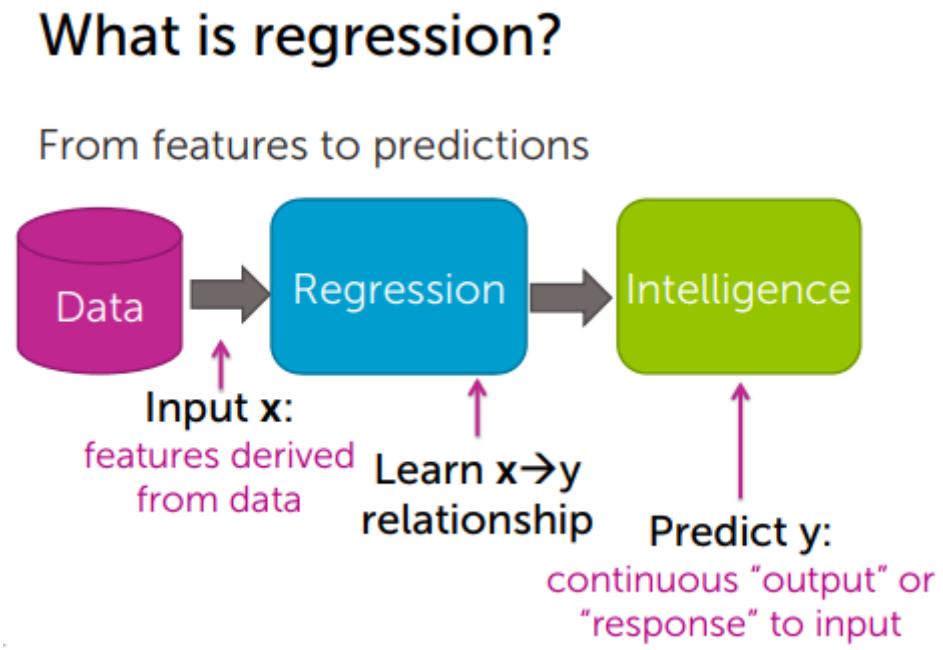
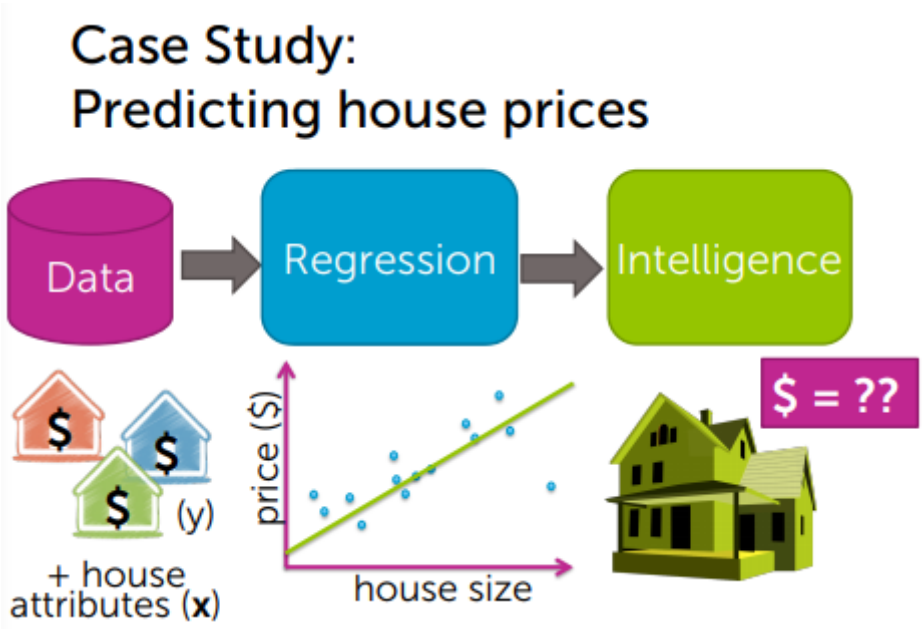
Regression





Module 1 : Simple Regression :

- 1 input and just fit a line to data.
- Define a goodness-of-fit metric for each possible line.
- Gradient Descent Algorithm : slope and intercept of the simple fitted line that minimizes the goodness-of-fit metrics;
- Get estimated parameters - intercept and slope - form the predictions.

Module 2 : Multiple Regression:

- Fits more complex relationships that just a line between a single input and an output.
- It also incorporates more inputs/features.

Module 3 : Assessing Performance:

- Overfit to Data model;
- Measures of Errors : Training, Test and True(generalization);
- Graph of Model complexity V/s Error -> understand errors and overfitting models.
- Bias-Variance Trade Off :
  - Simple models tend to be well behaved but are too simple to explain the relationship represented in the models dataset.
  - Complex Models are flexible enough to fit really intricate relationships that might be in the model by complex models could have strange behavior.

Module 4 : Ridge Regression:

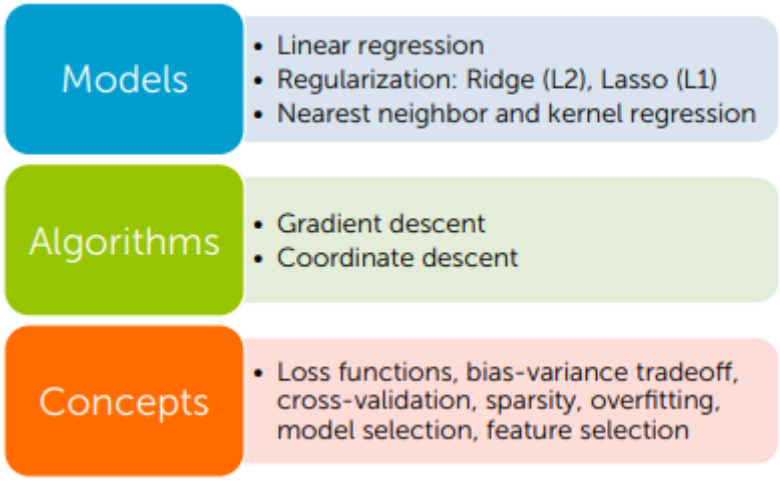
- Means to handle the Bias-Variance Trade off.
- Ridge total cost = measure of fit + measure of complexity. [bias-variance tradeoff]
- Cross validation - means to chose the emphases/tunning on the parameters of Ridge total cost (measure of fit ,measure of complexity);

Module 5 : Feature Selection and Lasso Regression :

- Feature selection.
- Useful for efficiency of predictions and interceptibility
- Lasso total cost -> Feature selection implicitly.
  - Lasso Total Cost = measure of fit + (different) measure of model complexity;
- Coordinate descent algorithm - optimizaion method - It goes co-ordinate by co-ordinate / variable-by-variable optimizing each in turn.

Module 6 : Nearest Neighbour and Kernel Regression :

- Nearest Neighbour Model - Find similar items in the dataset and attain information from it and base predictions from it.
- Kernel Regression - Include every observation in the dataset informing the predicted value. While computing the result weight the other observations by how close they are to the focus item.



Simple Regression

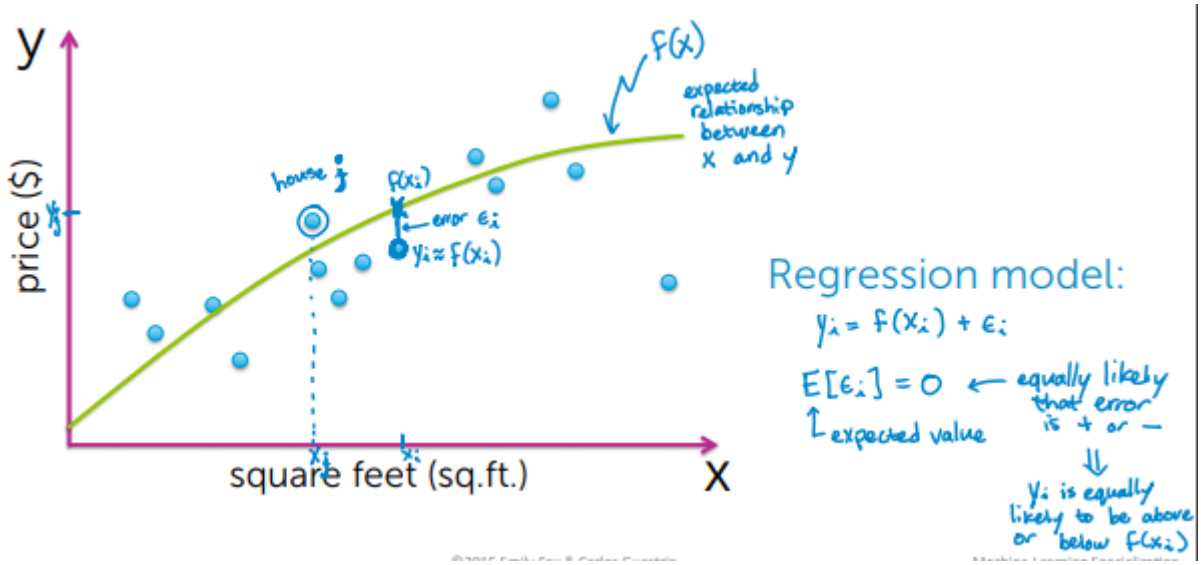
- Predicting house prices
- Look at the recent sales in the neighbourhood and predict the price of the focus house.

Regression fundamentals

Data : For every house - consider the input x = sqft, and output y = price. Record this for very house in model;

- y is the quantity of interest;
- assume y can be predicted from x;

Model - The fit to the dataset.



- Expected Error = weighted avaerage of the errors.
- The + or - means that the observation yi is equally likely that it can be above or below the expected value (f(xi));

Tasks

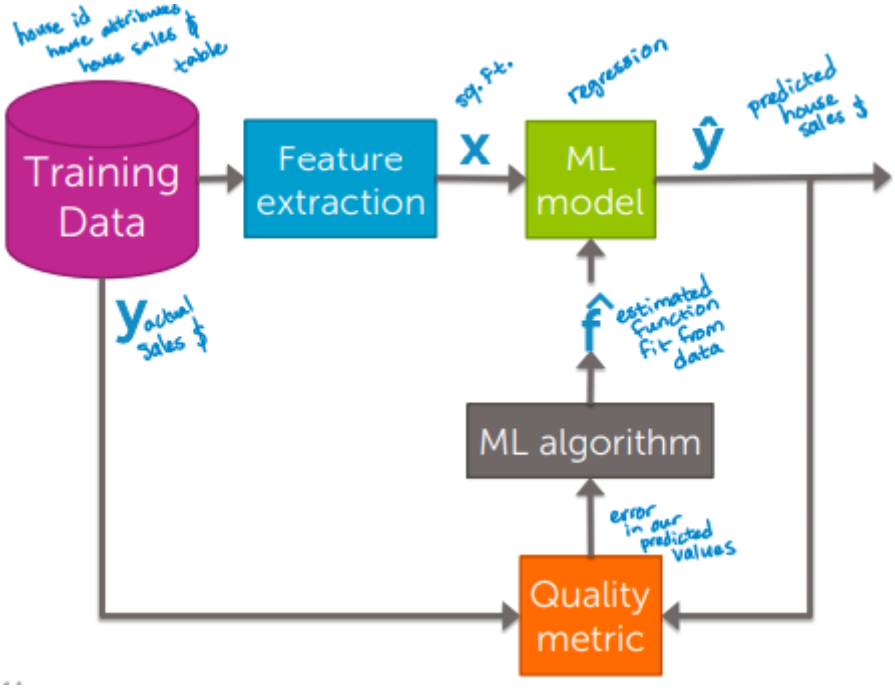
Task 1 : Which model is f(x) ?

- Straight line
- Linear line
- Quadratice relation
- polynomial,etc

Task 2 : For a given model f(x), estimate function f(hat)(x) from the data

- Different fits for the data, or differen curves for the data.
- Need to choose the best fit for the model.

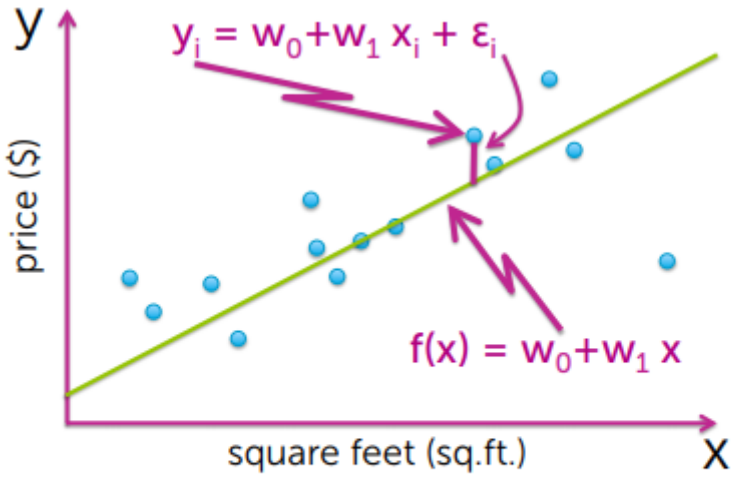
Regression ML Block Diagram



- Each block in the ML block diagram is discussed below

A. Model - Simple Linear Regression Model

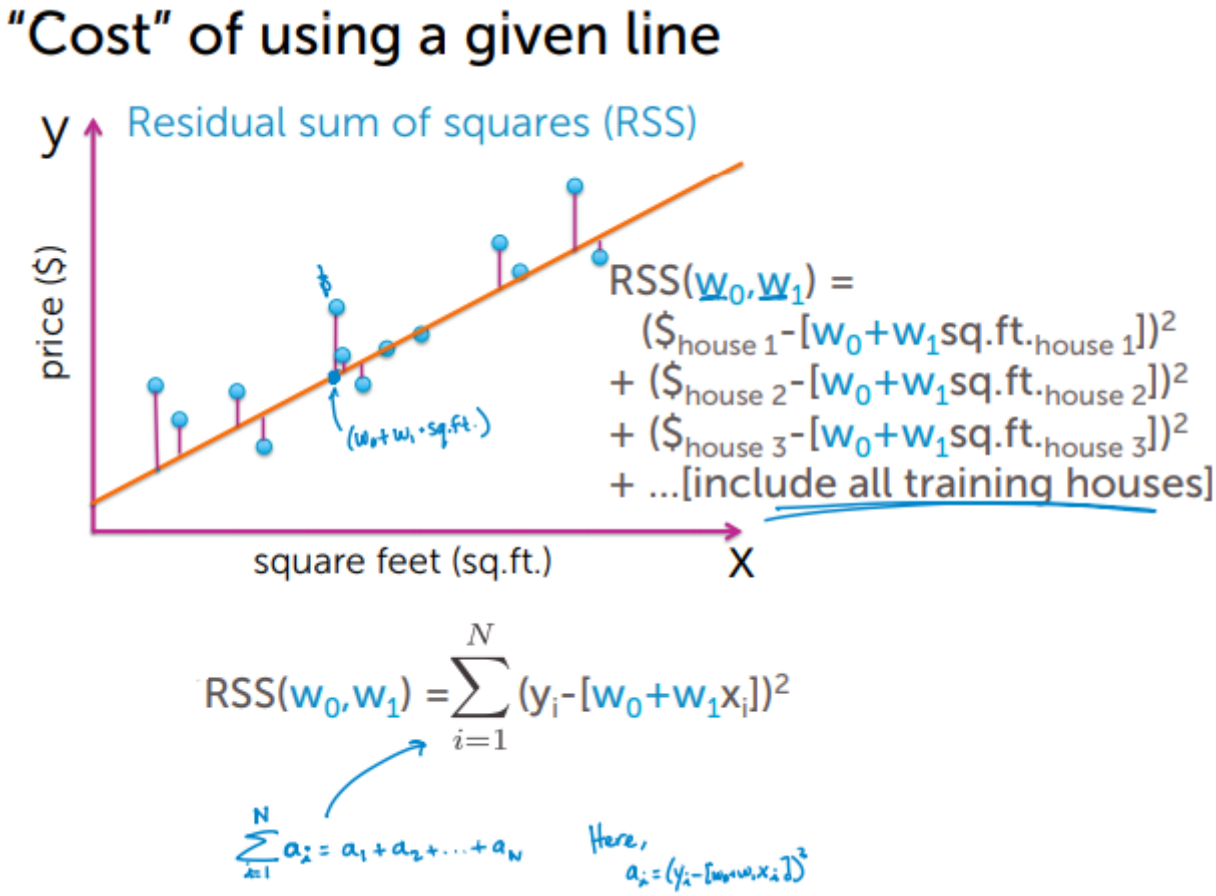
- One input(sqft) and One output(price)
- A line fit into the dataset.
- Equation of line = Intercept + slope *variable of interest; f(x) = w0 + w1*x
- Regression models for each observation yi = w0 + w1 * xi + εi;
- ε - epsilon - error term - distance from the line to the observation;
- w0 and w1 are parameters of regression coefficient.



B. Quality Metrics

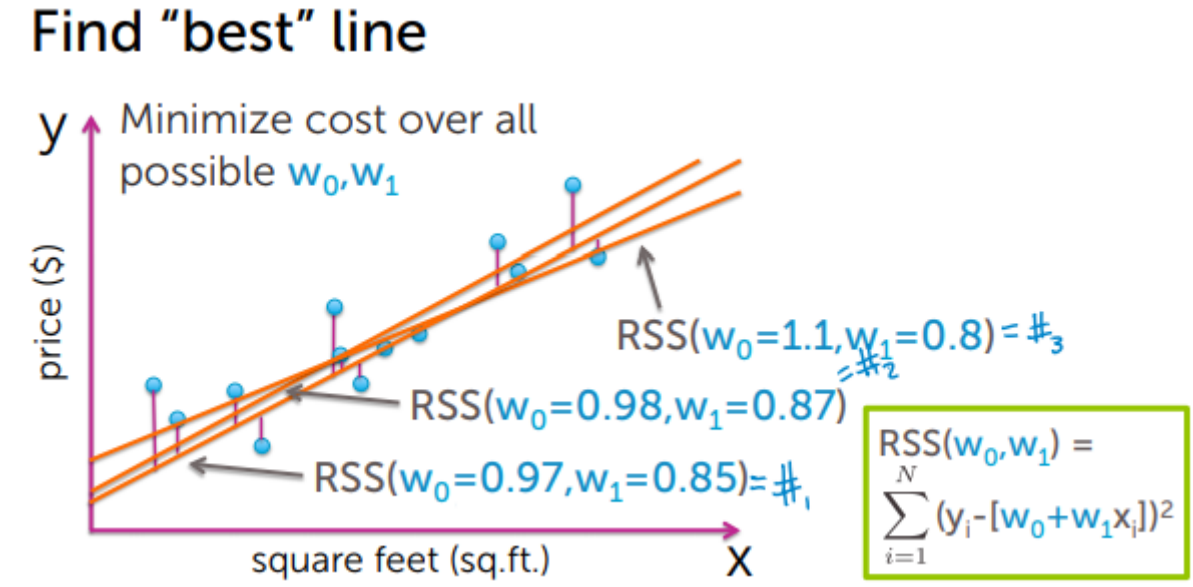
- The parameters of the simple linear regression model are w0 and w1 which can be represented as w(hat)/estimated parameters;
- There is a cost of using a particular line to fit the data.
- Residual Sum of Squares (RSS) : Add the errors between the observations and the line.

- Epsilon error is a part of the model; Residual is the difference betweem the prediction and the actual value.
- Its a function of two parameters w0 and w1;



Find the best line:

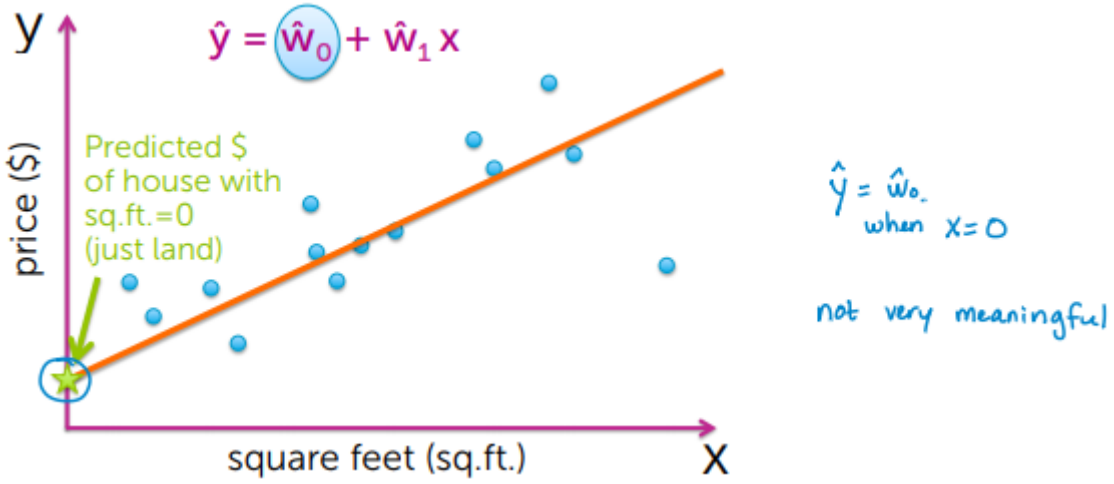
- For any given w0, w1 pair a spcific cost can be derived.
- Goal : Search over all possible lines and find the one with the smallest possible residual sum of squares.



Interpreting the coefficients

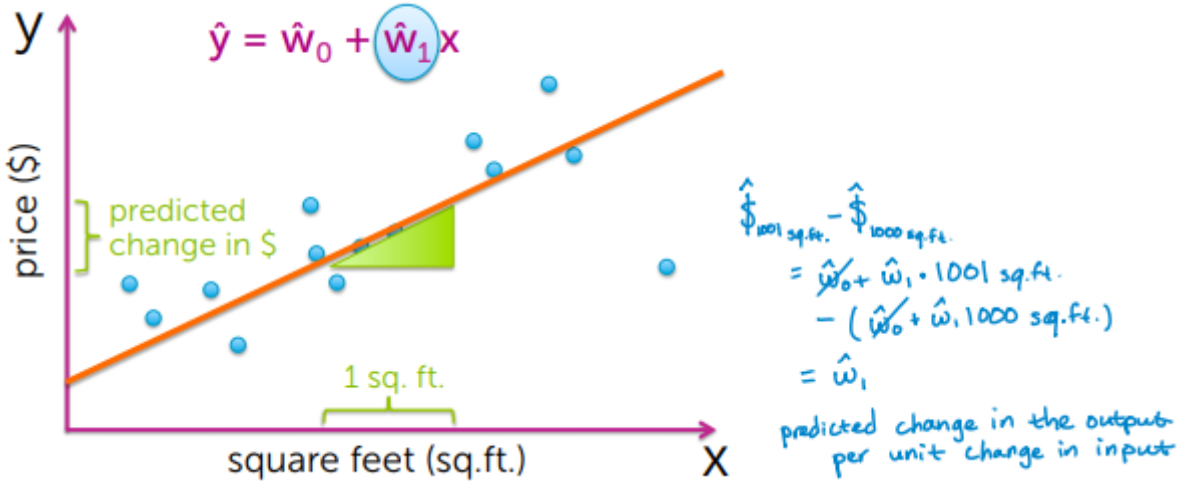
w(hat) 0 -> estimated intercept

- It indicates the predicted value of a house with 0 sqft (could be the price of the land);



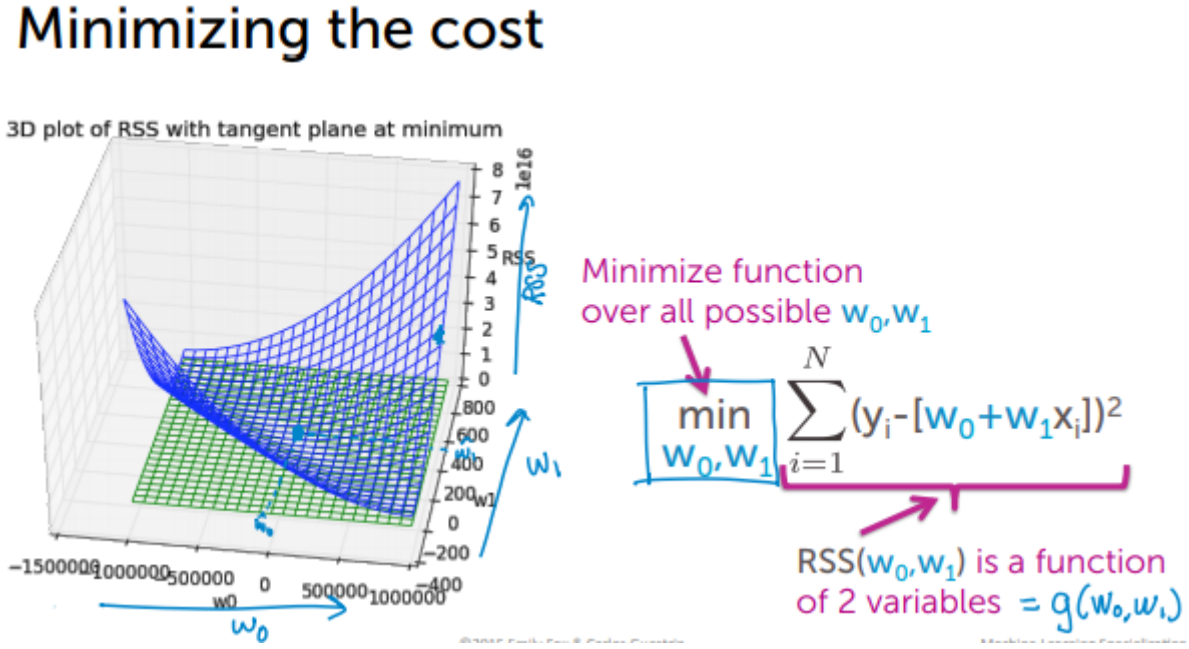
w(hat) 1 -> estimated slope

- per unit change in the input, what's the predicted change in the price.

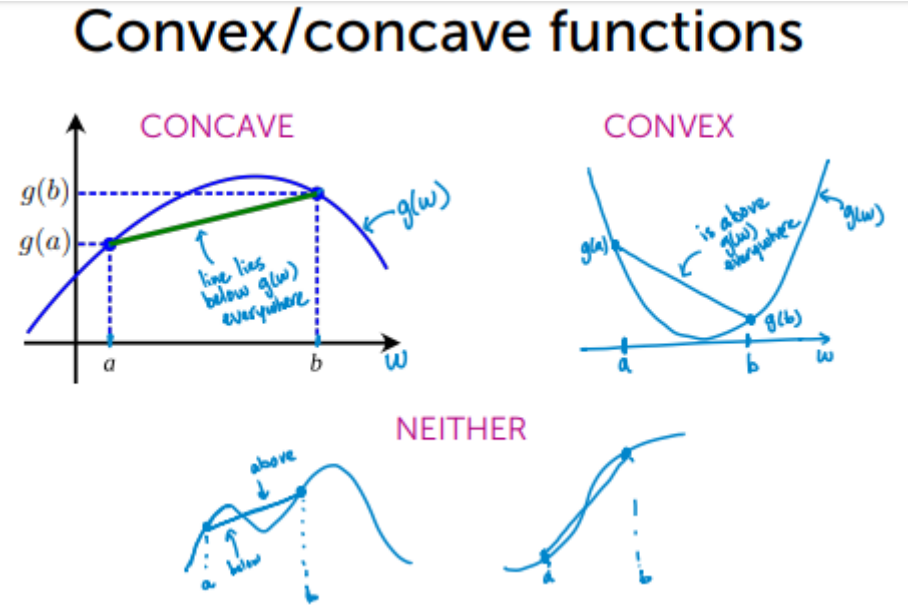


C. ML Algorithm

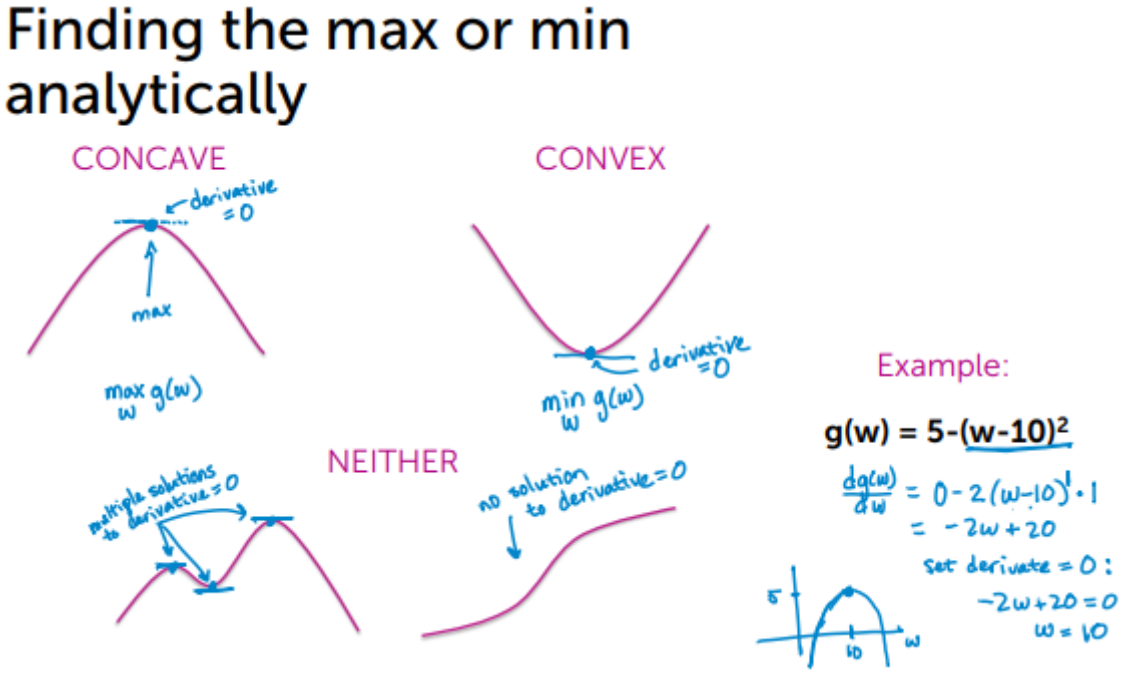
- Each model is fit with various lines, the line that best fits the data and minimizes the residual sum of squares (RSS) is the best fit line.



Optimization : Finding maxima / minima analytically



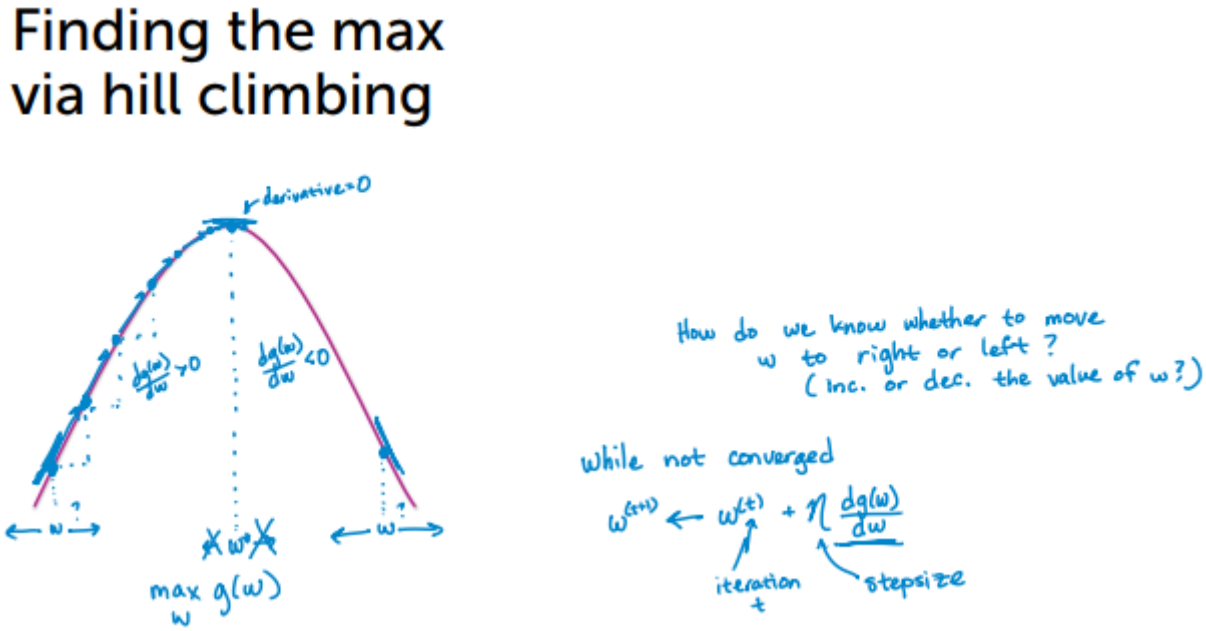
- concave -> maximum;
- convex -> minimum;



- The derivative = 0 will not implicitly tell us if it is a concave or a convex function, the second derivative of the function will provide us with that information.

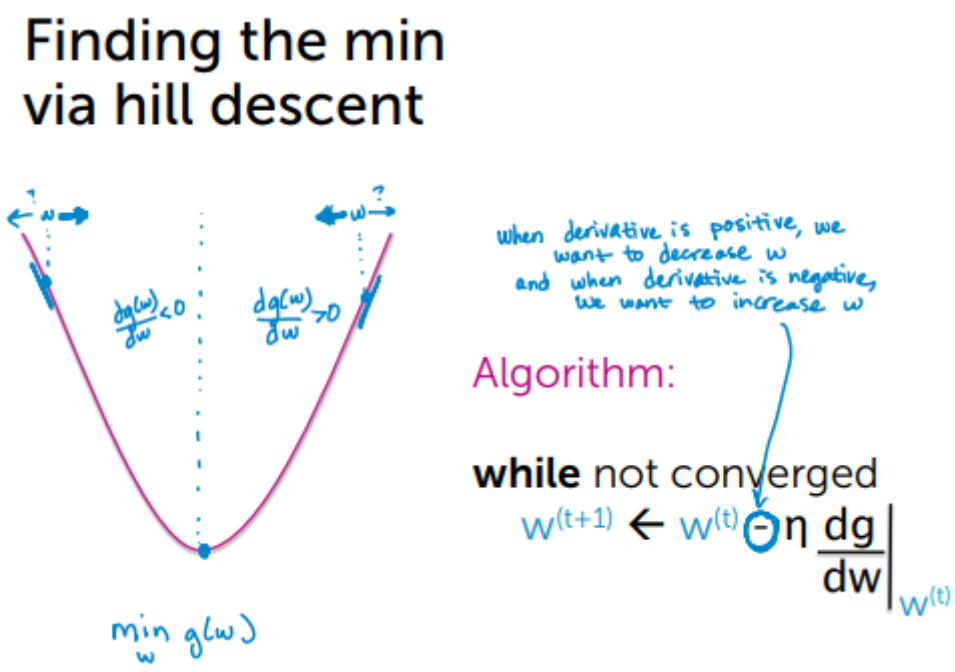
Finding the max via hill climbing algorithm :

- Taking the derivative via calculus ina means to find the min or max value.
- There are other means to find the min and max values - Hill climibing algorithm;
- Here we start somewhere in the space of all possible w, and keep changing w hoping to get closer and closer to the optimum.
- when you randomly choose w, and need to change it to reach the optimum then whether to move it to the right or left ?
- Look at the function and take the derivative - if it is positive then increase it or else decrease it.
- For a concave function -> left of the optimal the derivative increases, while on the right of the optimal the derivative decreases.



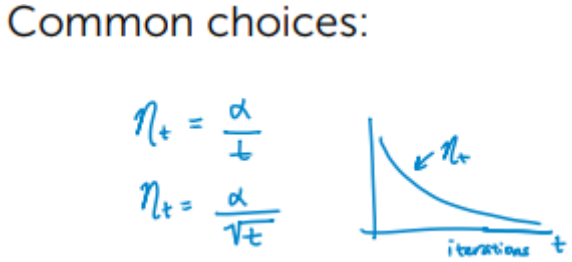
- Need to move the w value with eta(step-size) η ;

Finding the min via hill descent algorithm :



Step-Size : η

- Fixed step size - could jump to far, and takes long to find the optimum.
- Decrease step size with no off iteration. Step size schedule.
- Decrease step-size criteria



Convergence criteria :

- For a convex function, optimum occurs when the derivative is 0;
- In practice if the derivative is less than certain epsilon, then stop.

For convex functions, optimum occurs when

$$\frac{dg(w)}{dw} = 0$$

In practice, stop when

$$\left|\frac{dg(w)}{dw}\right| < \epsilon$$

↑ threshold to be set

Algorithm:

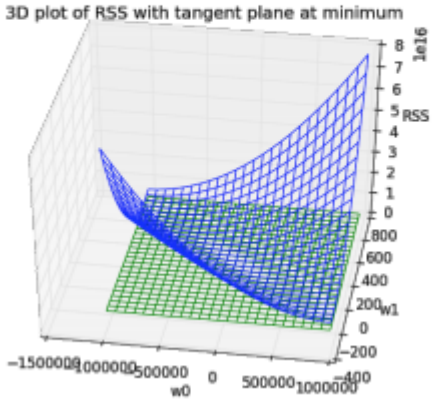
while not converged

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \left.\frac{dg}{dw}\right|_{w^{(t)}}$$

Gradients : Derivatives in multiple dimensions

- Thus far functions with just one variable were considered and their maximum and minimum were derived/found.
- Residual Sum of Squares (RSS) - takes two parameters - w0 and w1 - Gradients are needed for their derivatives.
- w -> vector of all the w's - w0, w1, w2,...,wn;
- gradient -> partial derivative of all the gradients;

Moving to multiple dimensions: Gradients



$$\nabla g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_0} \\ \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_p} \end{bmatrix}$$

gradient $[w_0, w_1, \ldots, w_p]$

(p+1)-dimensional vector

partial derivative is like a derivate with respect to w_i treating all other variables as constants

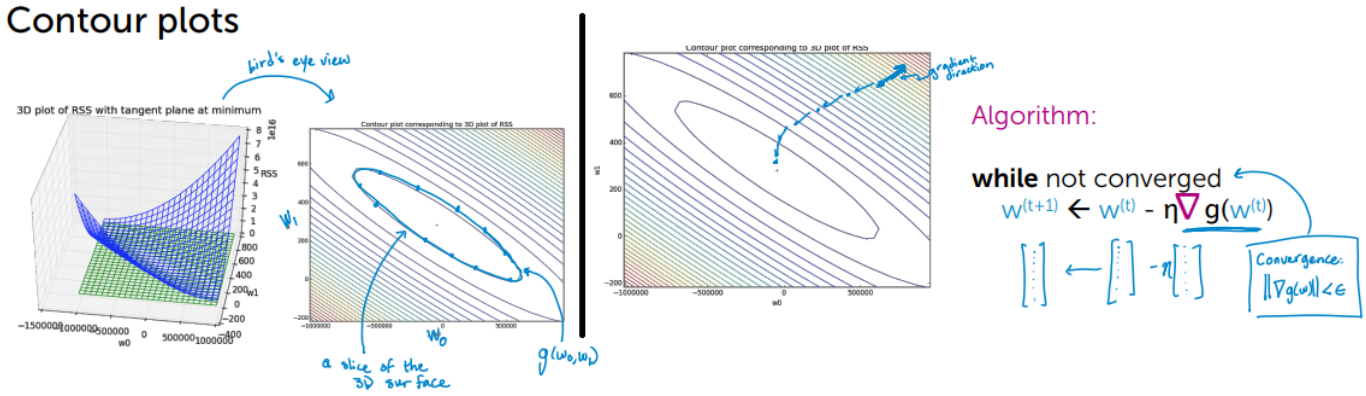
$$g(w) = 5w_0 + 10w_0w_1 + 2w_1^2$$

$$\frac{\partial g}{\partial w_0} = 5 + 10w_1$$

$$\frac{\partial g}{\partial w_1} = 10w_0 + 4w_1$$

$$\nabla g(w) = \begin{bmatrix} 5 + 10w_1 \\ 10w_0 + 4w_1 \end{bmatrix}$$

Contour Plots : A different visual view



- each curve is a slice through the 3-D mesh. Where all values along the elliptical curve has the same value of the function g(w0,w1)
- Inner elipses/rings in blue have lower value of the function and it keeps increasing outward; Red rings have higher values of the function.
- It is useful because it is a 2-d representation.

Gradient Descent via contour plots

- It is analogous to the hill descent algorithm.
- It's of the derivative of the function the gradient is specified.
- Algorithm: While not converged w(t+1) <- w(t) -η * ∇g(w(t))
- The gradient direction will be towards the higher function, i.e towards the red region. But the steps are gonna be in the opposite direction towards the optimal value - minima.
- Convergence to the function occurs : When the magnitude of the gradient < epsilon that is being fixed.
- convergence occurs := ||∇g(w)|| < ε

Finding the best line

Find the least square line

- Here a convex function is considered, therefore - the solution will be unique and the gradient descent algorithm will converge to this minimum.
- Minimize the cost over all possible w0, w1 is time-consuming using the gradient descent algorithm is more efficient;

Compute the Gradient Descent:

- Cost of the line : RSS for two parameters w0 and w1;

$$RSS(w_0,w_1) = \sum_{i=1}^{N} (y_i-[w_0+w_1x_i])^2$$

Aside:

$$\frac{d}{dw}\sum_{i=1}^{N} g_i(w) = \frac{d}{dw}(g_1(w)+g_2(w)+\dots g_N(w))$$
$$= \frac{d}{dw}g_1(w) + \frac{d}{dw}g_2(w) + \dots \frac{d}{dw}g_N(w)$$
$$= \sum_{i=1}^{N}\frac{d}{dw}g_i(w)$$

In our case
$g_i(w) = (y_i-[w_0+w_1x_i])^2$

$\frac{\partial RSS(w)}{\partial w_0} = \sum_{i=1}^{N}\frac{\partial}{\partial w_0}(y_i-[w_0+w_1x_i])^2$

same for w_1

Taking the derivative w.r.t. w_0

$\sum_{i=1}^{N} 2(y_i-[w_0+w_1x_i])^1\cdot(-1)$

$= -2\sum_{i=1}^{N}(y_i-[w_0+w_1x_i])$

Taking the derivative w.r.t. w_1

$\sum_{i=1}^{N} 2(y_i-[w_0+w_1x_i])^1\cdot(-x_i)$

$= -2\sum_{i=1}^{N}(y_i-[w_0+w_1x_i])x_i$

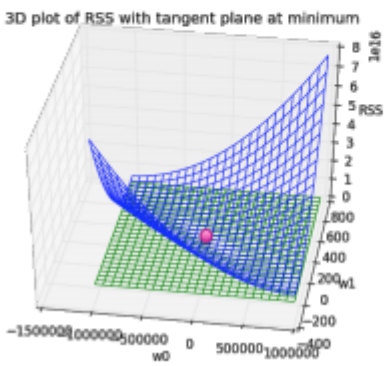
$$\nabla RSS(w_0,w_1) = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i-(w_0+w_1x_i)] \\ -2\sum_{i=1}^{N}[y_i-(w_0+w_1x_i)]x_i \end{bmatrix}$$

- Two approaches to find the minimum Approach 1 : set gradient = 0 and equate; Approach 2 : gradient descent

Approach 1 : set gradient = 0 and equate;

- The green surface is the optimum where the gradient = 0;
- Red dot is the optimum.

$$\nabla RSS(w_0,w_1) = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i-(w_0+w_1x_i)] \\ -2\sum_{i=1}^{N}[y_i-(w_0+w_1x_i)]x_i \end{bmatrix}$$



top term,

$\hat{w}_0 = \frac{\sum_{i=1}^{N} y_i}{N} - \hat{w}_1\frac{\sum_{i=1}^{N} x_i}{N}$

(average house price; estimate of the slope; average sq.ft.)

bottom term:

$\sum y_i x_i - \hat{w}_0\sum x_i - \hat{w}_1\sum x_i^2 = 0$

plug in

$\hat{w}_1 = \frac{\sum y_i x_i - \frac{\sum y_i \sum x_i}{N}}{\sum x_i^2 - \frac{\sum x_i \sum x_i}{N}}$

Note:

$\sum_{i=1}^{N} y_i$

$\sum_{i=1}^{N} x_i$

$\sum_{i=1}^{N} y_i x_i$

$\sum_{i=1}^{N} x_i^2$

Closed Form Solution - Approach 1 - Regression Parameters

The data

Consider the following 5 point synthetic data set:

	X	Y
1	0	1
2	1	3
3	2	7
4	3	13
5	4	21

top term: $\hat{w}_0 = \frac{\sum y_i}{N} - \hat{w}_1 \frac{\sum x_i}{N}$ (average house sales price; estimate the slope; average sq.ft.)

bottom term: $\sum y_i x_i - \hat{w}_0 \sum x_i - \hat{w}_1 \sum x_i^2 = 0$

plug in

$$\hat{w}_1 = \frac{\sum y_i x_i - \frac{\sum y_i \sum x_i}{N}}{\sum x_i^2 - \frac{\sum x_i \sum x_i}{N}}$$

```
numerator = (sum of X*Y) - (1/N)*((sum of X) * (sum of Y))
denominator = (sum of X^2) - (1/N)*((sum of X) * (sum of X))
```

- N = 5
- The sum of the Ys = 45
- The sum of the Xs = 10
- The sum of the product of the Xs and the Ys = 140
- The sum of the Xs squared = 30

So that:

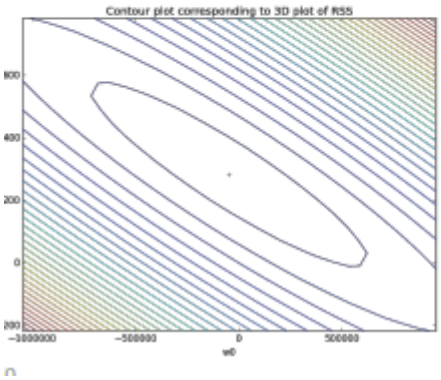
```
numerator = [(140) - (1/5) * (45*10)] = 50
denominator = [(30) - (1/5) * (10*10)] = 10

slope = 50/10 = 5
```

slope = 5, intercept = -1

```
intercept = (mean of Y) - slope * (mean of X)
intercept = 9 - 5 * 2 = -1
```

Approach 2 : Gradient Descent

$$\nabla RSS(w_0, w_1) = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0, w_1)] \\ -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0, w_1)]x_i \end{bmatrix}$$



while not converged $(-2)\cdot(-\eta)$

$$\begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \end{bmatrix} + 2\eta \begin{bmatrix} \sum_{i=1}^{N}[y_i - \hat{y}_i(w_0^{(t)}, w_1^{(t)})] \\ \sum_{i=1}^{N}[y_i - \hat{y}_i(w_0^{(t)}, w_1^{(t)})]x_i \end{bmatrix}$$

If overall, underpredicting $\hat{y}_i$, then $\sum[y_i - \hat{y}_i]$ is positive
→ w_0 is going to increase
similar intuition for w_1, but multiply by x_i

- yi -> actual observation.
- w0+w1x -> predicted value.

Interpreting the gradient:

actual house sales observation; predicted value $\hat{y}_i(w_0, w_1)$

$$\nabla RSS(w_0, w_1) = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - (w_0 + w_1 x_i)] \\ -2\sum_{i=1}^{N}[y_i - (w_0 + w_1 x_i)]x_i \end{bmatrix} = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0, w_1)] \\ -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0, w_1)]x_i \end{bmatrix}$$

Closed Form Solution - Approach 2 - Regression Parameters

- The derivative of the cost for the intercept is the sum of the errors
- The derivative of the cost for the slope is the sum of the product of the errors and the input
- Initial Values :
- initial_intercept = 0
- initial_slope = 0
- step_size = 0.05
- tolerance = 0.01

The algorithm

In each step of the gradient descent we will do the following:

1. Compute the predicted values given the current slope and intercept
2. Compute the prediction errors (prediction - Y)
3. Update the intercept:

```
compute the derivative: sum(errors)
compute the adjustment as step_size times the derivative -        decrease
 the intercept by the adjustment
w0(t+1) = w0(t) - η * Σyi; (η -> step size; Σyi -> predicted    errors sum)
```

4. Update the slope:

```
compute the derivative: sum(errors*input)
compute the adjustment as step_size times the derivative
decrease the slope by the adjustment
w0(t+1) = w0(t) - η * Σyi * xi;
```

5. Compute the magnitude of the gradient

```
sqrt((Σyi)^2 + (Σyi*xi)^2)
sqrt((square of the sum(error) + square of the sum(error * input)))
```

6. Check for convergence

```
magnitude < tolerance <- then converged else repeat the above steps.
```

The algorithm in action

First step:

Intercept = 0

Slope = 0

1. predictions = [0, 0, 0, 0, 0]

2. errors = [-1, -3, -7, -13, -21]

3. update Intercept

- sum([-1, -3, -7, -13, -21]) = -45
- adjustment = 0.05 * 45 = -2.25
- new_intercept = 0 - -2.25 = 2.25

4. update Slope

- sum([0, 1, 2, 3, 4] * [-1, -3, -7, -13, -21]) = -140
- adjustment = 0.05 * 45 = -7
- new_slope = 0 - -7 = 7

5. magnitude = sqrt((-45)^2 + (-140)^2) = 147.05

6. magnitude > tolerance: not converged

Second step:

Intercept = 2.25

Slope = 7

1. predictions = [2.25, 9.25, 16.25, 23.25, 30.25]

2. errors = [1.25, 6.35, 9.25, 10.25, 9.25]

3. update Intercept

- sum([1.25, 6.35, 9.25, 10.25, 9.25]) = 36.25
- adjustment = 0.05 * 36.25 = 1.8125
- new_intercept = 2.25-1.8125 = 0.4375

4. update Slope

- sum([0, 1, 2, 3, 4] * [1.25, 6.35, 9.25, 10.25, 9.25]) = 92.5
- adjustment = 0.05 * 92.5 = 4.625
- new_slope = 7 - 4.625 = 2.375

5. magnitude = sqrt((36.25)^2 + (92.5)^2) = 99.35

6. magnitude > tolerance: not converged

78th Step:

Intercept = -0.9937

Slope = 4.9978

1. predictions = [-0.99374, 4.00406, 9.00187, 13.99967, 18.99748]

2. errors = [-1.99374, 1.00406, 2.00187, 0.99967, -2.00252]

3. update Intercept

- sum([-1.99374, 1.00406, 2.00187, 0.99967, -2.00252]) = 0.009341224
- adjustment = 0.05 * 0.009341224 = 0.0004670612
- new_intercept = -0.9937 - 0.0004670612 = -0.994207

4. update Slope

- sum([0, 1, 2, 3, 4] * [-1.99374, 1.00406, 2.00187, 0.99967, -2.00252]) = -0.0032767
- adjustment = 0.05 *-0.0032767 = -0.00016383
- new_slope = 4.9978 --0.00016383 = 4.9979

5. magnitude = sqrt[()^2 + ()^2] = 0.0098992

6. magnitude < tolerance: converged!

Final slope: -0.994

Final Intercept: 4.998

```
Get the next set of predictions by inserting the
slope and intercept values
 y = mx+c
```

Comparing the approaches:

- For most ML problems, cannot solve gradient = 0 (approaach 1);
- Even if solving gradient = 0 is feasible, gradient descent (approach -2) can be more efficient.
- Gradient Descent relies on choosing "step-size" and "convergence-criteria".

Regression Demo - House Value v/s Crime Rate

In [1]:

```
import graphlab
```

Load some house value vs. crime rate data

Dataset is from Philadelphia, PA and includes average house sales price in number of neighbourhoods. The attributes of each neighbourhood we have include -> CrimeRate, MilesPhila(miles from city center), Name (town name), County.

In [2]:

```
sales = graphlab.SFrame.read_csv("Philadelphia_Crime_Rate_noNA.csv/");
```

```
This non-commercial license of GraphLab Create for academic use is assigned
to amitha353@gmail.com and will expire on May 07, 2019.

[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging:
C:\Users\Amitha\AppData\Local\Temp\graphlab_server_1531061875.log.0

Finished parsing file D:\UofWashington_Regression\Philadelphia_Crime_Rate_no
NA.csv

Parsing completed. Parsed 99 lines in 0.083122 secs.

------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[long,float,float,float,float,str,str]
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 D:\UofWashington_Regression\Philadelphia_Crime_Rate_no
NA.csv

Parsing completed. Parsed 99 lines in 0.067518 secs.
```

In [3]:

```
sales
```

Out[3]:

HousePrice	HsPrc ($10,000)	CrimeRate	MilesPhila	PopChg	Name
140463	14.0463	29.7	10.0	-1.0	Abington
113033	11.3033	24.1	18.0	4.0	Ambler
124186	12.4186	19.5	25.0	8.0	Aston
110490	11.049	49.4	25.0	2.7	Bensalem
79124	7.9124	54.1	19.0	3.9	Bristol B.
92634	9.2634	48.6	20.0	0.6	Bristol T.
89246	8.9246	30.8	15.0	-2.6	Brookhaven
195145	19.5145	10.8	20.0	-3.5	Bryn Athyn
297342	29.7342	20.2	14.0	0.6	Bryn Mawr
264298	26.4298	20.4	26.0	6.0	Buckingham

[99 rows x 7 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.

Explore the data

- The house price in a town in correlated with the crime rate of the town. Low crime towns tend to be associated with higher house prices and vice versa.

In [4]:

```
graphlab.canvas.set_target('ipynb')
sales.show(view="Scatter Plot", x="CrimeRate", y="HousePrice")
```

Fit the regression model using crime as the feature

In [5]:

```
crime_model = graphlab.linear_regression.create(sales,
                                                target='HousePrice',
                                                features=['CrimeRate'],
                                               validation_set=None,
                                               verbose=False)
```

Matplotlib is a Python plotting library that is also useful for plotting. You can install it with:

'pip install matplotlib'

In [6]:

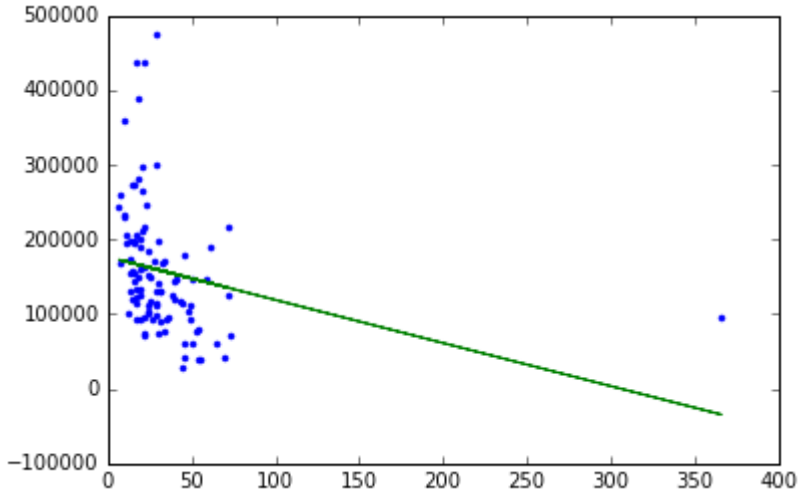
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [7]:

```
plt.plot(sales['CrimeRate'],sales['HousePrice'],'.',
        sales['CrimeRate'],crime_model.predict(sales),'-')
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x2b0ba748>,
 <matplotlib.lines.Line2D at 0x2af3bf28>]
```



Remove Center City and redo the analysis

Center City is the one observation with an extremely high crime rate, yet house prices are not very low. This point does not follow the trend of the rest of the data very well. A question is how much including Center City is influencing our fit on the other datapoints. Let's remove this datapoint and see what happens.

In [8]:

```
sales_noCC = sales[sales['MilesPhila'] != 0.0]
```

In [9]:

```
sales_noCC.show(view="Scatter Plot", x="CrimeRate", y="HousePrice")
```

Refit our simple regression model on this modified dataset:

In [10]:

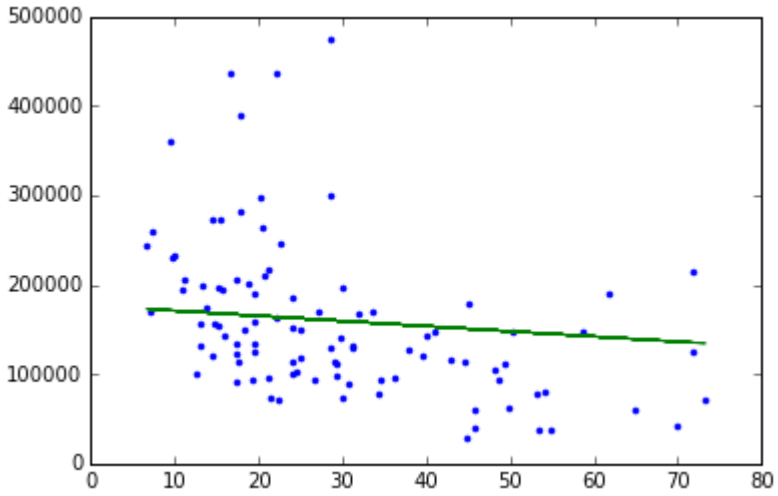
```
crime_model_noCC = graphlab.linear_regression.create(sales_noCC, target='HousePrice', featu
```

In [11]:

```
plt.plot(sales_noCC['CrimeRate'],sales_noCC['HousePrice'],'.',
        sales_noCC['CrimeRate'],crime_model.predict(sales_noCC),'-')
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x2b39c160>,
 <matplotlib.lines.Line2D at 0x2b019fd0>]
```



Compare coefficients for full-data fit versus no-Center-City fit

Visually, the fit seems different, but let's quantify this by examining the estimated coefficients of our original fit and that of the modified dataset with Center City removed.

In [12]:

```
crime_model.get('coefficients')
```

Out[12]:

name	index	value	stderr
(intercept)	None	176626.046881	11245.5882194
CrimeRate	None	-576.804949058	226.90225951

[2 rows x 4 columns]

In [13]:

```
crime_model_noCC.get('coefficients')
```

Out[13]:

name	index	value	stderr
(intercept)	None	225204.604303	16404.0247514
CrimeRate	None	-2287.69717443	491.537478123

[2 rows x 4 columns]

Above: We see that for the "no Center City" version, per unit increase in crime, the predicted decrease in house prices is 2,287. In contrast, for the original dataset, the drop is only 576 per unit increase in crime. This is significantly different!

- With center city -> $576/crime
- Without center city -> $2287/crime

High leverage points:

Center City is said to be a "high leverage" point because it is at an extreme x value where there are not other observations. As a result, recalling the closed-form solution for simple regression, this point has the *potential* to dramatically change the least squares line since the center of x mass is heavily influenced by this one point and the least squares line will try to fit close to that outlying (in x) point. If a high leverage point follows the trend of the other data, this might not have much effect. On the other hand, if this point somehow differs, it can be strongly influential in the resulting fit.

Influential observations:

An influential observation is one where the removal of the point significantly changes the fit. As discussed above, high leverage points are good candidates for being influential observations, but need not be. Other observations that are *not* leverage points can also be influential observations (e.g., strongly outlying in y even if x is a typical value).

Remove high-value outlier neighborhoods and redo analysis

Based on the discussion above, a question is whether the outlying high-value towns are strongly influencing the fit. Let's remove them and see what happens.

In [14]:

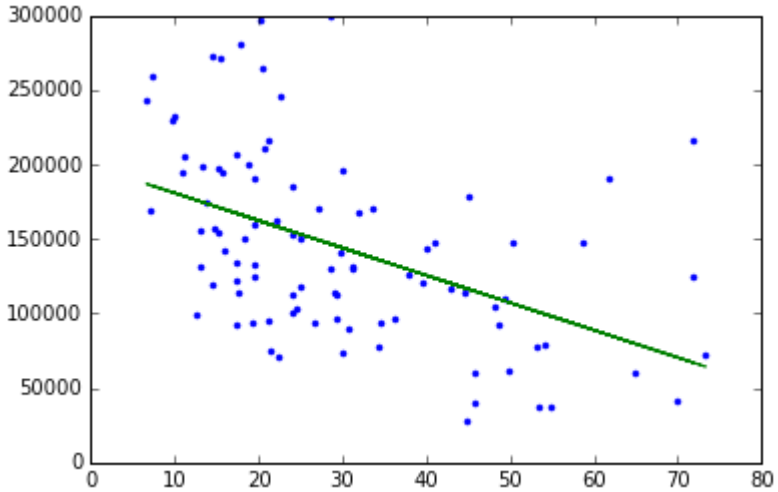
```
sales_nohighend = sales_noCC[sales_noCC['HousePrice'] < 350000]
crime_model_nohighend = graphlab.linear_regression.create(sales_nohighend, target='HousePri
```

In [17]:

```
plt.plot(sales_nohighend['CrimeRate'],sales_nohighend['HousePrice'],'.',
        sales_nohighend['CrimeRate'],
         crime_model_nohighend.predict(sales_nohighend),'-')
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x2ef2c7b8>,
 <matplotlib.lines.Line2D at 0x2ec47c18>]
```

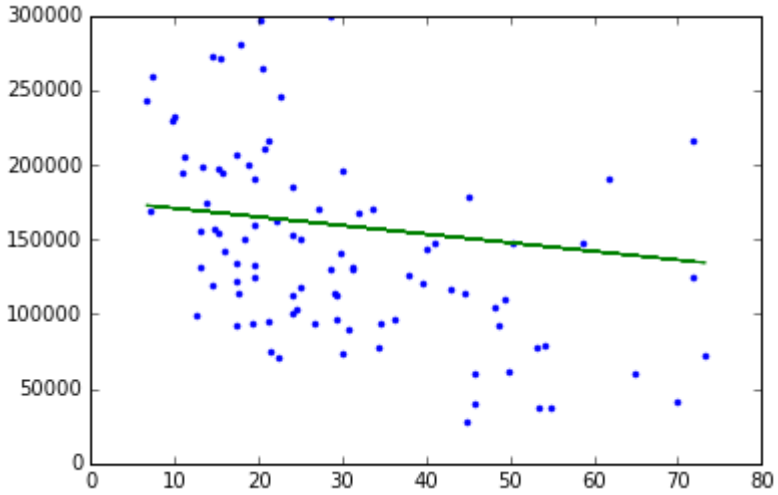


In [18]:

```
plt.plot(sales_nohighend['CrimeRate'],sales_nohighend['HousePrice'],'.',
        sales_nohighend['CrimeRate'],
         crime_model.predict(sales_nohighend),'-')
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x2f20be48>,
 <matplotlib.lines.Line2D at 0x2f20bef0>]
```



Do the coefficients change much?

In [19]:

```
crime_model_noCC.get('coefficients')
```

Out[19]:

name	index	value	stderr
(intercept)	None	225204.604303	16404.0247514
CrimeRate	None	-2287.69717443	491.537478123

[2 rows x 4 columns]

In [20]:

```
crime_model_nohighend.get('coefficients')
```

Out[20]:

name	index	value	stderr
(intercept)	None	199073.589615	11932.5101105
CrimeRate	None	-1837.71280989	351.519609333

[2 rows x 4 columns]

Above: We see that removing the outlying high-value neighborhoods has *some* effect on the fit, but not nearly as much as our high-leverage Center City datapoint.

Cost Function:

There are various cost function -> RSS (residual sum of squares), symmetric cost function, asymmetric cost function, etc

Symmetric Cost functions:

- The RSS is a symmetric cost function.
- It's cause overestimating the cost of house is same as underestimating the cost.

Asymmetric Cost functions:

- What if overestimatting and underestimating a factor has different influences, etc?
- Consider for a house: What is the cost of listing the house too high?

```
    - Too High - no offers($=0)

    - Too Low - offers for lower $
```

Quiz

Assume you fit a regression model to predict house prices from square feet based on a training data set consisting of houses with square feet in the range of 1000 and 2000. In which interval would we expect predictions to do best?

- ( ) [0, 1000]
- (•) [1000, 2000]
- ( ) [2000, 3000]

```
* The model predicts best in the same interval it was trained on.
```

2.
In a simple regression model, if you increase the input value by 1 then you expect the output to change by:

- ( ) Also 1
- (•) The value of the *slope* parameter
- ( ) The value of the *intercept* parameter
- ( ) Impossible to tell

```
* Intercept -> sum of error;
* Slope -> sum(error * input);
```

3.
Two people present you with fits of their simple regression model for predicting house prices from square feet. You discover that the estimated intercept and slopes are exactly the same. This necessarily implies that these two people fit their models on *exactly* the same data set.

- ( ) True
- (•) False

4.

You have a data set consisting of the sales prices of houses in your neighborhood, with each sale time-stamped by the month and year in which the house sold. You want to predict the average value of houses in your neighborhood over time, so you fit a simple regression model with average house price as the output and the time index (in months) as the input. Based on 10 months of data, the estimated intercept is $4569 and the estimated slope is 143 ($/month). If you extrapolate this trend forward in time, at which time index (in months) do you predict that your neighborhood's value will have doubled **relative to the value at month 10**? (Round to the nearest month).

52

```
* y = mx + c => price = slope * months + intercept
* Finding the price for 10 months:
price = 143 * 10 + 4569 = 5999;
* Find the month at which the price doubles
x = (y - c) / m
x = (2 * 5999 - 4569) / 143 = 51.95 ~ 52 months
```

5.

Your friend in the U.S. gives you a simple regression fit for predicting house prices from square feet. The estimated intercept is -44850 and the estimated slope is 280.76. You believe that your housing market behaves very similarly, but houses are measured in square meters. To make predictions for inputs in square meters, what **intercept** must you use? Hint: there are 0.092903 square meters in 1 square foot.

(Note: the next quiz question will ask for the slope of the new model.)

-44850

```
* Changing the unit of the input doesn't change the intercept, since intercept is
 calculated when the input(x) = 0;
```

6.

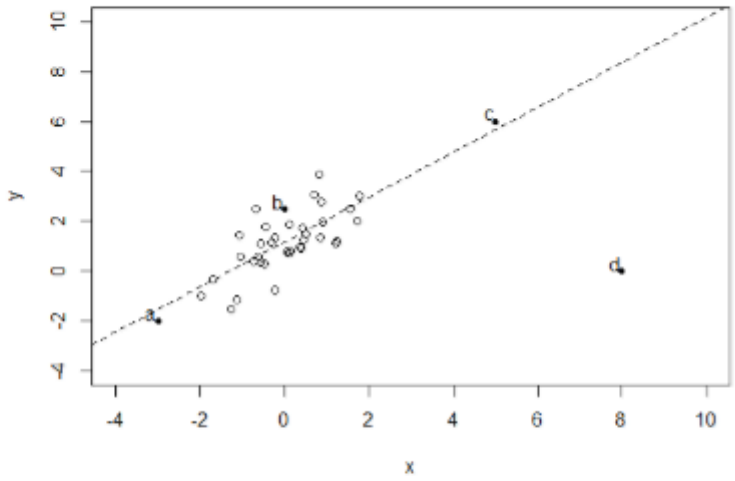
Your friend in the U.S. gives you a simple regression fit for predicting house prices from square feet. The estimated intercept is -44850 and the estimated slope is 280.76. You believe that your housing market behaves very similarly, but houses are measured in square meters. To make predictions for inputs in square meters, what **slope** must you use? Hint: there are 0.092903 square meters in 1 square foot.

3022

```
* the slope -> $280.76 / sqft; To convert sqft to sqmt

* the slope -> $280.76 / (0.092903) sqmt;

* 3022
```

1 point

7. Consider the following data set:



Which bold/labeled point, if removed, will have the largest effect on the fitted regression line (dashed)?

- a
- b
- c
- d

- Point D is the outlier and so if it is removed the fitted line could change.