Intern Project Report

On

Machine learning algorithms to predict efficiency curve

Submitted By

Shrey Mishra
Roll No: 149107438
Department of Electronics and Communication Engg.
Manipal University, Jaipur

Under Guidance of
Prof. Vinayak Kulkarni

Department of Mechanical Engineering

Indian Institute of Technology, Guwahati

June - July, 2016

Indian Institute of Technology Guwahati

CERTIFICATE

This is to certify that the summer intern project report entitled, "***Machine learning algorithms to predict efficiency curve***" submitted by **Shrey Mishra** in partial fulfillment of the requirement for the one-month (17-12-2017 to 16-01-2018) summer training program in Dept. of Mechanical Engineering, IIT Guwahati is carried out by him under my supervision and guidance.

Dr. Vinayak Kulkarni

Date: 16.01.2018

Professor, Dept. of Mechanical Engineering

IIT Guwahati

ACKNOWLEDGEMENT

It is with a feeling of great pleasure that I would like to express my most sincere heartfelt gratitude to **Dr. Vinayak Kulkarni**, Professor, Dept. of Mechanical Engineering, IIT Guwahati for suggesting the topic for my project work and for his ready and able guidance throughout my internship. I am greatly indebted to him for his constructive suggestion and criticism from time to time during the course of progress of my work.

I express my sincere thanks to Dr. A. K. Dass, HoD, Dept. of Mechanical Engineering, IIT Guwahati for allowing me to do my internship as well as providing the necessary facilities in the department. I express gratitude to **Mr. Saibal Kanchan Barik, Mr. Parag K. Talukdar and Mr. Shuvayan Brahmachary**, Research Scholar, Dept. of Mechanical Engineering for their advice, guidance and valuable suggestion for successfully completion of my intern project work.

I feel pleased and privileged to complete my intern in IIT Guwahati.

Date: 16.01.2018

Shrey Mishra

Roll No: 149107438

Manipal University, Jaipur

TABLES OF FIGURES

Figure 1.1: An example of a linear hypothesis curve. 7

Figure 1.2: The plot shows the varying Cost (J) is convex function 8

Figure 1.3: Shows how gradient converges to the global minima point.. 9

Figure 1.4: Shows the convergence of Cost (J) after a certain number of iterations…………….11

Figure 3.1: Shows the various curves obtained from training samples …………………………16

Figure 3.2:Shows the various curves obtained from training samples…………………………..18

Figure 3.3: Shows the convergence of gradient descent ……………………………………… 19

Figure 3.4:The comparison between experimental value and the prediction……………………20

CONTENT

ABSTRACT 6
CHAPTER 1 7
INTRODUCTION 7
CHAPTER 2 12
CHAPTER 3(COMPUTATIONAL MODEL) 8
    2.1 Tsr and efficiency 16
    2.2 Tsr and efficiency at various velocities 17
    2.3 Results and Discussions 20
    Conclusion 21
REFERENCES 21

ABSTRACT

Most of the Machine learning algorithms are classified into two categories: 1) Supervised 2) Unsupervised. The present investigation is mainly focussed on **Regression model** under Supervised learning to predict efficiency from test samples. For the sample datas we are taking quadratic model into account but we can fit almost any degree polynomial to the data set and predict the equation of the curve, we are also using **Octave** as our programming environment but the learning algorithms can be implemented on any other language such as Python, NumPy, R etc.

Chapter 1

Supervised learning (Regression)

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function.

In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories. Here is a description on Math is Fun on Continuous and Discrete Data.

Linear regression with one variable is also known as "univariate linear regression."

Univariate linear regression is used when you want to predict a single output value y from a single input value x. We're doing supervised learning here, so that means we already have an idea about what the input/output cause and effect should be.

The Hypothesis Function: -

Our hypothesis function has the general form:

$y\hat{} = h\theta(x) = \theta 0+\theta 1x$

Note that this is like the equation of a straight line. We give to $h\theta(x)$ values for $\theta 0$ and $\theta 1$ to get our estimated output $y\hat{}$. In other words, we are trying to create a function called $h\theta$ that is trying to map our input data (the x's) to our output data (the y's).

Example:

input (x)	output (y)
0	4
1	7
2	7
3	8

Suppose we have the following set of training data:
Now we can make a random guess about our *hθ* function: *θ*0=2 and *θ*1=2. The hypothesis function becomes *hθ*(*x*)=2+2*x*.

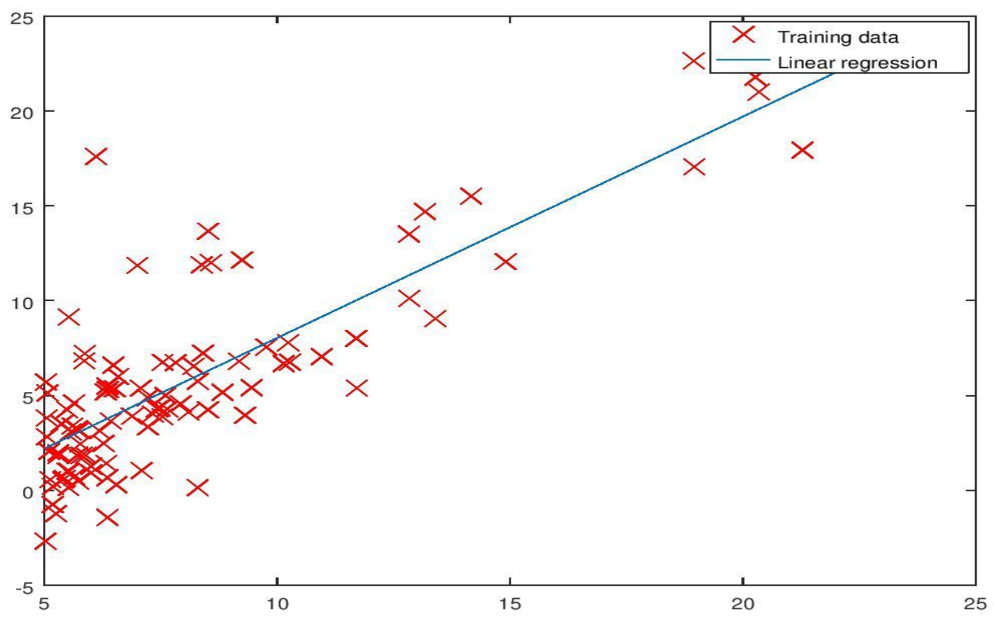


Figure1.1-An example of a linear hypothesis curve

So, for input of 1 to our hypothesis, y will be 4. This is off by 3. Note that we will be trying out various values of $\theta 0$ and $\theta 1$ to try to find values which provide the best possible "fit" or the most representative "straight line" through the data points mapped on the x-y plane.

Cost Function: -

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's compared to the actual output y's.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

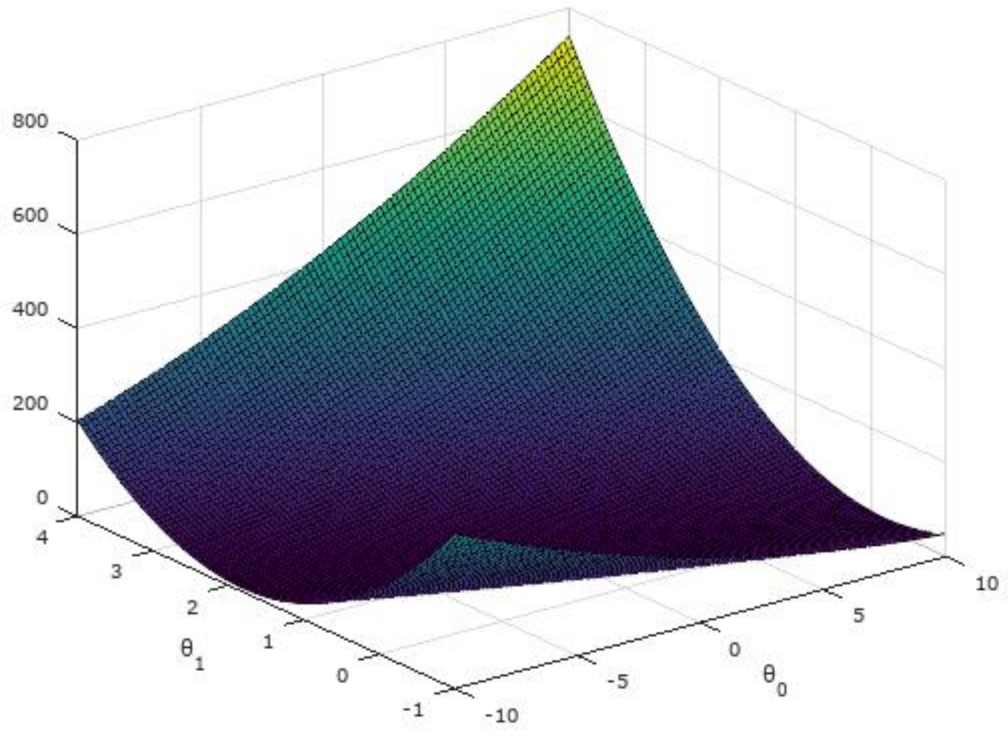


Figure 1.2-The plot shows the varying Cost (J) is convex function

To break it apart, it is 12 x^{-} where x^{-} is the mean of the squares of $h\theta(xi)-yi$, or the difference between the predicted value and the actual value.
This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved $(12m)$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the 12 terms.
Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make straight line (defined by $h\theta(x)$) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of $J(\theta0,\theta1)$ will be 0.

Gradient Descent: -

we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in hypothesis function. That's where gradient descent comes in.

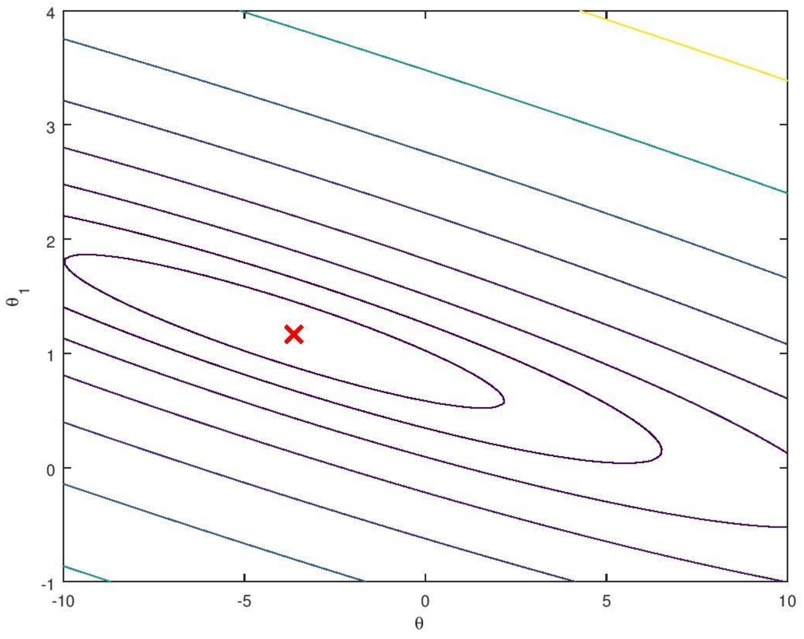


Figure1.3-Shows how gradient converges to the global minima point

Imagine that we graph our hypothesis function based on its fields $\theta 0$ and $\theta 1$ (actually we are graphing the cost function as a function of the parameter estimates). This can be kind of confusing; we are moving up to a higher level of abstraction. We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting particular set of parameters.

We put $\theta 0$ on the x axis and $\theta 1$ on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters.

We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent, and the size of each step is determined by the parameter α, which is called the learning rate.

The gradient descent algorithm is:

repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

where

j=0,1 represents the feature index number.

Intuitively, this could be thought of as:

repeat until convergence:

$\theta j := \theta j - \alpha$ [Slope of tangent aka derivative in j dimension][Slope of tangent aka derivative in j dimension]

Gradient Descent for Linear Regression:

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas is out of the scope of this course, but a really great one can be found here):

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

where m is the size of the training set, $\theta 0$ a constant that will be changing simultaneously with $\theta 1$ and *xi*, *yi* are values of the given training set (data).

Note that we have separated out the two cases for θj into separate equations for $\theta 0$ and $\theta 1$; and that for $\theta 1$ we are multiplying *xi* at the end due to the derivative. The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

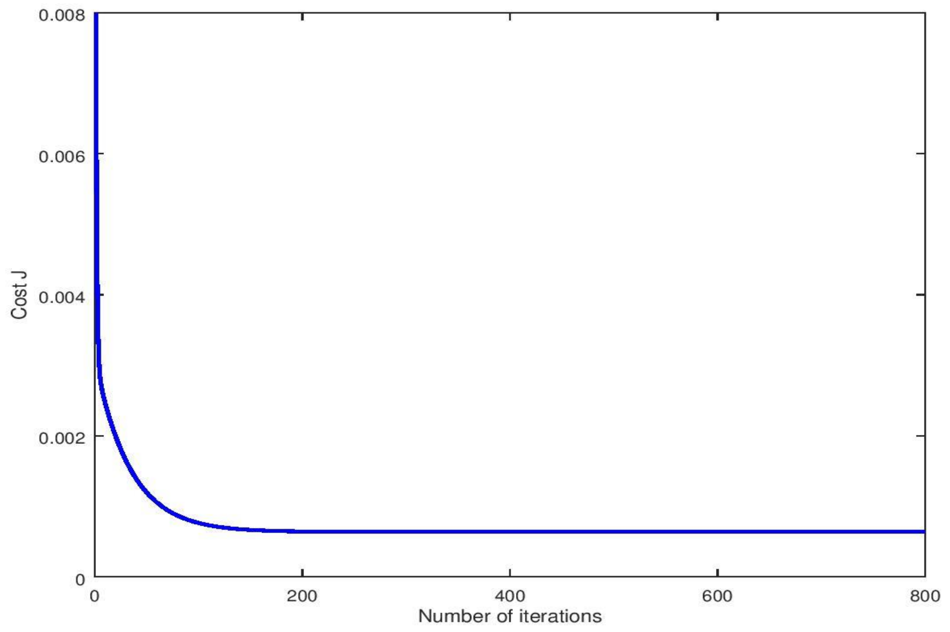


Figure1.4-Shows the convergence of Cost (J) after a certain number of iterations

Chapter 2

HOW TO SUIT YOUR MODEL ON VARIOUS DATA SETS?

Linear Regression with Multiple Variables

Linear regression with multiple variables is also known as "multivariate linear regression".

We now introduce notation for equations where we can have any number of input variables.

$$\begin{aligned} x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\ x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\ m &= \text{the number of training examples} \\ n &= \left| x^{(i)} \right| ; \text{(the number of features)} \end{aligned}$$

Now define the multivariable form of the hypothesis function as follows, accommodating these multiple features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

This is a vectorization of our hypothesis function for one training example.

The training examples are stored in X row-wise, like such:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

You can calculate the hypothesis as a column vector of size (m x 1) with:

$h_\theta(X) = X\theta$

Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

$$
\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \\
&\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)} \\
&\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)} \\
&\cdots \\
\}&
\end{aligned}
$$

In other words,

$$
\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0..n} \\
\}&
\end{aligned}
$$

Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways.

We can combine multiple features into one. For example, we can combine $x1$ and $x2$ into a new feature $x3$ by taking $x1 \cdot x2$.

Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can change the behavior or curve of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$$

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

Normal Equation method

The "Normal Equation" is a method of finding the optimum theta without iteration**.**

$$\theta = (X^T X)^{-1} X^T y$$

There is **no need** to do feature scaling with the normal equation.

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	**Normal Equation**
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn2)	O (n3), need to calculate inverse of XTX
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity O(n3). So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

Prerequisites to solving an Ml problem:

- Must be clear with the linear algebra
- Must be clear with the concept of Matrix operations
- Must be clear with the syntax of Octave programming language
- Must have the required data files, code files and supporting function files
- Must be clear with all the fundamental concepts of algorithms used
- Must be clear about feature scaling
- It is strongly recommended that you follow Andrew Ng’s course on machine learning

Chapter 3

Results and discussions

Turbine graph prediction (Normal equation method):

To begin with the problem of optimization lets visualize the training points to begin with the problem as shown in the figure below

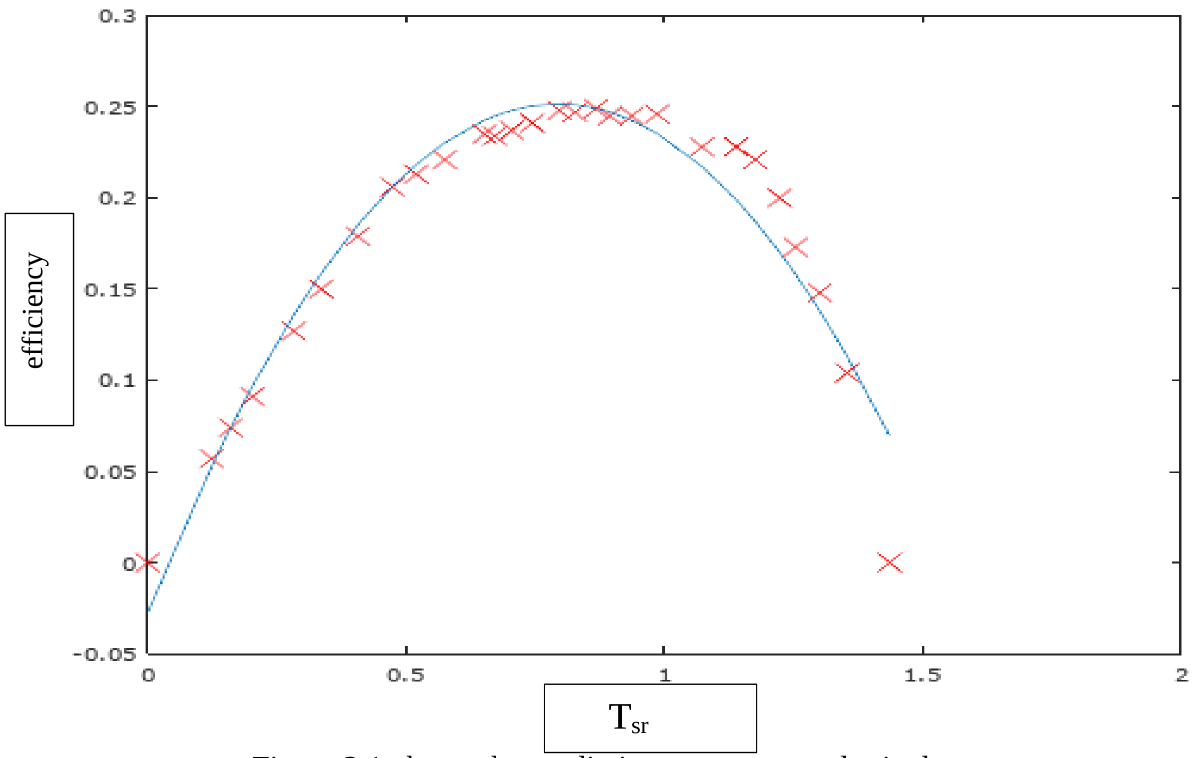
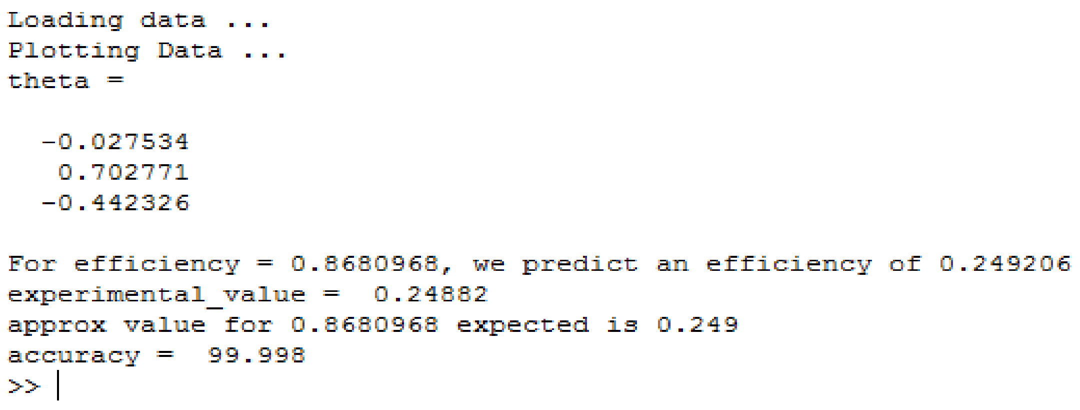


Figure 3.1-shows the prediction curve on quadratic data set

As we can clearly see that the test point clearly shows a quadratic relation Hence, we run the Normal equation method for polynomial hypothesis (Quadratic) to directly compute for the value of vector theta(θ) and then overlay the predicted curve on the training sample. With this curve we can predict the value of efficiency at any point on the curve by multiplying theta(θ) calculated for the predicted curve with the Parameters of X.



```
Loading data ...
Plotting Data ...
theta =

  -0.027534
   0.702771
  -0.442326

For efficiency = 0.8680968, we predict an efficiency of 0.249206
experimental_value =  0.24882
approx value for 0.8680968 expected is 0.249
accuracy =  99.998
>> |
```

Turbine graph prediction (Gradient descent method):

To begin with the problem, we should first understand there are two variables in the problem:

- Velocity (Efficiency varies linearly with change in velocity)
- TSR (Efficiency varies Quadratically with change in velocity)

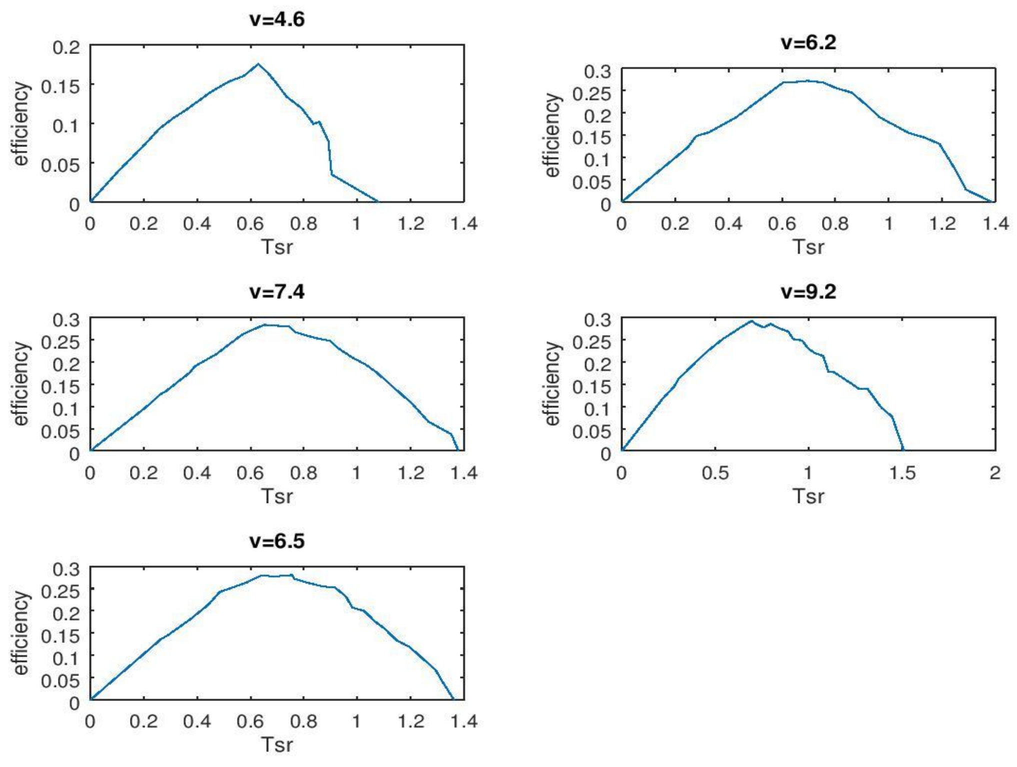


Figure 3.2-Shows the various curves obtained from training samples

And thus we take Gradient descent with multiple variables on polynomial regression model into account and hence compute the theta(θ) values for all the training samples in X (design matrix) .

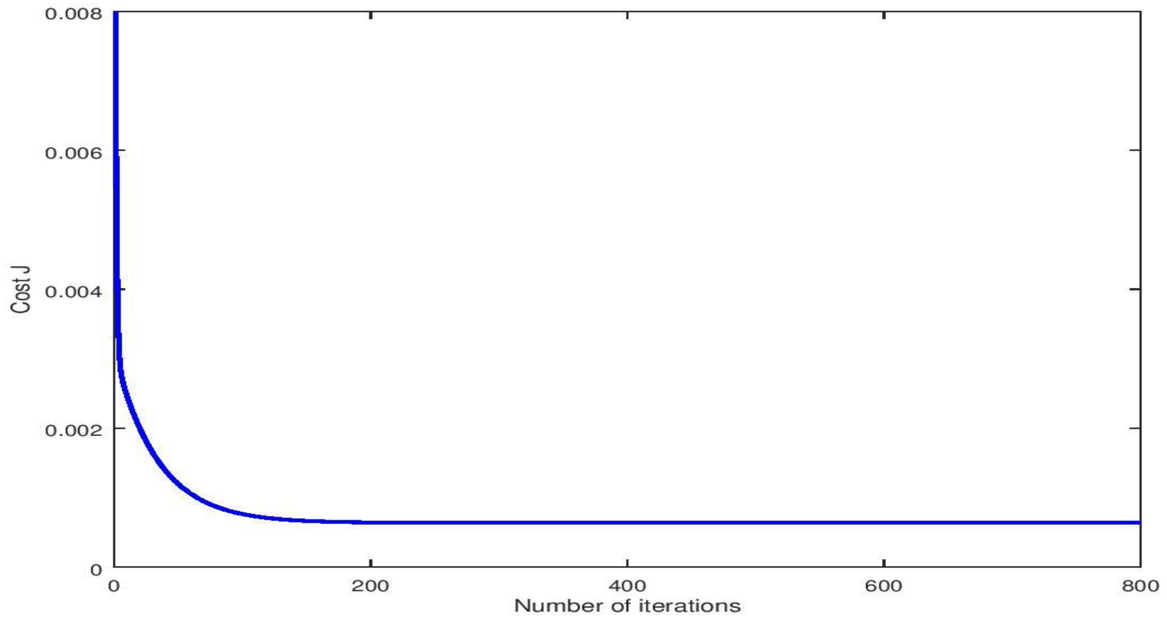


Figure 3.3-shows the convergence of gradient descent

similarly, we plot the predicted curve over the training samples and compute the accuracy.

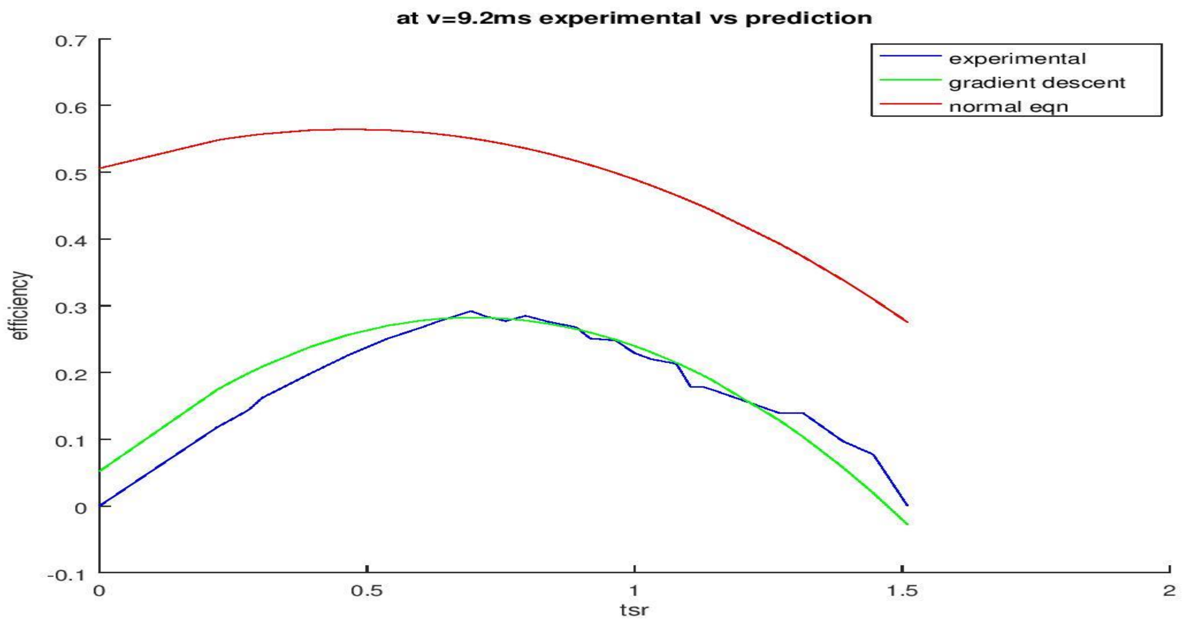
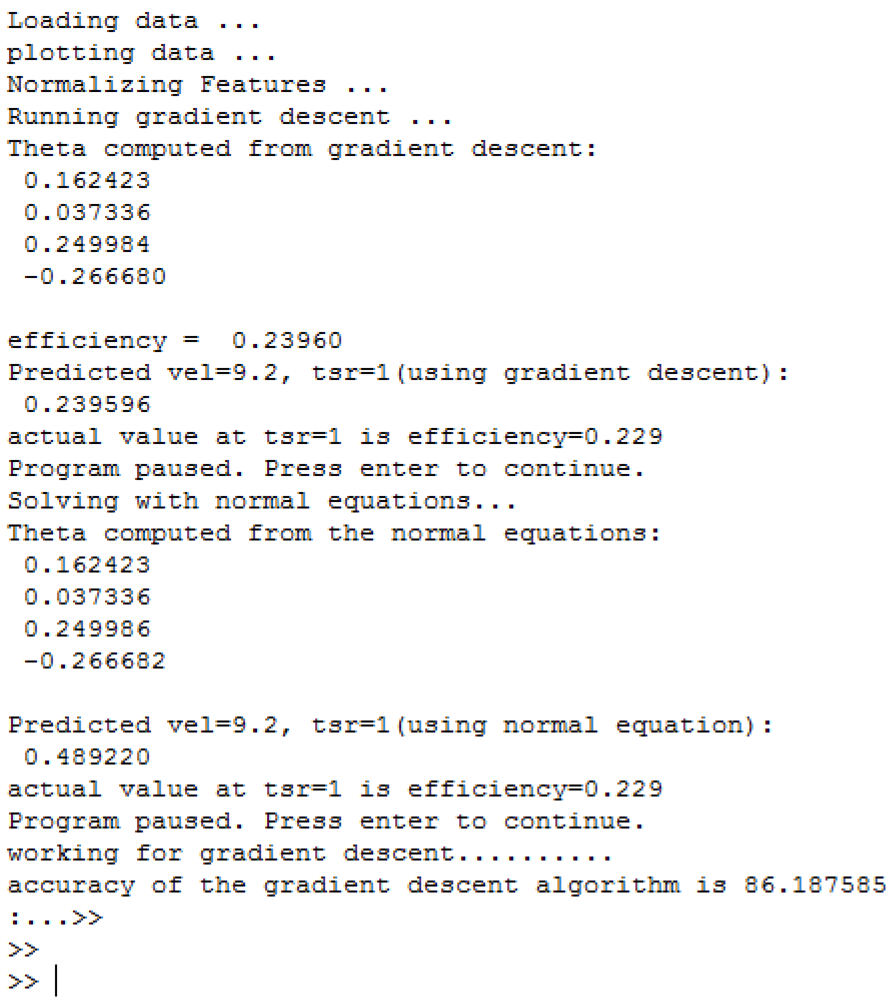


Figure 3.4 – Shows the comparison between experimental value and the prediction model

Results: The accuracy of prediction was found to be nearly 86%

Note-The accuracy is subject to change for the data set on addition/removal of new values of efficiency on various velocities and TSR values**.**



```
Loading data ...
plotting data ...
Normalizing Features ...
Running gradient descent ...
Theta computed from gradient descent:
 0.162423
 0.037336
 0.249984
 -0.266680

efficiency =  0.23960
Predicted vel=9.2, tsr=1(using gradient descent):
 0.239596
actual value at tsr=1 is efficiency=0.229
Program paused. Press enter to continue.
Solving with normal equations...
Theta computed from the normal equations:
 0.162423
 0.037336
 0.249986
 -0.266682

Predicted vel=9.2, tsr=1(using normal equation):
 0.489220
actual value at tsr=1 is efficiency=0.229
Program paused. Press enter to continue.
working for gradient descent.........
accuracy of the gradient descent algorithm is 86.187585
:...>>
>>
>> |
```

Conclusion: The given algorithm computed the values of theta and was able to successfully implement it with a working accuracy of 86.18% on various training samples. The Normal equation method had overshoot the result because of the normalization factor.

REFERENCES:

1. **Proofs of the algorithms used:** https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)
2. http://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression
3. Required software (Octave): https://www.gnu.org/software/octave/download.html
4. Bundle of code files: https://www.dropbox.com/s/hibyua12fso3fkz/iitg_project.rar?dl=0
5. Referred course: https://www.coursera.org/learn/machine-learning/home/welcome

Download reference code and helping functions at -

https://www.dropbox.com/s/hibyua12fso3fkz/iitg_project.rar?dl=0