The below model should not be considered as an advanced tool for inventory selection. Please consider leveraging backpropagation SNN (Standard Neural Network), or better RNN (Recurrent Neural Network). Both NN solutions will remove the constrain on number of feature input, and the concern on misfit hypothesis. (+ Numpy Library enables same level if not better vectorization computations over Octave. So, use Python) – Update on May 1, 2018

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

Model has since been updated, and ingests twenty-two features from DB (redshift). - Updated on March 5, 2018

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

**A Prediction Tool for AbeBooks Rare Product-Feed Selection Using “Batch Gradient Descent” and “Normal Equation” in a Machine Learning Framework**

joshua huang

joshuang@amazon.com

Feb-13, 2018

1. Abstract

In this document, I present a program and its underlying logic which finds two sets of parameters that best represent the relation of five selected NBC (Non-book Collectible) inventories’ attributes and their gross merchandise sales (GMS). In tandem, the new parameters are used for GMS prediction.

This program has only been tested with dummy data as illustrated in this document and is yet to produce actionable insights. Nonetheless, it is fully built to ingest real data files once available in the AbeBooks’ database; furthermore, it can be modified for other use cases, for instance, the weightings for twenty attributes, existing ISBN-based advertising feed, marketing channel budget allocation and etc.

In this document, the depth of the mathematical technicality, the selection of the algorithms to fit the ML framework, namely batch gradient descent and normal equation, should be greatly challenged. As it is outlined in the “next step” section, other solutions such as polynomial fitting, logistics and non-linear regressions should be sought after in the meantime while we learn more about the data and the performance challenges.

1. Prep and Basics (1)

Please **SKIP** this section IF: you are familiar with linear regression, and basic supervised machine learning intuition. This section is long.

Let’s say: Joshua is a stockbroker and he wants to decide whether he should buy Amazon stock. He pulled data from 1,000 US stocks and for each stock he has the 2 pieces of information: one is the [P/E ratio](https://www.investopedia.com/terms/p/price-earningsratio.asp) of the stock (at the specific time for all stocks), and two is the ROI 52 weeks after. Joshua plots them into a graph (fictitious number and graph below):

|  |  |  |
| --- | --- | --- |
| stock symbol | P/E | ROI (52 weeks after) |
| BNO | 99.15 | 21.8% |
| WMT | 39.96 | 13.7% |
| INTC | 49.82 | 29.3% |
| FISV | 35.58 | 29.3% |
| TSLA | 71.15 | 37.4% |
| NFLX | 47.83 | 42.0% |
| TWTR | 27.10 | 1.1% |
| APPL | 74.18 | 40.8% |
| DIS | 68.96 | 3.2% |
| … | 39.46 | 7.5% |

Joshua uses these data to create a linear algorithm to predict Amazon stock’s ROI 52 weeks from now given its P/E today. This process or problem is a regression problem. It is used to predict a real-valued output; simply put we want to predict a value, such as E%O or number of clicks or here the ROI of Amazon stock. It is in contrast to a classification problem where we will want to predict (most of the time) a binary output, for example, YES {1} for a rare-sale, or NO {0} for a textbook or trade sale.

To create this algorithm, Joshua uses the most basic machine learning process:

1. Extract data, e.g. from Nasdaq public data source, or Amazon Redshift database. And we call them the “training set”.
2. Pick a starting algorithm, or a hypothesis (term used by most early day machine learning practitioners, it does not mean anything fancy). The simplest hypothesis (univariate linear regression) can be y = x \* a + b, where y can be ROI, and “a”, “b” is “*parameters*”, and x represents input values such as a P/E ratio for Tesla Motors.
3. Pick a learning algorithm, e.g. in the later sections of this paper, we use Gradient Descent.
4. Feed “training sets” into the learning algorithm to refine the “*parameters*” in the hypothesis.
5. 1) Test how accurate the hypothesis is using known data outside of the training set. 2) if confident, Input a

new value, and use it for prediction.

Now, if this is confusing, don’t worry, it is likely my explanation is bad. Everything to this point is simple. Here we further explain what is “training” or “refining parameters”:

We can use a line (e.g. the black and red lines below) to try to best represent the correlation between p/e and ROI. But which one is better, the red or black or another? We judge it by saying, for every point on the “line”, we want it to be as close as to what the existing data (training set) is. Therefore, this becomes a minimization problem, where we want to minimize | (h(x) – y) |, or [(h(x) – y)]2. Here h(x) is the output of an algorithm (or hypothesis) for a given x. And y is the existing ROI given the existing x in the training set. For example, when we have Tesla Motor in our training set, and the p/e for Tesla is 71.15, this 71.15 is the x. And we know the y (ROI) for Tesla is 37.4%. h(x) is the algorithm we have for red line, and it can be h(x) = 71.15 \* 0.005 + 0.001, which outputs 35.7%. Here 0.005 and 0.001 are “parameters” and later we use to represent them.

However Tesla is only one example (data point), the minimization needs to cover all data points. The minimization goal thus becomes min{. This is to minimize the sum of the discrepancies (between a line and the reality) for each x. Here “m” is 1,000 ( the number of stocks Joshua decided to use for considerations).

Now we’ll quickly morph this minimization problem into a “cost function”. Before we do so, let me reiterate what we are doing at this point: we are trying to find the right “parameters” (that live in an algorithm, for example, the a, b, or 0.005, 0.001 above, or the in the later texts) to minimize the gaps between the output produced by the algorithm and the reality represented by the data we collected.

And this process is one way to explain what a supervised machine learning process is like. I kept using the term “supervised”, and here is what is means: in contrast to unsupervised, “supervised” simply means that we know what successes look like, or we know what we are looking for. For example, “make a sale” is better than “not make a sale”. Or in the prior text, high ROI is better than low ROI for a stock. “Unsupervised” sometimes called “clustering” means we do not know what we are looking for or at, so we build an algorithm to find patterns or potential insights.

Keep reading, it will become clearer.

1. Prep and Basics (2) – Cost function

Please **SKIP** this section IF you know how a cost function plays a role in simple linear regression model.

This section is long.

With minimal modification from the above minimization problem mentioned, a **cost function** looks like:

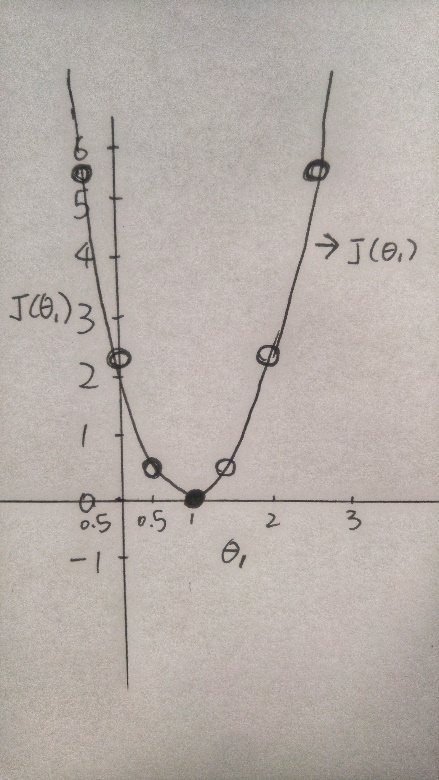
Let me visulize it so it will make more sense:

We have a data set: when p/e = 1, ROI = 1; when p/e = 2, ROI = 2; when p/e = 3, ROI = 3.

Joshua wrote down a baby “algorithm”: , if the greek symbol is hard to read, read it as y = a \* x. Note this is the same as

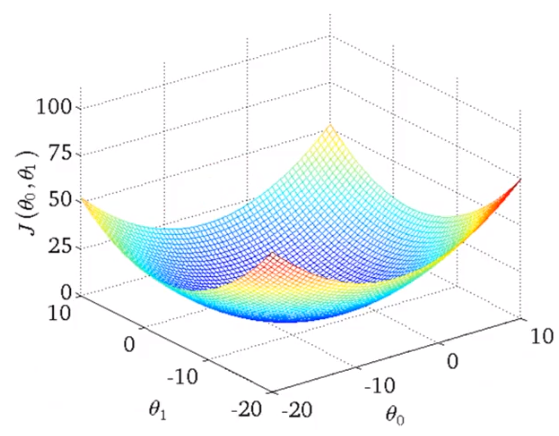
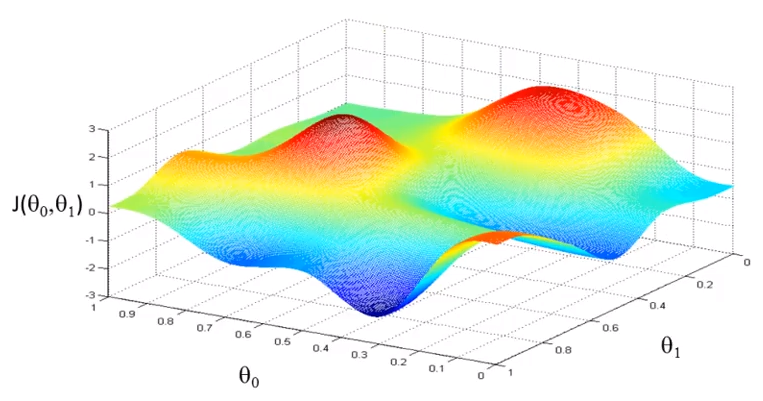
The cost function for this baby algorithm becomes or . (I will stop using “a” soon)

Now we plot three graphs, 1) the data set, 2) the baby algorithm(s), using these values for 0.5 (orange line), 2(blue line), 3(black line), -1(green line), and 3) its cost function (hand-drawn).



Res ipsa loquitur the minimal cost function value is achieved when = 1. When = 1, the algorithm (hypothesis) is y = x, or , and the cost function . And this fits our data set (when p/e = 1, ROI = 1; when p/e = 2, ROI = 2…) the best compared to any other algorithm with different .

Now we will add back (, so there are two “features’ or considerations. I hand-drew the visualization for a cost function with one feature above –-- a 2-D bell shape. For 2 features the cost function can look like the graph below on the left(linear), and if the hypothesis we picked is not non-linear, the cost function can look like the graph below on the right(non-linear), something I need to use a machine to draw:

(linear) (non-linear)

For the graph on the left, visually we can still easily identify that there is a lowest point in the cost function (the bottom of this bowl shape). That particular point is reached or represented by one combination of . But we no longer can use our eyes to find the exact numbers. For the graph on the right, visually we cannot even identify where the lowest point is. Now imagine we have a long list of , for example, BMS, BSA, Price, Seller\_Region and etc.

We need another algorithm, or a learning algorithm to help us find it(them).

1. Gradient Descent

Before we move on, here is a checkpoint on what we are doing.

* We want to build an algorithm that best represents how known input variables (“features”, or “” in a hypothesis) influence the known outputs. Thus we can use this algorithm to make predictions when we are given new input variables. In the scope of this paper, this algorithm is linear. From this point onwards, we call this algorithm a hypothesis. It is written like this: , where x0 = 1.
* We do so by finding the best parameters. Parameters are represented as . We believe when the Cost Function of this linear hypothesis against the known data set is minimized, that particular set of parameters are the best parameters.
* We need a learning algorithm to help us minimize the Cost Function, and therefore find the best parameters for the hypothesis. In this paper, we chose Gradient Descent as this learning algorithm.

The gradient descent algorithm is:

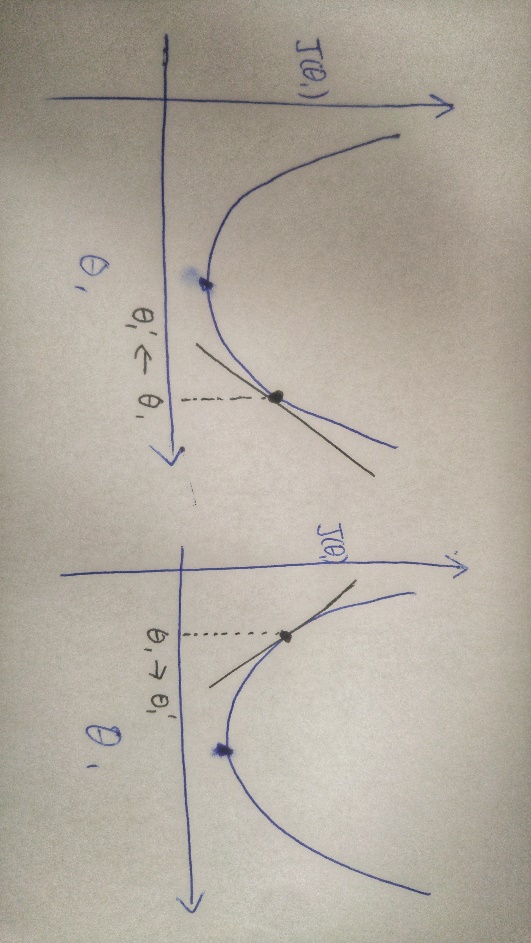
“repeat unitl convergence:

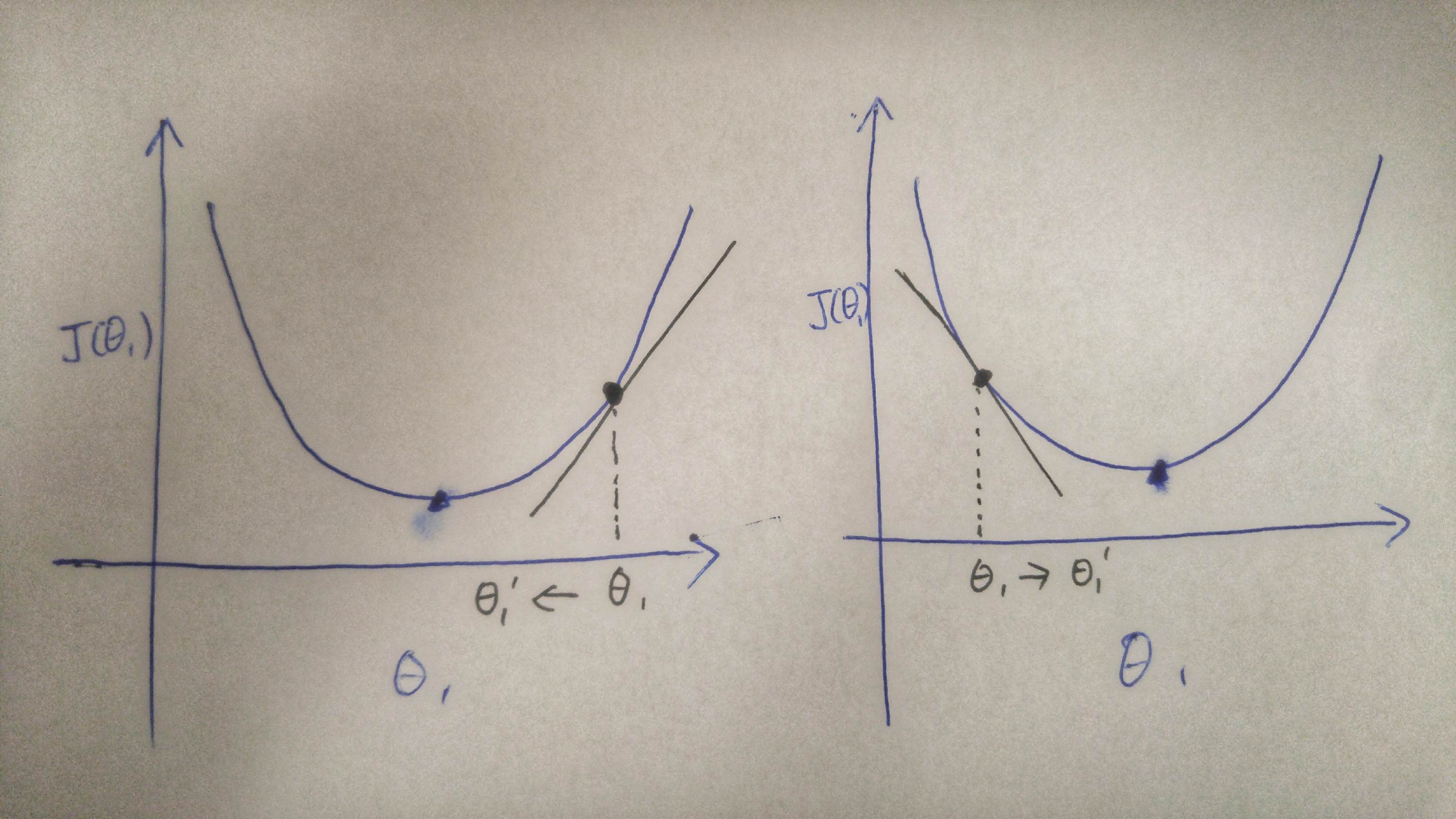
“

This algorithm comes out of nowhere and could look daunting. Let me explain how it will work on our case as intuitively as possible.

The sign “*:=*” represents an “assignment”, whereas “=” is an assertion. If X1 = 2, and X3 = 7, and we “do” X1 := X2, then X1 = 7. If we do X1 = X2, then we are making an assertion that X1 equals to X2, and this will return a “false”. – X means we assign the new value of to itself after each repetition. It’s a self-adjusting process. is the cost function for the studied hypothesis, if you are not familiar with this term, please refer to the earlier section of this paper.

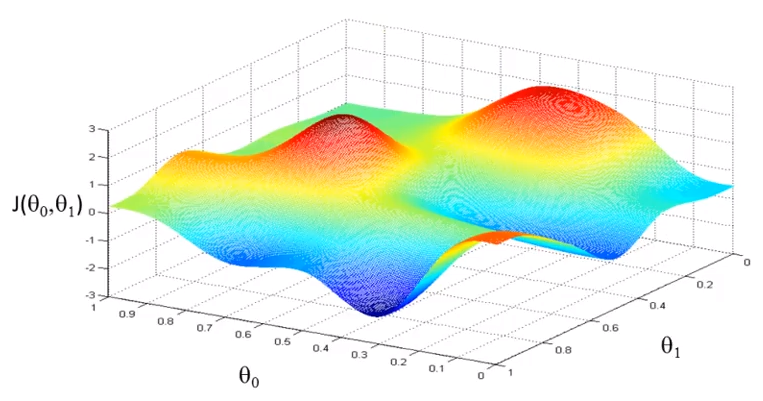
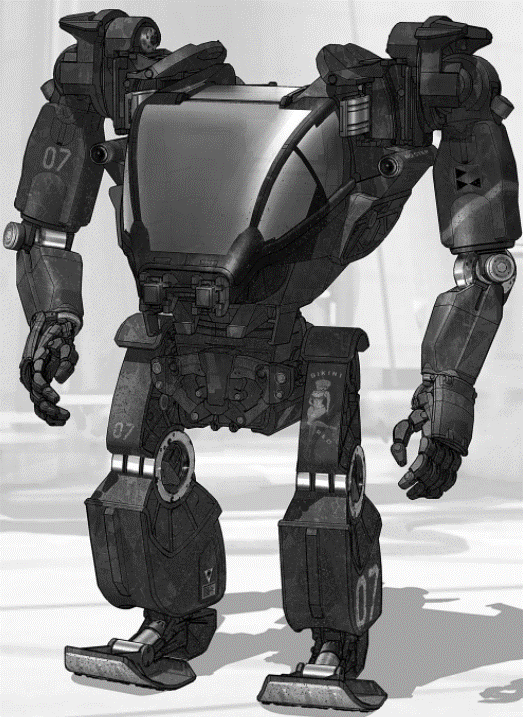
I plot a Cost Function of a hypothesis below (figure: GD1) and arbitrarily picked a for in the hypothesis; for example, let’s say represents “how many times this product has been viewed on the product page”, and we think it is important so we assign an initial value of 5 for . And is the [derivative](https://en.wikipedia.org/wiki/Derivative) of the cost function with respect to , and all it does is to take the tangent of this point ( and calculates the slop (black straight line). Here the slop is a positive number(when increases, increases, and vice versa). And in the gradient descent algorithm is a “learning rate factor”, and is always a positive value. Therefore, will drive to a smaller value (see ), or intuitively speaking will take a step to the left. Visually you can see that it is moving towards the lowest point.

(figure: GD1) (figure: GD2)



Now let’s assume the value of 5 for is not on the right-hand side of the lowest point for the Cost Function (like on figure GD1) but on the left-hand side. (figure: GD2) Because now the slop is a negative value,or < 0. will drive to a larger value (see ), or intuitively speaking will take a step to the right. Visually you can see that it is moving towards the lowest point as well.

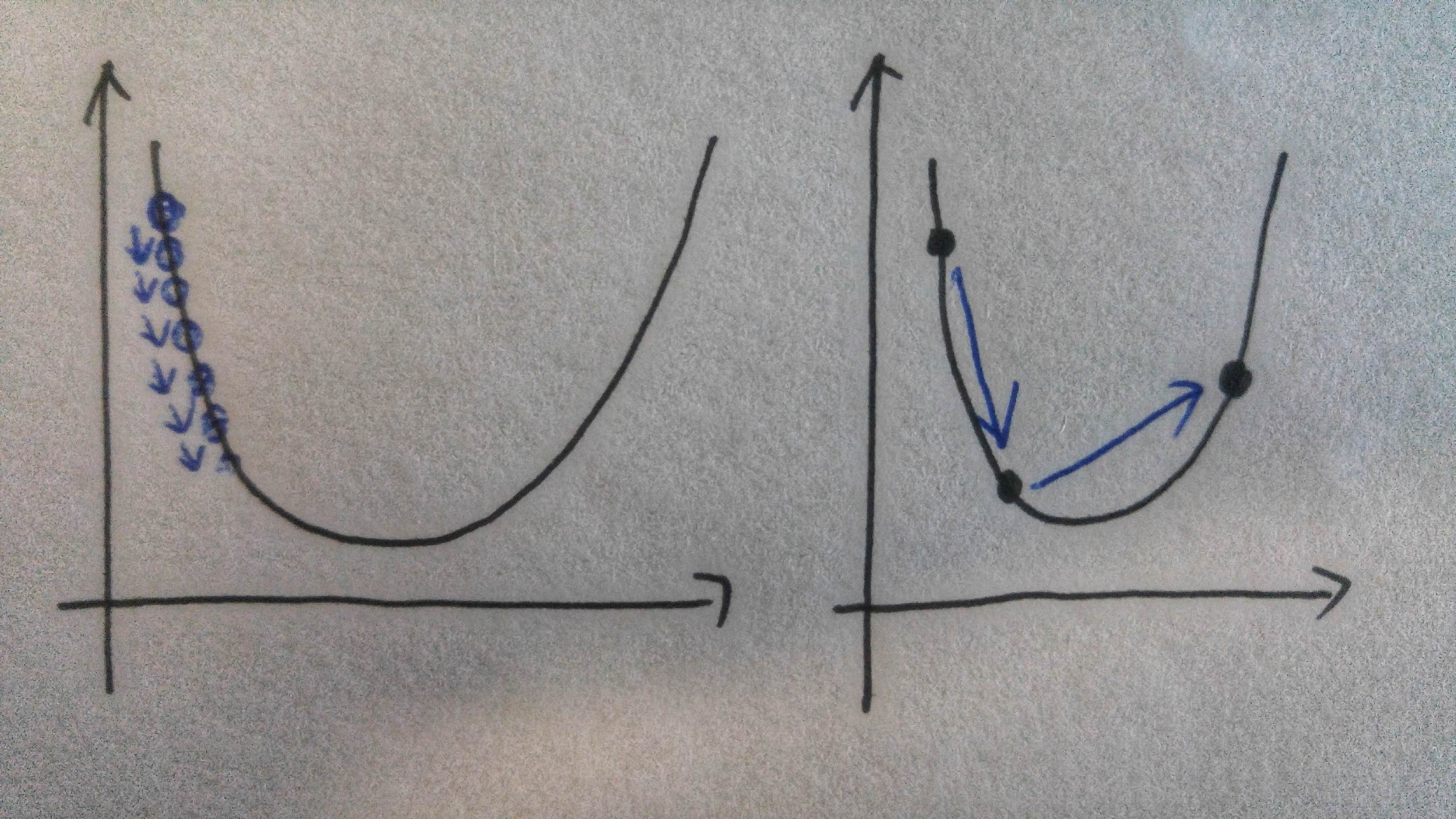
And this is how Gradient Descent works to minimize Cost Function by driving parameters to the lowest point (or normally called the Local Optimal Point). When there are more features or considerations built in for the hypothesis, the cost function will get more complicated, such as the one below (figure: GD3). Gradient Descent will function the same way: imagine “figure:GD3” is a representation of hills and valleys (red is the top of the hill, blue is deep valley), and a is randomly dropped somewhere in the area, Gradient Descent is an [AMP suit](http://james-camerons-avatar.wikia.com/wiki/Amplified_Mobility_Platform) machine (figure: GD4) that will carry and walk to the point in the area that is with the lowest elevation. Each time before AMP suit takes a stride, he will look around and find that one direction at which his next stride will create the maximum reduction in elevation.

(figure: GD3)  (figure: GD4)

Probable questions to be asked at this point (FAQ):

Q: How does learning rate matter, and how does it work?

is the learning rate and it influences how big a “step” gradient descent takes. When it is small, it may take gradient descent a great number of repetitions till it finds the lowest point. But when it is too large it may cause gradient descent to overshoot the lowest point. (see illustrations below)



Q: Are the steps Gradient Descent take equally paced?

No, when is closer to the local optimal, the “slope” is least steep and thus the will become smaller, and intuitively gradient descent will start taking smaller steps.

Q: What happens if the first I picked is already the lowest point (local/or global optimal)? Is Gradient Descent going to move it too?

The derivative of the cost function with respect to will be zero, and thus gradient descent will do nothing.

1. AbeBooks’ Rare Feed - Multiple Features & Hypothesis Set up

|  |  |
| --- | --- |
| **Features We Have** | **Notation** |
| Association |  |
| Seller Rating |  |
| Wants Match |  |
| BDP Views |  |
| Basket Adds |  |
| GMS |  |

Notation:

= number of features

= input (features) of training example

= value of feature j in training example

= number of trainings

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Example of dummy data** | | | | | value\_usd |
| Association | Seller Rating | Wants Match | BDP Views | Basket Adds | GMS |
|  |  |  |  |  |  |
| 1 | 4 | 2 | 6 | 2 | 29 |
| 1 | 1 | 3 | 3 | 1 | 15.1 |
| 1 | 4 | 3 | 10 | 0 | 32.6 |
| 1 | 5 | 3 | 15 | 1 | 41.7 |
| 0 | 2 | 2 | 3 | 1 | 12.6 |
| .. | .. | .. | .. | .. | .. |

Thus, = , a 4 dimension vector; and = 3.

Hypothesis basic set up:

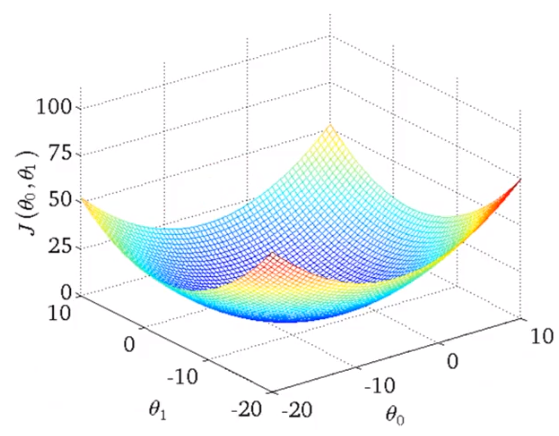
, where = 1; (multivariate liner regression)

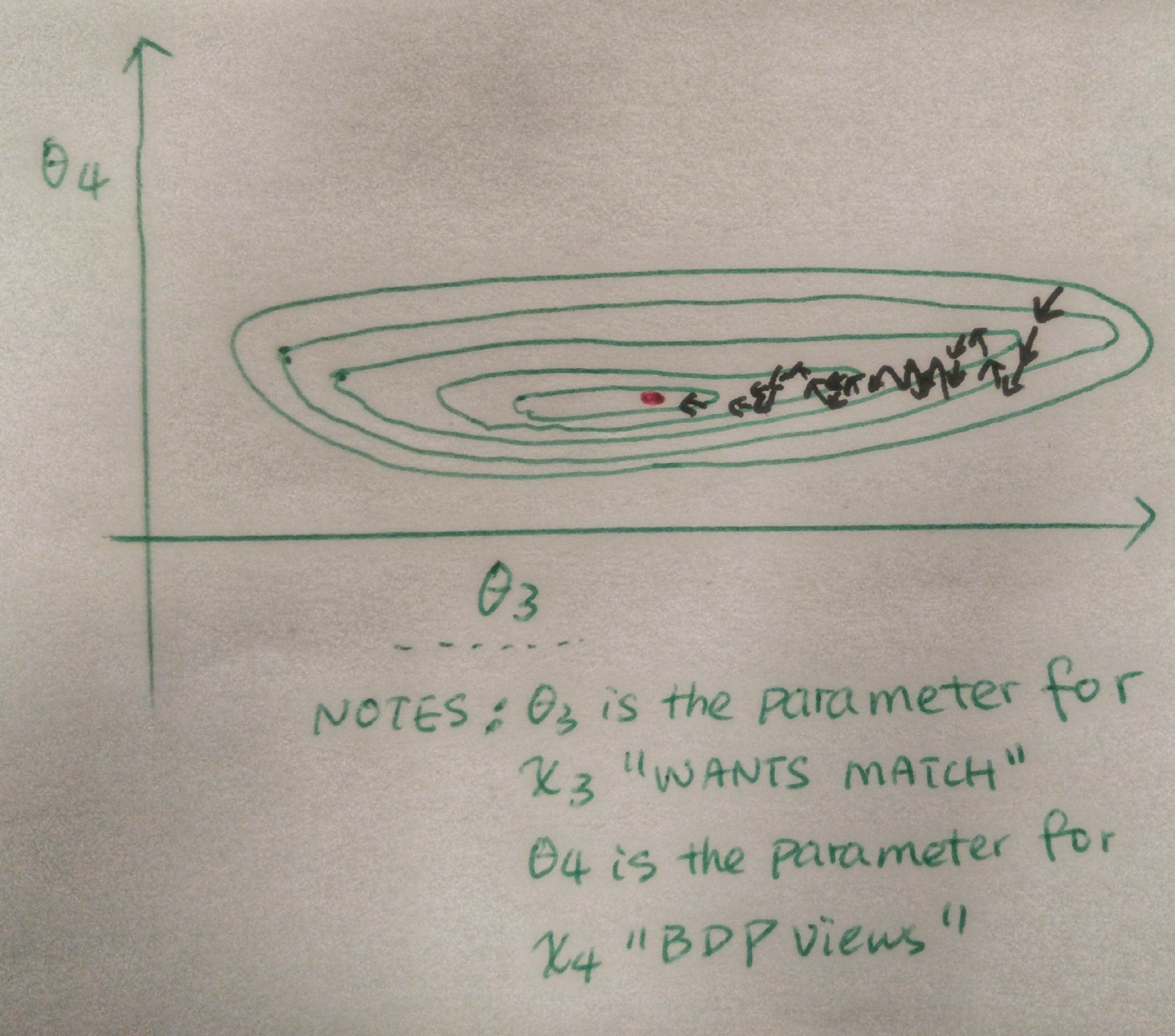
The input data we extract from redshift data sources are

= ; = ; Thus, **hypothesis**: . This is an [inner product multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication) of matrix which equals to: .

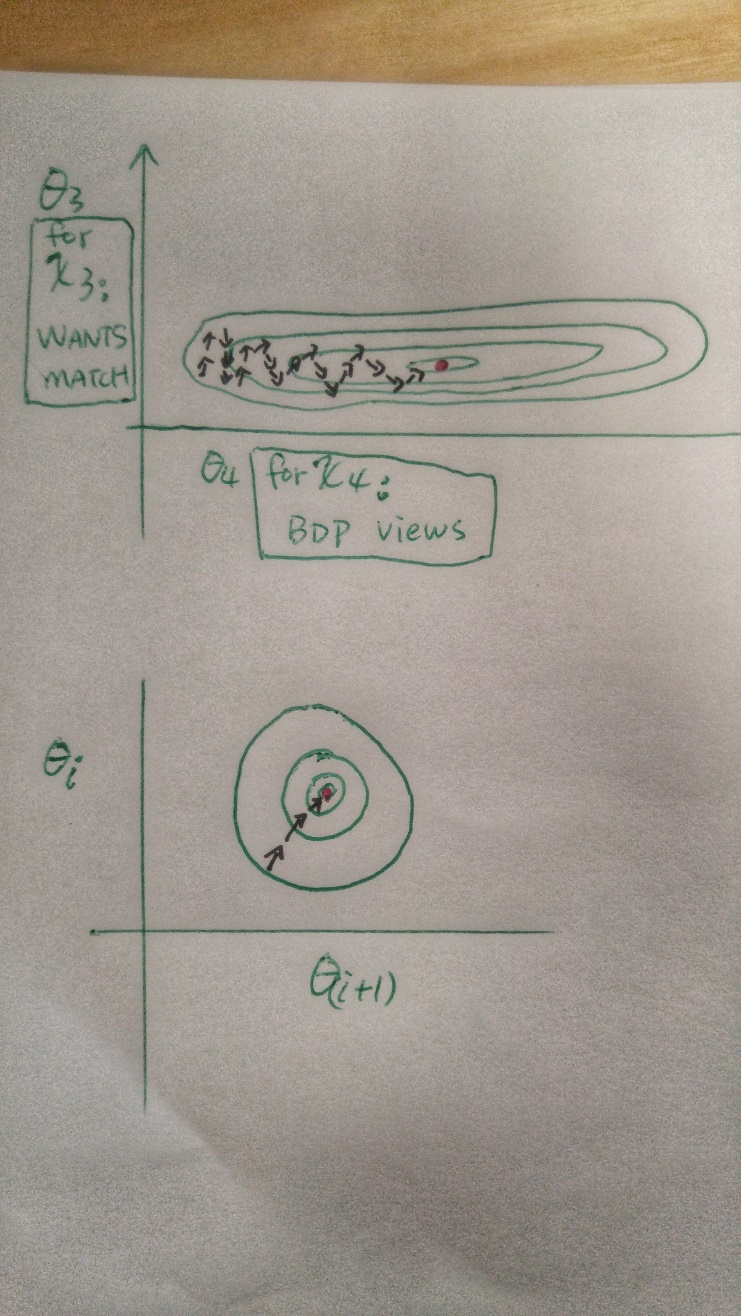
1. AbeBooks’ Rare Feed – Feature(s) Scaling

[Feature scaling](https://en.wikipedia.org/wiki/Feature_scaling) is a must-do step because it will make our programs later run more efficiently. This can also be explained visually. Let’s say we smash a 3-D representation of a Cost Function for which a Gradient Descent Algorism is about to minimize (figure: FS1) into a flat surface ([Contour Plot](http://www.itl.nist.gov/div898/handbook/eda/section3/contour.htm)). And let’s assume it looks like figure: FS2 below.

(figure: FS1) (figure: FS2)



Because the “number of BDP views” are in the scale of 0-10 (times) [assumptions], while the scale for “want matches” is 0-3 [assumptions]. An adjustment of has a considerably heavier influence on the Cost Function, and gradient descent takes the majority of the repetitions to run with smaller steps. The goal of feature scaling is to make most features on a similar scale, so gradient descent will be running more efficiently, e.g. look at figure: FS3 (2 features view).

 (figure: F3S)

One conventional method is to scale all of the features into range, for example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Features We Have** | **Notation** | **Scaling** | **Notes** |
| Association |  |  | This will be binary, either yes [1], or no[0]. |
| Seller Rating |  |  | Max rating = 5 |
| Wants Match |  |  | Assumed max wants matches for one listing\_id is <=3 |
| BDP Views |  |  | Assumed NBC BDP views per listing\_id <= 10 |
| Basket Adds |  |  | Assumed NBC basket adds per listing\_id <= 5 |
| Projected GMS |  |  |  |

Note: the program to be written will leverage another conventional method “[mean normalization](https://en.wikipedia.org/wiki/Feature_scaling#Mean_normalisation)”, it requires the knowledge/calculation of the standard deviation of each feature.

1. Gradient Descent Set Up

Let me merge 2 algorithms and this will make the subsequent efforts to put them into programming language easier.

We know Cost Function

We know gradient descent below.

“repeat unitl convergence:

“

For a hypothesis with only 1 feature, , let me insert Cost Function in (red highlighted portion).

=

=

After calculating the derivative portion with respect to both and gradient descent algorithm becomes:

“repeat until convergence {

}”

Now, the use case of AbeBooks Rare Product Feed will require multiple features, so we generalize gradient descent to the :

1. Normal Equation Set Up

In earlier text gradient descent algorithm was broken down into each element and explained. This document will not explain normal equation in such manner, primarily because there does not seem to be an intuitive way to do so (which the author is aware of). Normal equation can be explained fully mathematically. The x-coordinate for the lowest (or highest) point for [quadratic function with one unknown](http://www.biology.arizona.edu/biomath/tutorials/Quadratic/Roots.html) ( ) can be represented and calculated directly using a formula . This is also the result of taking the derivative of (which gives you ), and set it to zero with respect to (this give you ).

However, for AbeBooks’ rare feed there are more than one function (or one row of features and parameters ) like the quadratic function above, but a matrix of needed to compute and minimize the cost function .

For every row ( rows in total), there is a row of parameters . So if we want to take the derivate and set it to zero, we need to apply it to an entire matrix of :

If I turn the features and the y (success metric: GMS) into matrix and vector like the below; the summarized formula for is shown below (red color highlighted):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| - | Association | Seller Rating | Wants Match | BDP Views | Basket Adds | GMS |
|  |  |  |  |  |  |  |
| 1 | 1 | 4 | 2 | 6 | 2 | 29 |
| 1 | 1 | 1 | 3 | 3 | 1 | 15.1 |
| 1 | 1 | 4 | 3 | 10 | 0 | 32.6 |
| 1 | 1 | 5 | 3 | 15 | 1 | 41.7 |

; ;

Notes: is the [transpose](https://en.wikipedia.org/wiki/Transpose) of .

“” is the inverse of a matrix.

So the is the [inverse](https://www.mathsisfun.com/algebra/matrix-inverse.html) of matrix .

In the programming language [Octave](https://www.gnu.org/software/octave/) which I will use to build the program. The above formula is input as shown below, where “pinv” means inverse.

Pinv(x’\*x)\*x’\*y

And this is essentially the normal equation we will use, and because it is a one-formula process, it is in contrast to gradient descent which take iterations to find the best parameters, normal equation takes one step (calculation) and does not require a learning rate . The limitation of normal equation is that it will take longer computation time if (the number of features) is large, which is not the case for the practice we have in this document ( ).

1. Write the Program/Algorithms and Execute with Dummy Data

* Programming language used: [Octave](https://www.gnu.org/software/octave/); Octave is built to solve vectorized mathematical problems.
* Dummy Data is attached in this document at the appendix
  1. **Loading of features (from dummy file “abrfdata2.txt”) “abrf” stands for AbeBooks Rare Feed.**

%% Clear and Close Figures

clear ; close all; clc

fprintf('Loading data ...\n');

%% Load Data

data = load('abrfdata2.txt');

X = data(:, 1:5); % load all rows for the first 1-5 columns

y = data(:, 6);

m = length(y);

%% Let me print some data, so I know I pull the right AbeBooks data

fprintf('Let us load the First 10 rows from the dummy data for AbeBooks Rare Feed: \n');

fprintf(' x = [%.0f %.0f %.0f %.0f %.0f], y = %.0f \n', [X(1:10,:) y(1:10,:)]');

fprintf('Pausing, Press Enter to Continue.\n');

pause;

% Scale features and set them to zero mean

fprintf('Normalizing Features ...\n');

[X mu sigma] = featureNormalize(X);

% Add intercept term to X; this addition of a column of “1s” is not covered

% in this document, but it is mathematically required to make gradient descent run; simply put we add a 1 to each row of features in the matrix

X = [ones(m, 1) X];

* 1. **Function: Scaling/normalization for the features from our rare feed**

%% this portion is largely using exiting examples people built on github

function [X\_norm, mu, sigma] = featureNormalize(X)

X\_norm = X;

mu = zeros(1, size(X, 2));

sigma = zeros(1, size(X, 2));

mu = mean(X\_norm);

sigma = std(X\_norm);

tf\_mu = X\_norm - repmat(mu,length(X\_norm),1);

tf\_std = repmat(sigma,length(X\_norm),1);

X\_norm = tf\_mu ./ tf\_std;

end

* 1. **Function: Cost Function**

function J = computeCostMulti(X, y, theta)

m = length(y); % number of training examples (in AbeBooks dummy file: 97)

J = 0;

J = (1/(2\*m))\*sum(power((X\*theta - y),2));

end

* 1. **Function: Gradient Descent**

function [theta, J\_history] = gradientDescentMulti(X, y, theta, alpha, num\_iters)

m = length(y); % number of training examples

J\_history = zeros(num\_iters, 1);

for iter = 1:num\_iters

delta = (1/m)\*sum(X.\*repmat((X\*theta - y), 1, size(X,2)));

theta = (theta' - (alpha \* delta))';

J\_history(iter) = computeCostMulti(X, y, theta);

end

end

* 1. **Applying Gradient Descent to refine the parameters for 800 iterations**

**(800 is just an arbitrary number that is proved to work for this dummy data set)**

fprintf('Running gradient descent ...\n');

% arbitrary numbers for learning rate and number of repetitions

alpha = 0.01;

num\_iters = 800;

% Init Theta set to “0s”

% Run Gradient Descent

theta = zeros(6, 1);

[theta, J\_history] = gradientDescentMulti(X, y, theta, alpha, num\_iters);

% Plot the convergence graph - the sanity checks – will be covered later in this doc

figure;

plot(1:numel(J\_history), J\_history, '-b', 'LineWidth', 2);

xlabel('Number of iterations');

ylabel('Cost J - AbeBooks Rare Feed');

% Display gradient descent's result

fprintf('Theta computed from gradient descent: \n');

fprintf(' %f \n', theta);

fprintf('\n');

* 1. **Using the new parameters (thetas), and the hypothesis to make a prediction**

***% Predict the GMS of a listing\_id with 1) no association, 2) 3 star seller ratings, 3) 1 wants match, 4) 10 bdp views, and 5) 1 basket adds.***

d = [0 3 1 10 1];

d = (d - mu) ./ sigma;

d = [ones(1, 1) d];

price = d \* theta;

fprintf(['Predict the GMS of a listing\_id with 1) no association, 2) 3 star seller ratings, 3) 1 wants match, 4) 10 bdp views, and 5) 1 basket adds. ' ...

'(using gradient descent):\n $%f\n'], price);

fprintf('Program paused. Press enter to continue.\n');

pause;

* 1. **Function: Normal Equation**

function [theta] = normalEqn(X, y)

theta = zeros(size(X, 2), 1);

theta = pinv(X'\*X)\*X'\*y;

end

* 1. **Applying Normal Equation and calculating the new parameters (thetas); Use them to make a prediction with the same assumption we used for Gradient Descent**

fprintf('................................\n');

fprintf('Solving with normal equations...\n');

%% Load Data

data = load('abrfdata2.txt');

X = data(:, 1:5);

y = data(:, 6);

m = length(y);

% Add intercept term to X

X = [ones(m, 1) X];

% Calculate the parameters from the normal equation

theta = normalEqn(X, y);

% Display normal equation's result

fprintf('Theta computed from the normal equations: \n');

fprintf(' %f \n', theta);

fprintf('\n');

***% Predict the GMS of a listing\_id with 1) no association, 2) 3 star seller ratings, 3) 1 wants match, 4) 10 bdp views, and 5) 1 basket adds.***

d = [1 0 3 1 10 1];

price = d \* theta;

% ============================================================

fprintf(['Predict the GMS of a listing\_id with 1) no association, 2) 3 star seller ratings, 3) 1 wants match, 4) 10 bdp views, and 5) 1 basket adds. ' ...

'(using normal equation):\n $%f\n'], price);

* 1. **Function: Plotting**

function plotData(x, y)

figure; % open a new figure window

plot(x, y, 'rx', 'MarkerSize', 10); % Plot the data

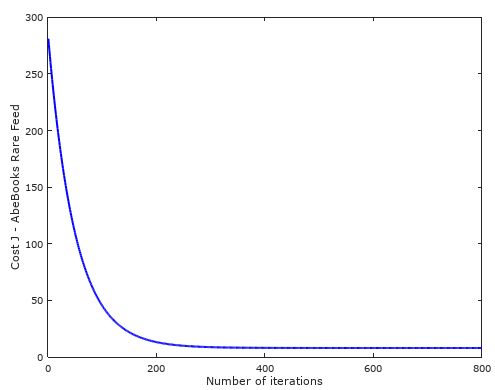
ylabel('Profit in $10,000s'); % Set the y?axis label

xlabel('Population of City in 10,000s'); % Set the x?axis label

end

1. Sanity Checks & Results

Two sanity checks can help us self-convince whether the program worked the way we intended it to. One is (visualization) taking a real look at how the Cost Function evolves throughout time. Intuitively speaking, we want to see whether the sum of the discrepancies between our hypothesis and the reality (dummy data file) is reducing every time after our gradient descent ([AMP suit](http://james-camerons-avatar.wikia.com/wiki/Amplified_Mobility_Platform) machine) takes a step. This is already built into the program if you took a close look. For our case, we arbitrarily ask Gradient Descent to run 800 times (iterations). Figure: QA1 below shows how it corresponds to our Cost Function. This sanity check or QA is used to primarily check whether the learning rate is sufficiently small. (remember we discussed this earlier when Gradient Descent takes too big a step, it can lead the algorism to overshoot the lowest point.)

 (figure: QA1)

The second sanity check is to literally do a napkin math with the set of parameters spit out by Gradient descent, and calculate the prediction manually and compare it with the program. To do so in this program, the data set for prediction “1) no association, 2) 3 star seller ratings, 3) 1 wants match, 4) 10 BDP views, and 5) 1 basket adds” should not be normalized, otherwise we need to do the napkin math for the feature normalization as well. Once the normailization is turned into a comment block (%%d = (d - mu) ./ sigma)), we can do a sumproduct of the parameters () and the features [0, 3, 1, 10, 1]. This is done and checked by the author and not demonstratedhere. Now, we both sanity checks completed. We look at the results below (also find from Appendix).

**Parameters () :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Features** | **Notation** | **By Gradient Descent** | **By Normal Equation** |
| - |  | = 21.978483 | = -1.30802 |
| Association |  | = 0.962034 | = 1.934372 |
| Seller Rating |  | = 1.817501 | = 1.300394 |
| Wants Match |  | = 2.014587 | = 1.959063 |
| BDP Views |  | = 7.926982 | = 1.726048 |
| Basket Adds |  | = 2.096466 | = 1.783615 |

**Predictions:**

With the 2 sets of new parameters, and the 2 hypothesis, the prediction of GMS for a listing\_id that is with no association, 3-star seller ratings, 1 wants match, 10 BDP views, and 1 basket adds, are shown below:

**By Gradient Descent**: GMS $23.585348

**By Normal Equation**: GMS $23.596325

1. Limitation & Next Step

The bold assumption that our hypothesis for inventories attributes () and weightings () can be fit into a linear regression model may underestimate the complexity of real case. If this program yields unrealistic prediction with actual data sets, a quick modification should be made to turn the hypothesis into a (regularized) polynomial model. In addition, logistics regression should be explored to build prediction tool for binary goals (or probability), for instance, a tool to predict the likelihood that a listing will generate impression. The author also aspires to seek support from Amazon/AbeBooks internal experts to 1) refine this model, 2) seek model selection algorithm, prior to experimenting non-linear hypothesis or/and neural network solutions.

More features can be fed into the model with minor change of the codes (matrix size, learning rate, and number of iteration) if the marketing team found BSA codes for instance is another signal (feature) worth taking into consideration.

Next step is to apply this prediction tool is to a larger set of inventories, and used the GMS prediction as a sorting criterion. Nonetheless, a risk assessment paper should be written and used to communicate the financial risks (loss) and gain consensus of a testing budget, prior to feeding the new sets of feed for advertising, especially for Google PLAs (Product Listing Ads).

1. Appendix

Dummy Data:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Listing\_id | Association  (Yes Or No) | Seller Rating  (1-5) | Wants Match | BDP Views  (Times) | Basket Adds  (Times) | GMS  ($) |
| - | 1 | 4 | 2 | 6 | 2 | 29.0 |
| - | 1 | 1 | 3 | 3 | 1 | 15.1 |
| - | 1 | 4 | 3 | 10 | 0 | 32.6 |
| - | 1 | 5 | 3 | 15 | 1 | 41.7 |
| - | 0 | 2 | 2 | 3 | 1 | 12.6 |
| - | 1 | 3 | 1 | 1 | 2 | 14.1 |
| - | 1 | 4 | 1 | 13 | 3 | 34.7 |
| - | 1 | 5 | 0 | 1 | 3 | 13.4 |
| - | 1 | 5 | 0 | 10 | 3 | 40.1 |
| - | 1 | 4 | 1 | 12 | 2 | 41.0 |
| - | 1 | 5 | 1 | 10 | 3 | 23.6 |
| - | 0 | 4 | 1 | 1 | 3 | 13.0 |
| - | 0 | 4 | 3 | 7 | 0 | 26.4 |
| - | 1 | 2 | 3 | 5 | 0 | 19.0 |
| - | 0 | 1 | 1 | 11 | 3 | 35.3 |
| - | 0 | 3 | 1 | 7 | 3 | 25.7 |
| - | 0 | 5 | 2 | 14 | 1 | 33.3 |
| - | 0 | 4 | 2 | 14 | 0 | 30.5 |
| - | 1 | 1 | 1 | 10 | 3 | 24.2 |
| - | 1 | 1 | 1 | 8 | 2 | 25.6 |
| - | 1 | 1 | 3 | 5 | 1 | 17.5 |
| - | 1 | 2 | 1 | 1 | 0 | 6.2 |
| - | 0 | 4 | 0 | 0 | 1 | 8.7 |
| - | 1 | 1 | 1 | 5 | 1 | 11.6 |
| - | 1 | 4 | 0 | 13 | 3 | 25.8 |
| - | 0 | 2 | 0 | 4 | 0 | 7.5 |
| - | 0 | 4 | 1 | 2 | 2 | 19.8 |
| - | 1 | 2 | 3 | 14 | 2 | 41.7 |
| - | 1 | 3 | 1 | 8 | 0 | 23.6 |
| - | 1 | 3 | 0 | 7 | 2 | 17.5 |
| - | 1 | 1 | 2 | 1 | 3 | 12.1 |
| - | 0 | 1 | 3 | 4 | 3 | 20.1 |
| - | 0 | 5 | 3 | 6 | 2 | 20.6 |
| - | 1 | 5 | 3 | 15 | 3 | 45.0 |
| - | 1 | 1 | 3 | 1 | 3 | 14.1 |
| - | 0 | 3 | 1 | 4 | 1 | 10.6 |
| - | 0 | 4 | 0 | 10 | 1 | 23.1 |
| - | 0 | 1 | 0 | 14 | 1 | 32.2 |
| - | 1 | 4 | 3 | 0 | 2 | 12.2 |
| - | 1 | 2 | 2 | 7 | 2 | 27.3 |
| - | 0 | 3 | 1 | 14 | 1 | 22.1 |
| - | 1 | 5 | 1 | 8 | 2 | 20.7 |
| - | 1 | 1 | 1 | 2 | 1 | 10.3 |
| - | 1 | 4 | 1 | 13 | 1 | 44.0 |
| - | 1 | 5 | 0 | 5 | 3 | 16.4 |
| - | 1 | 5 | 1 | 1 | 2 | 14.1 |
| - | 1 | 5 | 3 | 10 | 2 | 36.8 |
| - | 1 | 4 | 3 | 13 | 0 | 42.5 |
| - | 0 | 2 | 3 | 0 | 0 | 6.1 |
| - | 0 | 1 | 3 | 2 | 3 | 14.8 |
| - | 0 | 2 | 2 | 5 | 2 | 15.1 |
| - | 1 | 4 | 1 | 13 | 0 | 26.3 |
| - | 1 | 1 | 1 | 8 | 2 | 23.3 |
| - | 1 | 1 | 1 | 2 | 3 | 14.6 |
| - | 0 | 4 | 1 | 3 | 3 | 19.2 |
| - | 1 | 3 | 0 | 15 | 1 | 31.4 |
| - | 0 | 2 | 1 | 12 | 3 | 30.7 |
| - | 0 | 3 | 1 | 12 | 1 | 28.2 |
| - | 1 | 4 | 2 | 0 | 1 | 12.6 |
| - | 0 | 4 | 3 | 14 | 0 | 33.1 |
| - | 1 | 4 | 2 | 14 | 1 | 39.2 |
| - | 1 | 1 | 1 | 14 | 3 | 30.3 |
| - | 0 | 4 | 2 | 5 | 0 | 13.6 |
| - | 0 | 4 | 2 | 1 | 3 | 13.8 |
| - | 0 | 4 | 3 | 8 | 3 | 33.3 |
| - | 0 | 4 | 2 | 9 | 1 | 23.9 |
| - | 0 | 3 | 3 | 4 | 0 | 15.9 |
| - | 0 | 5 | 2 | 5 | 1 | 25.0 |
| - | 1 | 4 | 0 | 7 | 0 | 19.8 |
| - | 1 | 3 | 1 | 13 | 0 | 21.0 |
| - | 1 | 1 | 3 | 11 | 0 | 29.6 |
| - | 1 | 2 | 1 | 0 | 3 | 8.9 |
| - | 1 | 3 | 3 | 9 | 3 | 29.6 |
| - | 0 | 5 | 2 | 5 | 3 | 22.6 |
| - | 0 | 2 | 3 | 1 | 1 | 8.4 |
| - | 0 | 1 | 3 | 7 | 0 | 13.5 |
| - | 1 | 5 | 3 | 9 | 0 | 24.8 |
| - | 0 | 1 | 2 | 9 | 0 | 20.8 |
| - | 0 | 4 | 1 | 11 | 3 | 24.8 |
| - | 1 | 3 | 0 | 5 | 3 | 18.5 |
| - | 1 | 4 | 1 | 12 | 1 | 26.7 |
| - | 1 | 1 | 2 | 7 | 1 | 19.5 |
| - | 1 | 3 | 2 | 8 | 0 | 19.3 |
| - | 0 | 3 | 1 | 8 | 0 | 12.7 |
| - | 1 | 3 | 0 | 7 | 1 | 18.0 |
| - | 1 | 5 | 2 | 2 | 2 | 18.5 |
| - | 0 | 3 | 1 | 3 | 0 | 11.2 |
| - | 1 | 1 | 1 | 9 | 3 | 22.6 |
| - | 1 | 4 | 1 | 0 | 0 | 8.8 |
| - | 1 | 2 | 3 | 9 | 3 | 23.5 |
| - | 0 | 1 | 3 | 0 | 2 | 11.0 |
| - | 0 | 3 | 1 | 1 | 3 | 12.0 |
| - | 1 | 4 | 2 | 6 | 3 | 26.4 |
| - | 0 | 2 | 2 | 8 | 1 | 14.4 |
| - | 0 | 4 | 2 | 12 | 1 | 27.0 |
| - | 1 | 2 | 2 | 7 | 3 | 17.5 |
| - | 1 | 3 | 3 | 5 | 0 | 15.8 |

Results (Screenshot – command window):

