

Day 06 - Polynomial Regression Models

Sept. 24, 2020



From Pre-Class Assignment

Challenging bits

- How to fit data with more than one feature
- How to determine a good fit when multiple features are used
- How to make sure OLS is doing the fit that I want

You will get more practice with this today.

Things that are still important:

- What is a regression model?
- How does a regression model work?
- What are the concerns about fitting with a line (or other function)?
- How do we determine if the fit is good?

Let's start by making some data

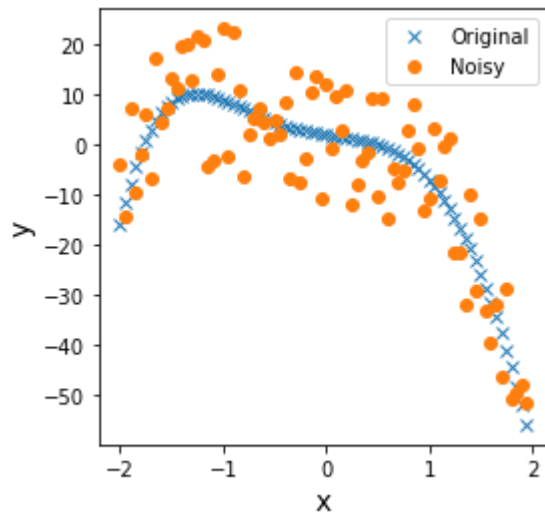
We will use quintic data: $y = 5x^5 + 3x^4 - 6x^3 + 2x^2 - 3x + 2$ from $x = -2$ to $+2$

```
In [1]: import statsmodels.api as sm
import numpy as np
import matplotlib.pyplot as plt

x_ary = np.arange(-2, 2, 0.05)
y_ary = +x_ary**5 - 3*x_ary**4 - 6*x_ary**3 + 2*x_ary**2 - 3*x_ary + 2
y_noisy = y_ary + np.random.uniform(-15, 15.0, len(y_ary))

fig = plt.figure(figsize=(4,4))
plt.plot(x_ary, y_ary, 'x')
plt.plot(x_ary, y_noisy, 'o')
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.legend(['Original', 'Noisy'])
```

Out[1]: <matplotlib.legend.Legend at 0x7ff780edef10>



Let's make a model using up to x^{10}

Our model will try to fit:

$$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5 + c_6 x^6 + c_7 x^7 + c_8 x^8 + c_9 x^9 + c_{10} x^{10} + \epsilon$$

to the noisy data. `statsmodels` will estimate the c_i each time we make the model. We will be able to set different c_i 's to zero to reduce the complexity of the model.

Ok let's run an OLS model using only the constant term

$$y_0 = c_0 + \epsilon$$

```
In [3]: model = sm.OLS(y_noisy, df['x0']) # make the model
results = model.fit()
results.summary()
```

Out[3]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.000
Model:	OLS	Adj. R-squared:	0.000
Method:	Least Squares	F-statistic:	nan
Date:	Thu, 24 Sep 2020	Prob (F-statistic):	nan
Time:	10:11:12	Log-Likelihood:	-344.10
No. Observations:	80	AIC:	690.2
Df Residuals:	79	BIC:	692.6
Df Model:	0		

Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
x0	-3.8682	2.009	-1.926	0.058	-7.867	0.130

Omnibus:	13.787	Durbin-Watson:	0.539
Prob(Omnibus):	0.001	Jarque-Bera (JB):	14.995
Skew:	-1.011	Prob(JB):	0.000554
Kurtosis:	3.642	Cond. No.	1.00

Warnings:

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

That was bad ($R^2 = 0$). Let's add the linear term.

$$y_1 = c_1 x + c_0 + \epsilon$$

That is better ($R^2 = 0.457$). Let's add the quadratic term

$$y_2 = c_2 x^2 + c_1 x + c_0 + \epsilon$$

Ok let's run this for each choice with increasing powers of x

$$y_0 = c_0 + \epsilon$$

$$y_1 = c_1 x + c_0 + \epsilon$$

$$y_{10} = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + \overset{\dots}{c_5 x^5} + c_6 x^6 + c_7 x^7 + c_8 x^8 + c_9 x^9 + c_{10} x^{10} + \epsilon$$

```

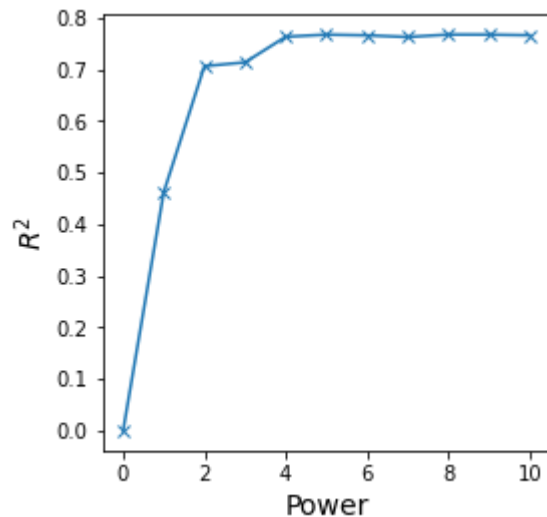
In [6]: rsqr_array = []
        columns = []

        for column in column_names:

            columns.append(column)
            model = sm.OLS(y_noisy, df[columns]) # make the model
            results = model.fit()                # run the OLS fit
            rsqr_array.append(results.rsquared_adj)

fig = plt.figure(figsize=(4,4))
plt.plot(np.arange(0,11,1), rsqr_array, 'x-')
plt.xlabel('Power', fontsize = 14)
plt.ylabel(r'$R^2$', fontsize = 14);

```



The quality of the fit "saturates" at x^5

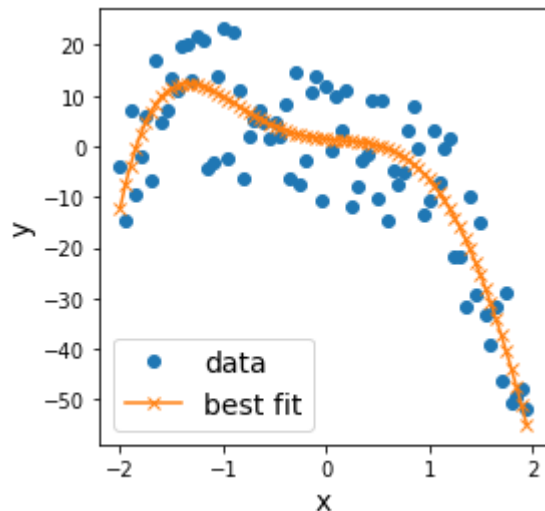
As we might have expected, $y_5 = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5 + \epsilon$ produces the "best" model

```
In [7]: best_fit = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5']

model = sm.OLS(y_noisy, df[best_fit]) # make the model
results_b = model.fit()

fig = plt.figure(figsize=(4,4))
plt.plot(x_ary, y_noisy, 'o')
plt.plot(x_ary, results_b.predict(), '-x')
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.legend(['data', 'best fit'], fontsize = 14)
```

Out[7]: <matplotlib.legend.Legend at 0x7ff7811ad910>



Let's look at the fits of other models

Linear fit: $y_1 = c_1 x + c_0 + \epsilon$

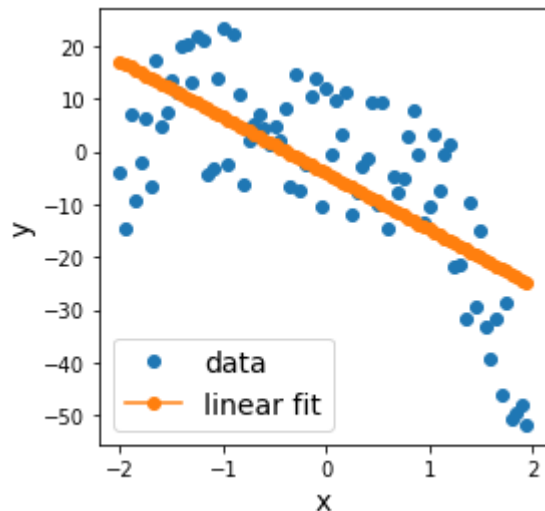
This is underfitting!

```
In [8]: linear_fit = ['x0', 'x1']

model = sm.OLS(y_noisy, df[linear_fit]) # make the model
results_l = model.fit()

fig = plt.figure(figsize=(4,4))
plt.plot(x_ary, y_noisy, 'o')
plt.plot(x_ary, results_l.predict(), '-o')
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.legend(['data', 'linear fit'], fontsize = 14)
```

Out[8]: <matplotlib.legend.Legend at 0x7ff770dacf50>



Let's look at the fits of other models

Full fit:

$$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5 + c_6 x^6 + c_7 x^7 + c_8 x^8 + c_9 x^9 + c_{10} x^{10} + \epsilon$$

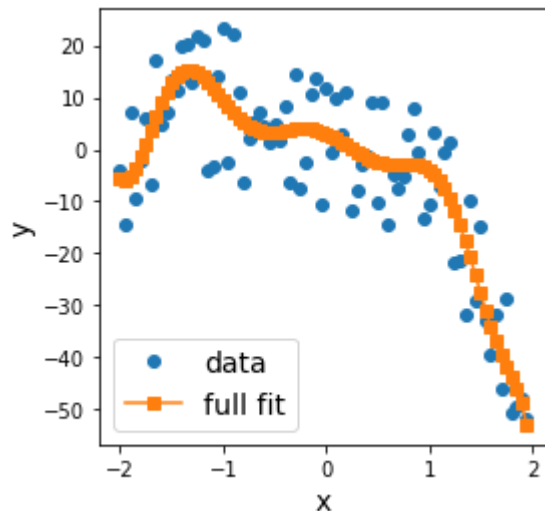
This is overfitting!

```
In [9]: full_fit = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10']

model = sm.OLS(y_noisy, df[full_fit]) # make the model
results_f = model.fit()

fig = plt.figure(figsize=(4,4))
plt.plot(x_ary, y_noisy, 'o')
plt.plot(x_ary, results_f.predict(), '-s')
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.legend(['data', 'full fit'], fontsize = 14)
```

Out[9]: <matplotlib.legend.Legend at 0x7ff7909f3a90>

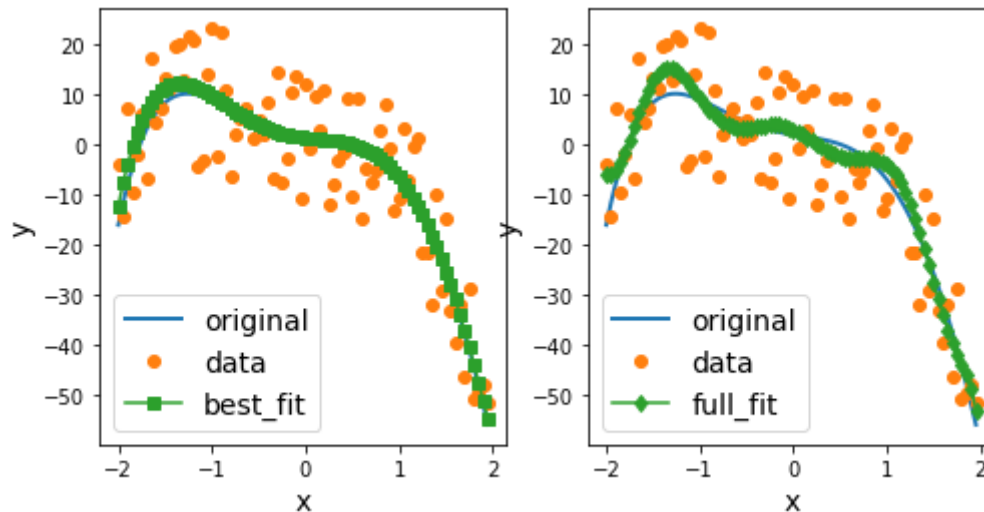


Let's compare the best fit to the full fit

```
In [10]: fig,(ax1, ax2) = plt.subplots(1, 2, figsize = (8,4))

ax1.plot(x_ary, y_ary, '-', lw = 2)
ax1.plot(x_ary, y_noisy, 'o')
ax1.plot(x_ary, results_b.predict(), '-s')
ax2.plot(x_ary, y_ary, '-', lw = 2)
ax2.plot(x_ary, y_noisy, 'o')
ax2.plot(x_ary, results_f.predict(), '-d')
ax1.set_xlabel('x', fontsize = 14)
ax2.set_xlabel('x', fontsize = 14)
ax1.set_ylabel('y', fontsize = 14)
ax2.set_ylabel('y', fontsize = 14)
ax1.legend(['original', 'data', 'best_fit'], fontsize = 14)
ax2.legend(['original', 'data', 'full_fit'], fontsize = 14)
```

Out[10]: <matplotlib.legend.Legend at 0x7ff790a31ed0>



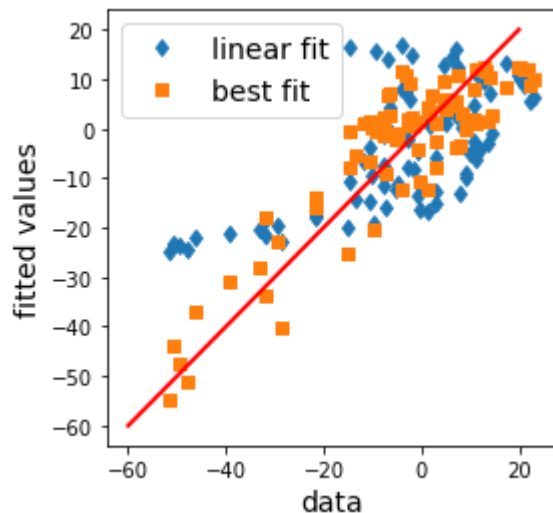
Let's compare the data to the fitted values

We would expect a perfect model to produce a line.

```
In [11]: fig = plt.figure(figsize = (4,4))
plt.plot(y_noisy, results_l.predict(), 'd')
plt.plot(y_noisy, results_b.predict(), 's')

plt.plot([-60,20], [-60,20], '-r', lw=2)
plt.xlabel('data', fontsize = 14)
plt.ylabel('fitted values', fontsize = 14)
plt.legend(['linear fit', 'best fit'], fontsize = 14)
```

Out[11]: <matplotlib.legend.Legend at 0x7ff760ee6d50>



Questions, Comments, Concerns?