Regression Week 3: Polynomial Regression Quiz

In this notebook you will compare different regression models in order to assess which model fits best. We will be using polynomial regression as a means to examine this topic. In particular you will:

- Write a function to take an an array and a degree and return an data frame where each column is the array to a polynomial value up to the total degree.
- Use a plotting tool (e.g. matplotlib) to visualize polynomial regressions
- Use a plotting tool (e.g. matplotlib) to visualize the same polynomial degree on different subsets of the data
- Use a validation set to select a polynomial degree
- Assess the final fit using test data

If you are doing the assignment with IPython Notebook

An IPython Notebook has been provided below to you for this quiz. This notebook contains the instructions, quiz questions and partially-completed code for you to use as well as some cells to test your code.

What you need to download

If you are using GraphLab Create:

- Download the King County House Sales data In SFrame format: kc_house_data.gl.zip
- Download the companion IPython Notebook: week-3-polynomial-regression-assignment-blank.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

If you are not using GraphLab Create:

- Download the King County House Sales data csv file: kc_house_data.csv

NOTE: The following files are different from weeks 1 and 2

- Download the King County House Sales training data csv file: wk3_kc_house_train_data.csv
- Download the King County House Sales validation data csv file: wk3_kc_house_valid_data.csv
- Download the King County House Sales testing data csv file: wk3_kc_house_test_data.csv
- Download the King County House Sales subset 1 data csv file: wk3_kc_house_set_1_data.csv
- Download the King County House Sales subset 2 data csv file: wk3_kc_house_set_2_data.csv
- Download the King County House Sales subset 3 data csv file: wk3_kc_house_set_3_data.csv
- Download the King County House Sales subset 4 data csv file: wk3_kc_house_set_4_data.csv
- **IMPORTANT: use the following types for columns when importing the csv files. Otherwise, they may not be imported correctly: [str, str, float, float, float, float, int, str, int, int, int, int, int, int, str, float, float, float, float, float}. If your tool of choice requires a dictionary of types for importing csv files (e.g. Pandas), use:**

```
dtype_dict = {'bathrooms':float, 'waterfront':int, 'sqft_above':int, 'sqft_living15':float, 'grade':int, 'yr_renovated':int, 'price':float, 'bedrooms':float, 'zipcode':str, 'long':float, 'sqft_lot15':float, 'sqft_living':float, 'floors':str, 'condition':int, 'lat':float, 'date':str, 'sqft_basement':int, 'yr_built':int, 'id':str, 'sqft_lot':int, 'view':int}
```

Useful resources

You may need to install the software tools or use the free Amazon EC2 machine. Instructions for both options are provided in the reading for Module 1.

If you are following the IPython Notebook and/or are new to numpy then you might find the following tutorial helpful: numpy-tutorial.ipynb

If instead you are using other tools to do your homework

You are welcome, however, to write your own code and use any other libraries, like Pandas or R, to help you in the process. If you would like to take this path, follow the instructions below.

1. You're going to write a function that adds powers of a feature to columns of a data frame. For those using SFrames:

Recall that if we have an SArray 'tmp' we can get a new SArray with all the values to the third power with:

```
tmp_cubed = tmp.apply(lambda x: x**3)
```

We can create an empty SFrame with:

```
my_SFrame = graphlab.SFrame()
```

And append the tmp to it with:

```
my_SFrame['power_1'] = tmp
```

Where here 'power_1' will refer to the power our feature was raised to.

2. Write your own function called 'polynomial_sframe' (or otherwise) which accepts an array 'feature' and a maximal 'degree' and returns an data frame (e.g. SFrame) with the first column equal to 'feature' and the remaining columns equal to 'feature' to increasing integer powers up to 'degree'.

e.g. if you're using SFrames, you can complete the following function:

```
def polynomial_sframe(feature, degree):
    # assume that degree >= 1
    # initialize the SFrame:
    poly_sframe = graphlab.SFrame()
    # and set poly_sframe['power_1'] equal to the passed feature
    ...
    # first check if degree > 1
    if degree > 1:
        # then loop over the remaining degrees:
        for power in range(2, degree+1):
            # first we'll give the column a name:
            name = 'power_' + str(power)
            # assign poly_sframe[name] to be feature^power
            ...
    return poly_sframe
```

e.g. if you're using Pandas, you can complete the following function:

```
def polynomial_dataframe(feature, degree): # feature is pandas.Series type
    # assume that degree >= 1
    # initialize the dataframe:
    poly_dataframe = pandas.DataFrame()
    # and set poly_dataframe['power_1'] equal to the passed feature
    ...
    # first check if degree > 1
    if degree > 1:
        # then loop over the remaining degrees:
        for power in range(2, degree+1):
            # first we'll give the column a name:
            name = 'power_' + str(power)
            # assign poly_dataframe[name] to be feature^power; use apply(*)
            ...
    return poly_dataframe
```

3. For the remainder of the assignment we will be working with the house Sales data as in the previous notebooks. Load in the data and also sort the sales SFrame by 'sqft_living'. When we plot the fitted values we want to join them up in a line and this works best if the variable on the X-axis (which will be 'sqft_living') is sorted. For houses with identical square footage, we break the tie by their prices.

e.g. if you're using SFrames

```
sales = graphlab.SFrame('kc_house_data.gl/')
sales = sales.sort(['sqft_living','price'])
```

e.g. if you're using Pandas

```
sales = pandas.read_csv('kc_house_data.csv', dtype=dtype_dict)
sales = sales.sort(['sqft_living','price'])
```

4. Make a 1 degree polynomial SFrame with sales['sqft_living'] as the the feature. Call it 'poly1_data'.

5. Add sales['price'] to poly1_data as this will be our output variable. e.g. if you're using SFrames

```
poly1_data = polynomial_sframe(sales['sqft_living'], 1)
poly1_data['price'] = sales['price']
```

6. Use graphlab.linear_regression.create (or another linear regression library) to compute the regression weights for predicting sales['price'] based on the 1 degree polynomial feature 'sqft_living'. The result should be an intercept and slope. e.g if you're using graphlab create:

```
model1 = graphlab.linear_regression.create(poly1_data, target = 'price', features = ['power_1'], validation_set = None)
```

If you use graphlab.linear_regression.create() to estimate these models please ensure that you set validation_set = None. This way you will get the same answer every time you run the code.

7. Next use the produce a scatter plot of the training data (just square feet vs price) and add the fitted model. e.g. with matplotlib and SFrames:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(poly1_data['power_1'],poly1_data['price'],'.',
poly1_data['power_1'], model1.predict(poly1_data),'-')
```

The resulting plot should look like a cloud of points with a straight line passing through.

8. Now that you have plotted the results using a 1st degree polynomial, try it again using a 2nd degree and 3rd degree polynomial. Look at the fitted lines, do they appear as you would expect?

9. Now try a 15th degree polynomial. Print out the coefficients and look at the resulted fitted line. Do you think this degree is appropriate for these data? If we were to use a different subset of the data do you think we would get pretty much the same curve?

10. If you're using SFrames then create four subsets as follows:

- first split sales into 2 subsets with .random_split(.5) use seed = 0!
- next split these into 2 more subsets (4 total) using random_split(0.5) again set seed = 0!
- you should have 4 subsets of (approximately) equal size, call them set_1, set_2, set_3, and set_4

If you're not using SFrames then please download the provided csv files for each subset.

11. Estimate a 15th degree polynomial on all 4 sets, plot the results and view the coefficients for all four models.

12. Quiz Question: Is the sign (positive or negative) for power_15 the same in all four models?

13. Quiz Question: True/False the plotted fitted lines look the same in all four plots

14. Since the "best" polynomial degree is unknown to us we will use cross validation to select the best degree. If you're using SFrames then create a training, validation and testing subsets as follows:

- First split sales into training_and_validation and testing with sales.random_split(0.9) use seed = 1!
- Next split training_and_validation into training and validation using .random_split(0.5) use seed = 1!

If you're not using SFrames then please download the provided csv files for training, validation and test data.

15. Now for each degree from 1 to 15:

- Build an polynomial data set using training_data['sqft_living'] as the feature and the current degree
- Add training_data['price'] as a column to your polynomial data set
- Learn a model on TRAINING data to predict 'price' based on your polynomial data set at the current degree
- Compute the RSS on VALIDATION for the current model (print or save the RSS)

Hint: in graphlab.linear_regression.create() you can set verbose = False if you want to suppress the interim output of linear_regression.create().

16. Quiz Question: Which degree (1, 2, …, 15) had the lowest RSS on Validation data?

17. Now that you have selected a degree compute the RSS on TEST data for the model with the best degree from the Validation data.

18. Quiz Question: what is the RSS on TEST data for the model with the degree selected from Validation data? (Make sure you got the correct degree from the previous question)