Regression Week 1: Simple Linear Regression

In this notebook we will use data on house sales in King County to predict house prices using simple (one input) linear regression. You will:

- Use graphlab SArray and SFrame functions to compute important summary statistics
- Write a function to compute the Simple Linear Regression weights using the closed form solution
- Write a function to make predictions of the output given the input feature
- Turn the regression around to predict the input given the output
- Compare two different models for predicting house prices

In this notebook you will be provided with some already complete code as well as some code that you should complete yourself in order to answer quiz questions. The code we provide to complte is optional and is there to assist you with solving the problems but feel free to ignore the helper code and write your own.

Fire up graphlab create

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

```
A newer version of GraphLab Create (v1.10) is available! Your current ver
sion is v1.8.3.

You can use pip to upgrade the graphlab-create package. For more informat
ion see https://dato.com/products/create/upgrade. (https://dato.com/produ
cts/create/upgrade.)
```

Load 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\App
Data\Local\Temp\graphlab_server_1465086244.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)
```

Useful SFrame summary functions

In order to make use of the closed form solution as well as take advantage of graphlab's built in functions we will review some important ones. In particular:

- Computing the sum of an SArray
- Computing the arithmetic average (mean) of an SArray
- multiplying SArrays by constants
- multiplying SArrays by other SArrays

In [ ]:
```
# Let's compute the mean of the House Prices in King County in 2 different w
prices = sales['price'] # extract the price column of the sales SFrame -- th

# recall that the arithmetic average (the mean) is the sum of the prices div
sum_prices = prices.sum()
num_houses = prices.size() # when prices is an SArray .size() returns its le
avg_price_1 = sum_prices/num_houses
avg_price_2 = prices.mean() # if you just want the average, the .mean() func
print "average price via method 1: " + str(avg_price_1)
print "average price via method 2: " + str(avg_price_2)
```

As we see we get the same answer both ways

In [ ]:
```
# if we want to multiply every price by 0.5 it's a simple as:
half_prices = 0.5*prices
# Let's compute the sum of squares of price. We can multiply two SArrays of
prices_squared = prices*prices
sum_prices_squared = prices_squared.sum() # price_squared is an SArray of th
print "the sum of price squared is: " + str(sum_prices_squared)
```

Aside: The python notation x.xxe+yy means x.xx * 10^(yy). e.g 100 = 10^2 = 1*10^2 = 1e2

Build a generic simple linear regression function

Armed with these SArray functions we can use the closed form solution found from lecture to compute the slope and intercept for a simple linear regression on observations stored as SArrays: input_feature, output.

Complete the following function (or write your own) to compute the simple linear regression slope and intercept:

In [8]:
```
def simple_linear_regression(input_feature, output):
    # compute the sum of input_feature and output
    sum_x = sum(input_feature)
    sum_y = sum(output)
    # compute the product of the output and the input_feature and its sum
    sum_yx = sum(input_feature*output)
    # compute the squared value of the input_feature and its sum
    sum_xx = sum(input_feature * input_feature)
    # use the formula for the slope
    n = len(output)
    slope = (sum_yx - ((sum_y*sum_x)/n))/(sum_xx - ((sum_x**2)/n))
    # use the formula for the intercept
    intercept = (sum_y/n) - slope*(sum_x/n)
    return (intercept, slope)
```

We can test that our function works by passing it something where we know the answer. In particular we can generate a feature and then put the output exactly on a line: output = 1 + 1*input_feature then we know both our slope and intercept should be 1

In [9]:
```
test_feature = graphlab.SArray(range(5))
test_output = graphlab.SArray(1 + 1*test_feature)
(test_intercept, test_slope) =  simple_linear_regression(test_feature, test_
print "Intercept: " + str(test_intercept)
print "Slope: " + str(test_slope)
```

```
Intercept: 1
Slope: 1
```

Now that we know it works let's build a regression model for predicting price based on sqft_living. Rembember that we train on train_data!

In [10]:
```
sqft_intercept, sqft_slope = simple_linear_regression(train_data['sqft_livin

print "Intercept: " + str(sqft_intercept)
print "Slope: " + str(sqft_slope)
```

```
Intercept: -47116.0765749
Slope: 281.958838568
```

Predicting Values

Now that we have the model parameters: intercept & slope we can make predictions. Using SArrays it's easy to multiply an SArray by a constant and add a constant value. Complete the following function to return the predicted output given the input_feature, slope and intercept:

In [13]:
```
def get_regression_predictions(input_feature, intercept, slope):
    # calculate the predicted values:
    return(intercept + slope*input_feature)
    return predicted_values
```

Now that we can calculate a prediction given the slope and intercept let's make a prediction. Use (or alter) the following to find out the estimated price for a house with 2650 squarefeet according to the squarefeet model we estiamted above.

Quiz Question: Using your Slope and Intercept from (4), What is the predicted price for a house with 2650 sqft?

In [14]:
```
my_house_sqft = 2650
estimated_price = get_regression_predictions(my_house_sqft, sqft_intercept,
print "The estimated price for a house with %d squarefeet is $%.2f" % (my_ho
```

```
The estimated price for a house with 2650 squarefeet is $700074.85
```

Residual Sum of Squares

Now that we have a model and can make predictions let's evaluate our model using Residual Sum of Squares (RSS). Recall that RSS is the sum of the squares of the residuals and the residuals is just a fancy word for the difference between the predicted output and the true output.

Complete the following (or write your own) function to compute the RSS of a simple linear regression model given the input_feature, output, intercept and slope:

In [19]:
```
def get_residual_sum_of_squares(input_feature, output, intercept, slope):
    y_hat = intercept + slope*input_feature
    return(sum((output - y_hat)**2))
```

Let's test our get_residual_sum_of_squares function by applying it to the test model where the data lie exactly on a line. Since they lie exactly on a line the residual sum of squares should be zero!

In [20]:
```
print get_residual_sum_of_squares(test_feature, test_output, test_intercept,
```

```
0.0
```

Now use your function to calculate the RSS on training data from the squarefeet model calculated above.

Quiz Question: According to this function and the slope and intercept from the squarefeet model What is the RSS for the simple linear regression using squarefeet to predict prices on TRAINING data?

In [21]:
```
rss_prices_on_sqft = get_residual_sum_of_squares(train_data['sqft_living'],
print 'The RSS of predicting Prices based on Square Feet is : ' + str(rss_pr
```

```
The RSS of predicting Prices based on Square Feet is : 1.20191835632e+15
```

Predict the squarefeet given price

What if we want to predict the squarefoot given the price? Since we have an equation y = a + b*x we can solve the function for x. So that if we have the intercept (a) and the slope (b) and the price (y) we can solve for the estimated squarefeet (x).

Comlplete the following function to compute the inverse regression estimate, i.e. predict the input_feature given the output!

In [22]:
```
def inverse_regression_predictions(output, intercept, slope):
    return((output - intercept)/slope)
```

Now that we have a function to compute the squarefeet given the price from our simple regression model let's see how big we might expect a house that costs $800,000 to be.

Quiz Question: According to this function and the regression slope and intercept from (3) what is the estimated square-feet for a house costing $800,000?

In [23]:
```
my_house_price = 800000
estimated_squarefeet = inverse_regression_predictions(my_house_price, sqft_i
print "The estimated squarefeet for a house worth $%.2f is %d" % (my_house_p
```

```
The estimated squarefeet for a house worth $800000.00 is 3004
```

New Model: estimate prices from bedrooms

We have made one model for predicting house prices using squarefeet, but there are many other features in the sales SFrame. Use your simple linear regression function to estimate the regression parameters from predicting Prices based on number of bedrooms. Use the training data!

In [32]:
```
# Estimate the slope and intercept for predicting 'price' based on 'bedrooms
#sqft_intercept, sqft_slope = simple_linear_regression(train_data['sqft_livi
bedroom_intercept, bedroom_slope = simple_linear_regression(train_data['bedr
print "Intercept: " + str(bedroom_intercept)
print "Slope: " + str(bedroom_slope)
```

```
Intercept: 109473.180469
Slope: 127588.952175
```

Test your Linear Regression Algorithm

Now we have two models for predicting the price of a house. How do we know which one is better? Calculate the RSS on the TEST data (remember this data wasn't involved in learning the model). Compute the RSS from predicting prices using bedrooms and from predicting prices using squarefeet.

Quiz Question: Which model (square feet or bedrooms) has lowest RSS on TEST data? Think about why this might be the case.

In [36]:
```
# Compute RSS when using bedrooms on TEST data:
rss_sqft = get_residual_sum_of_squares(test_data['sqft_living'], test_data['
rss_bedroom = get_residual_sum_of_squares(test_data['bedrooms'], test_data['
print "rss_sqft: " + str(rss_sqft)
print "rss_bedroom: " + str(rss_bedroom)
rss_sqft
rss_bedroom
```

```
rss_sqft: 2.75402936247e+14
rss_bedroom: 4.93364582868e+14
```

Out[36]:
```
493364582868287.4
```

In [ ]:
```
# Compute RSS when using squarefeet on TEST data:
# The model that uses house size (square feet) has the smallest RSS.
# This is likely do to the fact that square footage is more predictive of ho
```