. Linear regression, in its simple form, tries to model the relationship between two continous variables as a straight line. It interprets one variable as the input, and the other as the output.:

```
In [18]: demo_tb.scatter('Study_Hours','Grade')
```



In the example above, we're interested in `Study_Hours` and `Grade`. This is a natural "input" "output" situation. To plot the regression line, or ***best-fit***, we can feed in `fit_line=True` to the `scatter` method:

```
In [19]: demo_tb.scatter('Study_Hours','Grade', fit_line=True)
```



The better this line fits the points, the better we can predict one's `Grade` based on their `Study_Hours`, even if we've never seen anyone put in that number of study hours before.

The regression model above can be expressed as:

$GRADE_i = \alpha + \beta STUDYHOURS + \epsilon_i$

The variable we want to predict (or model) is the left side `y` variable, here `GRADE`. The variable which we think has an influence on our left side variable is on the right side, the independent variable `STUDYHOURS`. The α term is the y-intercept and the ϵ_i describes the randomness.

The β coefficient on `STUDYHOURS` gives us the slope, in a univariate regression. That's the factor on `STUDYHOURS` to get `GRADE`.

If we want to build a model for the regression, we can use the `sklearn` library. `sklearn` is by far the most popular machine learning library for Python, and its syntax is really important to learn. In the next cell we'll import the `Linear Regression` model and assign it to a `linreg` variable:

```
In [77]: from sklearn.linear_model import LinearRegression
         linreg = LinearRegression()
```

Before we go any further, `sklearn` likes our data in a very specific format. The `X` must be in an array of arrays, each sub array is an observation. Because we only have one independent variable, we'll have sub arrays of `len` 1. We can do that with the `reshape` method:

```
In [78]: X = demo_tb['Study_Hours'].reshape(-1,1)
         X

Out[78]: array([[ 2. ],
                [ 6.9],
                [ 1.6],
                [ 7.8],
                [ 3.1],
                [ 5.8],
                [ 3.4],
                [ 8.5],
                [ 6.7],
                [ 1.6],
                [ 8.6],
                [ 3.4],
                [ 9.4],
                [ 5.6],
                [ 9.6],
                [ 3.2],
                [ 3.5],
                [ 5.9],
                [ 9.7],
                [ 6.5]])
```

Your output, or dependent variable, is just one array with no sub arrays.

```
In [22]: y = demo_tb['Grade'].reshape(len(demo_tb['Grade']),)
         y

Out[22]: array([  67. ,   83.6,   35.4,   79.2,   42.4,   98.2,   67.6,   84. ,
                  93.8,   64.4,  100. ,   61.6,  100. ,   98.4,   98.4,   41.8,
                  72. ,   48.6,   90.8,  100. ])
```

We then use the `fit` method to fit the model. This happens in-place, so we don't have to reassign the variable:

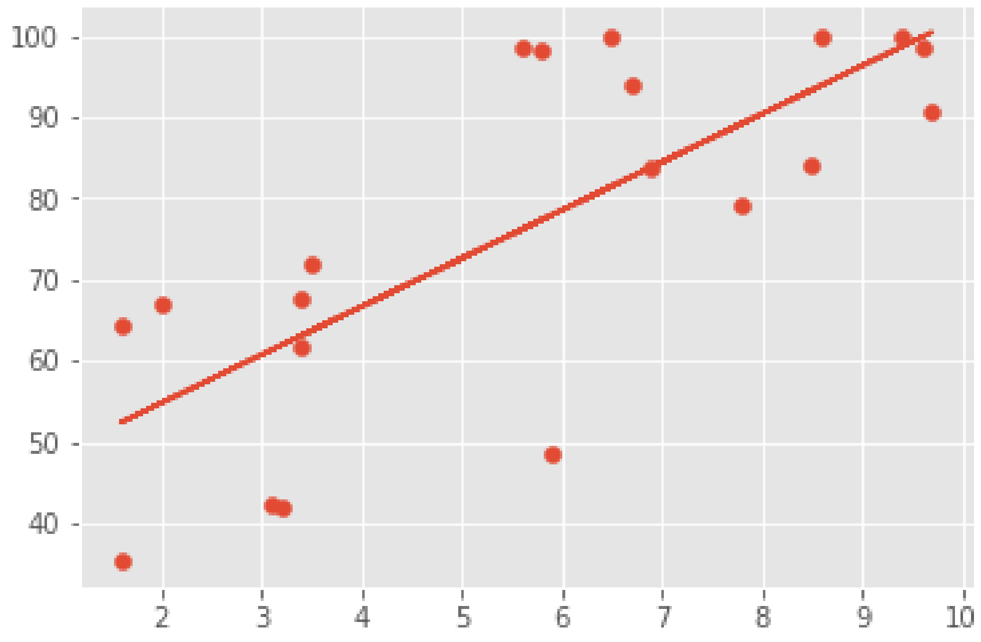
```
In [23]: linreg.fit(X, y)

Out[23]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

We can get back the `intercept_` and β `coef_` with attributes of the `linreg` object:

```
In [24]: B0, B1 = linreg.intercept_, linreg.coef_[0]
         B0, B1

Out[24]: (42.897229302892598, 5.9331153718275509)
```

So this means:
$GRADE_i = 42.897229302892598 + 5.9331153718275509 * STUDYHOURS + \epsilon_i$
As a linear regression this is simple to interpret. To get our grade score, we take the number of study hours and multipy it by 5.9331153718275509 then we add 42.897229302892598 and that's our prediction.
If we look at our chart again but using the model we just made, that looks about right:

```
In [25]: y_pred = linreg.predict(X)
         print(X)
         print(y_pred)
         plt.scatter(X, y)
         plt.plot(X, y_pred)

[[ 2. ]
 [ 6.9]
 [ 1.6]
 [ 7.8]
 [ 3.1]
 [ 5.8]
 [ 3.4]
 [ 8.5]
 [ 6.7]
 [ 1.6]
 [ 8.6]
 [ 3.4]
 [ 9.4]
 [ 5.6]
 [ 9.6]
 [ 3.2]
 [ 3.5]
 [ 5.9]
 [ 9.7]
 [ 6.5]]
[  54.76346005   83.83572537   52.3902139    89.1755292    61.28988696
   77.30929846   63.06982157   93.32870996   82.64910229   52.3902139
   93.9220215    63.06982157   98.6685138    76.12267539   99.85513687
   61.88319849   63.6631331    77.90261     100.44844841   81.46247922]

Out[25]: [<matplotlib.lines.Line2D at 0x7fb71d619c88>]
```



We can evaluate how great our model is with the `score` method. We need to give it the `X` and observed `y` values, and it will predict its own `y` values and compare:

```
In [26]: linreg.score(X, y)
```

```
Out[26]: 0.55865744244266069
```

For the Linear Regression, `sklearn` returns an **R-squared** from the `score` method. The R-squared tells us how much of the variation in the data can be explained by our model, .559 isn't that bad, but obviously more goes into your `Grade` than *just* `Study_Hours`.

Nevertheless we can still predict a grade just like we did above to create that line, let's say I studied for 5 hours:

```
In [27]: linreg.predict([[5]])
```

```
Out[27]: array([ 72.56280616])
```

Maybe I should study more?

```
In [28]: linreg.predict([[20]])
```

```
Out[28]: array([ 161.55953674])
```

Wow! I rocked it.