Predicting Abalone Age Through Ridge Regression

Michael Szczepaniak

Revised: January 2016

Contents

Introduction

There are a number of packages in R, Python, and other languages that can perform linear regression analysis (LRA) on a given dataset. While the use of these packages saves a data scientist or analysis quite a bit of time, it is important to do these from scratch every now and then to reenforce the concepts behind this most fundamental of machine learning algorithms. By doing so, we deepen our intuitions that allow us to make better use of the available tools.

In this project, an LRA was performed on a dataset of characteristics (features) related to the number of abalone rings which was treated as the independent or response variable. This analysis was performed in two ways. First, it was done "from scratch" using nothing but base R language constructs such as matrix operations. Second, the analysis is duplicate used the caret package.

The analysis starts with a quick exploratory data analyisis (EDA) which begins with standardizing the indepependent variables (features) before constructing a matrix of scatter plots in order to gain a high level understanding of the relationships between feature pairs. The features analyzed were: Sex (Male, Female, or Infant), Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, and Shell weight.

Following the pairs plot, the construction of the linear model was initiated by randomly partitioning the data into training and test sets. Following the data partitioning, the coefficients for the linear model were calculated for various ridge regression parameters (λ) on the training set. The λ which minimized the root mean square error (RMSE) was determined and then used in the model to make some predictions.

The parameters for the optimized linear model was as follows: [put table output here]

The optimized ridge regression parameter λ was found to be 0.8. The mean and the std dev columns above were used in the standardization which will be explained in detail later.

Background Information

What are Abalone?

Abalone are marine snails. Abalone shells have a low and open spiral structure, and are characterized by several respiratory holes in a row near the shell's outer edge. The innermost layer of the shell is composed of nacre or mother-of-pearl, which in many species is highly iridescent, giving rise to a range of strong and changeable colors which make them attractive to humans as decorative objects [1].

The abalone of the Northwest's Puget Sound are a delicacy in Asia, prized for their meat and beautiful shell. As a result they were poached nearly to extinction in the early 1990s, but with a little help from scientists, the wild abalone is slowly recovering [2].

Why Model Rings

Rings in abalone correspond to their age, similart to rings in trees. The age of abalone would be of interest to marine biologists as well as abalone farmers for a variety of reasons. Although other methods exist [3], the method described on the data repository page for determining the age of abalone involves cutting the shell through the cone, staining it, and counting the number of rings through a microscope which is a destructive and time-consuming task [4].

In order to make the task of predicting age faster and easier, other measurements, which are easier to obtain, were investigated to predict the age [4].

Data Preparation

Reading the Data

The data was first obtained from the UCI repository[4] and then read into R by running:

```r
abalone <- read.table("abalone.data", sep=',')
```

When running this command, one must make sure that the working directory is set to the same directory where the data file resides.

Converting Catagorical Data

The first column of the original data [4] was labeled **Sex** and was populated with the categorical values **M**, **F**, and **I** corresponding to males, females and infants respectively. In order to facilitate inclusion of the **Sex** variable into the model, these categorical values were converted to numerical values. This was done by replacing the original column with two new columns. The first new column was labeled **M** for male and the second was labeled **F** for female. Males were deignated by putting a 1 in the **M** column and a 0 in the **F** column. Females were designated by putting a 0 in the **M** column and a 1 in the **F** column. Infants were designated by a 0 in both the **M** and **F** columns.

The following code was used to convert the **Sex** (V1) column:

```r
convertSex <- function(data.df = abalone) {
    library(dplyr)
    male <- (data.df[, 1] == 'M') * 1
    female <- (data.df[, 1] == 'F') * 1  # if I, left as 0
    # leave out the first row and prepend the two new col's
    converted <- cbind(male, female, data.df[, 2:length(data.df)])
    return(converted)
}
```

Data Partitioning

As is standard practice, the model was constructed using randomly selected portions of the dataset. The portion used to create the model is referred to as the *training set* while the remaining portion which is used to test and validate predictions is referred to as the *testing set*.

During the analysis, many iterations based on different randomly selected training and test sets had to be constructed. The following two R functions were frequently called to accomplish this task:

```r
# Returns a randomly selected subset of rows from the original input dataframe.
#
# orig - original (complete) dataset (dataframe), assumes that samples
#        are stored in as rows
# fraction - fraction of the original dataset used for training
# randomSeed - integer used to set the seed for the random number generator
#              (used to provide reproducibility)
trainSetRows <- function(orig, fraction, randomSeed) {
  set.seed(randomSeed)
  randorder <- sample(nrow(orig))  # create set of randomly selected rows
  nTrain <- round(nrow(orig)*fraction) # calc # of rows in training set
  trainRows <- randorder[1:nTrain]  # 1st nTrain rows in rand order
  return(trainRows)
}

# Returns a vector of integers corresponding to row numbers of the test set
# which are assumed to be all the rows NOT in the training  set.
#
# allRows - a vector containing all the row numbers in the data table
# trainRows - the vector of training rows returned from the trainSetRows function
testSetRows <- function(allRows, trainRows) {
  testRows <- setdiff(allRows, trainRows) # take away train rows leaves test rows
  return(testRows)
}
```

Regressions Analysis

The Linear Model

Definition of the Linear Model

The model used to describe the data is as follows:

$$(1)\quad R = w_0 + \sum_{i=1}^{9} w_i x_i$$

In equation (1), R is the number of rings the abalone has, the w's are the weights (our fitted parameters) and the x_i's are the independent variables. The independent variables can be described as: x_1 = M, x_2 = F, x_3 = Length, x_4 = Diameter, x_5 = Height, x_6 = Whole weight, x_7 = Shucked weight, x_8 = Viscera weight, and x_9= Shell weight.

λ Parameterization

The goal was to fit the w vector described in (1), in such a way as to minimize the residual sum of squares (RSS) parameterized by the ridge regression factor λ which penalizes large coefficients. Since minimzing a constant times RSS is the same as minimizing RSS, the error which we want to minimize can be written in matrix form as:

$$(2)\quad \frac{1}{2}RSS = \frac{1}{2}\sum_{i=1}^{9}\left(t_n - \mathbf{w}^T\mathbf{x}_i\right)^2 + \frac{1}{2}\lambda\mathbf{w}^T\mathbf{w}$$

where t_n are know target (R) values and the constant $\frac{1}{2}$ has been multiplied by both sides to make the result of differentiation cleaner. Taking the gradient of (2), setting it equal to zero, and solving for the coefficients vector $\mathbf{w}$, yields the following solution in matrix notation [5]:

$$(3)\quad \mathbf{w} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{T}$$

The size of the independent variables matrix $\mathbf{X}$ is $(n \times d+1)$ where n is the number of samples and $d+1$ is the number of independent variables in the $\mathbf{w}$ vector which includes the w_0 bias term. The dependent variables matrix of targets $\mathbf{T}$ is a column vector of size $(n \times 1)$. The diagonal matrix $\lambda\mathbf{I}$ is size $(d+1 \times d+1)$.

Calculating weights

In order to use the `solve()` function in R, we first need to rewrite (3) in the following form:

$$(4)\quad \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w} = \mathbf{X}^T\mathbf{T}$$

With equation (4), we now have an expression that we can pass to `solve()` to calculate the weight vector $\mathbf{w}$ given parameter λ. The function `llsMakeST` was written to accomplish this task with care taken to ensure that the w_0 bias term was not penalized by zeroing out the first term in the $\lambda\mathbf{I}$ matrix as shown below.

```
## llsMakeST (Linear Least Squares Make for a Singe Target) Returns a vector of
## the linear least square regression weights for a single target variable with
## the Ridge Regression penalty factored in.
##
## For n = number of samples and d + 1 = number of coefficients including bias:
```

```r
## X - n x d+1 coefficient matrix wrt the independent variables
## Y - n x 1 vector of values for the dependent target variable
## lambda (optional) - ridge regression (square weight) penalty factor
##                      (0 by default, regular unpenalized regression)
##
## IMPORTANT USAGE NOTES:
## (1) Independent var's should NOT be standardized because it's done internally.
## (2) Samples w/ missing data need to be removed before running this function.
## (3) If lambda is not passed, straight (unpenalized) linear regression is done.
llsMakeST <- function(X, Y, lambda=0) {
   # if X or Y come in as a list, convert it to numeric
   X <- as.matrix(apply(X, 2, as.numeric))
   # standardize the independent variables
   means <- matrix(apply(X, 2, mean), ncol(X), 1)
   stdevs <- matrix(apply(X, 2, sd), ncol(X), 1)
   N <- nrow(X)  # number of samples
   D <- ncol(X)  # number of parameters (dimensions)
   Xs <- (X - matrix(rep(means, N), N, D, byrow=TRUE))/
          matrix(rep(stdevs, N), N, D, byrow=TRUE)

   Y <- as.matrix(apply(Y, 2, as.numeric))
   # account for the bias
   Xs1 <- cbind(1, Xs)
   colnames(Xs1)[1] <- "bias"
   # now solve for the weights, be sure not to penalize w0
   w <- solve((t(Xs1) %*% Xs1)+(lam*diag(c(0,rep(1,ncol(Xs1)-1)))),
               t(Xs1) %*% Y)
   # store means & std devs along with the weights so we can properly
   # standardize test values when we use our weights for predictions
   meansX <- matrix(apply(cbind(1,X),2,mean),ncol(X)+1,1)
   stdevsX <- matrix(apply(cbind(1,X),2,sd),ncol(X)+1,1)
   w <- cbind(w, meansX)
   w <- cbind(w, stdevsX)
   colnames(w) <- c("weight", "mean", "std dev")

   return(w)
}
```

The `llsMakeST` function returns a three column vector. In the first column are the weights as defined by $\mathbf{w}$. The second and third columns of this matrix contain the means and standard deviations of the independent variable that were used in training the model. By training, we mean solving for $\mathbf{w}$.

The second and third columns of the $\mathbf{w}$ matrix are used to standardize the independent variables. Standardizing takes the following form:

$$(5)\ \ \mathbf{X}_s = \frac{(\mathbf{X} - \mathbf{X}_{mean})}{\mathbf{X}_{stdev}}$$

In equation (5), $\mathbf{X}_s$ is the matrix of standardized inputs (independent variables) and the matrices $\mathbf{X}_{mean}$ and $\mathbf{X}_{stdev}$ are the means and standard deviations of the inputs. Standardization provides two advantages: (i) it allows us the ability to better account for significant difference between the regions of training and testing and (ii), it allows us to more easily compare the relative magnitude of the weights.

Making Predictions

To make predictions from the `llsMakeST` weights, the `llsUseST`function was created:

```
## This function returns a matrix of the predicted target values
## based on the weights determined in llsMakeST.
llsUseST <- function(weights, X) {
   X <- as.matrix(apply(X,2,as.numeric))
   # standardize X
   N <- nrow(X)  # number of samples
   D <- ncol(X)  # number of parameters of dimensions
   means <- weights[2:nrow(weights),2]
   stdevs <- weights[2:nrow(weights),3]
   # standardize the inputs
   Xs <- (X - matrix(rep(means, N), N, D, byrow=TRUE))/
          matrix(rep(stdevs, N), N, D, byrow=TRUE)

   Xs1 <- cbind(1, Xs)  # account for the bias
   colnames(Xs1)[1] <- "bias"
   weights[1, 3] <- 1  # replace bias stdev to avoid a division by 0
   model <- Xs1 %*% weights[,1]  # calc the prediction of a single target
   return(model)
}
```

The `llsUseST' function outputs the vector`model‘ which contains the model predictions for the independent variables **X** that were passed in. These predictions could then be used to evaluate RMSE which in turn allowed us to explore how λ and training partition impacted the models accuracy. This will be explored in detail in the last part of the **Experiments** which follows.

Experiments

A Quick Look at the Data

Before diving into the regression analysis, it's normally a good idea to take a look at the relationship between the variables - particularly the independent variables. To accomplish this, the following R code utilizing the *pairs* function was used:

```
panel.hist <- function(x) {
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.5))
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <-y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col="cyan")
    dev.copy2eps(file="scatter2.eps")
}

panel.cor <- function(x,y,digits=4, prefix="", cex.cor) {
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(0,1,0,1))
    r <- abs(cor(x,y))
    txt <- format(c(r, 0.123456789), digits=digits)[1]
    txt <- paste(prefix, txt, sep="")
    if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex=cex*r)
}

diag_hist <- function(y) {
    pairs(y, diag.panel=panel.hist, upper.panel=panel.cor)
}
```

This code produced the chart shown in Figure 1.

Results

A Personal Curiosity: Bias and Training Partition

Discussion

References

[1] Wikipedia - https://en.wikipedia.org/wiki/Abalone
[2] Living On Earth - http://loe.org/shows/segments.html?programID=15-P13-00031&segmentID=3
[3] Stable oxygen method - http://www.researchgate.net/profile/Craig_Mundy/publication/201169638_Determining_age_and_growth_of_abalone_using_stable_oxygen_isotopes_a_tool_for_fisheries_management
[4] UCI data repository (data source) - https://archive.ics.uci.edu/ml/datasets/Abalone
[5] [The Elements of Statistical Learning - 2nd Edition, page 64 http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
[9] Rings not annual - http://www.publish.csiro.au/?paper=MF9921215