solution

December 19, 2017

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        from scipy import stats
        import math
        import graphlab
        %matplotlib inline
        plt.rcParams["figure.figsize"] = [12,9]


        ---------------------------------------------------------------------------

        RuntimeError                              Traceback (most recent call last)

        RuntimeError: module compiled against API version 0xb but this version of numpy is 0xa
```

1 task

- Load house sales data: kc_house_data.csv.zip
- What is the content, could you read it? do you understand collumns?
- Explore the data for housing
- make scatter plot of selected features
- create simple regression model of sqft_living to price
- evaluate a simple model
- is linear function good enough? try quadratic polynomial

Read in the data. Just to check and know all the columns we visualize the SFrame. To later be able to add a constant part to the fits, we are adding a constant valued column. and for the square fit we create a column with the squared values of 'sqft_living'.

```
In [2]: houses = graphlab.SFrame("../lectures/data/kc_house_data.csv")
        houses['const'] = np.ones(len(houses['id']))
        houses['sqft_living_sq'] = houses['sqft_living']**2
        houses
```

```
This non-commercial license of GraphLab Create for academic use is assigned to niklas.eicker@rwt
```

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: /tmp/graphlab_server_15
```

```
Finished parsing file /home/nikl/DataScience/lectures/data/kc_house_data.csv
```

```
Parsing completed. Parsed 100 lines in 0.257914 secs.
```

```
------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,str,float,int,float,int,int,float,int,int,int,int,int,int,int,int,int,flo
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
```

```
Finished parsing file /home/nikl/DataScience/lectures/data/kc_house_data.csv
```

```
Parsing completed. Parsed 21613 lines in 0.263838 secs.
```

Out[2]:
```
Columns:
	id	int
	date	str
	price	float
	bedrooms	int
	bathrooms	float
	sqft_living	int
	sqft_lot	int
	floors	float
	waterfront	int
	view	int
	condition	int
	grade	int
	sqft_above	int
	sqft_basement	int
	yr_built	int
	yr_renovated	int
	zipcode	int
	lat	float
	long	float
	sqft_living15	int
	sqft_lot15	int
	const	float
	sqft_living_sq	float
```

```
Rows: 21613

Data:
+------------+-----------------+-----------+----------+-----------+-------------+
|     id     |       date      |   price   | bedrooms | bathrooms | sqft_living |
+------------+-----------------+-----------+----------+-----------+-------------+
| 7129300520 | 20141013T000000 |  221900.0 |    3     |    1.0    |     1180    |
| 6414100192 | 20141209T000000 |  538000.0 |    3     |    2.25   |     2570    |
| 5631500400 | 20150225T000000 |  180000.0 |    2     |    1.0    |     770     |
| 2487200875 | 20141209T000000 |  604000.0 |    4     |    3.0    |     1960    |
| 1954400510 | 20150218T000000 |  510000.0 |    3     |    2.0    |     1680    |
| 7237550310 | 20140512T000000 | 1225000.0 |    4     |    4.5    |     5420    |
| 1321400060 | 20140627T000000 |  257500.0 |    3     |    2.25   |     1715    |
| 2008000270 | 20150115T000000 |  291850.0 |    3     |    1.5    |     1060    |
| 2414600126 | 20150415T000000 |  229500.0 |    3     |    1.0    |     1780    |
| 3793500160 | 20150312T000000 |  323000.0 |    3     |    2.5    |     1890    |
+------------+-----------------+-----------+----------+-----------+-------------+
+----------+--------+------------+------+-----------+-------+------------+--------------
| sqft_lot | floors | waterfront | view | condition | grade | sqft_above | sqft_basement
+----------+--------+------------+------+-----------+-------+------------+--------------
|   5650   |  1.0   |     0      |  0   |     3     |   7   |    1180    |       0
|   7242   |  2.0   |     0      |  0   |     3     |   7   |    2170    |      400
|  10000   |  1.0   |     0      |  0   |     3     |   6   |    770     |       0
|   5000   |  1.0   |     0      |  0   |     5     |   7   |    1050    |      910
|   8080   |  1.0   |     0      |  0   |     3     |   8   |    1680    |       0
|  101930  |  1.0   |     0      |  0   |     3     |   11  |    3890    |      1530
|   6819   |  2.0   |     0      |  0   |     3     |   7   |    1715    |       0
|   9711   |  1.0   |     0      |  0   |     3     |   7   |    1060    |       0
|   7470   |  1.0   |     0      |  0   |     3     |   7   |    1050    |      730
|   6560   |  2.0   |     0      |  0   |     3     |   7   |    1890    |       0
+----------+--------+------------+------+-----------+-------+------------+--------------
+----------+--------------+---------+---------+----------+---------------+-----+
| yr_built | yr_renovated | zipcode |   lat   |   long   | sqft_living15 | ... |
+----------+--------------+---------+---------+----------+---------------+-----+
|   1955   |      0       |  98178  | 47.5112 | -122.257 |      1340     | ... |
|   1951   |     1991     |  98125  |  47.721 | -122.319 |      1690     | ... |
|   1933   |      0       |  98028  | 47.7379 | -122.233 |      2720     | ... |
|   1965   |      0       |  98136  | 47.5208 | -122.393 |      1360     | ... |
|   1987   |      0       |  98074  | 47.6168 | -122.045 |      1800     | ... |
|   2001   |      0       |  98053  | 47.6561 | -122.005 |      4760     | ... |
|   1995   |      0       |  98003  | 47.3097 | -122.327 |      2238     | ... |
|   1963   |      0       |  98198  | 47.4095 | -122.315 |      1650     | ... |
|   1960   |      0       |  98146  | 47.5123 | -122.337 |      1780     | ... |
|   2003   |      0       |  98038  | 47.3684 | -122.031 |      2390     | ... |
+----------+--------------+---------+---------+----------+---------------+-----+
[21613 rows x 23 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

Create a scatter plot of house price and sqft_living.

```
In [3]: graphlab.canvas.set_target('ipynb')
        houses.show(view="Scatter Plot",x="sqft_living",y="price")
```

We will do the linear and quadratic fit on one feature (sqft_living) with graphlab.

```
In [4]: sqft_model_lin = graphlab.linear_regression.create(houses, validation_set=None, target='
```

```
Linear regression:

--------------------------------------------------------

Number of examples          : 21613

Number of features          : 1

Number of unpacked features : 1

Number of coefficients    : 2

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+--------------------+---------------+

| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |

+-----------+----------+--------------+--------------------+---------------+

| 1         | 2        | 1.063856     | 4362074.683616     | 261440.790302 |

+-----------+----------+--------------+--------------------+---------------+

SUCCESS: Optimal solution found.
```

Now a model that takes the squared value of 'sqft_living' into account.

```
In [5]: sqft_model_quad = graphlab.linear_regression.create(houses, validation_set=None, target=
```

```
Linear regression:

--------------------------------------------------------

Number of examples          : 21613

Number of features          : 2

Number of unpacked features : 2

Number of coefficients    : 3

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+--------------------+---------------+

| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |

+-----------+----------+--------------+--------------------+---------------+

| 1         | 2        | 0.075018     | 5913021.143248     | 250948.367620 |

+-----------+----------+--------------+--------------------+---------------+

SUCCESS: Optimal solution found.
```

```
In [6]: # RMS of the lin fit
        print sqft_model_lin.evaluate(houses)
```

```
{'max_error': 4362074.683615588, 'rmse': 261440.79030169296}
```

```
In [7]: # RMS of the quadratic fit
        print sqft_model_quad.evaluate(houses)
```

```
{'max_error': 5913021.143247618, 'rmse': 250948.36761971583}
```

RMS of the quadratic method is a little bit better then the linear one, but still very big compared to the values.

2 task

- Split your data into training sample and test sample
- what is trainign error and testing error of your model?
- predict the house price for a given sqft_living
- predict the sqft_living for a given price of the house

```
In [8]: training , testing  = houses.random_split(.8,seed=0)
```

recalculte our linear model on the test data

```
In [9]: sqft_model_lin = graphlab.linear_regression.create(training, validation_set=None, target
```

```
Linear regression:


--------------------------------------------------------


Number of examples          : 17384


Number of features          : 1


Number of unpacked features : 1


Number of coefficients    : 2


Starting Newton Method
```

```
--------------------------------------------------------

+-----------+----------+--------------+--------------------+---------------+

| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |

+-----------+----------+--------------+--------------------+---------------+

| 1         | 2        | 0.045954     | 4349521.915863     | 262943.613519 |

+-----------+----------+--------------+--------------------+---------------+

SUCCESS: Optimal solution found.
```

Now compare the rms on training data with the rms on test data

```
In [10]: print "training: " + str(sqft_model_lin.evaluate(training)['rmse'])
         print "testing: " + str(sqft_model_lin.evaluate(testing)['rmse'])
```

```
training: 262943.613519
testing: 255191.027487
```

The errors are comparable...

2.1 Methods to predict prices and sqft_living with the fit parameters.

roots helps solving fot the sqft_living values

```
In [11]: def get_house_price(sqft_living):
             return (sqft_model_lin.get('coefficients')['value'][0] + sqft_model_lin.get('coeffi

In [12]: def get_house_sqft(price):
             coeff = [sqft_model_lin.get('coefficients')['value'][1], (sqft_model_lin.get('coeff
             return np.roots(coeff)[0]

In [13]: print "predict house sqft_living for prices:"
         print "    500.000 :   " + str(get_house_sqft(500000))
         print "    1.000.000 : " + str(get_house_sqft(1000000))
         print "    1.500.000 : " + str(get_house_sqft(1500000))
         print "    2.500.000 : " + str(get_house_sqft(2500000))
```

```
predict house sqft_living for prices:
    500.000 :   1940.41067055
    1.000.000 : 3713.72536216
    1.500.000 : 5487.04005378
    2.500.000 : 9033.66943701
```

```
In [14]: print "predict house prices for given sqft:"
         print "    1000 : "+ str(get_house_price(1000))
         print "    2000 : "+ str(get_house_price(2000))
         print "    3000 : "+ str(get_house_price(3000))
         print "    4000 : "+ str(get_house_price(4000))
```

```
predict house prices for given sqft:
    1000 : 234843.82806
    2000 : 516801.679289
    3000 : 798759.530518
    4000 : 1080717.38175
```

3 task

- add more feaures
- is the model better now?
- maybe using range of data would work better?

```
In [15]: sqft_model = graphlab.linear_regression.create(training, target='price',features=['sqft
         sqft_model.evaluate(training)
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
          You can set ``validation_set=None`` to disable validation tracking.



Linear regression:


--------------------------------------------------------


Number of examples          : 16450


Number of features          : 4


Number of unpacked features : 4
```

```
Number of coefficients    : 5

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+--------------------+---------------------+------------

| Iteration | Passes   | Elapsed Time | Training-max_error | Validation-max_error | Training-rms

+-----------+----------+--------------+--------------------+---------------------+------------

| 1         | 2        | 0.106160     | 4502064.109799     | 2987342.356958       | 231173.72689

+-----------+----------+--------------+--------------------+---------------------+------------

SUCCESS: Optimal solution found.
```

```
Out[15]: {'max_error': 4502064.109799153, 'rmse': 232189.4072274167}
```

Yes more featerues lowered our error, but it is still quite high.

```
In [16]: range_houses = houses[(houses['sqft_living'] <= 5000) & (houses['price'] <= 2000000)]
         range_training , range_testing  = range_houses.random_split(.8,seed=0)
         sqft_model_selected = graphlab.linear_regression.create(range_training, target='price',
         sqft_model_selected.evaluate(range_training)
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
          You can set ``validation_set=None`` to disable validation tracking.



Linear regression:

--------------------------------------------------------
```

```
Number of examples          : 16302

Number of features          : 4

Number of unpacked features : 4

Number of coefficients    : 5

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+--------------------+----------------------+-------------

| Iteration | Passes   | Elapsed Time | Training-max_error | Validation-max_error | Training-rms

+-----------+----------+--------------+--------------------+----------------------+-------------

| 1         | 2        | 0.104579     | 1330911.700288     | 983736.696828        | 179574.56947

+-----------+----------+--------------+--------------------+----------------------+-------------

SUCCESS: Optimal solution found.
```

```
Out[16]: {'max_error': 1330911.7002882427, 'rmse': 179424.43561122433}
```

on our limited range for prices < 2mil and sqft_living < 5000 the model works much better! On the other hand the points are now also closer together anyways.

4 task

- predict house price for a house id = 5309101299 (does not exists! using 2008000270 instead)
- what is this house like?
- predict house price for a house id = 1925069082

```
In [17]: houses[houses['id'] == 2008000270]
```

```
Out[17]: Columns:
                id        int
                date        str
                price        float
                bedrooms        int
                bathrooms        float
                sqft_living        int
                sqft_lot        int
                floors        float
                waterfront        int
                view        int
                condition        int
                grade        int
                sqft_above        int
                sqft_basement        int
                yr_built        int
                yr_renovated        int
                zipcode        int
                lat        float
                long        float
                sqft_living15        int
                sqft_lot15        int
                const        float
                sqft_living_sq        float

        Rows: Unknown

        Data:
        +------------+-----------------+----------+----------+-----------+-------------+
        |     id     |       date      |  price   | bedrooms | bathrooms | sqft_living |
        +------------+-----------------+----------+----------+-----------+-------------+
        | 2008000270 | 20150115T000000 | 291850.0 |    3     |    1.5    |     1060    |
        +------------+-----------------+----------+----------+-----------+-------------+
        +----------+--------+------------+------+-----------+-------+------------+-------------
        | sqft_lot | floors | waterfront | view | condition | grade | sqft_above | sqft_basemen
        +----------+--------+------------+------+-----------+-------+------------+-------------
        |   9711   |  1.0   |     0      |  0   |     3     |   7   |    1060    |       0
        +----------+--------+------------+------+-----------+-------+------------+-------------
        +----------+--------------+---------+---------+----------+---------------+-----+
        | yr_built | yr_renovated | zipcode |   lat   |   long   | sqft_living15 | ... |
```

```
+----------+-------------+---------+---------+----------+---------------+-----+
|   1963   |      0      |  98198  | 47.4095 | -122.315 |      1650     | ... |
+----------+-------------+---------+---------+----------+---------------+-----+
[? rows x 23 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.
```

4.1 house nr 2008000270

It is a rather small house, with 1060 sqft living and only one floor. The price is one of the lowest of the dataset with 291850 dollar. There are 3 bedrooms and 1.5(?!) bathrooms. This house was build in 1963.

```
In [18]: print "predict price of house nr. 1925069082"
         print "         predicted: "+ str(get_house_price(houses[houses['id'] == 1925069082]['s
         print "              real: "+ str(houses[houses['id'] == 1925069082]['price'][0])
```

```
predict price of house nr. 1925069082
         predicted: 1261170.40653
              real: 2200000.0
```

The prediciton for the price is only ~6% off