MCS507 Project Two:
Study a Mathematical Software Package: Earth

Prepared By: Adam McElhinney October 28, 2012

---

1 Assignment One: Overview and Illustrative Example

Our first objective is to identify the main problem this software aims to solve. We then provide an overview of the software package and the algorithms used. We conclude with an illustrative example that displays the utility of this software.

1.1 Main Problem this Software Aims to Solve

The Earth software package is an implementation of the Multivariate Adaptive Regression Splines (MARS) technique developed by Jerome H. Friedman of Stanford University. Interestingly, the reference MARS is copy written, thus open source implementations of the software are referred to as Earth. The purpose of this technique is similar to that of all regression techniques, namely to fit a function to a set of data. However, the Earth software specifically seeks to do this in a manner that has the following advantages:

1. Ability to easily handle non-linear data sets
2. Ease of interpretation
3. Ability to handle large datasets
4. Automatic variable selection

1.2 Overview of the Software and Algorithms

The Earth software package relies on the application of splines to construct the function to predict the target variable. Broadly speaking, a spline is simply a function constructed in various segments from other polynomial functions. The MARS technique relies on a type of splines known as hinge functions, which can be represented in the form:

$$h_j = \max(0, x - c) \tag{1}$$

The constant c is referred to as the knot. The Earth software then uses these functions to divide the data into mutually exclusive segments and then fit a model to each individual function. This is done in such a manner as to minimize some objective function. One commonly used objective function is the residual sum of squares (RSS).

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{2}$$

To illustrate, let us consider the typical regression equation where we seek to build a model that predicts the value of the y_i-th obvseration based on some values of x.

$$y_i = \beta_0 + \sum_{i=1}^{M} \beta_i * x_i \tag{3}$$

The Earth software then modifies this model to have the following form (where h_i represents the hinge function in equation 1).

$$y_i = \beta_0 + \sum_{i=1}^{M} \beta_i * h_i(x_i) \tag{4}$$

To fit this model, the Earth software relies on two distinct steps.

1. Forward Pass: In this step, the model starts with just one intercept term composed of the mean of the target (y) variables. Then the software divides the data into two distinct sets via a pair of basis functions. The area dividing point between these two areas is the knot referred to in equation 1. The position of this knot is calculated such that it results in the maximum reduction in the RSS in equation 2. This process is repeated until the software has reached a maximum number of terms (as defined by the user prior to initializing the model), or until the change in RSS is too small to continue.

2. Backward Pass: The result of the first step will be a model that fits the data set extremely closely, however it is likely that the model will generalize well to other data. Thus, the backward pass (sometimes referred to as pruning) is applied. In this step, the model searches for the basis functions that contribute to the smallest increase in the goodness of fit. However, the RSS will always favor a model with more parameters, so a goodness of fit measure that penalizes additional parameters is needed. This is referred to as the Generalized Cross Validation GCV.

$$GCV = \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{1 - \frac{C}{n}} \tag{5}$$

where $C = 1 + c * d$, n is the number of observations in the data, d is the number of independent basis functions and c is a penality for adding a basis function.

1.3 Illustrative Example of the Earth Software

To illustrate usage of the Earth software, a simple example data set was generated using the Data Painter tool from the Orange Machine Learning Python library. This data set is composed of an x variable which we attempt to use to predict the value of the y variable. We begin by loading the data, verifying the column names, and then inspect the data.

```
15
16  # Remember to add the "class" keyword to the third line, under the target variable
        . See here:
17  # http://orange.biolab.si/doc/tutorial/load-data/
18  data = orange.ExampleTable("painted_data_wo_outlier")
19  print data.domain.attributes
20  print data[:4]
```

Listing 1: Load and Inspect the Data

After reviewing the data, we convert it to *Numpy* arrays and plot the data using *Matplotlib*.

```
21 X, Y = data.to_numpy("A/C")
22
23 pl.plot(X, Y, ".r")
24 pl.title('Example Data Set')
25 pl.show()
```

Listing 2: Convert to Array and Plot

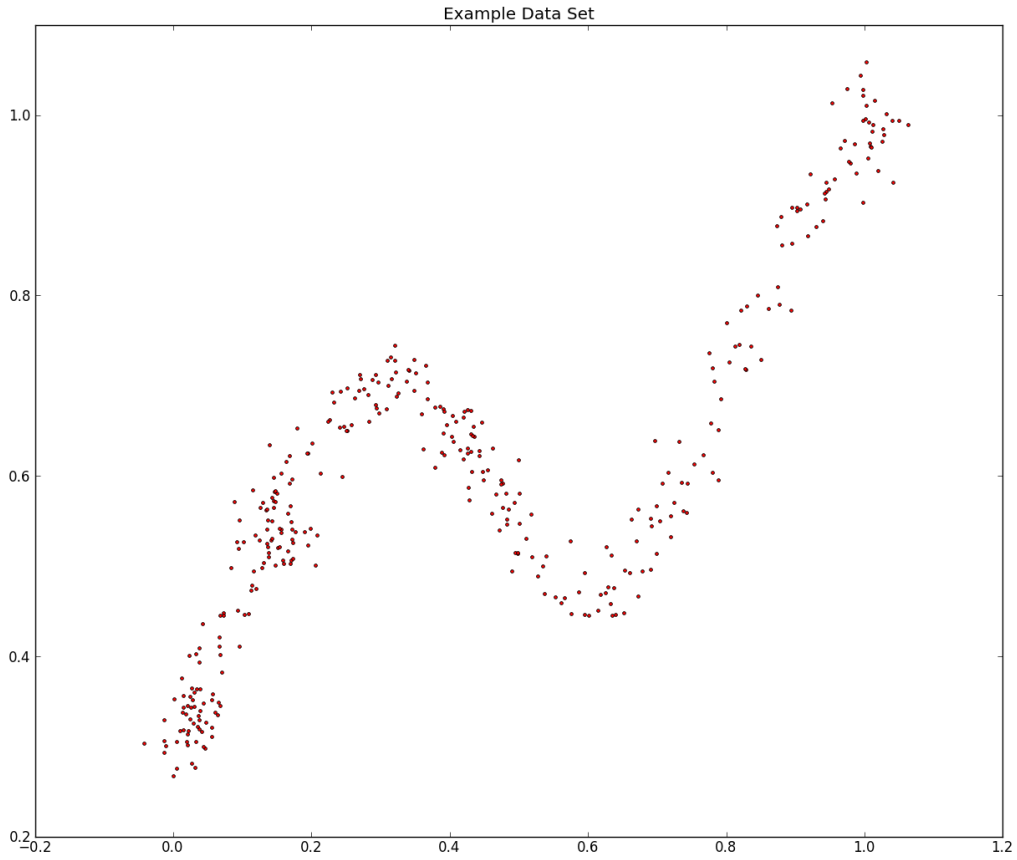

Fig. 1: Example Data

As can be seen from the graph, the data exhibits an irregular shape and the density of the points varies as one moves along the x-axis. Thus, this example should prove interesting to test to utility of the Earth software. Next, we use Earth to fit the data and examine the model.

```
27 earth_predictor = earth.EarthLearner(data)
28 print earth_predictor
```

Listing 3: Fit the Data

$$Y = 1.486 + 1.445 * \max(0, X - 0.744) - 1.590 * \max(0, 0.744 - X) - 1.818 * \max(0, X - 0.244) \quad (6)$$

$$+2.625 * \max(0, X - 0.640) - 0.682 * \max(0, X - 0.404) - 1.661 * \max(0, X - 0.979)$$

Next, we would like to see how well this model predicts the data. One strategy is to plot the values the Earth model predicts and compare them with the actual data. This is done by creating a *Numpy* array of x-values and then using the Earth model to predict the values of y.

```
earth_predictor = earth.EarthLearner(data)
print earth_predictor
linspace = numpy.linspace(min(X), max(X), 20)
predictions = [earth_predictor([s, "?"]) for s in linspace]
pl.plot(X, Y, ".r")
pl.plot(linspace, predictions, "-b")
pl.title('Example Data Set with Line Fit by MARS')
pl.show()
```

Listing 4: Compare the Predicted vs Actual Values and Plot the Results

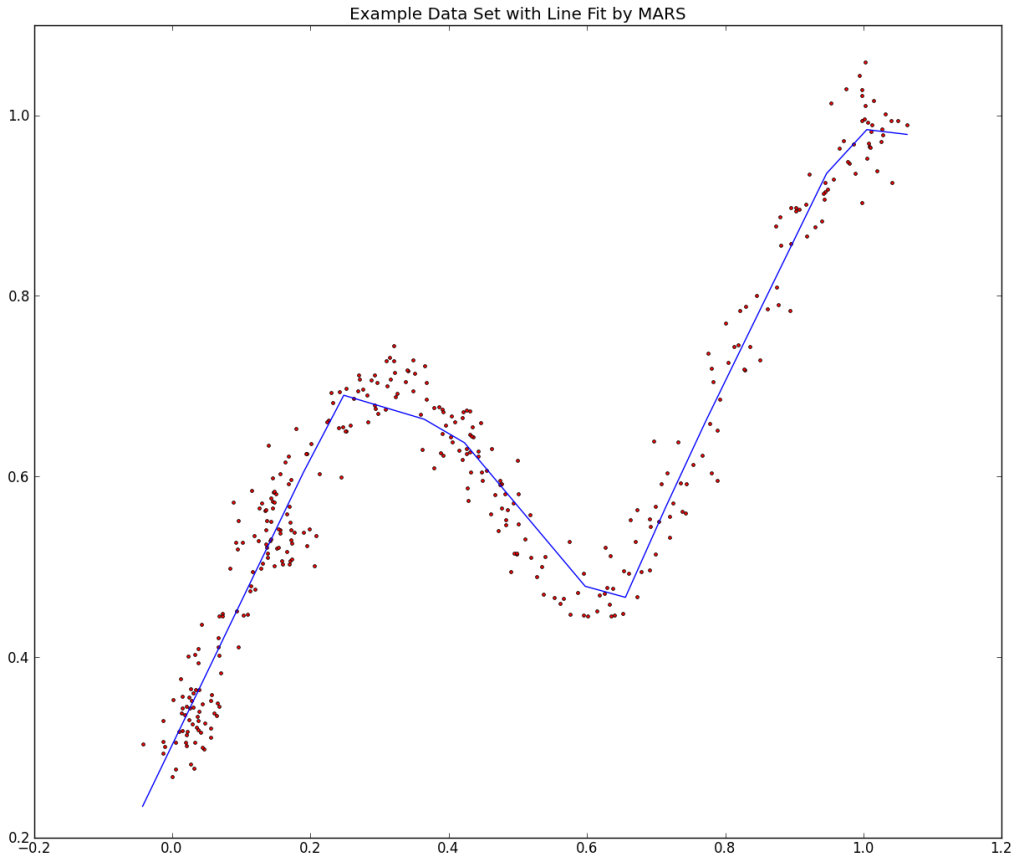

Fig. 2: Example Data Set with Line Fit by MARS

Figure 2 shows us that the model output by the Earth software fits the data quite well. It appears that the knots are located close to the inflection points in the data.

2 Assignment Two: Using the Software from Python and Extending Illustrative Example

Our second objective is to review how to utilize this software from Python. We then extend the illustrative example from Assignment One to further display the advantages and disadvantages of the Earth software.

2.1 Utilizing the Earth Software from Python

The Earth software is created by Stephen Milborrow and runs in an R environment. R is an open source enviroment for statistical computer. Interestingly, the R language is an extension of a language developed at Bell Laboratories in the late 1970's and early 1980's. Although the Earth software runs in R, it is also available as a standalone program and as a C library.

Access to the Earth software from Python is done via the Orange module, which is a module focused on data mining and machine learning. It is also available as a standalone software program complete with a GUI. The Orange module and software is developed by the Bioinformatics Laboratory at the Faculty of Computer and Information Science in the University of Ljubljana, Slovenia. In addition to the implementation of the Earth software, the Orange module features implementations of the majority of cutting-edge machine learning and data mining techniques, including techniques from the following categories:

1. Summary Statistics
2. Classification
3. Regression
4. Ensemble Algorithms
5. Clustering
6. Network Analysis

2.2 Extending the Illustrative Example

We now seek to extend our Illustrative Example from Assignment One. The natural question arries, "How does the Earth software compare with other traditional techniques?" To answer this question, we utilize the *polyfit* functions from the *NumPy* module to fit polynomial functions of the 3rd, 4th and 5th degrees. We then plot the predicted versus actual values and compare them to the Earth software.

```
43 p3=polyfit(x,y,3)
44 pp3=poly1d(p3)
45
46 p4=polyfit(x,y,4)
47 pp4=poly1d(p4)
48
49 p5=polyfit(x,y,5)
50 pp5=poly1d(p5)
51
52 pl.plot(X, Y, ".r")
53 pl.plot(linspace, predictions, "-b")
54 pl.plot(linspace,pp3(linspace),"-g")
55 pl.plot(linspace,pp4(linspace),"-m")
56 pl.plot(linspace,pp5(linspace),"-y")
57 pl.title('Example Data Set with Line Fit by MARS and Higher Order Polynomials')
58 pl.legend(['Data','Mars','3rd Degree','4th Degree','5th Degree'],loc=2)
59 pl.show()
```

Listing 5: Fit Polynomial Functions of the 3rd, 4th and 5th Degrees.

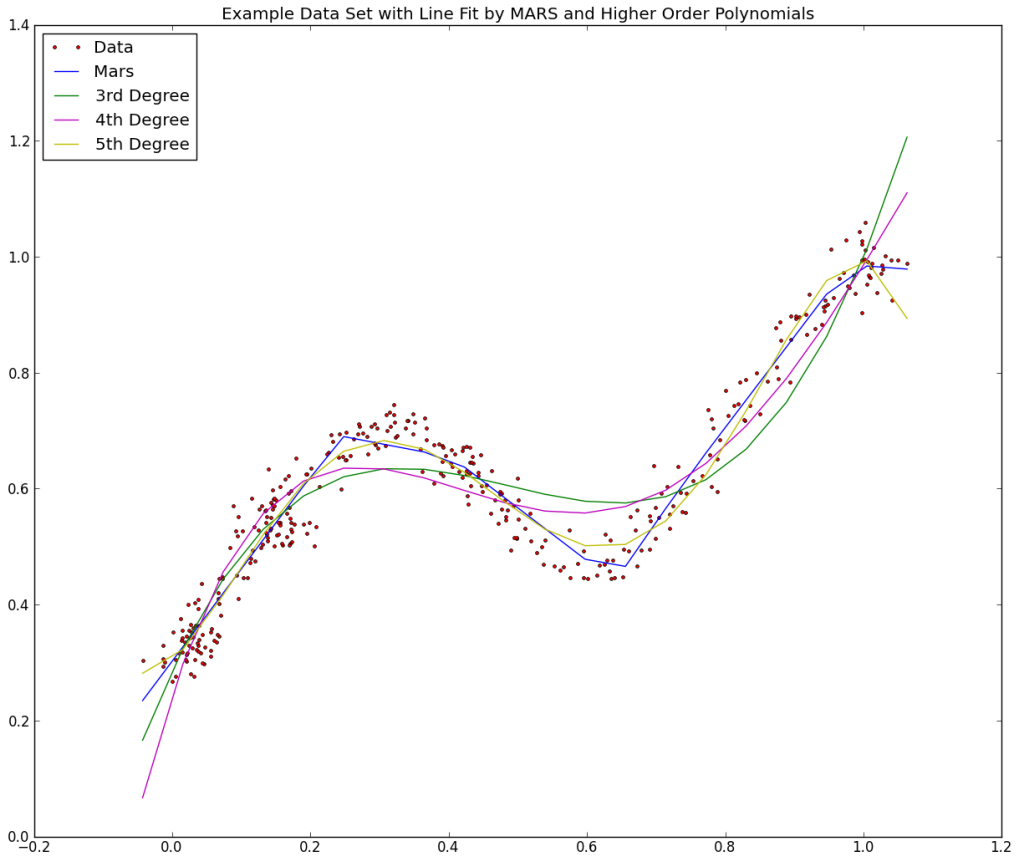

Fig. 3: Comparison of the Fit of Mars vs Polynomial Functions

As Figure 3 shows, visually it appears that the Mars model fitted via the Earth software has closer fit than the polynomial functions. Further, the reader should bear in mind that the fitting of the polynomial functions would not be automatic and requires manual input from the modeler, versus the Earth software which automatically selects the best model structure. To empirically test the fit of the 4 models, we define a function to compute the RSS.

```
63 # Create function to calc RSS
64 def rss(y_obs, y_predicted):
65     r=0
66     for i in range(len(y_obs)):
67         t=(y_obs[i]-y_predicted[i])**2.
68         r=r+t
69     return r
70
71 # Calc RSS for all models
72 y_predicted1=[earth_predictor([X[i], "?"]) for i in range(len(X))]
73 y_predicted3=pp3(X)
74 y_predicted4=pp4(X)
```

```
y_predicted5=pp5(X)

r1=rss(Y, y_predicted1)
r3=rss(Y, y_predicted3)
r4=rss(Y, y_predicted4)
r5=rss(Y, y_predicted5)
```

Listing 6: Calculate the RSS for the Four Models

Tab. 1: RSS For the Four Models

Method	RSS
MARS	0.59
3rd Degree	1.57
4th Degree	1.32
5th Degree	0.60

As Table 1 shows, the MARS model has the lowest RSS, indicating superior fit as compared to the 3 other competing models.

3 Assignment Three: More Challenging Inputs and Examples

3.1 The Boston Housing Data Set

To provide a more interesting data set to examine the Earth software implementation of the MARS algorithm, we sought assistance from the UCI Machine Learning Repository which publishes numerous data sets for the benefit of researchers. We chose the Boston Housing Data Set, which is a famous data set outlining housing prices in suburban Boston. The data set contains the following variables:

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000's

This data set was chosen for the following reasons:

1. The dataset contains categorical, integer and real variables
2. The number of instances ($n = 506$) and the number of attributes ($m = 14$) is representative of many typical real-world data sets
3. The data set does not contain missing values, which the MARS algorithm does not address in its basic form

To begin the analysis, we first load in the data. Then, to explore the data set, we constructed a function to compute scatter plots of all the data versus the target variable.

```
# Bring in the data using the C4.5 format, which brings in the .data file and then
    the .names file
data = orange.ExampleTable("housing")
print data.domain.attributes

names=str(data.domain).replace('[','').replace(']','').split(',')
# Make Scatterplots of all the variables versus the target variable
def scatterplot(data, data_name, target_var):
    import matplotlib.pyplot as plt
    M = len(data[0])-1
    N=len(data)
    fig = plt.figure()

    y=[float(data[l][target_var]) for l in range(N)]

    for i in range(M):
        ax = fig.add_subplot(int(M/4.)+1,4,i+1)
        py.setp(ax.get_xticklabels(), visible=False)
        x=[float(data[k][i]) for k in range(N)]
        ax.scatter(x,y)
        #ax.set_xlabel(data_name[i],size='10')
        ax.set_title(data_name[i],size='10')
        #ax.set_ylabel(data_name[target_var],size='10')
    return fig
fig = scatterplot(data, names, len(data[0])-1)
```

Listing 7: Load the Data and Explore

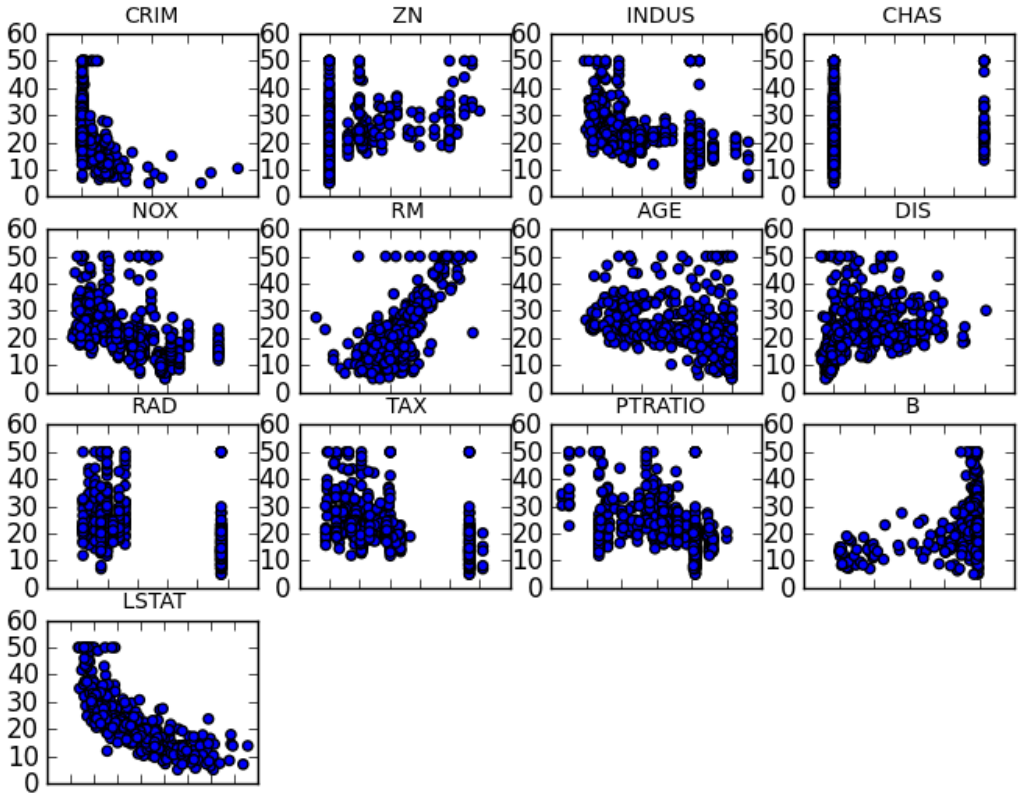

Fig. 4: Scatterplot of the Independent Variables vs the Target Variable

3.2 Fitting the MARS Model

Simiarly to previous examples, the Earth software lets us very quickly view the data, fit the data to the model and then view the model.

```
53 # View the data
54 for i in range(5):
55     print data[i]
56
57
58 earth_predictor = earth.EarthLearner(data)
59
60 print earth_predictor
```

Listing 8: Load the Data and Explore

$$MEDV = 33.518 - 0.687 * \max(0, CRIM - 4.422) - 1.132 * \max(0, 4.422 - CRIM) \quad (7)$$
$$+0.560 * \max(0, CRIM - 12.048) - 28.029 * \max(0, NOX - 0.488) + 7.127 * \max(0, RM - 6.431)$$
$$-0.662 * \max(0, DIS - 2.436) + 5.010 * \max(0, 2.436 - DIS) + 0.031 * \max(0, 296.000 - TAX)$$

$$-0.637 * \max(0, PTRATIO - 14.700) + 1.692 * \max(0, 14.700 - PTRATIO) - 0.355 * \max(0, B - 393.360)$$
$$-0.007 * \max(0, 393.360 - B) - 0.598 * \max(0, LSTAT - 6.120)$$
$$+2.428 * \max(0, 6.120 - LSTAT) + 0.716 * \max(0, LSTAT - 25.790)$$

We then score the data and utilize our previous function to compute the RSS

```
62 dl=list(data)
63
64 # Predict all the data
65 y_hat=[]
66 for i in range(0,len(dl)):
67     t=earth_predictor.predict(dl[i])
68     y_hat.append(t[0])
69
70 os.chdir("C:/Documents and Settings/amcelhinney/My Documents/GitHub/MCS507ProjectTwo
       /src/")
71 from earth_example import rss
72 X, Y = data.to_numpy("A/C")
73
74 y_1=rss(Y,y_hat)
```

Listing 9: Compute the RSS

The Earth software also allows us to compute and graph the relative importance of each of the variables on the target variable via the *evimp* and *plot_evimp* methods.

```
75 Orange.regression.earth.plot_evimp(earth_predictor.evimp())
```

Listing 10: Load the Data and Explore

As the graph below shows, the number of rooms the house has (variable RM) and the percentage of the population that is lower status in a given area (variable $LSTAT$) are by far the two strongest predictors of housing price.

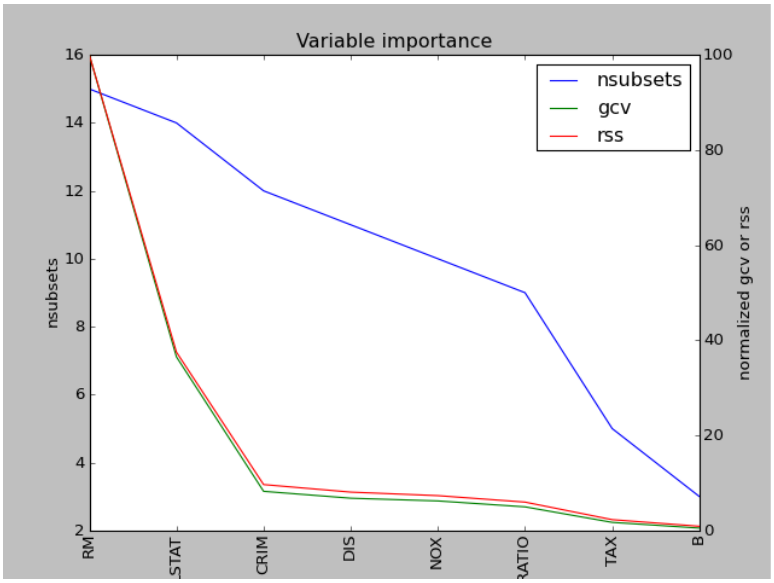

Fig. 5: Relative Importance of the Independent Variables on the Target Variable

Similarly to Assignment 2, we compare the performance of the MARS algorithm versus the traditional least squares regression. However, this time we utilize the *OLS* class from the *SciPy* module.

```
77 # Compute using the standard regression model
78 import ols
79 model=ols.ols(Y,X,names[len(names)-1],names[:len(names)-1])
80 model.summary()
```

Listing 11: Fit the OLS Model

Unlike the regression fitting packages used in Example 2, the *Ols* class provides us with a full regression output, similar to what one would find in R, SAS, or any other dedicated statistics package.

```
================================================================================
Dependent Variable:  MEDV
Method: Least Squares
Date:  Sun, 28 Oct 2012
Time:  20:47:55
# obs:                 506
# variables:        14
================================================================================
variable     coefficient     std. Error      t-statistic     prob.
================================================================================
const           36.459491      5.103459      7.144075      0.000000
CRIM            -0.108011      0.032865     -3.286517      0.001087
 ZN              0.046420      0.013727      3.381576      0.000778
 INDUS           0.020559      0.061496      0.334311      0.738287
 CHAS            2.686734      0.861580      3.118381      0.001925
 NOX           -17.766615      3.819744     -4.651258      0.000004
 RM              3.809865      0.417925      9.116140      0.000000
 AGE             0.000692      0.013210      0.052402      0.958229
 DIS            -1.475567      0.199455     -7.398003      0.000000
 RAD             0.306050      0.066346      4.612900      0.000005
 TAX            -0.012335      0.003761     -3.280009      0.001112
 PTRATIO          -0.952747      0.130827     -7.282511      0.000000
 B               0.009312      0.002686      3.466793      0.000573
 LSTAT           -0.524758      0.050715     -10.347146      0.000000
================================================================================
Models stats                        Residual stats
================================================================================
R-squared             0.740643          Durbin-Watson stat    1.078375
Adjusted R-squared    0.733790          Omnibus stat          178.041109
F-statistic           108.076667          Prob(Omnibus stat)    0.000000
Prob (F-statistic)    0.000000                  JB stat               783.126151
Log likelihood       -1498.804297                       Prob(JB)              0.000000
AIC criterion         5.979464          Skew                  1.520713
BIC criterion         6.096403          Kurtosis              8.281482
================================================================================
```

Fig. 6: Output from the *Ols* Class in *SciPy*

Interestingly, the t-statistics for the RM and $LSTAT$ variables are among the largest. This is in alignment with the graph of relative variable importance examined previously.

```
83 # Get the regression coefficients
84 coeff=model.b
85 # Score the data
86 y_hat_ols=[]
87 for i in range(len(data)):
```

```
88      y_hat=coeff[0]
89      for j in range(len(coeff)-1):
90          #print coeff[j+1]
91          #print data[i][j]
92          y_hat=y_hat+data[i][j]*coeff[j+1]
93      y_hat_ols.append(y_hat)
94  # Compute the RSS
95  y_2=rss(Y, y_hat_ols)
```

Listing 12: Calculate the RSS for the OLS Model

Again following the convention from Example 2, we score the data using the OLS Model and compute the RSS.

Tab. 2: RSS For MARS vs OLS

Method	RSS
MARS	6311
OLS	11079

Once again the Earth software shows its utility because the MARS algorithm was better able to predict the target variable as compared to the OLS.

4 Conclusion

We have shown how the Earth software package implements the MARS algorithm and how it can be very useful for quickly solving real world problems. We demonstrated how it easily beats least squares regression, while requiring less work from the actual modeler. However, this is by no means a definitive answer. Some factors beyond the scope of this assignment, but relevant for future inquiry are:

1. Execution time (particularly on large data sets)
2. High dimensional data (data sets where there are a much larger amount of explanatory variables as compared with the number of observations)
3. Various variable transformations and their ability to improve the quality of the OLS models
4. Calculating statistical significance and variance with the MARS model
5. Available heuristics to improve MARS calculation time, including the Fast Mars model
6. Utilizing MARS for purposes of classification
7. Implementing our own MARS algorithm and comparing performance to that of the Earth software

In summation, the Earth software package is a cross-platform, versatile tool with applications to real world data sets. It should be considered by any modeler seeking a fast, intuitive and accurate regression modeling technique.