Linear Regression and Regression Trees

Avinash Kak
Purdue University

April 28, 2019
4:04pm

An RVL Tutorial Presentation
Originally presented on April 29, 2016
Minor edits made in April 2019

CONTENTS

1. Regression In General

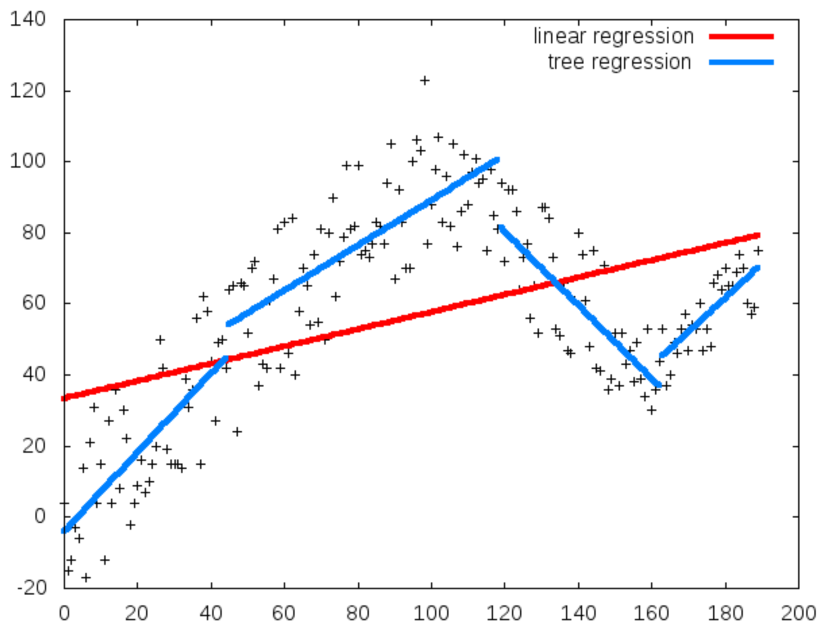
- Regression, in general, helps us understand relationships between variables that are not amenable to analysis through causal phenomena. This is the case with many variables about us as human beings and about many socioeconomic aspects of our societies.

- As a case in point, we know intuitively that our heights and weights are correlated — in the colloquial sense that the taller one becomes, the more likely that one's weight will go up. [If we were to sample, say, 1000 individuals and make a plot of the height versus weight values, we are likely to see a scatter plot in which for each height you will see several different weights. But, overall, the scatter plot will show a trend indicating that, on the average, we can expect the weight to go up with the height.]

- Central to the language of regression are the notions of one designated *dependent variable* and one or more *predictor variables*. In the height-vs.-weight example, we can think of the weight as the dependent variable and the height as the predictor variable. For that example, the goal of a regression algorithm would be to return the best (best in some average sense) value for the weight for a given height.

- Of all the different regression algorithms — and there are virtually hundreds of them out there now — linear regression is arguably the most commonly used. What adds to the "versatility" of linear regression is the fact that it includes polynomial regression in which you are allowed to use powers of the predictor variables for estimating the likely values for the dependent variable.

- While linear regression has sufficed for many applications, there are many others where it fails to perform adequately. Just to illustrate this point with a simple example, shown below is some noisy data for which linear regression yields the line shown in red.



- The blue line is the output of the tree regression algorithm that is presented in the second half of this tutorial.

2. Introduction to Linear Regression

- The goal of linear regression is to make a "best" possible estimate of the general trend regarding the relationship between the predictor variables and the dependent variable with the help of a curve that most commonly is a straight line, but that is allowed to be a polynomial also.

- The fact that the relationship between the predictor variables and the dependent variable can be nonlinear adds to the power of linear regression — for obvious reasons. [As to why the name of the algorithm has "Linear" in it considering that it allows for polynomial relationships between the predictor and the dependent variables, the linearity in the name refers to the relationship between the dependent variable and the regression coefficients.]

- Let's say we have two predictor variables x_1 and x_2; and that our dependent variable is denoted y. Then, both of the following relationships are examples of what is handled by linear regression:

$$\begin{aligned} y &= a_1 \cdot x_1 \;+\; a_2 \cdot x_2 \;+\; b \\ y &= a_1 \cdot x_1 \;+\; a_2 \cdot x_1^2 \;+\; a_3 \cdot x_1^3 \;+\; a_4 \cdot x_2 \;+\; a_5 . x_2^2 \;+\; b \end{aligned}$$

  In both cases, the relationship between the dependent variable and *the regression coefficients* is linear.

- The regression coefficients for the first case are a_1 and a_2 and the same for the second case are a_1, a_2, a_3, a_4, and a_5. The parameter b in both cases is called the *intercept*.

- The fact that linear regression allows for the powers of the predictor variables to appear in the relationship between the dependent and the predictor variables is commonly referred to by saying that it includes *polynomial regression.*

3. Linear Regression Through Equations

- In this tutorial, we will always use y to represent the dependent variable. A dependent variable is the same thing as the predicted variable. And we use the vector $\vec{\mathbf{x}}$ to represent a p-dimensional predictor.

- In other words, we have p predictor variables, each corresponding to a different dimension of $\vec{\mathbf{x}}$.

- Linear regression is based on assuming that the relationship between the dependent and the predictor variables can be expressed as:

$$y \quad = \quad \vec{\mathbf{x}}^T \vec{\beta} \quad + \quad b \tag{1}$$

- In the equation at the bottom of the previous slide, the p-dimensional predictor vector $\vec{\mathbf{x}}$ is given by

$$\vec{\mathbf{x}} \quad = \quad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \qquad (2)$$

- The scalar b in the same equation is to allow for the existence of a nonzero base for y.

- You could say that the role assigned to the predictor variables is to tell us how much the value of y is changed beyond b by given values for the predictor variables.

- In Eq. (1), the vector $\vec{\beta}$ consists of p coefficients as shown below:

$$\vec{\beta} \quad = \quad \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} \tag{3}$$

that reflect the weights to be given to the different predictor variables in $\vec{\mathbf{x}}$ with regard to their predictive power.

- The very first thought that pops up in one's head when one looks at Eq. (1) is that the scalar y depends linearly on the predictor variables $\{x_1, x_2, \cdots, x_p\}$. It is NOT this linearity that the linear regression refers to through its name. As mentioned earlier, the predictor variables are allowed to be powers of what it is that is doing the prediction.

- Let's say that in reality we have only two predictor variables x_1 and x_2 and that the relationship between y and these two predictors is given be $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1^3 + b$ where we have now included the second and the third powers of the variable x_1. For linear regression based computations, we express such a relationship as $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + b$ where $x_3 = x_1^2$ and $x_4 = x_1^3$. The linearity that linear regression refers to is related to linearity with respect to the regression coefficients $\{\beta_1, \beta_2, \cdots, \beta_p, b\}$.

- For the reasons state above, linear regression includes polynomial regression in which the prediction is made from the actual predictor variables and their various powers.

- For a more efficient notation and for positioning ourselves for certain algebraic manipulations later on, instead of using predictor vector $\vec{\mathbf{x}}$ as shown in Eq. (2), we will use its augmented form shown below:

$$\vec{\mathbf{x}} \quad = \quad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ 1 \end{bmatrix} \tag{4}$$

- Additionally, we will now denote the base-value scalar b by the notation:

$$\beta_{p+1} \quad = \quad b \tag{5}$$

- Denoting b in this manner will allow us to make b a part of the vector of regression coefficients as shown on the next slide.

- The notation $\vec{\beta}$ will now stand for:

$$\vec{\beta} \quad = \quad \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \\ b \end{bmatrix} \tag{6}$$

- Using the forms in Eqs. (4) and (6), the relationship in Eq. (1) can now be expressed more compactly as

$$y \quad = \quad \vec{\mathbf{x}}^T \vec{\beta} \tag{7}$$

When using this more compact representation of the relationship between the dependent variable and the predictor variables, we must not forget that the last element of the $\vec{\mathbf{x}}$ is set to 1 and the last element of the coefficient vector $\vec{\beta}$ is supposed to be the base value b.

4. A Compact Representation for All Observed Data

- Now let's say we have N observations available for the relationship between the dependent variable y and the predictor variables $\vec{\mathbf{x}}$. We can express these as

$$\begin{aligned} y_1 &= \vec{\mathbf{x}}_1^T \vec{\beta} \\ y_2 &= \vec{\mathbf{x}}_2^T \vec{\beta} \\ &\vdots \\ y_N &= \vec{\mathbf{x}}_N^T \vec{\beta} \end{aligned} \tag{8}$$

where each observed $\vec{\mathbf{x}}_i$ is given by

$$\vec{\mathbf{x}}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,p} \\ 1 \end{bmatrix} \tag{9}$$

- Constructing a vertical stack of all of the equalities in Eq. (8), we can express them together through the following form:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{x}}_1^T \\ \vec{\mathbf{x}}_2^T \\ \vdots \\ \vec{\mathbf{x}}_N^T \end{bmatrix} \vec{\beta}$$

$$= \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} & 1 \\ x_{2,1} & x_{2,2} & \cdots & x_{2,p} & 1 \\ \vdots & & & & \\ x_{N,1} & x_{N,2} & \cdots & x_{N,p} & 1 \end{bmatrix} \vec{\beta} \qquad (10)$$

- We now express the system of equations shown above in the following more compact form:

$$\vec{\mathbf{y}} = \mathbf{X}\,\vec{\beta} \qquad (11)$$

- In the last equation on the previous slide, the $N \times p$ matrix $\mathbf{X}$ is given by

$$\mathbf{X} \quad = \quad \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} & 1 \\ x_{2,1} & x_{2,2} & \cdots & x_{2,p} & 1 \\ \vdots & & & & \\ x_{N,1} & x_{N,2} & \cdots & x_{N,p} & 1 \end{bmatrix} \qquad (12)$$

- The matrix $\mathbf{X}$ is sometimes called the "design matrix". Note that, apart from the number 1 for its last element, each row of the design matrix $\mathbf{X}$ corresponds to **one** observation for all p predictor variables. In total, we have N observations.

- **The goal of linear regression is to estimate the $p+1$ regression coefficients in the vector $\vec{\beta}$ from the system of equations in Eq. (11).** Recall the form of the $\vec{\beta}$ vector as provided in Eq. (6).

5. Estimating the $p+1$ Regression Coefficients

- To account for noise in the measurement of the dependent variable y (and assuming that the values for the predictor variables are known exactly for each of the N observations), we may prefer to write Eq. (11) as

$$\vec{\mathbf{y}} \quad = \quad \mathbf{X}\,\vec{\beta} \quad + \quad \vec{\epsilon} \qquad (13)$$

- Assuming that $N > p+1$ and that the different observations for the dependent variable are more or less independent, the rank of $\mathbf{X}$ is $p+1$. Our goal now is to estimate the $p+1$ dimensional vector $\vec{\beta}$ from the overdetermined system of equations through the minimization of the cost function

$$C(\vec{\beta}) \quad = \quad \|\vec{\mathbf{y}} \; - \; \mathbf{X}\,\vec{\beta}\|^2 \qquad (14)$$

- Ignoring for now the error term $\vec{\epsilon}$ in Eq. (13), note that what we are solving is an inhomogeneous system of equations given by $\vec{\mathbf{y}} = \mathbf{X}\ \vec{\beta}$. This system of equations is inhomogeneous since the vector $\vec{\mathbf{y}}$ cannot be all zeros.

- The optimum solution for $\vec{\beta}$ that minimizes the cost function $C(\vec{\beta})$ in Eq. (14) possesses the following geometrical interpretation: Focusing on the equation $\vec{\mathbf{y}} = \mathbf{X}\vec{\beta}$, the measured vector $\vec{\mathbf{y}}$ on the left resides in a large N dimensional space. On the other hand, as we vary $\vec{\beta}$ in our search for the best possible solution, the space spanned by the product $\mathbf{X}\vec{\beta}$ will be a $(p+1)$-dimensional subspace (a hyperplane, really) in the N dimensional space in which $\vec{\mathbf{y}}$ resides. The question now is: which point in the hyperplane spanned by $\mathbf{X}\vec{\beta}$ is the best approximation to the point $\vec{\mathbf{y}}$ which is outside the hyperplane. For any selected value for $\vec{\beta}$, the "error" vector $\vec{\mathbf{y}} - \mathbf{X}\vec{\beta}$ will go from the tip of the vector $\mathbf{X}\vec{\beta}$ to the tip of the $\vec{\mathbf{y}}$ vector. Minimization of the cost function C in Eq. (14) amounts to minimizing the norm of this difference vector.

- The norm in question is minimized when we choose for $\vec{\beta}$ a vector so that $\mathbf{X}\vec{\beta}$ is the **perpendicular projection** of $\vec{\mathbf{y}}$ into the $(p+1)$-dimensional row space of the matrix $\mathbf{X}$. The difference vector $\vec{\mathbf{y}} - \mathbf{X}\vec{\beta}$ at the point in the row space of $\mathbf{X}$ where this perpendicular projection of $\vec{\mathbf{y}}$ falls satisfies the following constraint:

$$\mathbf{X}^T \cdot (\vec{\mathbf{y}} - \mathbf{X}\,\vec{\beta}) = 0 \qquad (15)$$

- This implies the following solution for $\vec{\beta}$:

$$\vec{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{\mathbf{y}} \qquad (16)$$

- If we denote the pseudoinverse of $\mathbf{X}$ by $\mathbf{X}^+$, we can write for the solution:

$$\vec{\beta} = \mathbf{X}^+\vec{\mathbf{y}} \qquad (17)$$

where the pseudoinverse is given by $\mathbf{X}^+ = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$.

- Note that $\mathbf{X}^T\mathbf{X}$ is a square $(p+1) \times (p+1)$ matrix whose rank is exactly $p+1$. That is a consequence of the fact that $rank(\mathbf{X}) = p+1$. $\mathbf{X}^T\mathbf{X}$ being of **full rank** implies that the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ is guaranteed to exist.

- In general, for real valued matrices, $\mathbf{X}^T\mathbf{X}$ will always be symmetric and positive semidefinite. The eigenvalues for a such a product matrix will always be real and nonnegative.

- At this point, it is good to reflect on the composition of the matrix $\mathbf{X}$ shown in Eq. (12). Each row of $\mathbf{X}$ consists of one observation for all the p predictor variables following by the number 1.

- We say that the solution shown in Eq. (16) is a a result of linear least squares minimization from a system of inhomogeneous equations. The characterization "linear" in "linear least squares minimization" refers to the fact that we developed our solution from a system of *linear* equations.

6. Refining the Estimates for the Regression Coefficients

- The solution that is obtained with linear least-squares minimization can be further refined with **nonlinear least-squares minimization**.

- **An informed reader would say: Where is the need for such refinement?** Such a reader would go on to say that since the linear least-squares estimate for the regression coefficients in the previous section was obtained by minimizing a convex cost function (implying that this function has a unique global minimum and no other local minima), we can be reasonably certain that the minimum yielded by the solution in the previous section is as good as it can get.

- Nonetheless, in order to allow for the fact that real data can be complex and a linear regression model is at best an approximation to whatever true relationship there exists between the predictor variables and the dependent variable, I am going to go ahead and develop a nonlinear least-squares based estimation refinement method in this section.

- Before going ahead, though, I wish to say at the very outset that for all the synthetic examples I will be presenting in a later section, you do not need any refinement at all. If you do try to refine them with the formulas shown in this section, yes, you do get an answer that looks slightly different (in some cases, perhaps visually more appropriate) — but with an MSE (mean-squared error) that is very slightly greater than what you get with just plain least-squares. **I attribute this increase in MSE to issues related to how you terminate a gradient-descent path to the minimum.**

- With all the caveats out of the way, let me now focus on how we may go about "refining" the estimates of the previous section.

- With nonlinear least-squares minimization, we now think of a cost function in the space spanned by the the $p+1$ elements of the $\vec{\beta}$ vector. (Recall, in linear least-squares minimization, we were focused on how to best project the N-dimensional measurement vector $\vec{\mathbf{y}}$ into the $(p+1)$-dimensional space spanned by the row vectors of the matrix $\mathbf{X}$.)

- Think now of the hyperplane as spanned by the elements of $\vec{\beta}$ and a cost function $C(\vec{\beta})$ as the height of a surface above the hyperplane.

- Our goal is find that point in the hyperplane where the height of the surface is the least. In other words, our goal is the find that point $\vec{\beta}$ that gives us the global minimum for the height of the surface. Such a global minimum is generally found by an iterative algorithm that starts somewhere in the hyperplane, looks straight up at the surface, and then takes small incremental steps along those directions in the hyperplane that yield descents on the surface towards the global minimum. This, as you surely know already, is the essence of the **gradient descent algorithms**.

- In general, though, the surface above the hyperplane may possess both a global minimum and an arbitrary number of local minima. (Cost functions must be convex for there to exist just a single global minimum. In our case, there is no guarantee that $C(\vec{\beta})$ is convex.)

- Considering that, in general, a cost function surface may possess local minima, our only hope is to start the search for the global minimum from somewhere that is in its vicinity. The consequences of starting the search from an arbitrary point in the hyperplane are obvious — we could get trapped in a local minimum.

- Experience has shown that if we start the search at that point in the hyperplane spanned the unknown vector $\vec{\beta}$ that corresponds to somewhere in the vicinity of the linear least-squares solution, we are likely to not get trapped in a local minimum. If $\vec{\beta}_0$ denotes this point in the hyperplane, we set

$$\vec{\beta}_0 \quad = \quad \alpha \cdot (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{\mathbf{y}} \tag{18}$$

for some value of the multiplier α between 0 and 1.

- Subsequently, in the gradient descent approach, we calculate the next point in the hyperplane through the following calculation:

$$\vec{\beta}_{k+1} \quad = \quad \vec{\beta}_k \quad - \quad \gamma \cdot \nabla C \Big|_{\vec{\beta}=\vec{\beta}_k} \qquad (19)$$

  where γ is called the *step size controller* and ∇C is the gradient of the surface at point $\vec{\beta} = \vec{\beta}_k$ in the supporting hyperplane for the cost function surface.

- We stop the iterations either after a certain fixed number of them or when the step size $\|\vec{\beta}_{k+1} - \vec{\beta}_k\|$ falls below some threshold.

- Before the iterative formula dictated by Eq. (19) can be used, the issue of how to compute the gradient $\nabla C(\vec{\beta})$ remains.

- In order to compute this gradient, we express our cost function $C(\vec{\beta})$ in the following form:

$$C(\vec{\beta}) \quad = \quad \|\vec{\mathbf{y}} \; - \; \vec{\mathbf{g}}(\vec{\beta})\|^2 \qquad (20)$$

  where $\vec{\mathbf{g}}(\vec{\beta})$ is a function that relates the parameters to be estimated, $\vec{\beta}$, and the actual measurements $\vec{\mathbf{y}}$. For the linear least-squares solution in the previous section, we assumed that $\vec{\mathbf{g}}(\vec{\beta}) = \mathbf{X} \cdot \vec{\beta}$. But now we want to allow for the possibility that while, nominally, the relationship between the measurements $\vec{\mathbf{y}}$ and $\vec{\beta}$ is linear, actual data may demand nonlinear degrees of freedom with regard to how the parameters $\vec{\beta}$ relate to the measurements. [The parameter β_i reflects the weight to be given to the predictor x_i for making a y prediction. What if this weight does not remain exactly the same over the entire scale over which x_i can vary?]

- We will express the cost function of Eq. (20) in the form shown at right below:

$$C(\vec{\beta}) \;=\; \|\vec{\mathbf{y}} - \vec{\mathbf{g}}(\vec{\beta})\|^2 \;=\; \vec{\epsilon}^T(\vec{\beta}) \cdot \vec{\epsilon}(\vec{\beta}) \tag{21}$$

where by $\vec{\epsilon}$, we mean the difference

$$\vec{\epsilon}(\vec{\beta}) \;=\; \vec{\mathbf{y}} \;-\; \vec{\mathbf{g}}(\vec{\beta}) \tag{22}$$

which, at point $\vec{\beta}_k$ in the parameter hyperplane, is the difference between the measurement $\vec{\mathbf{y}}$ and and $\vec{\mathbf{g}}(\vec{\beta}_k)$ at that location.

- In terms of these difference vectors, the gradient of the cost function is given by

$$\nabla C(\vec{\beta}) \;=\; 2 \cdot J^T_{\vec{\epsilon}}(\vec{\beta}) \cdot \vec{\epsilon}(\vec{\beta}) \tag{23}$$

where $J_{\vec{\epsilon}}(\vec{\beta})$ is the Jacobian of the vector function $\vec{\epsilon}(\vec{\beta})$.

- Since $\vec{\mathbf{y}}$ in the difference $\vec{\epsilon}(\vec{\beta}) = \vec{\mathbf{y}} - \vec{\mathbf{g}}(\vec{\beta})$ is a constant vector, we have

$$J_{\vec{\epsilon}}(\vec{\beta}) \quad = \quad -\, J_{\vec{\mathbf{g}}}(\vec{\beta}) \tag{24}$$

That means we can write the following expression for the gradient of the cost function:

$$\nabla C(\vec{\beta}) \quad = \quad -\, 2 \cdot J^T_{\vec{\mathbf{g}}}(\vec{\beta}) \cdot \vec{\epsilon}(\vec{\beta}) \tag{25}$$

- As for the Jacobian $J_{\vec{\mathbf{g}}}$, it is given by

$$J_{\vec{\mathbf{g}}} \quad = \quad \begin{bmatrix} \frac{\delta g_1}{\delta \beta_1} & \cdots & \frac{\delta g_1}{\delta \beta_{p+1}} \\ \vdots & & \vdots \\ \frac{\delta g_N}{\delta \beta_1} & \cdots & \frac{\delta g_N}{\delta \beta_{p+1}} \end{bmatrix} \tag{26}$$

- So far we have not committed ourselves to any particular form for the relationship $\vec{\mathbf{g}}(\vec{\beta})$ between the parameters $\vec{\beta}$ and the measured $\vec{\mathbf{y}}$. All we can assume is that $\vec{\mathbf{g}}(\vec{\beta}) \approx \mathbf{X} \cdot \vec{\beta}$.

- Keeping in mind the approximation shown in the previous bullet, one can argue that we may approximate the Jacobian shown in Eq. (26) by first calculating the difference between the vectors $\mathbf{X}\cdot(\vec{\beta}+\delta\vec{\beta})$ and $\mathbf{X}\vec{\beta}$, dividing each element of the difference vector by the β *increment*, and finally distributing the elements of the vector thus obtained according to the matrix shown in Eq. (26). [We will refer to this as the "`jacobian_choice=2`" option in the next section.]

- As a counterpoint to the arguments made so far for specifying the Jacobian, it is interesting to think that if one yielded to the temptation of creating an analytical form for the Jacobian using the $\vec{\mathbf{y}} = \mathbf{X}\cdot\vec{\beta}$ relationship, one would end up (at least theoretically) with the same solution as the linear least-squares solution of Eq. (17). A brief derivation on the next slide establishes this point.

- From the first element of the first row of the matrix in Eq. (26), we know from Eq. (10) that

$$(\mathbf{X}\vec{\beta})_i \quad = \quad x_{i,1}\beta_1 + \cdots + x_{i,p}\beta_p + \beta_{p+1} \qquad i = 1, \cdots, N \tag{27}$$

  Therefore,

$$\begin{aligned} \frac{\delta(\mathbf{X}\vec{\beta})_i}{\delta\beta_j} \quad &= \quad x_{i,j} \qquad i = 1, \cdots, N \quad and \quad j = 1, \cdots, p \\ \frac{\delta(\mathbf{X}\vec{\beta})_i}{\delta\beta_{p+1}} \quad &= \quad 1 \qquad i = 1, \cdots, N \end{aligned} \tag{28}$$

- Substituting these partial derivatives in Eq. (26), we can write for the $N \times (p+1)$ Jacobian:

$$J_{\mathbf{X}\vec{\beta}} \quad = \quad \begin{bmatrix} x_{1,1} & \cdots & x_{1,p} & 1 \\ \vdots & & & \vdots \\ x_{N,1} & \cdots & x_{N,p} & 1 \end{bmatrix} \quad = \quad \mathbf{X} \tag{29}$$

- Substituting this result in Eq. (25), we can write for the gradient of the cost function surface at a point $\vec{\beta}$ in the $\vec{\beta}$-hyperplane:

$$\nabla C(\vec{\beta}) \quad = \quad -2 \cdot \mathbf{X}^T \cdot \vec{\epsilon}(\vec{\beta}) \qquad (30)$$

- The goal of a gradient-descent algorithm is to find that point in the space spanned by the $p+1$ dimensional β vector where the gradient given by Eq. (30) is zero. When you set this gradient to zero while using $\vec{\epsilon}(\vec{\beta}) = \vec{\mathbf{y}} - \mathbf{X} \cdot \vec{\beta}$, you get exactly the same solution as given by Eq. (17).

- Despite the conclusion drawn above, it's interesting nonetheless to experiment with the refinement formulas when the Jacobian is to the matrix $\mathbf{X}$ as dictated by Eq. (29). [We will refer to this as the option "`jacobian_choice=1`" in the next section.]

- Shown below is a summary of the Gradient Descent Algorithm for refining the estimate produced by the linear least-squares algorithm of the previous section.

Step 1: Given the N observations of the dependent variable, while knowing at the same time the corresponding p predictor variables, we set up an algebraic system of equations as shown in Eq. (8). Note that in these equations, we have already augmented the predictor vector $\vec{\mathbf{x}}$ by the number 1 for its last element and the $\vec{\beta}$ vector by the intercept b (as argued in Section 3).

Step 2: We stack all the equations together into the form shown in Eq. (11) and (12):

$$\vec{\mathbf{y}} \quad = \quad \mathbf{X}\,\vec{\beta} \tag{31}$$

Step 3: We construct a least-squares estimate for the regression coefficients as shown in Eq. (18)

Step 4: We use the linear least-squares estimate as our starting point for the nonlinear least-squares refinement of the estimate:

$$\vec{\beta}_0 \ = \ \alpha \cdot (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{\mathbf{y}} \qquad (32)$$

This point is our starting point in the hyperplane spanned by the $\vec{\beta}$ vectors in our iterative search for the global minimum of the surface defined by the cost function $C(\beta)$ as given by Eq. (20):

$$C(\vec{\beta}) \ = \ \|\vec{\mathbf{y}} \ - \ \vec{\mathbf{g}}(\vec{\beta})\|^2 \qquad (33)$$

Step 5: We next estimate the Jacobian at the current solution point by using the matrix in Eq. (26).

Step 6: We now carry out the iterative march toward the global minimum using the formulas shown in Eqs. (19) and (25):

$$\vec{\beta}_{k+1} \quad = \quad \vec{\beta}_k \ + \ \gamma \cdot J_{\vec{\mathbf{g}}}^T(\vec{\beta}) \cdot (\vec{\mathbf{y}} - \mathbf{X}\vec{\beta}) \quad (34)$$

where we use for the Jacobian either of the two options "`jacobian_choice=1`" or "`jacobian_choice=2`" described earlier in this section. To these two options, we will add the "`jacobian_choice=0`" option to indicate the case when we do not want any refinement.

Step 7: We stop the iterations either after a prespecified number of them have been executed or when the step size $\|\vec{\beta}_{k+1} - \vec{\beta}_k\|$ falls below a prespecified threshold.

7. Regression Trees

- You can think of regression with a regression tree as a powerful generalization of the linear regression algorithm we have presented so far in this tutorial.

- Although you can certainly carry out polynomial regression with run-of-the-mill linear regression algorithms for modeling nonlinearities between the predictor variables and the dependent variable, specifying the degree of the polynomial is often a tricky business.

- Additionally, a polynomial can inject continuities between the predictor and the predicted variables that may not actually exist in the real data.

- Regression trees, on the other hand, give you a piecewise linear relationship between the predictor and the predicted variables that is freed from the constraints of superimposed continuities at the joins between the different segments.

8. The `RegressionTree` Class in My Python and Perl `DecisionTree` Modules

- The rest of this tutorial is about the `RegressionTree` class that is a part of my Python and Perl `DecisionTree` modules. More precisely speaking, `RegressionTree` is a subclass of the main `DecisionTree` class in both cases. Here are the links to the download pages for the `DecisionTree` modules:

  `https://pypi.python.org/pypi/DecisionTree/3.4.3`

  `http://search.cpan.org/~avikak/Algorithm-DecisionTree-3.43/lib/Algori`

  Just clicking on the links should take you directly to the relevant webpages at the Python and Perl repositories from where you can get access to the module files.

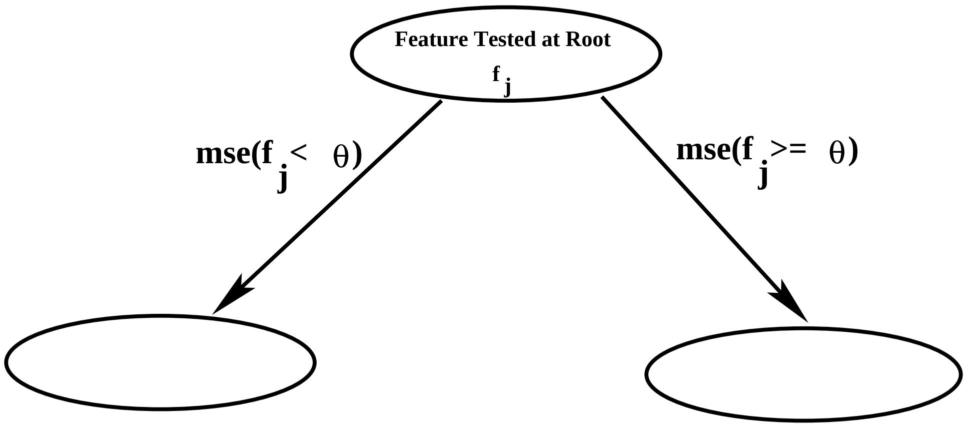
- By making `RegressionTree` a subclass of `DecisionTree`, the former is able to call upon the latter's functionality for computations needed for growing a tree. [These computations refer to keeping track of the feature names and their associated values, figuring the data samples relevant to a node in the tree taking into account the threshold inequalities on the branches from the root to the node, etc.]

- The current version of the `RegressionTree` class can only deal with purely numerical data. This is unlike what the `DecisionTree` module is capable of. The `DecisionTree` module allows for arbitrary mixtures of symbolic and numerical features in your training dataset.

- My goal is that a future version of `RegressionTree` will call upon some of the additional functionality already built into the `DecisionTree` class to allow for a mixture of symbolic and numerical features to be used for creating a predictor for numeric variable.

- The basic idea involved in using formulas derived in the previous section in order to grow a regression tree is simple. The calculations we carry out at each step are listed in the next several bullets.

- As illustrated by the figure shown below, you start out with all of your data at the root node and you apply the linear regression formulas there for every bipartition of every feature.



- For each feature, you calculate the MSE (Mean Squared Error) per sample for every possible partition along the feature axis.

- For each possible bipartition threshold θ, you associate it with the larger of the MSE for the two partitions. [To be precise, this search for the best possible partitioning point along each feature is from the 10th point to the 90th point in rank order of all the sampling points along the feature axis. This is done in order to keep the calculations numerically stable. If the number of points retained for a partition is too few, the matrices you saw earlier in the least-squares formulas may become singular.]

- And, with each feature overall, you retain the minimum of the MSE value calculated in the previous step. We refer to retaining this minimum for all threshold-based maximum values as the `minmax` operation applied to a features.

- You select that feature at a node that yields the smallest value for the MSE values associated with the feature in the previous step.

- At each node below the root, you only deal with the data samples that are relevant to that node. You find these samples by applying the branch thresholds along the path from the root to the node to all of the training data.

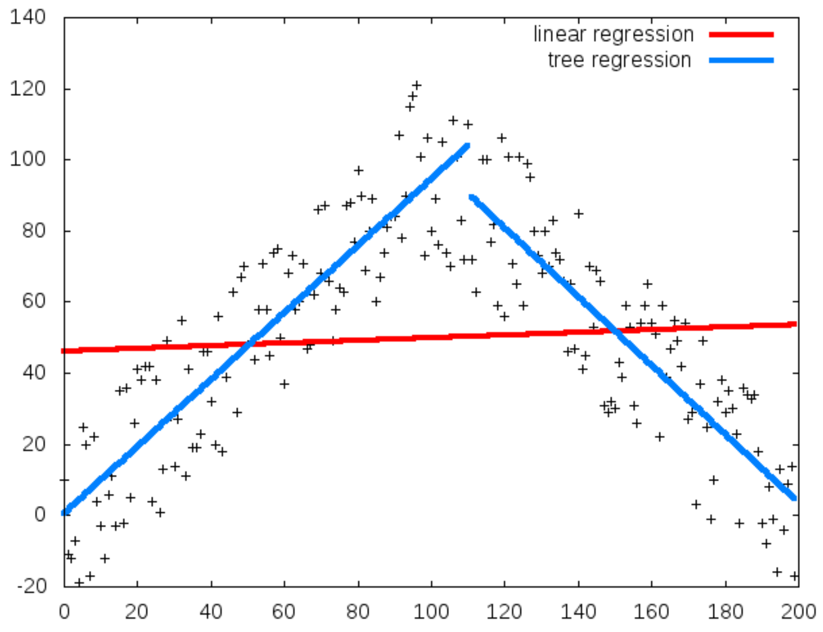
- You use the following criteria for the termination condition in growing a tree:

  - A node is not expanded into child nodes if the minmax value at that node is less than the user-specified `mse_threshold`.

  - If the number of data samples relevant to a node falls below 30, you treat that node as a leaf node.

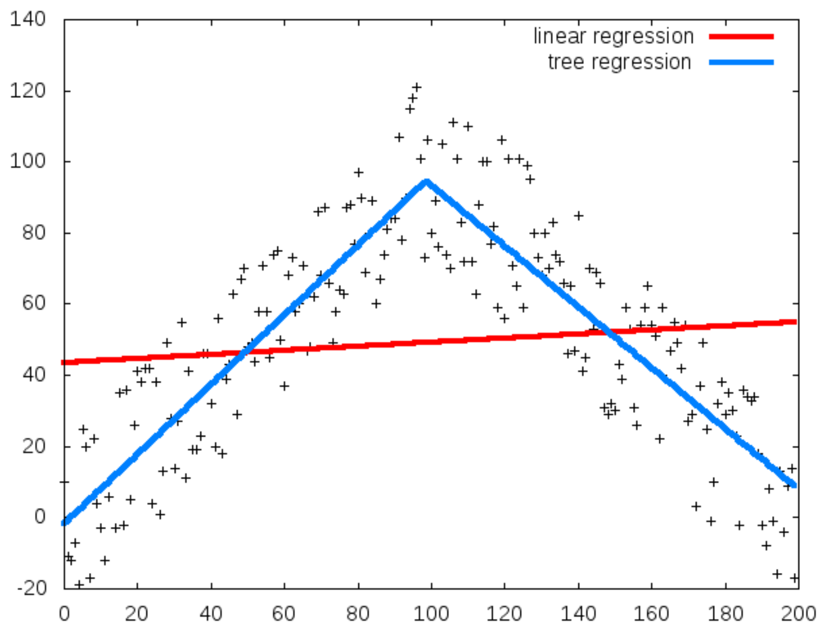
- I'll now show the sort of results you can obtain with the `RegressionTree` class.

- Shown below is the result returned by the script whose basename is `regression4` in the `ExamplesRegression` subdirectory of the `DecisionTree` module. In this case, we have a single predictor variable and, as you'd expect, one dependent variable. [The predictor variable is plotted along the horizontal axis and the dependent variable along the vertical.]

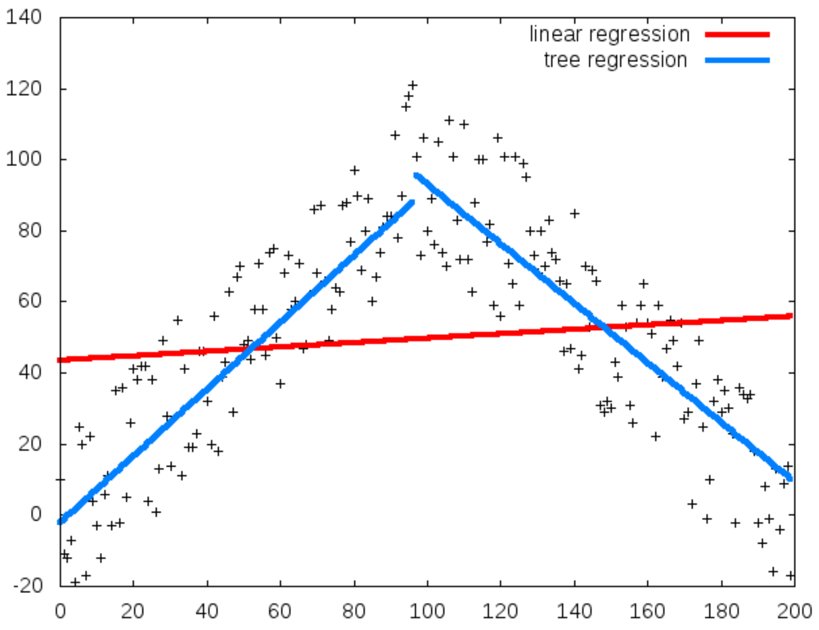


- The blue line shows the result returned by tree regression and the red line the result of linear regression.

- The result shown in the previous slide was obtained with the "jacobian_choice=0" option for the Jacobian. As the reader will recall from the previous section, this option means that we did not use the refinement. When we turn on the option "jacobian_choice=1" for refining the coefficients, we get the result shown below:

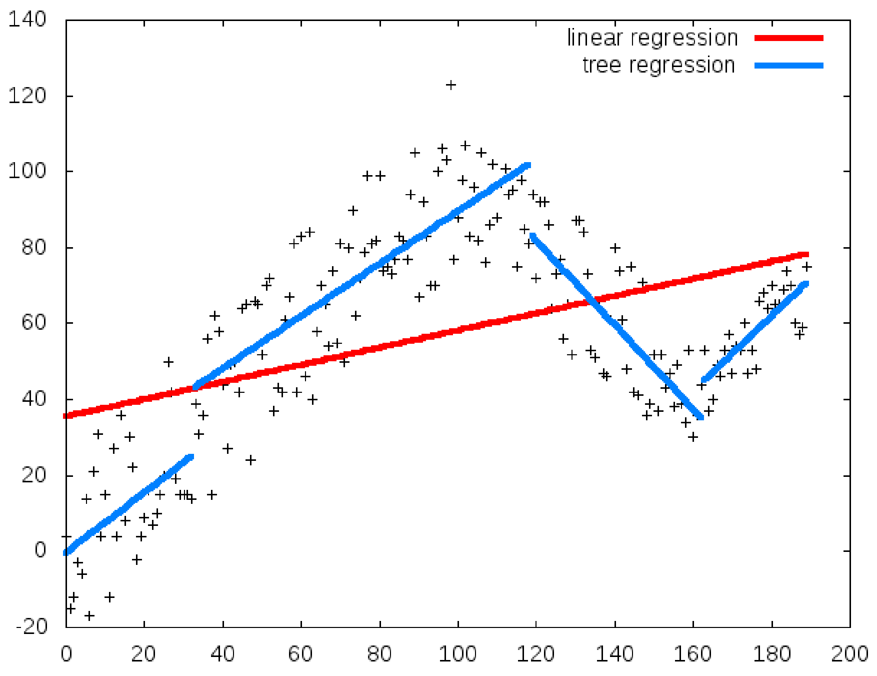


- And with the option "`jacobian_choice=2`", we get the result shown below:



- To me, all three results shown above look comparable. So it doesn't look like that we again anything by refining the regression coefficients through gradient descent. This observation is in agreement with the statements made at the beginning of Section 6 of this tutorial.

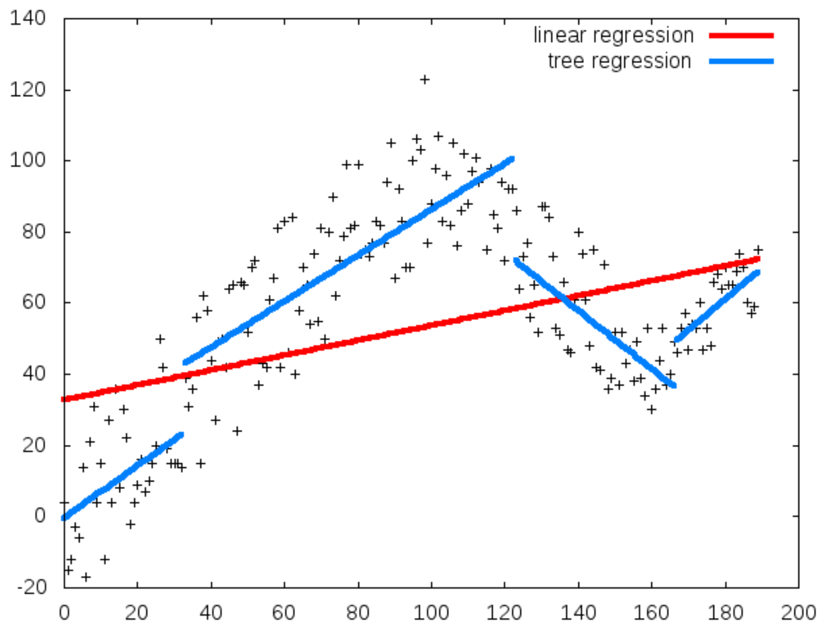
- The next example, shown below, also involves only a single predictor variable. However, in this case, we have a slightly more complex relationship between the predictor variable and the dependent variable. Shown below is the result obtained with the "`jacobian_choice=0`" option.



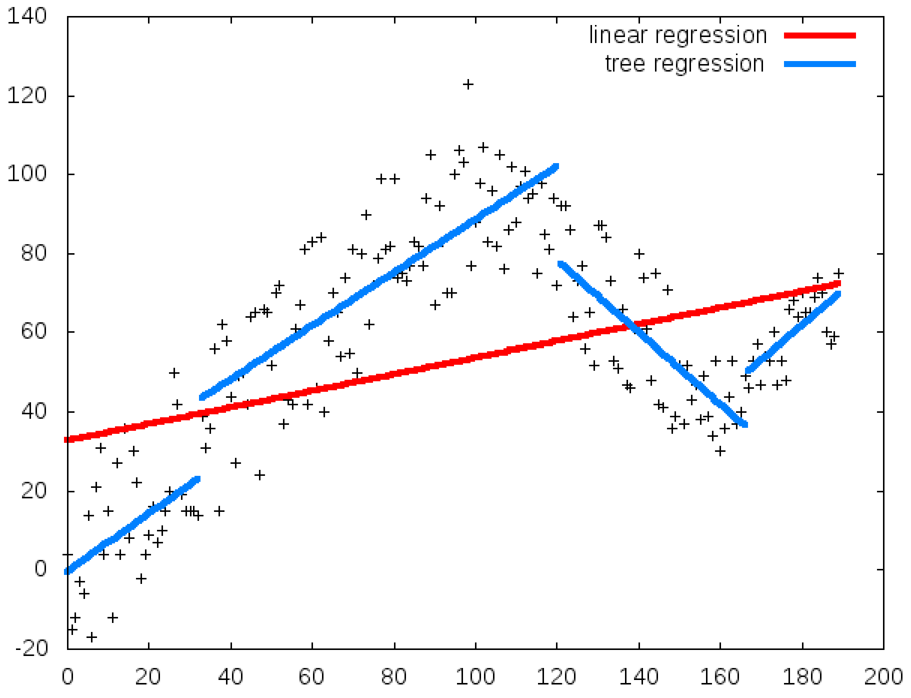
- The red line again is the linear regression fit to the data, and the blue line the output of tree regression.

- The result shown on the previous slide was calculated by the script whose basename is `regression5` in the `ExamplesRegression` directory.

- If we use the "`jacobian_choice=1`" option for the same data, we get the following result:

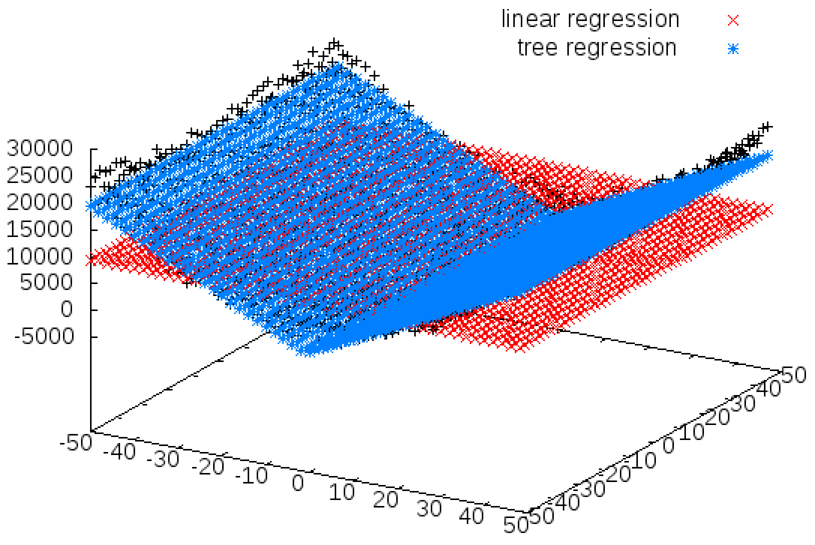


- And, shown on the next slide is the result with the "`jacobian_choice=2`" option.



- Based on the results shown in the last three figures, we have the same overall conclusion as presented earlier: the refinement process, while altering the shape of the regression function, does not play that noticeable role in improving the results.

- Shown below is the result for the case when we have two predictor variables. [Think of the two variables as defining a plane and the dependent variable as representing the height above the plane.]



- The red scatter plot shows the fit created by linear regression. The tree based regression fits two separate planes to the data that are shown as the blue scatter plot. The original noisy data is shown as the black dots.

9. API of the `RegressionTree` Class

- Here's how you'd call the `RegressionTree` constructor in Python:

```
import RegressionTree
training_datafile = "gendata6.csv"
rt = RegressionTree.RegressionTree(
                 training_datafile = training_datafile,
                 dependent_variable_column = 3,
                 predictor_columns = [1,2],
                 mse_threshold = 0.01,
                 max_depth_desired = 2,
                 jacobian_choice = 0,
     )
```

and here's you would call it in Perl:

```
use Algorithm::RegressionTree;
my $training_datafile = "gendata6.csv";
my $rt = Algorithm::RegressionTree->new(
                 training_datafile => $training_datafile,
                 dependent_variable_column => 3,
                 predictor_columns => [1,2],
                 mse_threshold => 0.01,
                 max_depth_desired => 2,
                 jacobian_choice => 0,
         );
```

- Note in particular the constructor parameters:

```
dependent_variable

predictor_columns

mse_threshold

jacobian_choice
```

- The first of these constructor parameters, `dependent_variable`, is set to the column index in the CSV file for the dependent variable. The second constructor parameter, `predictor_columns`, tells the system as to which columns contain values for the predictor variables. The third parameter, `mse_threshold`, is for deciding when to partition the data at a node into two child nodes as a regression tree is being constructed. Regarding the parameter `jacobian_choice`, we have already explained what that stands for in Section 6.

- If the minmax of MSE (Mean Squared Error) that can be achieved by partitioning any of the features at a node is smaller than `mse_threshold`, that node becomes a leaf node of the regression tree.

- What follows is a list of the methods defined for the `RegressionTree` class that you can call in your own scripts:

get_training_data_for_regression()

Only CSV training datafiles are allowed. Additionally, the first record in the file must list the names of the fields, and the first column must contain an integer ID for each record.

construct_regression_tree()

As the name implies, this method actually construct a regression tree.

display_regression_tree(" ")

Displays the regression tree, as the name implies. The white-space string argument specifies the offset to use in displaying the child nodes in relation to a parent node.

prediction_for_single_data_point(root_node, test_sample)

You call this method after you have constructed a regression tree if you want to calculate the prediction for one sample.

The parameter `root_node` is what is returned by the call `construct_regression_tree()`. The formatting of the argument bound to the `test_sample` parameter is important. To elaborate, let's say you are using two variables named `xvar1` and `xvar2` as your predictor variables. In this case, the `test_sample` parameter will be bound to a list that will look like

```
['xvar1 = 23.4', 'xvar2 = 12.9']
```

Arbitrary amount of white space, including none, on the two sides of the equality symbol is allowed in the construct shown above.

A call to this method returns a dictionary with two <key,value> pairs. One of the keys is called `solution_path` and the other is called `prediction`. The value associated with key `solution_path` is the path in the regression tree to the leaf node that yielded the prediction. And the value associated with the key `prediction` is the answer you are looking for.

predictions_for_all_data_used_for_regression_estimation(root_node)

This call calculates the predictions for all of the predictor variables data in your training file. The parameter `root_node` is what is returned by the call to `construct_regression_tree()`. The values for the dependent variable thus predicted can be seen by calling `display_all_plots()`, which is the method mentioned below.

display_all_plots()

This method displays the results obtained by calling the prediction method of the previous entry. This method also creates a hardcopy of the plots and saves it as a '.png' disk file. The name of this output file is always "`regression_plots.png`".

mse_for_tree_regression_for_all_training_samples(root_node)

This method carries out an error analysis of the predictions for the samples in your training datafile. It shows you the overall MSE (Mean Squared Error) with tree-based regression, the MSE for the data samples at each of the leaf nodes of the regression tree, and the MSE for the plain old linear regression as applied to all of the data. The parameter `root_node` in the call syntax is what is returned by the call to `construct_regression_tree()`.

bulk_predictions_for_data_in_a_csv_file(root_node, filename, columns)

Call this method if you want to apply the regression tree to all your test data in a disk file. The predictions for all of the test samples in the disk file are written out to another file whose name is the same as that of the test file except for the addition of '`_output`' in the name of the file. The parameter `filename` is the name of the disk file that contains the test data. And the parameter `columns` is a list of the column indices for the predictor variables.

10. THE ExamplesRegression DIRECTORY

- The `ExamplesRegression` subdirectory in the main installation directory shows example scripts that you can use to become familiar with the regression trees and how they can be used for nonlinear regression. If you are new to the concept of regression trees, start by executing the following Python scripts without changing them and see what sort of output is produced by them (I show the Python examples on the left and the Perl examples on the right):

```
regression4.py     regression4.pl
regression5.py     regression5.pl
regression6.py     regression6.pl
regression8.py     regression8.pl
```

- The `regression4` script involves just one predictor variable and one dependent variable. The training data for this exercise is drawn from the `gendata4.csv` file.

- Th data file `gendata4.csv` contains strongly nonlinear data. When you run the script `regression4`, you will see how much better the result from tree regression is compared to what you can get with linear regression.

- The `regression5` script is essentially the same as the previous script except for the fact that the training datafile in this case, `gendata5.csv`, consists of three noisy segments, as opposed to just two in the previous case.

- The script `regression6` deals with the case when we have two predictor variables and one dependent variable. You can think of the data as consisting of noisy height values over an $(x1, x2)$ plane. The data used in this script is drawn from the `gen3Ddata1.csv` file.

- Finally, the script `regression8` shows how you can carry out bulk prediction for all your test data records in a disk file. The script writes all the calculated predictions into another disk file whose name is derived from the name of the test datafile.

11. ACKNOWLEDGMENT

The presentation in Section 6 of nonlinear least-squares for the refinement of the regression coefficients was greatly influenced by the comments received from Tanmay Prakash during my presentation of this tutorial. Tanmay has the uncanny ability to spot the minutest of flaws in any explanation.