Regression Week 4: Ridge Regression (interpretation)

In this notebook, we will run ridge regression multiple times with different L2 penalties to see which one produces the best fit. We will revisit the example of polynomial regression as a means to see the effect of L2 regularization. In particular, we will:

- Use a pre-built implementation of regression (GraphLab Create) to run polynomial regression
- Use matplotlib to visualize polynomial regressions
- Use a pre-built implementation of regression (GraphLab Create) to run polynomial regression, this time with L2 penalty
- Use matplotlib to visualize polynomial regressions under L2 regularization
- Choose best L2 penalty using cross-validation.
- Assess the final fit using test data.

We will continue to use the House data from previous notebooks. (In the next programming assignment for this module, you will implement your own ridge regression learning algorithm using gradient descent.)

Fire up graphlab create

In [1]:

```
import graphlab
```

Polynomial regression, revisited

We build on the material from Week 3, where we wrote the function to produce an SFrame with columns containing the powers of a given input. Copy and paste the function `polynomial_sframe` from Week 3:

In [4]:

```
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
    poly_sframe['power_1'] = feature
    # first check if degree > 1
    if degree > 1:
        # then loop over the remaining degrees:
        # range usually starts at 0 and stops at the endpoint-1. We want it to start
        for power in range(2, degree+1):
            # first we'll give the column a name:
            name = 'power_' + str(power)
            # then assign poly_sframe[name] to the appropriate power of feature
            poly_sframe[name] = feature.apply(lambda x: x**power)
    return poly_sframe
```

Let's use matplotlib to visualize what a polynomial regression looks like on the house data.

In [5]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [11]:

```
sales = graphlab.SFrame('C:/courses/Coursera/Current/ML Regression/Week4/kc_house_dat
```

As in Week 3, we will use the sqft_living variable. For plotting purposes (connecting the dots), you'll need to sort by the values of sqft_living. For houses with identical square footage, we break the tie by their prices.

In [12]:

```
sales = sales.sort(['sqft_living','price'])
```

Let us revisit the 15th-order polynomial model using the 'sqft_living' input. Generate polynomial features up to degree 15 using `polynomial_sframe()` and fit a model with these features. When fitting the model, use an L2 penalty of `1e-5`:

In [13]:

```
l2_small_penalty = 1e-5
```

Note: When we have so many features and so few data points, the solution can become highly numerically unstable, which can sometimes lead to strange unpredictable results. Thus, rather than using no regularization, we will introduce a tiny amount of regularization (`l2_penalty=1e-5`) to make the solution numerically stable. (In lecture, we discussed the fact that regularization can also help with numerical stability, and here we are seeing a practical example.)

With the L2 penalty specified above, fit the model and print out the learned weights.

Hint: make sure to add 'price' column to the new SFrame before calling `graphlab.linear_regression.create()`. Also, make sure GraphLab Create doesn't create its own validation set by using the option `validation_set=None` in this call.

In [16]:

```
poly15_data = polynomial_sframe(sales['sqft_living'], 15)
my_features = poly15_data.column_names() # get the name of the features
poly15_data['price'] = sales['price'] # add price to the data since it's the target
model15 = graphlab.linear_regression.create(poly15_data, target = 'price', features =
model15.get("coefficients").print_rows(num_rows=16)
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 21613
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.031251     | 2662555.738027     |
245656.462165 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   167924.857726    |
|   power_1   |  None |   103.090951289    |
|   power_2   |  None |   0.13460455096    |
|   power_3   |  None | -0.000129071363752 |
|   power_4   |  None | 5.18928955754e-08  |
|   power_5   |  None | -7.77169299595e-12 |
|   power_6   |  None | 1.71144842837e-16  |
|   power_7   |  None | 4.51177958161e-20  |
|   power_8   |  None | -4.78839816249e-25 |
|   power_9   |  None | -2.33343499941e-28 |
|   power_10  |  None | -7.29022428496e-33 |
|   power_11  |  None | 7.22829146954e-37  |
|   power_12  |  None |  6.9047076722e-41  |
|   power_13  |  None | -3.65843768148e-46 |
|   power_14  |  None | -3.79575941941e-49 |
|   power_15  |  None |  1.1372314991e-53  |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

QUIZ QUESTION: What's the learned value for the coefficient of feature `power_1`?

Observe overfitting

Recall from Week 3 that the polynomial fit of degree 15 changed wildly whenever the data changed. In particular, when we split the sales data into four subsets and fit the model of degree 15, the result came out to be very different for each subset. The model had a *high variance*. We will see in a moment that ridge regression reduces such variance. But first, we must reproduce the experiment we did in Week 3.

First, split the data into split the sales data into four subsets of roughly equal size and call them `set_1`, `set_2`, `set_3`, and `set_4`. Use `.random_split` function and make sure you set `seed=0`.

In [17]:

```
(semi_split1, semi_split2) = sales.random_split(.5,seed=0)
(set_1, set_2) = semi_split1.random_split(0.5, seed=0)
(set_3, set_4) = semi_split2.random_split(0.5, seed=0)
```

Next, fit a 15th degree polynomial on `set_1`, `set_2`, `set_3`, and `set_4`, using 'sqft_living' to predict prices. Print the weights and make a plot of the resulting model.

Hint: When calling `graphlab.linear_regression.create()`, use the same L2 penalty as before (i.e. `l2_small_penalty`). Also, make sure GraphLab Create doesn't create its own validation set by using the option `validation_set = None` in this call.

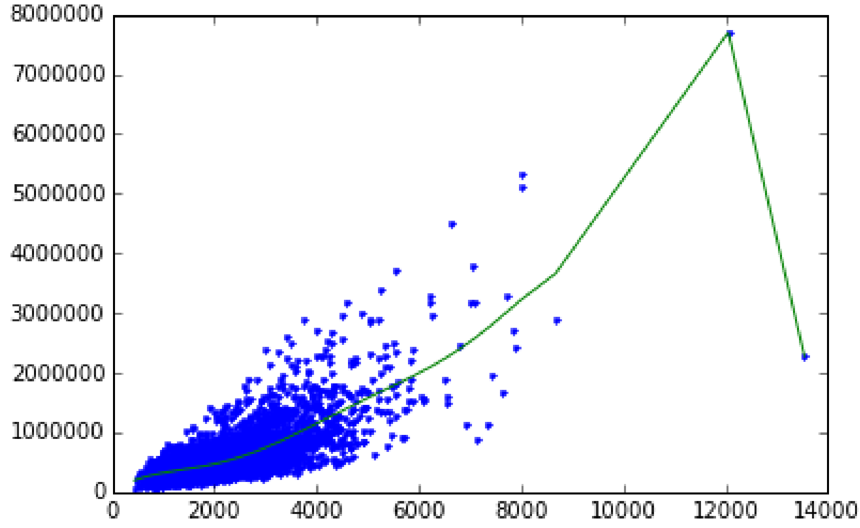
In [18]:

```
poly_data = polynomial_sframe(set_1['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_1['price'] # add price to the data since it's the target
model_1 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_1.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_1.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5404
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.015624     | 2191984.900939     |
248699.117254 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   9306.46397814    |
|   power_1   |  None |    585.86581347    |
|   power_2   |  None |  -0.397305884724   |
|   power_3   |  None | 0.000141470894825  |
|   power_4   |  None | -1.52945974394e-08 |
|   power_5   |  None | -3.79756526062e-13 |
|   power_6   |  None |  5.9748184732e-17  |
|   power_7   |  None | 1.06888505979e-20  |
|   power_8   |  None | 1.59344052349e-25  |
|   power_9   |  None | -6.9283495515e-29  |
|   power_10  |  None | -6.83813368045e-33 |
|   power_11  |  None | -1.62686203908e-37 |
|   power_12  |  None | 2.85118784287e-41  |
|   power_13  |  None | 3.79998146917e-45  |
|   power_14  |  None | 1.52652617058e-49  |
|   power_15  |  None | -2.33807310252e-53 |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x1bbf7a58>,
 <matplotlib.lines.Line2D at 0x1bbf7c88>]
```


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000
12000
14000

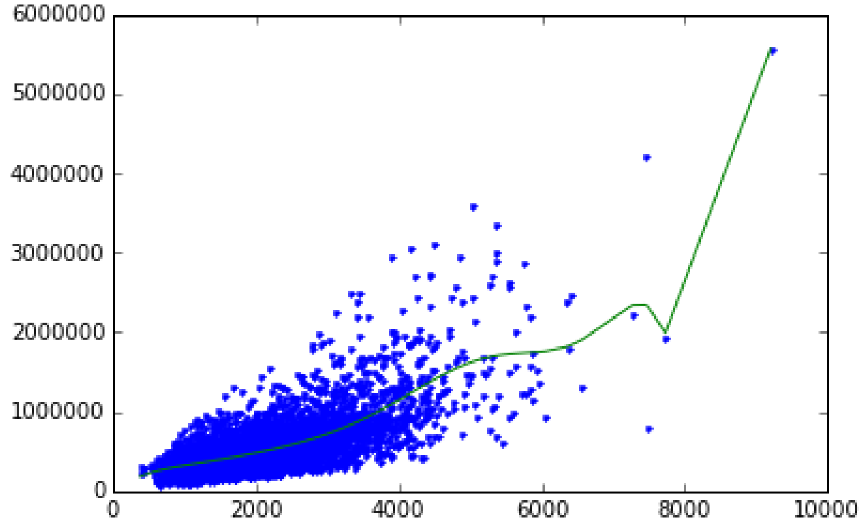
In [19]:

```
poly_data = polynomial_sframe(set_2['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_2['price'] # add price to the data since it's the target
model_2 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_2.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_2.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5398
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.000000     | 1975178.190682     |
234533.610645 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   -25115.9059869   |
|   power_1   |  None |   783.493802508    |
|   power_2   |  None |  -0.767759300173   |
|   power_3   |  None | 0.000438766361934  |
|   power_4   |  None | -1.15169161152e-07 |
|   power_5   |  None | 6.84281148707e-12  |
|   power_6   |  None |  2.5119522464e-15  |
|   power_7   |  None | -2.06440624344e-19 |
|   power_8   |  None | -4.59673058828e-23 |
|   power_9   |  None | -2.71277342492e-29 |
|   power_10  |  None | 6.21818505057e-31  |
|   power_11  |  None | 6.51741311744e-35  |
|   power_12  |  None | -9.41316275987e-40 |
|   power_13  |  None | -1.02421363129e-42 |
|   power_14  |  None | -1.00391099753e-46 |
|   power_15  |  None | 1.30113366848e-50  |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x1be89400>,
 <matplotlib.lines.Line2D at 0x1be895f8>]
```


6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000

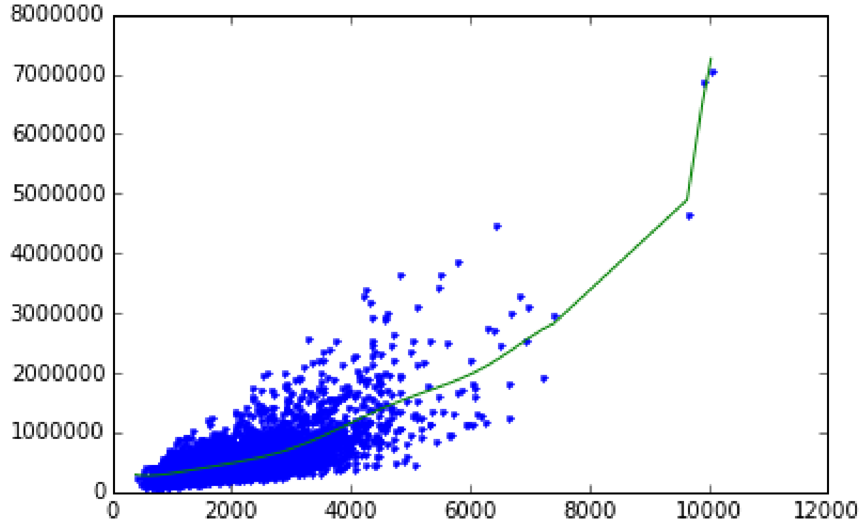
In [20]:

```
poly_data = polynomial_sframe(set_3['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_3['price'] # add price to the data since it's the target
model_3 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_3.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_3.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5409
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.000000     | 2283722.683933     |
251097.728054 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   462426.565731    |
|   power_1   |  None |   -759.251842854   |
|   power_2   |  None |    1.0286700473    |
|   power_3   |  None | -0.000528264527386 |
|   power_4   |  None | 1.15422908385e-07  |
|   power_5   |  None | -2.26095948062e-12 |
|   power_6   |  None | -2.08214287571e-15 |
|   power_7   |  None | 4.08770475709e-20  |
|   power_8   |  None |  2.570791329e-23   |
|   power_9   |  None | 1.24311265196e-27  |
|   power_10  |  None | -1.72025834939e-31 |
|   power_11  |  None | -2.96761071315e-35 |
|   power_12  |  None | -1.06574890499e-39 |
|   power_13  |  None | 2.42635621458e-43  |
|   power_14  |  None |  3.5559876473e-47  |
|   power_15  |  None | -2.85777468723e-51 |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

Out[20]:

```
[<matplotlib.lines.Line2D at 0x1bf1b198>,
 <matplotlib.lines.Line2D at 0x1bf1b390>]
```


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000
12000

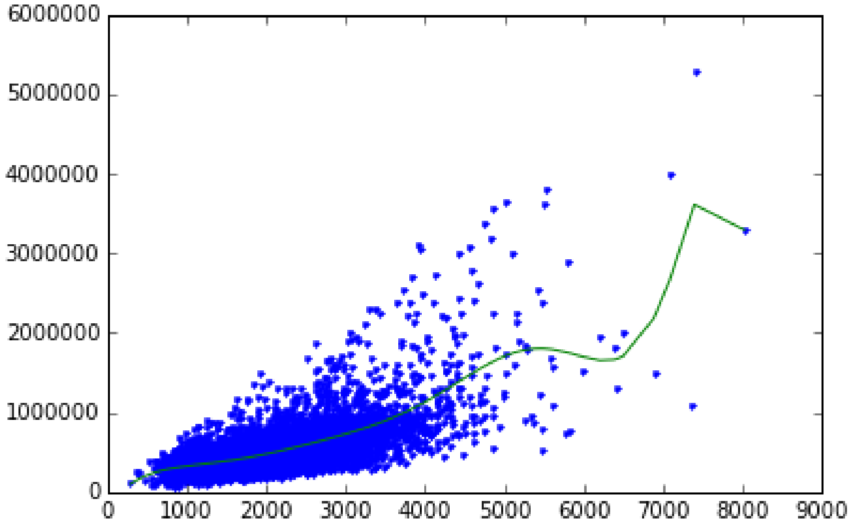
In [21]:

```
poly_data = polynomial_sframe(set_4['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_4['price'] # add price to the data since it's the target
model_4 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_4.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_4.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5402
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.009008     | 2378292.371612     |
244341.293203 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   -170240.034791   |
|   power_1   |  None |   1247.59035088    |
|   power_2   |  None |   -1.2246091264    |
|   power_3   |  None | 0.000555254626787  |
|   power_4   |  None | -6.38262361929e-08 |
|   power_5   |  None | -2.20215996475e-11 |
|   power_6   |  None | 4.81834697594e-15  |
|   power_7   |  None |  4.2146163248e-19  |
|   power_8   |  None | -7.99880749051e-23 |
|   power_9   |  None | -1.32365907706e-26 |
|   power_10  |  None | 1.60197797139e-31  |
|   power_11  |  None | 2.39904337326e-34  |
|   power_12  |  None | 2.33354505765e-38  |
|   power_13  |  None | -1.79874055895e-42 |
|   power_14  |  None | -6.02862682894e-46 |
|   power_15  |  None | 4.39472672531e-50  |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

Out[21]:

```
[<matplotlib.lines.Line2D at 0x1c1aed68>,
 <matplotlib.lines.Line2D at 0x1c1aef60>]
```



The four curves should differ from one another a lot, as should the coefficients you learned.

QUIZ QUESTION: For the models learned in each of these training sets, what are the smallest and largest values you learned for the coefficient of feature `power_1`? (For the purpose of answering this question, negative numbers are considered "smaller" than positive numbers. So -5 is smaller than -3, and -3 is smaller than 5 and so forth.)

Ridge regression comes to rescue

Generally, whenever we see weights change so much in response to change in data, we believe the variance of our estimate to be large. Ridge regression aims to address this issue by penalizing "large" weights. (Weights of `model15` looked quite small, but they are not that small because 'sqft_living' input is in the order of thousands.)

With the argument `l2_penalty=1e5`, fit a 15th-order polynomial model on `set_1`, `set_2`, `set_3`, and `set_4`. Other than the change in the `l2_penalty` parameter, the code should be the same as the experiment above. Also, make sure GraphLab Create doesn't create its own validation set by using the option `validation_set = None` in this call.

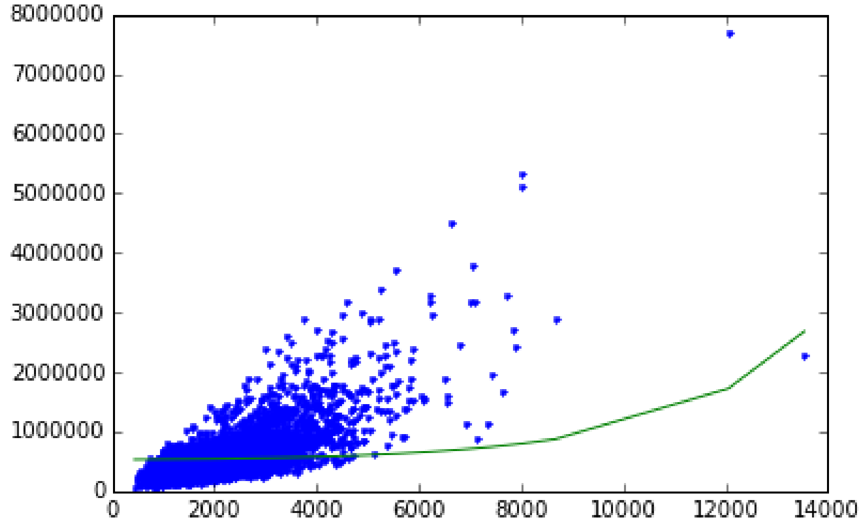
In [23]:

```
poly_data = polynomial_sframe(set_1['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_1['price'] # add price to the data since it's the target
model_1 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_1.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_1.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5404
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.006457     | 5978778.434729     |
374261.720860 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+-------------------+
|     name    | index |       value       |
+-------------+-------+-------------------+
| (intercept) |  None |   530317.024516   |
|   power_1   |  None |   2.58738875673   |
|   power_2   |  None |  0.00127414400592 |
|   power_3   |  None | 1.74934226932e-07 |
|   power_4   |  None | 1.06022119097e-11 |
|   power_5   |  None | 5.42247604482e-16 |
|   power_6   |  None | 2.89563828343e-20 |
|   power_7   |  None | 1.65000666351e-24 |
|   power_8   |  None | 9.86081528409e-29 |
|   power_9   |  None | 6.06589348254e-33 |
|   power_10  |  None |  3.7891786887e-37 |
|   power_11  |  None | 2.38223121312e-41 |
|   power_12  |  None | 1.49847969215e-45 |
|   power_13  |  None | 9.39161190285e-50 |
|   power_14  |  None | 5.84523161981e-54 |
|   power_15  |  None | 3.60120207203e-58 |
+-------------+-------+-------------------+
[16 rows x 3 columns]
```

Out[23]:

```
[<matplotlib.lines.Line2D at 0x1c5fb0f0>,
 <matplotlib.lines.Line2D at 0x1c5fb2e8>]
```


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000
12000
14000

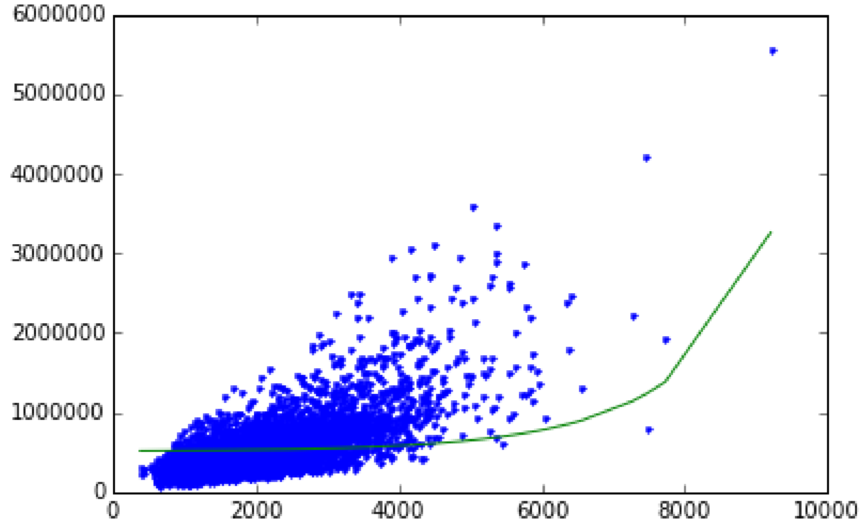
In [24]:

```
poly_data = polynomial_sframe(set_2['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_2['price'] # add price to the data since it's the target
model_2 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_2.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_2.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5398
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.000000     | 2984894.541944     |
323238.809634 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+-------------------+
|     name    | index |       value       |
+-------------+-------+-------------------+
| (intercept) |  None |   519216.897383   |
|   power_1   |  None |   2.04470474182   |
|   power_2   |  None |  0.0011314362684  |
|   power_3   |  None | 2.93074277549e-07 |
|   power_4   |  None | 4.43540598453e-11 |
|   power_5   |  None | 4.80849112204e-15 |
|   power_6   |  None | 4.53091707826e-19 |
|   power_7   |  None | 4.16042910575e-23 |
|   power_8   |  None | 3.90094635128e-27 |
|   power_9   |  None |  3.7773187602e-31 |
|   power_10  |  None | 3.76650326842e-35 |
|   power_11  |  None | 3.84228094754e-39 |
|   power_12  |  None | 3.98520828414e-43 |
|   power_13  |  None | 4.18272762394e-47 |
|   power_14  |  None | 4.42738332878e-51 |
|   power_15  |  None | 4.71518245412e-55 |
+-------------+-------+-------------------+
[16 rows x 3 columns]
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x1c6960b8>,
 <matplotlib.lines.Line2D at 0x1c6962b0>]
```


6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000

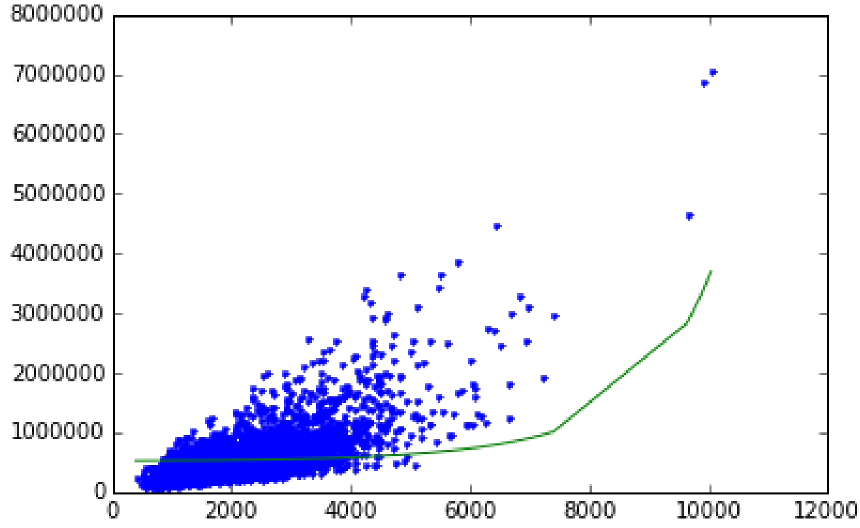
In [25]:

```
poly_data = polynomial_sframe(set_3['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_3['price'] # add price to the data since it's the target
model_3 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_3.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_3.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5409
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.000000     | 3695342.767093     |
350033.521294 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+-------------------+
|     name    | index |       value       |
+-------------+-------+-------------------+
| (intercept) |  None |   522911.518048   |
|   power_1   |  None |   2.26890421877   |
|   power_2   |  None |  0.00125905041842 |
|   power_3   |  None | 2.77552918155e-07 |
|   power_4   |  None |  3.2093309779e-11 |
|   power_5   |  None | 2.87573572364e-15 |
|   power_6   |  None | 2.50076112671e-19 |
|   power_7   |  None | 2.24685265906e-23 |
|   power_8   |  None | 2.09349983135e-27 |
|   power_9   |  None | 2.00435383296e-31 |
|   power_10  |  None | 1.95410800249e-35 |
|   power_11  |  None | 1.92734119456e-39 |
|   power_12  |  None | 1.91483699013e-43 |
|   power_13  |  None | 1.91102277046e-47 |
|   power_14  |  None | 1.91246242302e-51 |
|   power_15  |  None | 1.91699558035e-55 |
+-------------+-------+-------------------+
[16 rows x 3 columns]
```

Out[25]:

```
[<matplotlib.lines.Line2D at 0x1ed77f28>,
 <matplotlib.lines.Line2D at 0x1ed86160>]
```


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000
12000

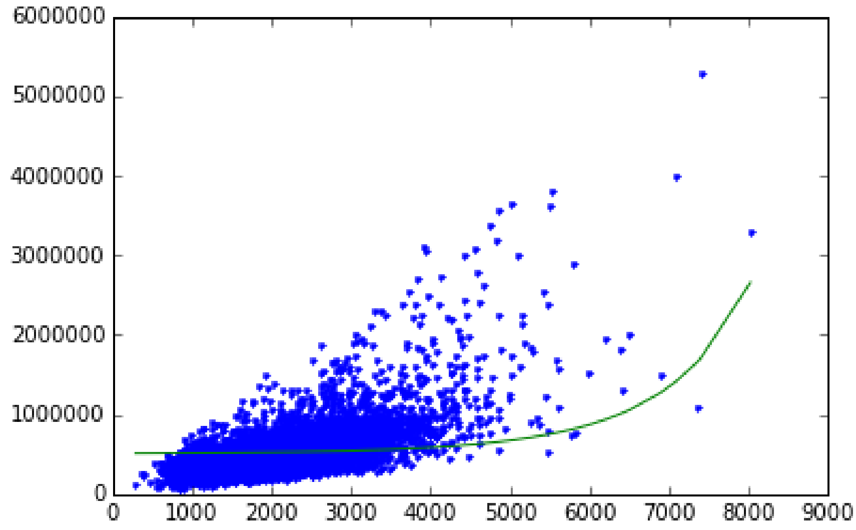
In [26]:

```
poly_data = polynomial_sframe(set_4['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = set_4['price'] # add price to the data since it's the target
model_4 = graphlab.linear_regression.create(poly_data, target = 'price', features = m
model_4.get("coefficients").print_rows(num_rows=16)
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model_4.p
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 5402
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.015625     | 3601895.280124     |
323111.582889 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+-------------------+
|     name    | index |       value       |
+-------------+-------+-------------------+
| (intercept) |  None |   513667.087087   |
|   power_1   |  None |   1.91040938244   |
|   power_2   |  None |  0.00110058029175 |
|   power_3   |  None | 3.12753987879e-07 |
|   power_4   |  None | 5.50067886825e-11 |
|   power_5   |  None | 7.20467557825e-15 |
|   power_6   |  None | 8.24977249384e-19 |
|   power_7   |  None | 9.06503223498e-23 |
|   power_8   |  None | 9.95683160453e-27 |
|   power_9   |  None | 1.10838127982e-30 |
|   power_10  |  None | 1.25315224143e-34 |
|   power_11  |  None | 1.43600781402e-38 |
|   power_12  |  None |  1.662699678e-42  |
|   power_13  |  None |  1.9398172453e-46 |
|   power_14  |  None |  2.2754148577e-50 |
|   power_15  |  None | 2.67948784897e-54 |
+-------------+-------+-------------------+
[16 rows x 3 columns]
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x1f04f748>,
 <matplotlib.lines.Line2D at 0x1f04f940>]
```



These curves should vary a lot less, now that you applied a high degree of regularization.

QUIZ QUESTION: For the models learned with the high level of regularization in each of these training sets, what are the smallest and largest values you learned for the coefficient of feature `power_1`? (For the purpose of answering this question, negative numbers are considered "smaller" than positive numbers. So -5 is smaller than -3, and -3 is smaller than 5 and so forth.)

Selecting an L2 penalty via cross-validation

Just like the polynomial degree, the L2 penalty is a "magic" parameter we need to select. We could use the validation set approach as we did in the last module, but that approach has a major disadvantage: it leaves fewer observations available for training. **Cross-validation** seeks to overcome this issue by using all of the training set in a smart way.

We will implement a kind of cross-validation called **k-fold cross-validation**. The method gets its name because it involves dividing the training set into k segments of roughly equal size. Similar to the validation set method, we measure the validation error with one of the segments designated as the validation set. The major difference is that we repeat the process k times as follows:

Set aside segment 0 as the validation set, and fit a model on rest of data, and evalutate it on this validation set
Set aside segment 1 as the validation set, and fit a model on rest of data, and evalutate it on this validation set
...
Set aside segment k-1 as the validation set, and fit a model on rest of data, and evalutate it on this validation set

After this process, we compute the average of the k validation errors, and use it as an estimate of the generalization error. Notice that all observations are used for both training and validation, as we iterate over segments of data.

To estimate the generalization error well, it is crucial to shuffle the training data before dividing them into segments. GraphLab Create has a utility function for shuffling a given SFrame. We reserve 10% of the data as the test set and shuffle the remainder. (Make sure to use `seed=1` to get consistent answer.)

In [27]:

```
(train_valid, test) = sales.random_split(.9, seed=1)
train_valid_shuffled = graphlab.toolkits.cross_validation.shuffle(train_valid, random
```

Once the data is shuffled, we divide it into equal segments. Each segment should receive `n/k` elements, where `n` is the number of observations in the training set and `k` is the number of segments. Since the segment 0 starts at index 0 and contains `n/k` elements, it ends at index `(n/k)-1`. The segment 1 starts where the segment 0 left off, at index `(n/k)`. With `n/k` elements, the segment 1 ends at index `(n*2/k)-1`. Continuing in this fashion, we deduce that the segment `i` starts at index `(n*i/k)` and ends at `(n*(i+1)/k)-1`.

With this pattern in mind, we write a short loop that prints the starting and ending indices of each segment, just to make sure you are getting the splits right.

In [28]:

```
n = len(train_valid_shuffled)
k = 10 # 10-fold cross-validation

for i in xrange(k):
    start = (n*i)/k
    end = (n*(i+1))/k-1
    print i, (start, end)
```

```
0 (0, 1938)
1 (1939, 3878)
2 (3879, 5817)
3 (5818, 7757)
4 (7758, 9697)
5 (9698, 11636)
6 (11637, 13576)
7 (13577, 15515)
8 (15516, 17455)
9 (17456, 19395)
```

Let us familiarize ourselves with array slicing with SFrame. To extract a continuous slice from an SFrame, use colon in square brackets. For instance, the following cell extracts rows 0 to 9 of `train_valid_shuffled`. Notice that the first index (0) is included in the slice but the last index (10) is omitted.

In [29]:

```
train_valid_shuffled[0:10] # rows 0 to 9
```

Out[29]:

id	date	price	bedrooms	bathrooms
2780400035	2014-05-05 00:00:00+00:00	665000.0	4.0	2.5
1703050500	2015-03-21 00:00:00+00:00	645000.0	3.0	2.5
5700002325	2014-06-05 00:00:00+00:00	640000.0	3.0	1.75
0475000510	2014-11-18 00:00:00+00:00	594000.0	3.0	1.0
0844001052	2015-01-28 00:00:00+00:00	365000.0	4.0	2.5
2781280290	2015-04-27 00:00:00+00:00	305000.0	3.0	2.5
2214800630	2014-11-05 00:00:00+00:00	239950.0	3.0	2.25
2114700540	2014-10-21 00:00:00+00:00	366000.0	3.0	2.5
2596400050	2014-07-30 00:00:00+00:00	375000.0	3.0	1.0
4140900050	2015-01-26 00:00:00+00:00	440000.0	4.0	1.75

view	condition	grade	sqft_above	sqft_basement	yr_built
0	3	8	1660	1140	1963
0	3	9	2490	0	2003
0	5	7	1170	1170	1917
0	4	7	1090	230	1920
0	5	7	1904	0	1999
0	3	8	1610	0	2006
0	4	7	1560	0	1979
0	3	6	660	660	1918
0	4	7	1260	700	1963
2	3	8	2000	180	1966

long	sqft_living15	sqft_lot15
-122.28583258	2580.0	5900.0
-122.02177564	2710.0	6629.0
-122.28796	1360.0	4725.0

```
Now let us extract individual segments with array slicing. Consider the scenario
where we group the houses in the `train_valid_shuffled` dataframe into k=10
segments of roughly equal size, with starting and ending indices computed as above.
Extract the fourth segment (segment 3) and assign it to a variable called
`validation4`.
```

In [31]:

```
validation4 = train_valid_shuffled[5818:7757]
```

To verify that we have the right elements extracted, run the following cell, which computes the average price of the fourth segment. When rounded to nearest whole number, the average should be $536,234.

In [32]:

```
print int(round(validation4['price'].mean(), 0))
```

```
536353
```

After designating one of the k segments as the validation set, we train a model using the rest of the data. To choose the remainder, we slice (0:start) and (end+1:n) of the data and paste them together. SFrame has `append()` method that pastes together two disjoint sets of rows originating from a common dataset. For instance, the following cell pastes together the first and last two rows of the `train_valid_shuffled` dataframe.

In [33]:

```
n = len(train_valid_shuffled)
first_two = train_valid_shuffled[0:2]
last_two = train_valid_shuffled[n-2:n]
print first_two.append(last_two)
```

```
+------------+---------------------------+-----------+----------+------
-----+
|     id     |            date           |   price   | bedrooms | bathr
ooms |
+------------+---------------------------+-----------+----------+------
-----+
| 2780400035 | 2014-05-05 00:00:00+00:00 |  665000.0 |   4.0    |
2.5    |
| 1703050500 | 2015-03-21 00:00:00+00:00 |  645000.0 |   3.0    |
2.5    |
| 4139480190 | 2014-09-16 00:00:00+00:00 | 1153000.0 |   3.0    |
3.25   |
| 7237300290 | 2015-03-26 00:00:00+00:00 |  338000.0 |   5.0    |
2.5    |
+------------+---------------------------+-----------+----------+------
-----+
+-------------+----------+--------+------------+------+-----------+----
---+------------+
| sqft_living | sqft_lot | floors | waterfront | view | condition | gra
de | sqft_above |
+-------------+----------+--------+------------+------+-----------+----
---+------------+
|    2800.0   |   5900   |   1    |     0      |  0   |     3     |   8
|    1660    |
|    2490.0   |   5978   |   2    |     0      |  0   |     3     |   9
|    2490    |
|    3780.0   |  10623   |   1    |     0      |  1   |     3     |   1
1  |    2650    |
|    2400.0   |   4496   |   2    |     0      |  0   |     3     |   7
|    2400    |
+-------------+----------+--------+------------+------+-----------+----
---+------------+
+---------------+----------+-------------+---------+-------------+
| sqft_basement | yr_built | yr_renovated | zipcode |     lat     |
+---------------+----------+-------------+---------+-------------+
|      1140     |   1963   |      0      |  98115  | 47.68093246 |
|       0       |   2003   |      0      |  98074  | 47.62984888 |
|      1130     |   1999   |      0      |  98006  | 47.55061236 |
|       0       |   2004   |      0      |  98042  | 47.36923712 |
+---------------+----------+-------------+---------+-------------+
+---------------+---------------+-----+
|      long     | sqft_living15 | ... |
+---------------+---------------+-----+
| -122.28583258 |     2580.0    | ... |
| -122.02177564 |     2710.0    | ... |
| -122.10144844 |     3850.0    | ... |
| -122.12606473 |     1880.0    | ... |
+---------------+---------------+-----+
[4 rows x 21 columns]
```

```
Extract the remainder of the data after *excluding* fourth segment (segment 3) and
assign the subset to `train4`.
```

In [34]:

```
train4 = train_valid_shuffled[:5818].append(train_valid_shuffled[7758:])
```

To verify that we have the right elements extracted, run the following cell, which computes the average price of the data with fourth segment excluded. When rounded to nearest whole number, the average should be $539,450.

In [35]:

```
print int(round(train4['price'].mean(), 0))
```

```
539450
```

Now we are ready to implement k-fold cross-validation. Write a function that computes k validation errors by designating each of the k segments as the validation set. It accepts as parameters (i) k, (ii) `l2_penalty`, (iii) dataframe, (iv) name of output column (e.g. `price`) and (v) list of feature names. The function returns the average validation error using k segments as validation sets.

- For each i in [0, 1, ..., k-1]:
  - Compute starting and ending indices of segment i and call 'start' and 'end'
  - Form validation set by taking a slice (start:end+1) from the data.
  - Form training set by appending slice (end+1:n) to the end of slice (0:start).
  - Train a linear model using training set just formed, with a given l2_penalty
  - Compute validation error using validation set just formed

In [36]:

```
def k_fold_cross_validation(k, l2_penalty, data, output_name, features_list):
    n, RSS = len(data), 0.0
    for i in xrange(k):
        start = (n*i)/k
        end = (n*(i+1))/k-1
        validation_i = data[start:end+1]
        train_i = data[0:start].append(data[end+1:n])
        model_i = graphlab.linear_regression.create(train_i, target = output_name, fe
        RSS += sum((validation_i[output_name] - model_i.predict(validation_i))**2)
    return RSS / k
```

Once we have a function to compute the average validation error for a model, we can write a loop to find the model that minimizes the average validation error. Write a loop that does the following:

- We will again be aiming to fit a 15th-order polynomial model using the `sqft_living` input
- For `l2_penalty` in [10^1, 10^1.5, 10^2, 10^2.5, ..., 10^7] (to get this in Python, you can use this Numpy function: `np.logspace(1, 7, num=13)`.)
  - Run 10-fold cross-validation with `l2_penalty`

- Report which L2 penalty produced the lowest average validation error.

Note: since the degree of the polynomial is now fixed to 15, to make things faster, you should generate polynomial features in advance and re-use them throughout the loop. Make sure to use `train_valid_shuffled` when generating polynomial features!

In [40]:

```
import numpy as np
poly_data = polynomial_sframe(train_valid_shuffled['sqft_living'], 15)
features_list = poly_data.column_names() # get the name of the features
poly_data['price'] = train_valid_shuffled['price'] # add price to the data since it's
RSS = {}
for l2_penalty in np.logspace(1, 7, num=13):
    RSS[l2_penalty] = k_fold_cross_validation(10, l2_penalty, poly_data, 'price', fea
    print l2_penalty, RSS[l2_penalty]
print sorted([(RSS[k],k) for k in RSS])
```

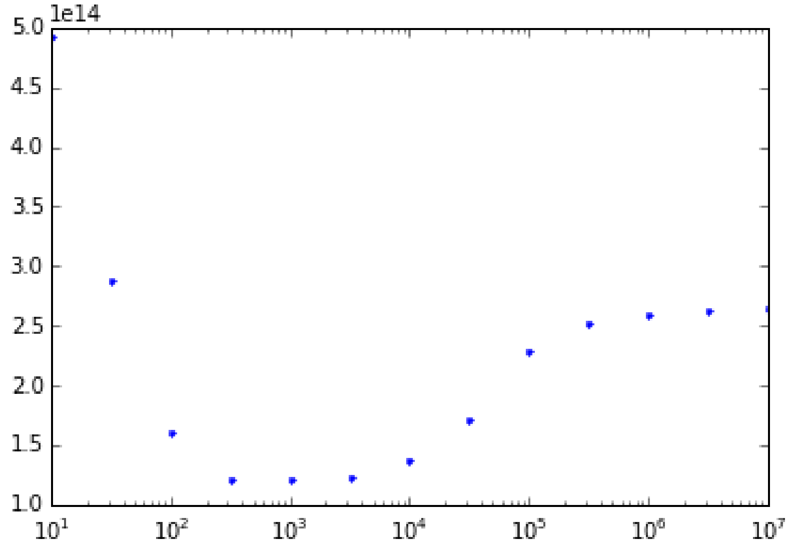
```
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+-------------+------------------
-+---------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error
| Training-rmse |
PROGRESS: +-----------+----------+-------------+------------------
-+---------------+
PROGRESS: | 1         | 2        | 0.015049     | 5750349.023107
| 354206.158132 |
PROGRESS: +-----------+----------+-------------+------------------
-+---------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 17456
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
```

QUIZ QUESTIONS: What is the best value for the L2 penalty according to 10-fold validation?

You may find it useful to plot the k-fold cross-validation errors you have obtained to better understand the behavior of the method.

In [41]:

```
# Plot the l2_penalty values in the x axis and the cross-validation error in the y ax
# Using plt.xscale('log') will make your plot more intuitive.
plt.plot(RSS.keys(),RSS.values(),'.')
plt.xscale('log')
```



Once you found the best value for the L2 penalty using cross-validation, it is important to retrain a final model on all of the training data using this value of `l2_penalty`. This way, your final model will be trained on the entire dataset.

In [43]:

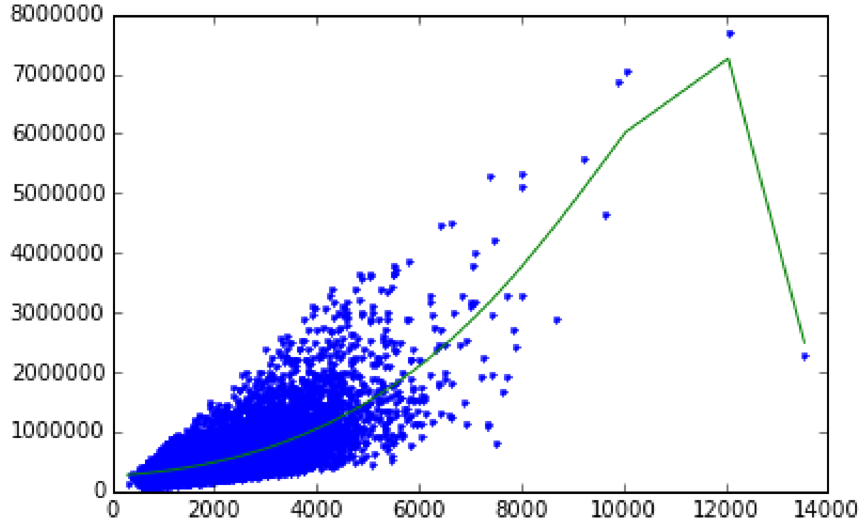
```
poly_data = polynomial_sframe(train_valid['sqft_living'], 15)
my_features = poly_data.column_names() # get the name of the features
poly_data['price'] = train_valid['price'] # add price to the data since it's the targ
model = graphlab.linear_regression.create(poly_data, target = 'price', features = my_
model.get("coefficients").print_rows(num_rows=16)
poly_data_test = polynomial_sframe(test['sqft_living'], 15)
RSS = sum((test['price'] - model.predict(test))**2)
print RSS
plt.plot(poly_data['power_1'],poly_data['price'],'.', poly_data['power_1'], model.pre
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 19396
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: | 1         | 2        | 0.031251     | 2461778.986191     |
248914.007014 |
PROGRESS: +-----------+----------+--------------+--------------------+-
--------------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   253972.377679    |
|   power_1   |  None |   57.3754544277    |
|   power_2   |  None |   0.023783252166   |
|   power_3   |  None | 2.64303484986e-06  |
|   power_4   |  None | 6.53250411225e-11  |
|   power_5   |  None | -1.0280259983e-15  |
|   power_6   |  None | -1.17062658026e-19 |
|   power_7   |  None | -5.32286483343e-24 |
|   power_8   |  None | -2.90413735131e-28 |
|   power_9   |  None | -2.90869143388e-32 |
|   power_10  |  None | -3.44379026738e-36 |
|   power_11  |  None | -3.74459613949e-40 |
|   power_12  |  None | -3.68892060043e-44 |
|   power_13  |  None | -3.37485950649e-48 |
|   power_14  |  None | -2.92982311462e-52 |
|   power_15  |  None | -2.45030811724e-56 |
+-------------+-------+--------------------+
[16 rows x 3 columns]

2.52897427447e+14
```

Out[43]:

```
[<matplotlib.lines.Line2D at 0x1f8cca90>,
 <matplotlib.lines.Line2D at 0x9d62940>]
```



QUIZ QUESTION: Using the best L2 penalty found above, train a model using all training data. What is the RSS on the TEST data of the model you learn with this L2 penalty?

In [ ]:

```

```