Regression Week 2: Multiple Regression (Interpretation)

The goal of this first notebook is to explore multiple regression and feature engineering with existing graphlab functions.

In this notebook you will use data on house sales in King County to predict prices using multiple regression. You will:

- Use SFrames to do some feature engineering
- Use built-in graphlab functions to compute the regression weights (coefficients/parameters)
- Given the regression weights, predictors and outcome write a function to compute the Residual Sum of Squares
- Look at coefficients and interpret their meanings
- Evaluate multiple models via RSS

Fire up graphlab create

In [1]:
```
import graphlab
```

```
A newer version of GraphLab Create (v1.10.1) is available! Your current version
 is v1.8.3.

You can use pip to upgrade the graphlab-create package. For more information se
e https://dato.com/products/create/upgrade. (https://dato.com/products/create/u
pgrade.)
```

Load in house sales data

Dataset is from house sales in King County, the region where the city of Seattle, WA is located.

In [2]:
```
sales = graphlab.SFrame('kc_house_data.gl/')
```

```
[INFO] GraphLab Create v1.8.3 started. Logging: C:\Users\DORINA~1.STR\AppData\L
ocal\Temp\graphlab_server_1465780353.log.0
```

Split data into training and testing.

We use seed=0 so that everyone running this notebook gets the same results. In practice, you may set a random seed (or let GraphLab Create pick a random seed for you).

In [4]:
```
train_data,test_data = sales.random_split(.8,seed=0)
```

Learning a multiple regression model

Recall we can use the following code to learn a multiple regression model predicting 'price' based on the following features: example_features = ['sqft_living', 'bedrooms', 'bathrooms'] on training data with the following code:

(Aside: We set validation_set = None to ensure that the results are always the same)

In [5]:
```
example_features = ['sqft_living', 'bedrooms', 'bathrooms']
example_model = graphlab.linear_regression.create(train_data, target = 'price', fe
                                                  validation_set = None)
```

Now that we have fitted the model we can extract the regression weights (coefficients) as an SFrame as follows:

In [6]:
```
example_weight_summary = example_model.get("coefficients")
print example_weight_summary
```

```
+-------------+-------+----------------+---------------+
|     name    | index |     value      |     stderr    |
+-------------+-------+----------------+---------------+
| (intercept) |  None | 87910.0724924  |  7873.3381434 |
| sqft_living |  None | 315.403440552  | 3.45570032585 |
|   bedrooms  |  None | -65080.2155528 | 2717.45685442 |
|  bathrooms  |  None | 6944.02019265  | 3923.11493144 |
+-------------+-------+----------------+---------------+
[4 rows x 4 columns]
```

Making Predictions

In the gradient descent notebook we use numpy to do our regression. In this book we will use existing graphlab create functions to analyze multiple regressions.

Recall that once a model is built we can use the .predict() function to find the predicted values for data we pass. For example using the example model above:

In [7]:
```
example_predictions = example_model.predict(train_data)
print example_predictions[0] # should be 271789.505878
```

```
271789.505878
```

Compute RSS

Now that we can make predictions given the model, let's write a function to compute the RSS of the model. Complete the function below to calculate RSS given the model, data, and the outcome.

In [12]:
```
def get_residual_sum_of_squares(model, data, outcome):
    # First get the predictions
    predicted_price = model.predict(data)
    # Then compute the residuals/errors
    residuals = predicted_price - outcome
    # Then square and add them up
    RSS = (residuals * residuals).sum()
    return(RSS)
```

Test your function by computing the RSS on TEST data for the example model:

In [13]:
```
rss_example_train = get_residual_sum_of_squares(example_model, test_data, test_dat
print rss_example_train # should be 2.7376153833e+14
```

```
2.7376153833e+14
```

Create some new features

Although we often think of multiple regression as including multiple different features (e.g. # of bedrooms, squarefeet, and # of bathrooms) but we can also consider transformations of existing features e.g. the log of the squarefeet or even "interaction" features such as the product of bedrooms and bathrooms.

You will use the logarithm function to create a new feature. so first you should import it from the math library.

In [15]:
```
from math import log
```

```
Next create the following 4 new features as column in both TEST and TRAIN data:
* bedrooms_squared = bedrooms\*bedrooms
* bed_bath_rooms = bedrooms\*bathrooms
* log_sqft_living = log(sqft_living)
* lat_plus_long = lat + long
As an example here's the first one:
```

In [16]:
```
train_data['bedrooms_squared'] = train_data['bedrooms'].apply(lambda x: x**2)
test_data['bedrooms_squared'] = test_data['bedrooms'].apply(lambda x: x**2)
```

In [17]:
```
# create the remaining 3 features in both TEST and TRAIN data
train_data['bed_bath_rooms'] =  train_data['bedrooms'] * train_data['bathrooms']
test_data['bed_bath_rooms'] =  test_data['bedrooms'] * test_data['bathrooms']

train_data['log_sqft_living'] = train_data['sqft_living'].apply(lambda x: log(x))
test_data['log_sqft_living'] = test_data['sqft_living'].apply(lambda x: log(x))

train_data['lat_plus_long'] = train_data['lat'] + train_data['long']
test_data['lat_plus_long'] = test_data['lat'] + test_data['long']
```

- Squaring bedrooms will increase the separation between not many bedrooms (e.g. 1) and lots of bedrooms (e.g. 4) since 1^2 = 1 but 4^2 = 16. Consequently this feature will mostly affect houses with many bedrooms.
- bedrooms times bathrooms gives what's called an "interaction" feature. It is large when *both* of them are large.
- Taking the log of squarefeet has the effect of bringing large values closer together and spreading out small values.
- Adding latitude to longitude is totally non-sensical but we will do it anyway (you'll see why)

Quiz Question: What is the mean (arithmetic average) value of your 4 new features on TEST data? (round to 2 digits)

In [19]:
```
print test_data['bedrooms_squared'].mean()
print test_data['bed_bath_rooms'].mean()
print test_data['log_sqft_living'].mean()
print test_data['lat_plus_long'].mean()
```

```
12.4466777016
7.50390163159
7.55027467965
-74.6533349722
```

Learning Multiple Models

Now we will learn the weights for three (nested) models for predicting house prices. The first model will have the fewest features the second model will add one more feature and the third will add a few more:

- Model 1: squarefeet, # bedrooms, # bathrooms, latitude & longitude
- Model 2: add bedrooms*bathrooms
- Model 3: Add log squarefeet, bedrooms squared, and the (nonsensical) latitude + longitude

In [20]:
```
model_1_features = ['sqft_living', 'bedrooms', 'bathrooms', 'lat', 'long']
model_2_features = model_1_features + ['bed_bath_rooms']
model_3_features = model_2_features + ['bedrooms_squared', 'log_sqft_living', 'lat
```

Now that you have the features, learn the weights for the three different models for predicting target = 'price' using graphlab.linear_regression.create() and look at the value of the weights/coefficients:

In [21]:
```
# Learn the three models: (don't forget to set validation_set = None)
model_1 = graphlab.linear_regression.create(train_data, target = 'price', features
                                                   validation_set = None)
model_2 = graphlab.linear_regression.create(train_data, target = 'price', features
                                                   validation_set = None)
model_3 = graphlab.linear_regression.create(train_data, target = 'price', features
                                                   validation_set = None)
```

```
Linear regression:

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

Number of examples          : 17384

Number of features          : 5

Number of unpacked features : 5

Number of coefficients    : 6

Starting Newton Method

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

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

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

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

| 1         | 2        | 0.015012     | 4074878.213096     | 236378.596455 |

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

SUCCESS: Optimal solution found.

Linear regression:

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

Number of examples          : 17384

Number of features          : 6

Number of unpacked features : 6

Number of coefficients    : 7

Starting Newton Method

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

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

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

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

| 1         | 2        | 0.018015     | 4014170.932927     | 235190.935428 |
```

```
+-----------+----------+--------------+--------------------+---------------+
SUCCESS: Optimal solution found.

Linear regression:
--------------------------------------------------------
Number of examples          : 17384
Number of features          : 9
Number of unpacked features : 9
Number of coefficients    : 10
Starting Newton Method
--------------------------------------------------------
+-----------+----------+--------------+--------------------+---------------+
| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |
+-----------+----------+--------------+--------------------+---------------+
| 1         | 2        | 0.013010     | 3193229.177894     | 228200.043155 |
+-----------+----------+--------------+--------------------+---------------+
```

In [22]:
```
# Examine/extract each model's coefficients:
model_1_weight_summary = model_1.get("coefficients")
model_2_weight_summary = model_2.get("coefficients")
model_3_weight_summary = model_3.get("coefficients")
print model_1_weight_summary
print model_2_weight_summary
print model_3_weight_summary
```

```
+-------------+-------+----------------+---------------+
|     name    | index |     value      |     stderr    |
+-------------+-------+----------------+---------------+
| (intercept) |  None | -56140675.7444 | 1649985.42028 |
| sqft_living |  None | 310.263325778  | 3.18882960408 |
|   bedrooms  |  None | -59577.1160682 | 2487.27977322 |
|  bathrooms  |  None | 13811.8405418  | 3593.54213297 |
|     lat     |  None | 629865.789485  | 13120.7100323 |
|     long    |  None | -214790.285186 | 13284.2851607 |
+-------------+-------+----------------+---------------+
[6 rows x 4 columns]

+---------------+-------+----------------+---------------+
|      name     | index |     value      |     stderr    |
+---------------+-------+----------------+---------------+
|  (intercept)  |  None | -54410676.1152 | 1650405.16541 |
|  sqft_living  |  None | 304.449298057  | 3.20217535637 |
|    bedrooms   |  None | -116366.043231 | 4805.54966546 |
|   bathrooms   |  None | -77972.3305135 | 7565.05991091 |
|      lat      |  None | 625433.834953  | 13058.3530972 |
|      long     |  None | -203958.60296  | 13268.1283711 |
| bed_bath_rooms |  None | 26961.6249092  | 1956.36561555 |
+---------------+-------+----------------+---------------+
[7 rows x 4 columns]

+-----------------+-------+----------------+---------------+
|       name      | index |     value      |     stderr    |
+-----------------+-------+----------------+---------------+
|   (intercept)   |  None | -52974974.0602 |  1615194.9439 |
|   sqft_living   |  None | 529.196420564  | 7.69913498511 |
|     bedrooms    |  None | 28948.5277313  | 9395.72889106 |
|    bathrooms    |  None |  65661.207231  | 10795.3380703 |
|       lat       |  None | 704762.148408  | 1292011141.66 |
|       long      |  None | -137780.01994  | 1292011141.57 |
|  bed_bath_rooms |  None | -8478.36410518 | 2858.95391257 |
| bedrooms_squared |  None | -6072.38466067 | 1494.97042777 |
| log_sqft_living |  None | -563467.784269 | 17567.8230814 |
|  lat_plus_long  |  None | -83217.1979248 | 1292011141.58 |
+-----------------+-------+----------------+---------------+
[10 rows x 4 columns]
```

Quiz Question: What is the sign (positive or negative) for the coefficient/weight for 'bathrooms' in model 1?

Quiz Question: What is the sign (positive or negative) for the coefficient/weight for 'bathrooms' in model 2?

Think about what this means.

Comparing multiple models

Now that you've learned three models and extracted the model weights we want to evaluate which model is best.

First use your functions from earlier to compute the RSS on TRAINING Data for each of the three models.

In [23]:
```
# Compute the RSS on TRAINING data for each of the three models and record the val
rss_train_model_1 = get_residual_sum_of_squares(model_1, train_data, train_data['p
rss_train_model_2 = get_residual_sum_of_squares(model_2, train_data, train_data['p
rss_train_model_3 = get_residual_sum_of_squares(model_3, train_data, train_data['p
print rss_train_model_1
print rss_train_model_2
print rss_train_model_3
```

```
9.71328233544e+14
9.61592067856e+14
9.05276314555e+14
```

Quiz Question: Which model (1, 2 or 3) has lowest RSS on TRAINING Data? Is this what you expected?

Now compute the RSS on on TEST data for each of the three models.

In [24]:
```
# Compute the RSS on TESTING data for each of the three models and record the valu
rss_test_model_1 = get_residual_sum_of_squares(model_1, test_data, test_data['pric
rss_test_model_2 = get_residual_sum_of_squares(model_2, test_data, test_data['pric
rss_test_model_3 = get_residual_sum_of_squares(model_3, test_data, test_data['pric
print rss_test_model_1
print rss_test_model_2
print rss_test_model_3
```

```
2.26568089093e+14
2.24368799994e+14
2.51829318952e+14
```

Quiz Question: Which model (1, 2 or 3) has lowest RSS on TESTING Data? Is this what you expected?Think about the features that were added to each model from the previous.