Principles of Statistical Mining 03/10/2010

Lecture Notes [Intro Regression]: Introduction to Regression with R

Professor: Ernest Fokoué **Note**: This is just a draft

1 Simple Linear Regression Analysis

Consider the data set `golf2008.csv` encountered earlier. One might be interested in finding out if Earnings can be predicted based on information measuring the technical performance of the golfer on the field. Is it possible to build a model such that through that model, a change of one unit in a characteristic leads to an average constant change in Earnings? For instance, how much of the Earnings of a golfer can be explained by his average score through a linear relationship? Does a nonzero number β_1 exist such that

$$\texttt{Earnings} = \beta_0 + \beta_1 \texttt{Avg.Score} + \varepsilon \tag{1}$$

Equation (1) is just an application of the more general model of Equation (2) better known as the simple linear regression model.

$$Y = \beta_0 + \beta_1 \mathrm{x} + \varepsilon \tag{2}$$

Before we get to details in the analysis of the above model, let's look at the variables that might have such a relationship with Earnings. Since correlation measures the strength of the linear relationship between two variables, we present below the sample correlation coefficient values between Earnings and all the remaining variables in the Golf 2008 data set.

```
 > cor(golf[,-22], golf[,22])
                          [,1]
Age                   -0.09341
Yards.Drive            0.16436
Driving.Acc           -0.17701
Drive.Total            0.18699
Greens.Reg             0.38437
Putting.Avg           -0.30435
Save.Pct               0.21435
Scrambling.Pct         0.19935
Putts.Round           -0.19229
Birdie.Conversions     0.31492
```

```
Par.5.Birdies          0.48170
Par.4.Birdies          0.32660
Par.3.Birdies         -0.09289
Avg.Score             -0.80776
Avg.Score.Before.Cut  -0.46674
Avg.Finish            -0.81112
Events                -0.10868
Rounds                 0.17337
Cuts.Made              0.49080
Top.10                 0.85295
FedEx.Points           0.61314
```

The above sample correlation coefficient values clearly suggest that `Earnings` will likely have a meaningful linear relationship with the following characteristics: `Avg.Score, Avg.Finish, Top.10, FedEx.Points`. Of the remaining variables `Avg.Score.Before.Cut, Cuts.Made, Par.5.Birdies` may also be individually good at explaining a good proportion of the variation in `Earnings`. Let's consider building the model that relates `Earnings` to `Avg.Score` linearly. This activity is deemed consequential because we found that

$$\texttt{cor}(\texttt{Avg.}\widehat{\texttt{Score}}, \texttt{Earnings}) = -0.81112$$

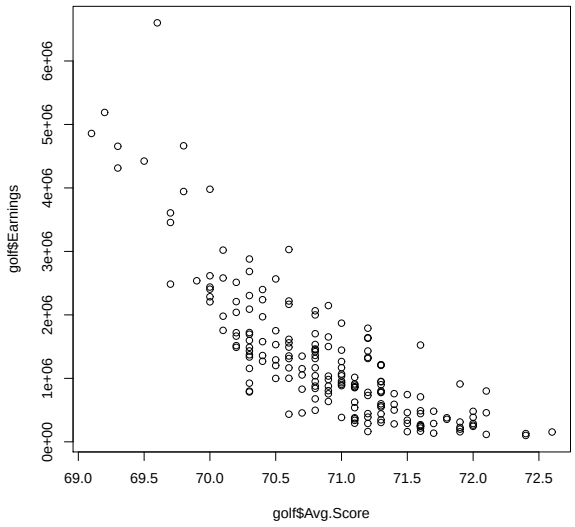
which is indicates that there is a moderately strong linear relationship between , `Earnings` and `Avg.Score`. The hat is used because the value obtained is the sample correlation coefficient r, defined by

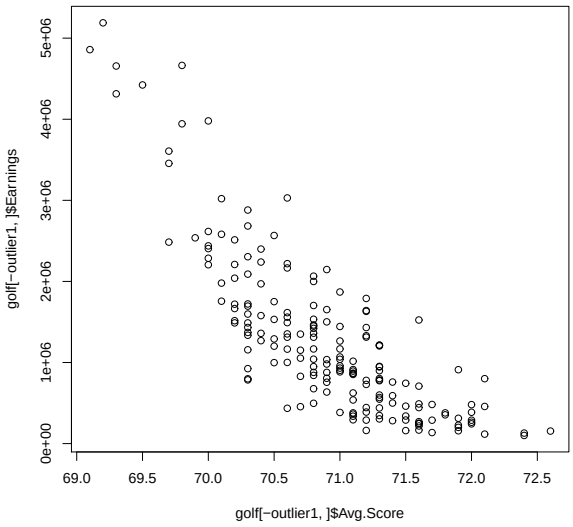
$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{\mathrm{x}_i - \bar{\mathbf{x}}}{\mathtt{s}_\mathrm{x}}\right)\left(\frac{\mathrm{y}_i - \bar{\mathbf{y}}}{\mathtt{s}_\mathrm{y}}\right)$$

which is a point estimate of the population correlation coefficient

$$\rho = \frac{\texttt{cov}(X,Y)}{\sigma_\mathrm{x}\sigma_\mathrm{y}} = \frac{\mathbb{E}[(X-\mu_\mathrm{x})(Y-\mu_\mathrm{y})]}{\sigma_\mathrm{x}\sigma_\mathrm{y}}.$$

whose main property is $-1 \leq \rho \leq +1$, also expressed as $|\rho| \leq 1$. Therefore a sample correlation coefficient r close to -1 (negative relationship i.e. downward trend) or to $+1$ (positive relationship i.e upward trend) is an indication that there is a strong linear component in the relationship between X and Y.
Let's visualize that relationship on a scatter plot.
The scatterplot clearly shows a downward trend (negative relationship) that is moderately strong. Although the linear relationship is clear, there might be a need for more than just the linear term, judging from the shape of the plot at the bottom end. Besides - and importantly so - there is a clear outlier on the plot was influence might compromise the validity and even correctness of the model built: this is because the correlation coefficient is influenced by outliers. With the outlier removed, (see right plot), the scatterplot looks better. To fit a linear model in `R`, we simply use the command `lm()`.






```
x.avg.score <- golf[-outlier1,]$Avg.Score  # Define x
y.earnings  <- golf[-outlier1,]$Earnings   # Define y
earn.score <- lm(y.earnings~x.avg.score)   # Fit the model
lm.earn.score <- summary(earn.score)       # Extract the summary
print(lm.earn.score)                       # Print the summary
par(mfrow=c(2,2))                          # Prepare to plot residuals
plot(earn.score)                           # Plot residuals
```

The details of the fitted model are given below, and we need to learn to interpret this type of output.

```
    Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept) 90922196    4632501    19.6   <2e-16 ***
x.avg.score -1264346      65339   -19.4   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
Residual standard error: 581000 on 184 degrees of freedom
Multiple R-squared: 0.671,      Adjusted R-squared: 0.669
F-statistic:  374 on 1 and 184 DF,  p-value: <2e-16
```

The above output has two important items of information about: (a) The measure of the overall importance of the model (b) the significance of each coefficient.

- R^2, also known as coefficient of determination, measures the proportion of variation in Earnings explained by the predictor variables through the linear model. $0 \leq R^2 \leq 1$. Therefore, a value

R^2 close to 1 is typically a "good" thing. In this case, $R^2 = 0.671$, which means that 67% of the variation in `Earnings` is captured by the knowledge of `Avg.Score`. That's decent, but one may require higher is `x` is the only variable one has.

- The second crucial item in the output, are the `P-value` associated with each coefficient. Basically, under the column named `Pr(>|t|)` are values that correspond to half the `P-value` delivered by the test on the corresponding coefficient. The most important one correspond to the row that has the explanatory variable, in this case `x.avg.score`. Formally, it is the `P-value` for the test

$$\begin{cases} \mathtt{H_0}: \ \beta_1 = 0 & \texttt{slope is zero, therefore NO linear relationship} \\ \mathtt{H_a}: \ \beta_1 \neq 0 & \texttt{slope is nonzero, therefore there IS linear relationship} \end{cases}$$

  In this case, $\mathtt{P-value} = 2 \times 0 = 0$. With a `P-value` equal to 0 we reject the null hypothesis, and conclude that the slope is nonzero, and therefore there is a meaningful linear relationship between `Earnings` and `Avg.Score`. Typically we need this `P-value` to be as small as it can get (close to zero) for the variable under consideration to be a significant (relevant) predictor of the response.

  The Mighty `R` makes the decision about the significance of a predictor easy by its system of stars. (***) stands for very strong significance, while no star means irrelevant (non significant) variable.

  We also have a test on the intercept, namely

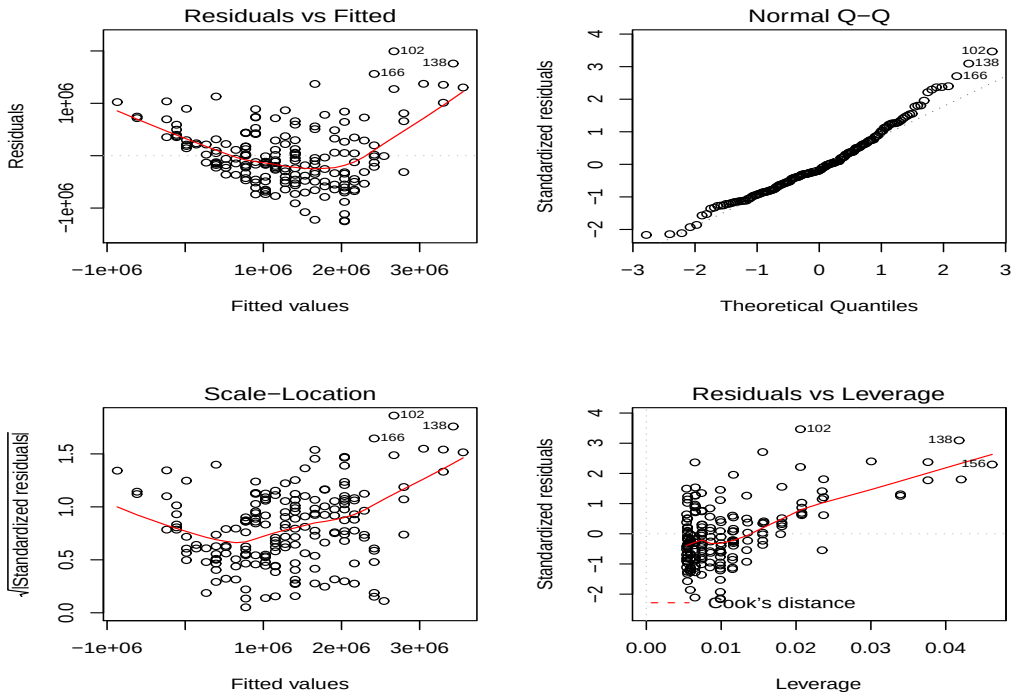
$$\begin{cases} \mathtt{H_0}: \ \beta_0 = 0 & \texttt{intercept is zero, therefore NO need for intercept} \\ \mathtt{H_a}: \ \beta_0 \neq 0 & \texttt{intercept is nonzero, intercept is needed in model} \end{cases}$$

  For this particular example, the corresponding `P-value` $= 2 \times 0 = 0$. Therefore, we conclude that the intercept is statistically significantly different from zero and therefore needed in the model.

- The other items in the output are such things as the `F`-statistic and the overall `P-value` of the regression model. The Adjusted R^2 is another quantity that will be touched on when we study multiple linear regression.

`Residual Analysis`

To summarize, we have a linear model relating `Earnings` to `Avg.Score`. However, we cannot conclude until we have checked the residuals to make sure that the assumptions underlying the linear model are met. The four plots below provide the raw material for this crucial residual analysis.
If the fitted linear model is an adequate representation of the relationship between the response variable and the explanatory variable, then the residual vs fitted values plot (top left) must be a random scatter with no pattern whatsoever left in it. In this particular case unfortunately, the linear relationship did not capture all the structure in the data, because the residuals still shows what looks like a quadratic term. There are also seem to be some outliers clearly marked out by `R`. The only satisfactory thing so far is the fact the residuals seem to have mean 0. The top right plot checks the normality of the residuals. In



this case, the residuals fail normality miserably, because of all the outliers present in it. In fact, checking normality is meaningless if the residual versus fitted values is not conclusive. We seem to need: (a) address the remaining quadratic term by adding it to the model (b) understand the present outliers and remove them if need be to build a better model[1].

$$\texttt{Earnings} = \beta_0 + \beta_1 \texttt{Avg.Score} + \beta_2 \texttt{Avg.Score}^2 + \varepsilon \tag{3}$$

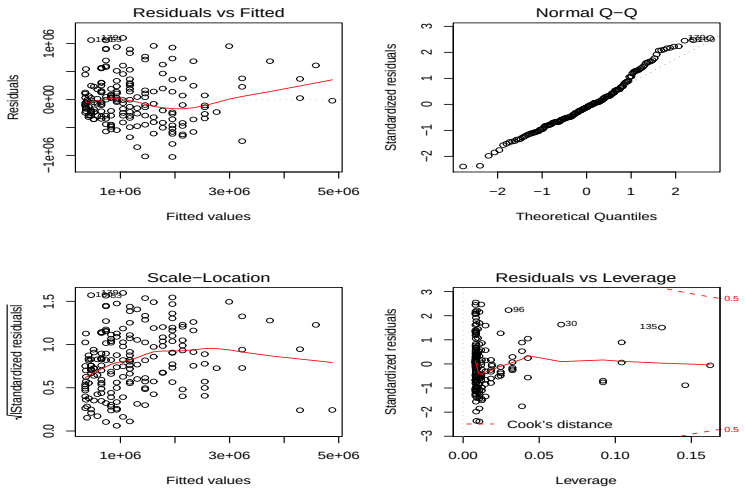
```
outlier2 <- c(outlier1, 102, 151, 166, 17, 50)
x.avg.score <- golf[-outlier2,]$Avg.Score
y.earnings  <- golf[-outlier2,]$Earnings
earn.score <- lm(y.earnings~x.avg.score+I(x.avg.score^2))
lm.earn.score <- summary(earn.score)
print(lm.earn.score)

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       2.69e+09   2.71e+08    9.93   <2e-16 ***
x.avg.score      -7.47e+07   7.65e+06   -9.76   <2e-16 ***
I(x.avg.score^2)  5.18e+05   5.40e+04    9.60   <2e-16 ***
```

[1]It could well be that the thirteen riches golfers in the data are making it difficult for a single regression line to capture the variation in Earnings. It could also be that transforming Earnings to log scale might help lessen that influence.

```
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 433000 on 178 degrees of freedom
Multiple R-squared: 0.799,      Adjusted R-squared: 0.797
F-statistic:  354 on 2 and 178 DF,  p-value: <2e-16
```

Not only is the quadratic term statistically significant, but it has also led to 13% extra variation explained as we now have 80% explained as opposed to 67% earlier. However, the residual versus fitted values is still not satisfactory, namely because the noise variance is not constant as it should be. Besides, the residuals fail the normality test.

```
par(mfrow=c(2,2))
plot(earn.score)
```



Clearly, `Avg.Score` is a good predictor of `Earnings`, but other variables (characteristics of golfers) are needed here as the model based on `Avg.Score` alone cannot seem to capture all the structure. The other variables that were important as measured by their sample correlation coefficients are also turned out to be significant predictor of `Earnings` when used in a simple linear regression model.

1.1 Extracting Information from models built in R

Once a model is fitted, it is always of interest to extract the information it calculated. The command `summary()` does precisely that. For instance, we fit a linear model and store it in `earn.score`, then we display the summary of the fitted model.

```
earn.score <- lm(y.earnings~x.avg.score+I(x.avg.score^2))
summary(earn.score)
```

Notice that earlier, we stored the summary itself, and then printed it. The result is the same. However, sometimes, we may want to store a particular piece of the summary. Then the stored summary becomes interesting. By displaying the summary of the summary of an `lm()` model object in R, we get to know what information can be extracted. For instance,

```
summary(lm.earn.score)
```

gives the following output

```
              Length Class  Mode
call            2    -none- call
terms           3    terms  call
residuals     186    -none- numeric
coefficients   12    -none- numeric
aliased         3    -none- logical
sigma           1    -none- numeric
df              3    -none- numeric
r.squared       1    -none- numeric
adj.r.squared   1    -none- numeric
fstatistic      3    -none- numeric
cov.unscaled    9    -none- numeric
```

With that, I can now store pieces of the summary for later use and comparisons. For instance

```
Rsquared.score <- lm.earn.score$r.squared
sigma.score    <- lm.earn.score$sigma
Fstat.score    <- lm.earn.score$fstatistic
```

However, it must be noted that the itself direct delivers vital information pertaining to the model, namely,

```
coefficients(earn.score)      # Estimates of coefficients beta_hat_i, i=1,..,p
fitted.values(earn.score)     # Fitted values y_hat_i =f(x_i) for i=1,..,n
fitted(earn.score)            # Same as fitted.values(earn.score)
residuals(earn.score)         # Residuals y_i-y_hat_i for i=1,..,n
```

`Note:` More detailed information about what the method delivers can be found using `help(lm)`. For the linear model, the following important pieces of information can be obtained

```
confint(earn.score)          # CIs for coefficients beta_hat_i, i=1,..,p
logLik(earn.score)           # log likelihood of the model
AIC(earn.score)              # Akaike Information Criterion measure
deviance(earn.score)         # Model deviance
```

`Summaries are technique-dependent:` It is important to note that the summary of summary differs from method to method. For instance, Support Vector Machines optimize a different objective function, and will therefore provide a different summarization.

1.2 Prediction with models built in R

`In sample predictions:` The command `fitted(earn.score)` produces the vector $(\hat{y}_1, \cdots, \hat{y}_n)$ where each $\hat{y}_i = \hat{f}(x_i)$, with $\hat{f}$ representing the model just built, and the x_i are the values used to build it. In other words, predictions (or better yet estimations) are made for observations within the present sample.

`Out of sample predictions:` How does one obtain $\hat{y}_{\texttt{new}} = \hat{f}(x_{\texttt{new}})$ where $x_{\texttt{new}}$ is new and unseen during model fitting? The answer to this question help assess `generalization`.

The generic R command `predict()` delivers elements that help answer the above question. For instance, for the ubiquitous linear model command `lm()`, the command `predict()` is

```
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, type = c("response", "terms"),
        terms = NULL, na.action = na.pass,
        pred.var = res.var/weights, weights = 1, ...)
```

For instance, using `predict()` without specifying new data delivers the same thing we obtained with `fitted()`. Notice however, that even if you only deal with the in sample data, `predict()` command provides extra information of `great statistical importance`, namely: (a) `standard error estimates` by way of the `se.fit = TRUE` switch; and (b) computation of $100(1-\alpha)\%$ confidence bands and prediction bands, by way of the `interval` switch plus the `level` switch. Indeed, when we fit the above univariate linear regression, each fitted value represents

$$\widehat{\mu_{Y_i|x_i}} = \widehat{\mathbb{E}[Y_i|x_i]}$$

which is really an estimate of the average response $\mathbb{E}[Y_i|x_i]$. Clearly, $\mathbb{E}[Y_i|x_i]$ is the population "parameter" (as it were), and our little $\widehat{\mathbb{E}[Y_i|x_i]} = \hat{y}_i$ is the corresponding statistic. Statistically, $\widehat{\mathbb{E}[Y_i|x_i]} = \hat{y}_i$ is just a point estimate, and really need to make inference about the true yet unknown population characteristic, which is the average response $\mathbb{E}[Y_i|x_i]$. We therefore need, the sampling distribution of $\widehat{\mathbb{E}[Y_i|x_i]} = \hat{y}_i$. It turns out that for an arbitrary x^* in the domain of x, the random - sampling dependent - variable

$$T = \frac{\widehat{\mathbb{E}[Y|x^*]} - \mathbb{E}[Y|x^*]}{s\sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}}}$$

follows a Student's t-distribution with $\texttt{df} = n - 2$ degrees of freedom, where

$$s = \hat{\sigma} = \sqrt{\frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

is an estimate of the noise standard deviation, and the estimated standard error is given by

$$\texttt{ese}(\widehat{\mathbb{E}[Y|\mathrm{x}^*]}) = \mathtt{s}\sqrt{\frac{1}{n} + \frac{(\mathrm{x}^* - \bar{\mathbf{x}})^2}{\sum_{i=1}^n (\mathrm{x}_i - \bar{\mathbf{x}})^2}}.$$

With all that, a $100(1-\alpha)\%$ `confidence interval` for $\mathbb{E}[Y|\mathrm{x}^*]$ is given by

$$\left[\widehat{\mathbb{E}[Y|\mathrm{x}^*]} - t_{\mathtt{df},\alpha/2} \times \texttt{ese}(\widehat{\mathbb{E}[Y|\mathrm{x}^*]}), \widehat{\mathbb{E}[Y|\mathrm{x}^*]} + t_{\mathtt{df},\alpha/2} \times \texttt{ese}(\widehat{\mathbb{E}[Y|\mathrm{x}^*]})\right]$$

Prediction interval and prediction bands: Now, if instead of estimating the average response corresponding to x^* one wants to predict the actual value of the response for that arbitrary x^*, then the problem gets harder: the extra difficulty comes from the fact that the response itself is random by virtue of the model definition. Therefore prediction error has in it both (a) estimation error (b) random sampling error.

$$\texttt{pese}(\widehat{Y|\mathrm{x}^*}) = \mathtt{s}\sqrt{1 + \frac{1}{n} + \frac{(\mathrm{x}^* - \bar{\mathrm{x}})^2}{\sum_{i=1}^n (\mathrm{x}_i - \bar{\mathbf{x}})^2}}$$

With all that, a $100(1-\alpha)\%$ `prediction interval` for $Y|\mathrm{x}^*$ is given by

$$\left[\widehat{\mathbb{E}[Y|\mathrm{x}^*]} - t_{\mathtt{df},\alpha/2} \times \texttt{pese}(\widehat{\mathbb{E}[Y|\mathrm{x}^*]}), \widehat{\mathbb{E}[Y|\mathrm{x}^*]} + t_{\mathtt{df},\alpha/2} \times \texttt{pese}(\widehat{\mathbb{E}[Y|\mathrm{x}^*]})\right]$$

Note: Confidence is about a fixed average $\mathbb{E}[Y|\mathrm{x}^*]$, whereas prediction is about a random variable $Y|\mathrm{x}^*$

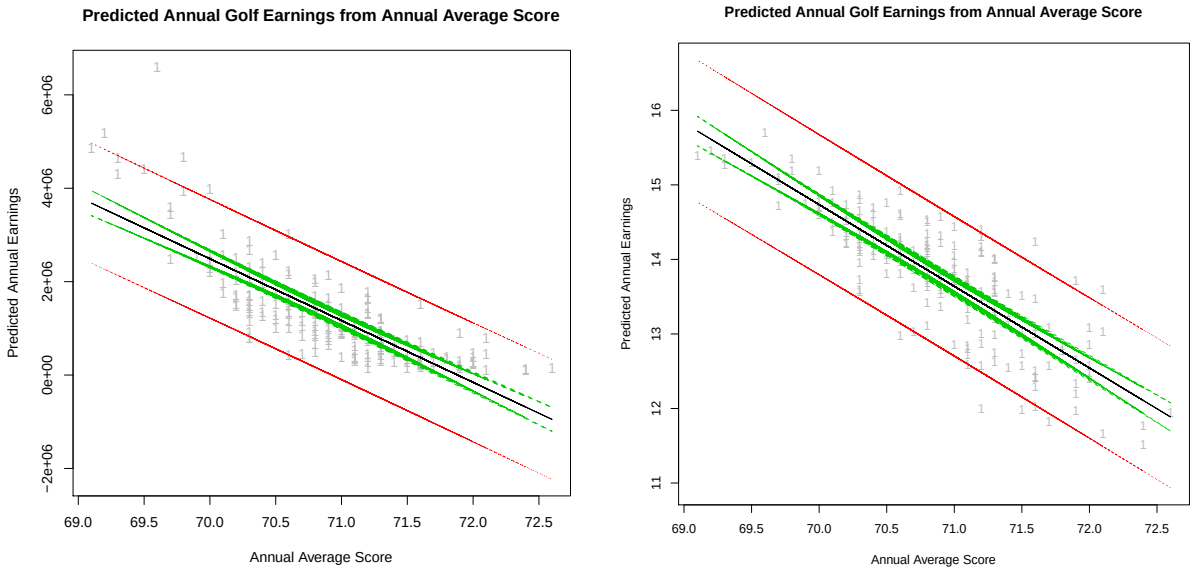


图 1: *Regression of Earnings on Average Score with both confidence bands (green) and prediction bands (red) on the training data. (left) Response on the original scale (right) Response on the log scale*

```
# Predicting Annual Golf Earnings using Average Score with Bands
 x  <- golf$Avg.Score          #  Defining predictor variable
```

```
y  <- golf$Earnings           #  Defining response variable
xy <- data.frame(x,y)         #  Prepare data frame
rownames(xy)<-1:length(x)     #  Put in the row names
lm.xy <- lm(y~x, data=xy)     #  Fit the linear model

pred.band <- predict(lm.xy, data.frame(xy$x), interval="prediction")
conf.band <- predict(lm.xy, data.frame(xy$x), interval="confidence")

matplot(xy$x,cbind(y, conf.band, pred.band[,-1]), col=c(8,1,3,3,2,2),
    lty=c(8,1,2,2,3,3), type=c("p","l","l","l","l","l"),
    xlab = 'Annual Average Score', ylab="Predicted Annual Earnings",
    main= "Predicted Annual Golf Earnings from Annual Average Score")
```

1.3 Assessing generalization with models built in R

The confidence and prediction bands were all created for all the observations within the sample that was used to build the model. In Machine Learning and Data Mining a single sample estimate is never conclusive, and clearly should not be: Regression models are often built with an intention of using them to predict outcomes (responses) for cases that were not encountered during model building.

⊡ The data used for fitting the model is called the training set

⊡ The data used for testing the generalization ability of the model is called the test set.

A wee bit of Statistical Learning Theory: Assume that the x_i in your study are sampled from $\mathcal{X}$ and the responses y_i come from $\mathcal{Y}$. You consider using a function, say f to capture the dependencies between X and Y so that the prediction of Y from the knowledge of X alone is *as good as it can get*. You first define how to measure good prediction (versus bad prediction that is). That corresponds to defining (choosing or specifying) a loss function $\ell(Y, f(X))$. For instance, if X and Y are numbers, then you are doing regression, and a candidate loss function is the so-called squared error loss

$$\ell(Y, f(X)) = (Y - f(X))^2$$

Since X and Y are random, `as good as it gets` translates into minimizing the expected (average) loss, also known as the risk functional and defined as

$$R(f) = \mathbb{E}[\ell(Y, f(X))] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(\mathrm{y}, f(\mathrm{x}))p(\mathrm{x})p(\mathrm{y}|\mathrm{x})d\mathrm{x}d\mathrm{y}$$

For the above squared error loss, that corresponds to the Mean (Integrated) Squared Error (MSE)

$$\texttt{MSE}(f) = \int_{\mathcal{X}\times\mathcal{Y}} (\mathrm{y} - f(\mathrm{x}))^2 p(\mathrm{x})p(\mathrm{y}|\mathrm{x})d\mathrm{x}d\mathrm{y}$$

The Fact that the densities $p(\mathrm{x})$ and $p(\mathrm{y}|\mathrm{x})$ are unknown in practice leads to the need to use an empirical counterpart of $\texttt{MSE}(f)$ based on collected samples, namely

$$\widehat{\texttt{MSE}}(f) = \frac{1}{n}\sum_{i=1}^{n} (\mathrm{y}_i - f(\mathrm{x}_i))^2$$

where the pairs $(\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \cdots, (\mathrm{x}_n, \mathrm{y}_n)$ are assumed to be sampled identically and independently from the joint distribution with density $p(\mathrm{x}, \mathrm{y}) = p(\mathrm{x})p(\mathrm{y}|\mathrm{x})$. Our goal is to find the function f that minimizes $\widehat{\texttt{MSE}}(f)$. This goal can be borderline unreachable if we seek to find the universally best function. However, in the presence of a collection of functions - starting with two functions for example - this gives us a criterion for deciding which function to prefer (choose). Let f_i and f_j be two functions. Then

$$\textit{if } \widehat{\texttt{MSE}}(f_j) < \widehat{\texttt{MSE}}(f_i) \quad \textit{then select } f_j \tag{4}$$

The trouble - limitation - with naively using criterion prescribed by (4) lies in the fact, given a sample $(\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \cdots, (\mathrm{x}_n, \mathrm{y}_n)$, the function $\hat{f}_{\texttt{memory}}$ defined by

$$\hat{f}_{\texttt{memory}}(\mathrm{x}_i) = \mathrm{y}_i, \qquad i = 1, \cdots, n$$

always achieves the best performance, since $\widehat{\texttt{MSE}}(\hat{f}_{\texttt{memory}}) = 0$, which is the minimum achievable.

Where does the limitation of $\hat{f}_{\texttt{memory}}$ come from? Well, $\hat{f}_{\texttt{memory}}$ does not really learn the dependency between X and Y. While it may have some of it, it also grabs a lot of the noise in the data, and ends overfitting the data. As a result of not really learning the structure of the relationship between X and Y and only merely memorizing the present sample values, $\hat{f}_{\texttt{memory}}$ **will predict very poorly when presented with observations that were not in the sample.**

Splitting the data into training set and test set: It therefore makes sense to judge models (functions), not on how they perform with in sample observations, but instead how they perform on **out of sample cases**. Given a collection $\mathcal{D} = (\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \cdots, (\mathrm{x}_n, \mathrm{y}_n)$ of pairs,

- ⊡ Randomly split $\mathcal{D}$ into training set of size `ntr` and test set of size `nte`, such that $\texttt{ntr} + \texttt{nte} = n$
  - ⊞ Training set $\texttt{Tr} = \left\{(\mathrm{x}_1^{(\texttt{tr})}, \mathrm{y}_1^{(\texttt{tr})}), (\mathrm{x}_2^{(\texttt{tr})}, \mathrm{y}_2^{(\texttt{tr})}), \cdots, (\mathrm{x}_{\texttt{ntr}}^{(\texttt{tr})}, \mathrm{y}_{\texttt{ntr}}^{(\texttt{tr})})\right\}$
  - ⊞ Training set $\texttt{Te} = \left\{(\mathrm{x}_1^{(\texttt{te})}, \mathrm{y}_1^{(\texttt{te})}), (\mathrm{x}_2^{(\texttt{te})}, \mathrm{y}_2^{(\texttt{te})}), \cdots, (\mathrm{x}_{\texttt{nte}}^{(\texttt{te})}, \mathrm{y}_{\texttt{nte}}^{(\texttt{te})})\right\}$
- ⊡ For each function class $\mathcal{F}$ (linear models, nonparametric smoothers, etc...) under consideration,
  - ⊞ Find the best in its class based on the training set `Tr`
  - ⊞ For all the estimated functions $\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_m$ constructed, compute the training error

$$\widehat{\texttt{MSE}}_{\texttt{Tr}}(\hat{f}_j) = \frac{1}{\texttt{ntr}} \sum_{i=1}^{\texttt{ntr}} (\mathrm{y}_i^{(\texttt{tr})} - \hat{f}_j(\mathrm{x}_i^{(\texttt{tr})}))^2$$

  - ⊞ For all the estimated functions $\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_m$ constructed, compute the test error

$$\widehat{\texttt{MSE}}_{\texttt{Te}}(\hat{f}_j) = \frac{1}{\texttt{nte}} \sum_{i=1}^{\texttt{nte}} (\mathrm{y}_i^{(\texttt{te})} - \hat{f}_j(\mathrm{x}_i^{(\texttt{te})}))^2$$

- ⊡ Compute the averages of both $\widehat{\texttt{MSE}}_{\texttt{Tr}}$ and $\widehat{\texttt{MSE}}_{\texttt{Te}}$ over many random splits of the data, and tabulate (if necessary) those averages.
- ⊡ Select $\hat{f}_{j^*}$ such that

$$\texttt{mean}[\widehat{\texttt{MSE}}_{\texttt{Te}}(\hat{f}_{j^*})] < \texttt{mean}[\widehat{\texttt{MSE}}_{\texttt{Te}}(\hat{f}_j)], \qquad j = 1, 2, \cdots, m, \quad j \neq j^*$$

Best of the bunch: When $\hat{f}_{j^*}$ as defined above exists, it is said to have the best generalization ability of the bunch $\hat{f}_1, \cdots, \hat{f}_m$.

1.3.1 Example: Best approximator of a simple non linear function

Consider the following simple nonlinear univariate function

$$f(\mathrm{x}) = -\mathrm{x} + \sqrt{2}\sin(\pi^{3/2}\mathrm{x}^2), \qquad \mathrm{x} \in [-1, +1]. \tag{5}$$

The goal is to find the best generalizing approximator of f from a sample $\mathcal{D} = \{(\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \cdots, (\mathrm{x}_n, \mathrm{y}_n)\}$ of pairs drawn from its population. You are given the collection

$$\mathcal{F} = \{\texttt{Polynomial Regression}, \texttt{SVM}, \texttt{RVM}, \texttt{Gaussian Processes}\}$$

as your family (class) of approximating schemes. Write an `R` script that tabulates the average training and test errors and selects the best.

1.3.2 Implementation in `R` of the split for the example

We implement this framework of splitting the data in training set and test set as follows.

```
f <- function(x){-x + sqrt(2)*sin(pi^(3/2)*x^2)}
a    <- -1;   b    <- +1
sig  <- 0.3;  n   <- 300
x   <- seq(a,b,length=n)
y   <-  f(x)+ rnorm(n, 0, sig)
XY  <- data.frame(x,y)
n   <- nrow(XY);  p   <- ncol(XY)
colnames(XY)[p]<- 'Y'
m   <- 10
mse.tr <- mse.te <- matrix(0, ncol=4, nrow=m)
for(r in 1:m)
{
   s <- sample(sample(n))
   ntr <- round(2*n/3)
   nte <- n - ntr
   id.tr <- s[1:ntr]
   id.te <- (1:n)[-id.tr]

   XY.tr <- XY[id.tr,]
   XY.te <- XY[id.te,]

   #  Polynomial regression
   mse.tr[r,1] <-  mean((fitted(lm.mod)-XY.tr$Y)^2)
```

```
    mse.te[r,1] <-  mean((predict(lm.mod,xxnew)-XY.te$Y)^2)

    svm.mod <- ksvm(Y~., data=XY.tr, kernel='laplacedot')
    mse.tr[r,2]   <- mean((fitted(svm.mod)-XY.tr$Y)^2)
    mse.te[r,2]   <- mean((predict(svm.mod,data.frame(x=XY.te[,-p]))-XY.te$Y)^2)

    rvm.mod <- rvm(Y~., data=XY.tr, kernel='laplacedot')
    mse.tr[r,3]   <- mean((fitted(rvm.mod)-XY.tr$Y)^2)
    mse.te[r,3]   <- mean((predict(rvm.mod,data.frame(x=XY.te[,-p]))-XY.te$Y)^2)

    gpr.mod <- gausspr(Y~., data=XY.tr, kernel='laplacedot')
    mse.tr[r,4]   <- mean((fitted(gpr.mod)-XY.tr$Y)^2)
    mse.te[r,4]   <- mean((predict(gpr.mod,data.frame(x=XY.te[,-p]))-XY.te$Y)^2)
}
```

The commands `ksvm` for Suppport Vector Machine, `RVM` for Relevance Vector Machine and `GPR` for Gaussian Process Regression are provided by the `R` package `kernlab`. The complete script with computation of average errors is provided on the course website. Be sure to download it and explore it. You should try various datasets to demonstrate to yourself the celebrated No Free Lunch Theorem which essentially stipulates that *no method can be that universally does well on all possible data sets*. In a sense, no matter how hard one tries to build a technique that universally does well, aspects will be ignored or simply not accounted for that turn out to be the characteristics of some data somewhere.

The Power of the Weak? In the table that follows, we see sophisticated techniques like Support Vector Machines doing better than simple polynomial regression. *Does this superiority remain with problems that are inherently linear and therefore better suited to our old friend MLR?*

表 1: *Average Training Error and Average Test Error over* $m = 10$ *random splits of* $n = 300$ *observations generated from a population with true function* $f(\mathrm{x}) = -\mathrm{x} + \sqrt{2}\sin(\pi^{3/2}\mathrm{x}^2)$ *for* $\mathrm{x} \in [-1, +1]$. *The noise variance in this case is* $\sigma^2 = 0.3^2$. *Each split has* `ntr`$= 2n/3$.

		Approximating Function Class			
		Poly	SVM	RVM	GPR
Average	Training Error	0.0998	0.0335	0.0295	0.1861
	Test Error	0.3866	0.1465	0.1481	0.1556

The above table says it loud and clear that the polynomial approximation class has an inferior performance relative to the other approximating schemes, as judged by the average test error. This should not

be too surprising as the other techniques are specifically designed to tackle nonlinear problems of this type and should do better.

`Insights into model selection:` *The question should arise in your mind as to what happens within each class? In other words, how what the polynomial selected? We address this issue when we explore model selection techniques and bias-variance trade-off in data ming and machine learning.*

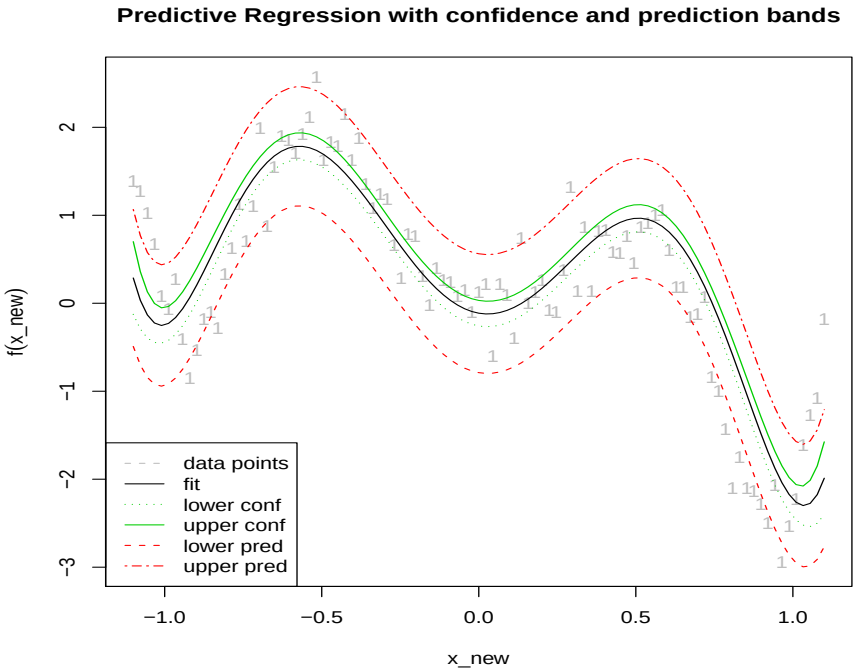


图 2: *Simple Orthogonal Polynomial Regression of with both confidence bands and prediction bands on the test set. The true function is* $f(\mathrm{x}) = -\mathrm{x} + \sqrt{2}\sin(\pi^{3/2}\mathrm{x}^2)$ *for* $\mathrm{x} \in [-1, +1]$. *The noise variance in this case is* $\sigma^2 = 0.3^2$.

1.4 Multiple Linear Regression (MLR)

With none of the single variable models capturing the whole whole structure underlying the golf Earnings data, it is reasonable to attempt fitting a model with many variables. The theoretical model of the form

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon \tag{6}$$

Where ε is assumed to have a normal distribution with zero mean and constant variance σ^2.
The `R` command for fitting the MLR with `Earnings` as the response and everything else being the explanatory variables, is

```
earn.mod <- lm(Earnings~., data=golf)  # fit the model with data from "golf"
lm.earn  <- summary(earn.mod)          # Store model object for summaries
print(lm.earn)                         # Print summary of model

par(mfrow=c(2,2))                      # Prepare plot of residuals
plot(earn.mod)                         # Plot residuals
```

The output of the model summary is

```
Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          3.90e+07   1.72e+07    2.27    0.025 *
Age                 -1.83e+03   5.40e+03   -0.34    0.735
Yards.Drive         -5.96e+03   8.67e+03   -0.69    0.493
Driving.Acc          2.04e+04   3.85e+04    0.53    0.596
Drive.Total          3.17e+03   3.50e+03    0.90    0.367
Greens.Reg          -2.97e+03   1.81e+04   -0.16    0.869
Putting.Avg          3.47e+06   5.19e+06    0.67    0.505
Save.Pct            -9.41e+02   8.31e+03   -0.11    0.910
Scrambling.Pct      -5.83e+04   2.63e+04   -2.21    0.028 *
Putts.Round         -2.66e+05   2.45e+05   -1.09    0.277
Birdie.Conversions   2.19e+03   7.09e+04    0.03    0.975
Par.5.Birdies        6.96e+03   2.06e+04    0.34    0.736
Par.4.Birdies       -1.05e+04   6.53e+04   -0.16    0.872
Par.3.Birdies       -5.41e+04   2.58e+04   -2.10    0.037 *
Avg.Score           -2.94e+05   2.12e+05   -1.39    0.166
Avg.Score.Before.Cut -1.40e+05   1.47e+05   -0.95    0.343
Avg.Finish          -2.31e+04   1.20e+04   -1.93    0.056 .
Events              -2.94e+04   4.91e+04   -0.60    0.550
```

```
Rounds                1.98e+04   2.13e+04    0.93    0.354
Cuts.Made            -7.95e+04   5.01e+04   -1.59    0.115
Top.10                2.61e+05   2.86e+04    9.13  2.4e-16 ***
FedEx.Points          9.04e-01   1.32e+00    0.68    0.496
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 485000 on 165 degrees of freedom
Multiple R-squared: 0.821,      Adjusted R-squared: 0.798
F-statistic: 36.1 on 21 and 165 DF,  p-value: <2e-16
```

Some crucial remarks are in order

- ⊡ `[Multicollinearity]:` The first surprising remark here is that many of the variables that were strongly significant when taken alone are no longer significant at all when combined with other variables. It turns out as we shall explained later that variables that are highly correlated might end up hurting one another up to the point of canceling one another's influence.
- ⊡ `[Principle of Parsimony]:` The second really important remark is that with some many variable combined together, the resulting model is not substantially better in terms of R^2 than the single variable model on `Avg.Score` after the addition of the quadratic term. While an R^2 of 82% is better than 80% (`Avg.Score` alone), the improvement is too tiny compared to the price paid (a complex model with many more variables)

  `Principle of parsimony`: *Simple models should be preferred over unnecessarily complex ones.*

It is important therefore to find strategy for selecting only those variables that really add substantial value to the model. For the MLR we will use the technique known as `Stepwise Regression`. However, before using the procedure of stepwise regression, it is important to mention something we already mentioned earlier:

`Model selection criterion`: *In the presence of two models, we need a criterion for deciding which one is the best. For example, the coefficient of determination R^2 encountered earlier in one such criterion. The model with the highest R^2 is deemed better. Other criterion exist, among which, Akaike Information Criterion (AIC), Bayesian Information Ciriterion (BIC), Mallows' Cp and Cross Validation. Read the lecture notes on model selection for more details.*

Stepwise regression in `R`.

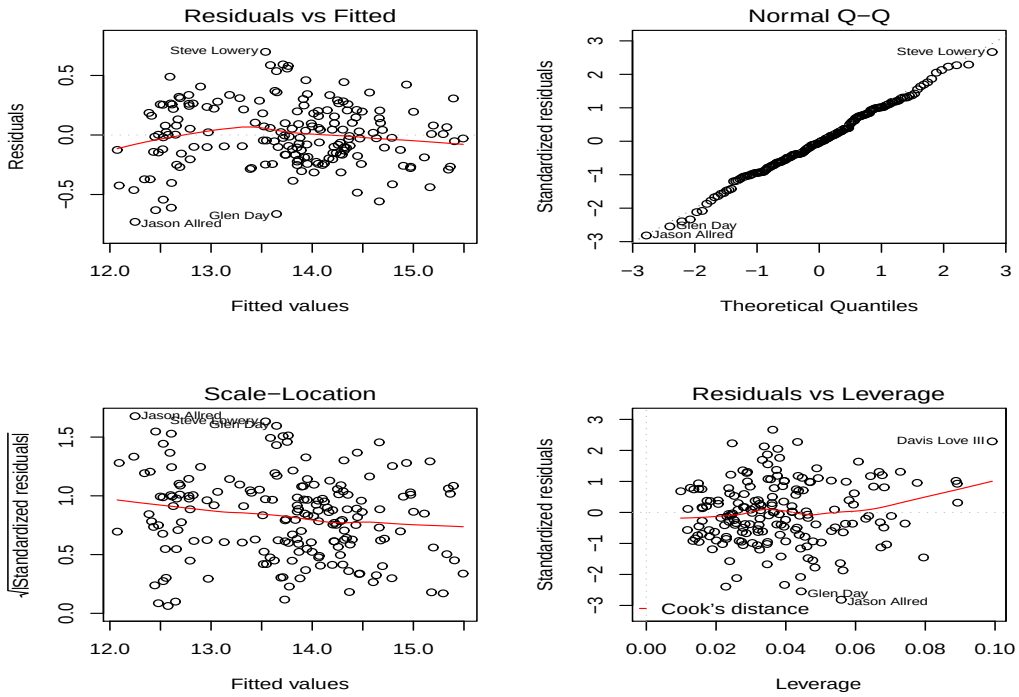
```
out1<-which(rownames(golf)=='Cameron Beckman'|
            rownames(golf)=='Marc Turnesa'|
```

```
            rownames(golf)==’Will MacKenzie’)

  lgolf   <- golf
  lgolf$Earnings <- log(lgolf$Earnings)
  lgolf <- lgolf[-out1, ]
  earn.mod <- lm(Earnings~., data=lgolf)

  earn.clean <- step(earn.mod, Earnings~.,
  direction="both", trace=0, k=log(nrow(lgolf)))

  par(mfrow=c(2,2))
  plot(earn.clean)

  lm.earn.clean <- summary(earn.clean)
  print(lm.earn.clean)


# Extract useful stuff
  lm.earn.clean$residuals    # Residual normality
```

The output is then given by

```
Call:
lm(formula = Earnings ~ Driving.Acc + Avg.Score + Avg.Score.Before.Cut +
    Rounds + Top.10 + FedEx.Points, data = lgolf)
Coefficients:
                     Estimate Std. Error t value Pr(>|t|)l
(Intercept)           3.98e+01   3.87e+00   10.30  < 2e-16 ***
Driving.Acc          -1.07e-02   3.86e-03   -2.76  0.00632 **
Avg.Score            -5.80e-01   6.47e-02   -8.97  4.1e-16 ***
Avg.Score.Before.Cut  2.03e-01   5.60e-02    3.63  0.00038 ***
Rounds                5.33e-03   1.65e-03    3.24  0.00142 **
Top.10                1.28e-01   1.49e-02    8.58  4.5e-15 ***
FedEx.Points          6.51e-06   6.87e-07    9.46  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.267 on 177 degrees of freedom
```

```
Multiple R-squared: 0.909,      Adjusted R-squared: 0.906
F-statistic:  293 on 6 and 177 DF,  p-value: <2e-16
```



All looks reasonable now, and even a formal test of the normality of the residuals reveals that the residuals are indeed normally distributed.

```
        Shapiro-Wilk normality test

data:  lm.earn.clean$residuals
 W = 0.9946, p-value = 0.7475
```

1.5 Correlation tests and Variable Selection

Variable selection is arguably one of the hottest topics in both theoretical and applied statistics nowadays. This appeal of variable selection is certainly due to the fact that modern day data mining has to deal with a very large number of variables, of which many are either noise variables or redundant variables. One of the simplest techniques of variable selection, at least at any initial stage consists of using the sample correlation coefficient. In other words, if Y is the response variable, and there are p explanatory variables $X_1, X_2, \cdots, X_p$, one seeks to retain only those variables whose correlation with the response passes a certain threshold. Let

$$r_j = \widehat{\texttt{cor}(X_j, Y)}$$

be the sample correlation coefficient between X_j and Y. Then we may have as a rule that variables with $|r_j| > 0.3$ are retained as relevant[2]. An even more emphatic way to decide which variable to retain is to perform the correlation test and only keep variables that are significant. The test is simply

$$\left[\begin{array}{lll} \texttt{H}_0: & \rho_j = \texttt{cor}(X_j, Y) = 0 & X_j \text{ and } Y \text{ are uncorrelated} \\ \texttt{H}_\texttt{a}: & \rho_j = \texttt{cor}(X_j, Y) \neq 0 & X_j \text{ and } Y \text{ are correlated} \end{array}\right.$$

This correlation test can be performed readily in `R` using `cor.test()`. For instance, the test for the correlation between Annual Golf Earnings and Annual Golf Average Score is done as follows:

```
> cor.test(golf$Avg.Score,golf$Earnings)
     Pearson's product-moment correlation
data:  golf$Avg.Score and golf$Earnings
t = -18.64, df = 185, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
       -0.8524 -0.7514
```

For the Golf Earnings, we can run a little `R` script to automatically select the meaningful (relevant) significant variables.

```
#    Select those variables that have high correlation
#        With the response variable being studied
XY       <- golf
p        <- ncol(XY)
good.var <- c()
for(i in 1:(p-1))
{
   if (abs(cor(XY[,i],XY[,p]))>=0.30){
```

[2] I read somewhere that sociologists consider 0.3 to be very good correlation.

```
  if (cor.test(XY[,i],XY[,p])$p.value < 0.05)
  {good.var <- c(good.var, colnames(XY)[i])}}
}
```

The output is

```
> good.var
[1] "Greens.Reg"           "Putting.Avg"          "Birdie.Conversions"   "Par.5.Birdies"
[5] "Par.4.Birdies"        "Avg.Score"            "Avg.Score.Before.Cut" "Avg.Finish"
[9] "Cuts.Made"            "Top.10"               "FedEx.Points"
```

1.5.1 Friedman #1 function Benchmark task

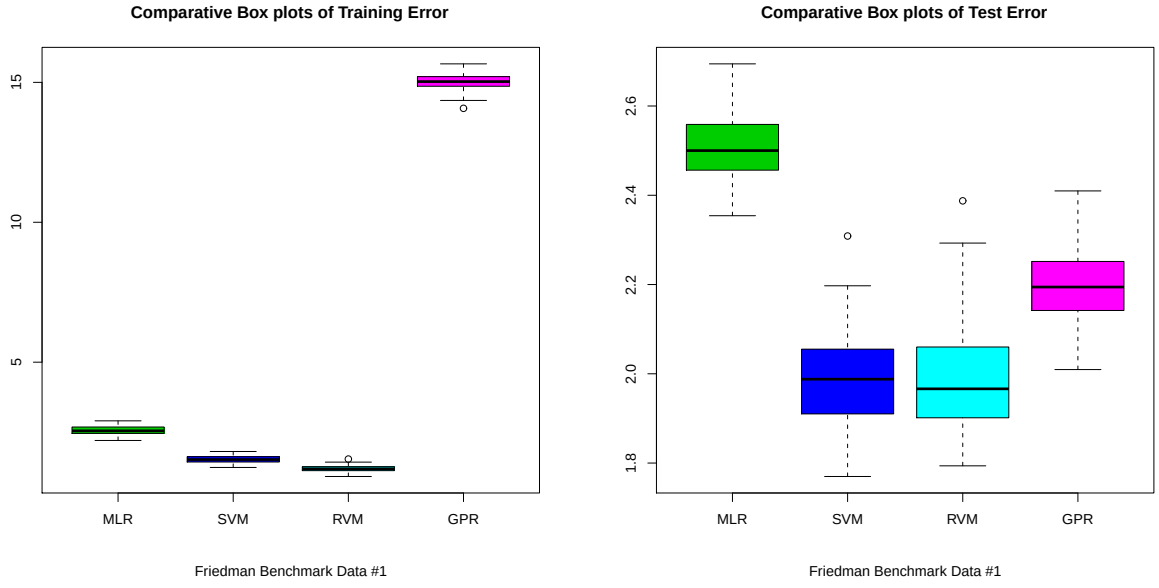
MLR has so many benefits that practitioners and sometimes theoreticians go to extreme lengths to perform transformation on the data so as to use linear model tools. However, there are cases where the model underlying the data is so inherently nonlinear that such cosmetics on the data won't achieve much. A very simple illustrative example is the collection of the so-called Friedman's benchmark function for regression analysis which are described below and also found the R package mlbench. We will go over these functions in great details in our lecture notes entitled Exploration 2.

The first Friedman benchmark function has intrinsic dimensional $p = 5$ with $\mathrm{x}_j \in \mathsf{Uniform}(0,1)$ for $i = 1, .., p$. The true underlying function

$$f(\mathbf{x}) = 10\sin(\pi \mathrm{x}_1 \mathrm{x}_2) + 20\left(\mathrm{x}_3 - \frac{1}{2}\right)^2 + 10\mathrm{x}_4 + 5\mathrm{x}_5.$$

To add spice to the problem, the data is generated with 5 additional noise variables, making the input dimension manifestly (visibly) $q = 10$. Finally, the true response is corrupted with a noise term $\epsilon \sim \mathsf{N}(0,1)$.

	MLR	SVM	RVM	GPR
Training	6.601	2.312	1.421	225.449
Test	6.291	3.929	3.921	4.814



1.5.2 Friedman #2 function Benchmark task

The second Friedman function lives in 4 dimensions

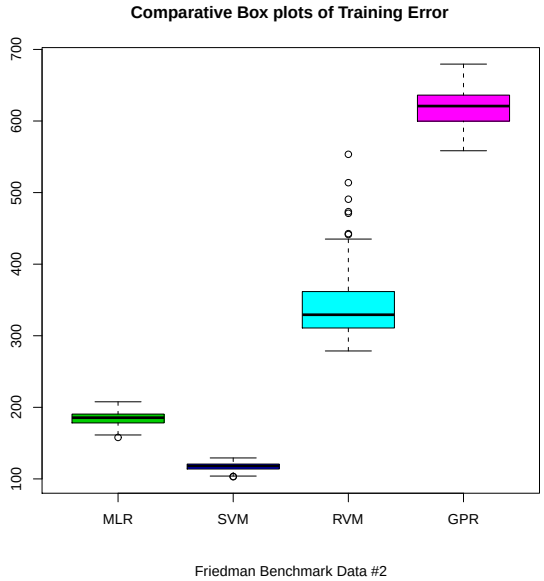
$$f(\mathbf{x}) = \sqrt{\mathrm{x}_1^2 + \left(\mathrm{x}_2\mathrm{x}_3 - \frac{1}{\mathrm{x}_2\mathrm{x}_4}\right)^2}$$

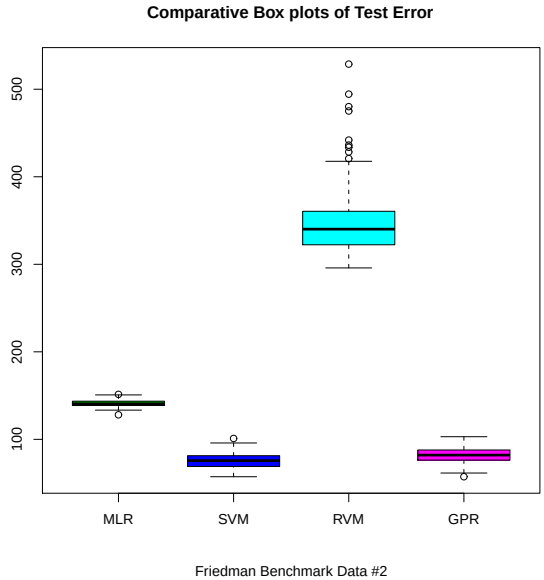
where the 4 independent variables $\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4$ are uniformly distributed in the following domains:

- $0 \leq \mathrm{x}_1 \leq 100$,
- $40\pi \leq \mathrm{x}_2 \leq 560\pi$,
- $0 \leq \mathrm{x}_3 \leq 1$,
- $1 \leq \mathrm{x}_4 \leq 11$

The noise is Gaussian, i.e., $\epsilon \sim \mathbf{N}(0, \sigma^2)$, with standard deviation $\sigma = 125$ used by default. You should experiment with various values of σ.

	MLR	SVM	RVM	GPR
Training	34015	13757	119052	384062
Test	19967	5744	122787	6777





1.5.3 Friedman #3 function Benchmark task

The third Friedman function lives in the same 4 dimensions above

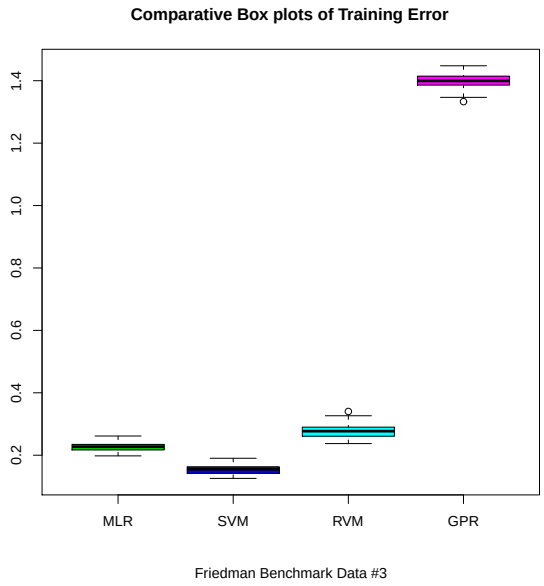
$$f(\mathbf{x}) = \tan^{-1}\left\{\frac{1}{\mathrm{x}_1}\left(\mathrm{x}_2\mathrm{x}_3 - \frac{1}{\mathrm{x}_2\mathrm{x}_4}\right)\right\}$$

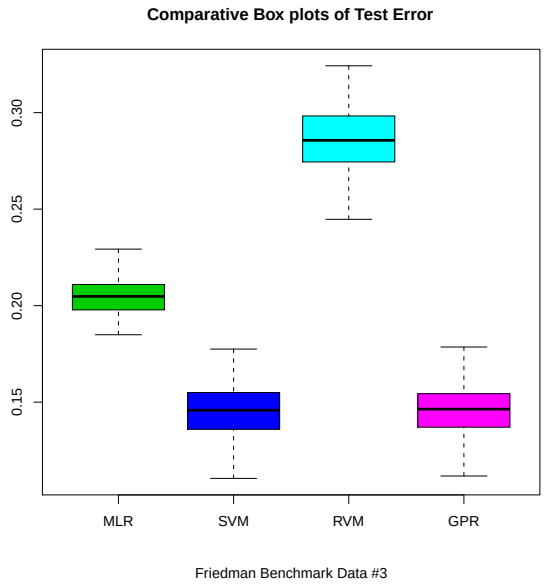
where the 4 independent variables $\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4$ are uniformly distributed in the following domains:

- ⊡ $0 \leq \mathrm{x}_1 \leq 100$,
- ⊡ $40\pi \leq \mathrm{x}_2 \leq 560\pi$,
- ⊡ $0 \leq \mathrm{x}_3 \leq 1$,
- ⊡ $1 \leq \mathrm{x}_4 \leq 11$

The noise is Gaussian, i.e., $\epsilon \sim \mathbf{N}(0, \sigma^2)$, with standard deviation $\sigma = 125$ used by default. You should experiment with various values of σ.

	MLR	SVM	RVM	GPR
Training	0.05084	0.02350	0.07665	1.95543
Test	0.04203	0.02121	0.08170	0.02121





1.5.4 Fokoue #1 function Benchmark task

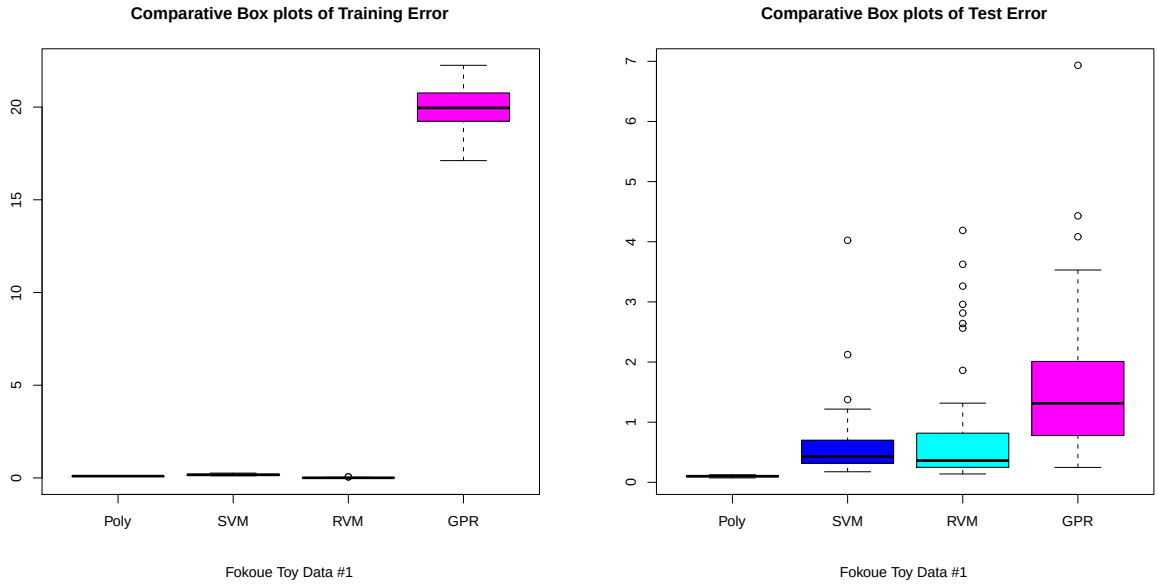
Consider the following simple univariate function

$$f(\mathrm{x}) = 1 + 9\mathrm{x}, \qquad \mathrm{x} \in [-1, +1]. \tag{7}$$

The goal is to find the best generalizing approximator of f from a sample $\mathcal{D} = \{(\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \cdots, (\mathrm{x}_n, \mathrm{y}_n)\}$ of pairs drawn from its population. You are given the collection

$$\mathcal{F} = \{\texttt{Polynomial Regression}, \texttt{SVM}, \texttt{RVM}, \texttt{Gaussian Processes}\}$$

	Poly	SVM	RVM	GPR
Training Error	0.0969	0.1735	0.0105	20.028
Test Error	0.1014	0.6016	0.8472	1.593



1.5.5 Fokoue #2 function Benchmark task

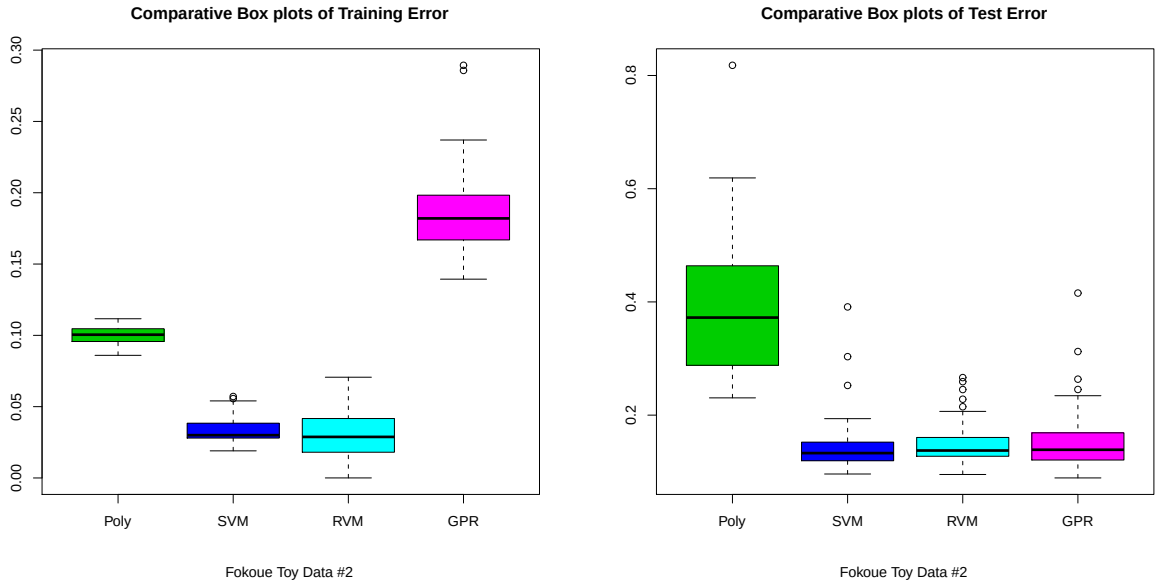
Consider the following simple nonlinear univariate function

$$f(\mathrm{x}) = -\mathrm{x} + \sqrt{2}\sin(\pi^{3/2}\mathrm{x}^2), \qquad \mathrm{x} \in [-1, +1]. \tag{8}$$

The goal is to find the best generalizing approximator of f from a sample $\mathcal{D} = \{(\mathrm{x}_1, \mathrm{y}_1), (\mathrm{x}_2, \mathrm{y}_2), \cdots, (\mathrm{x}_n, \mathrm{y}_n)\}$ of pairs drawn from its population. You are given the collection

$$\mathcal{F} = \{\texttt{Polynomial Regression}, \texttt{SVM}, \texttt{RVM}, \texttt{Gaussian Processes}\}$$

	Poly	SVM	RVM	GPR
Training Error	0.0998	0.0335	0.0295	0.1861
Test Error	0.3866	0.1465	0.1481	0.1556



Although the above functions are synthetic, they help illustrate the fact that the linear model is often just an approximation of the true underlying function, and can be very very far from the truth. (See Exercise with diagnostics on Friedman). When the true underlying function is not linear, We need better approximations that capture the nonlinearity inherent in the data:

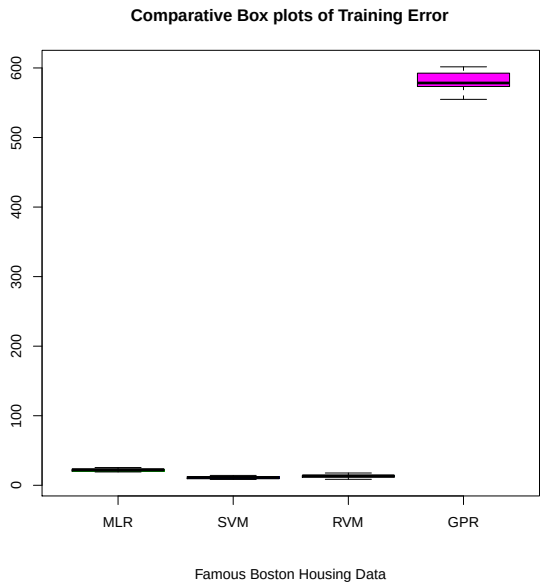
1.5.6 Boston Housing Benchmark task

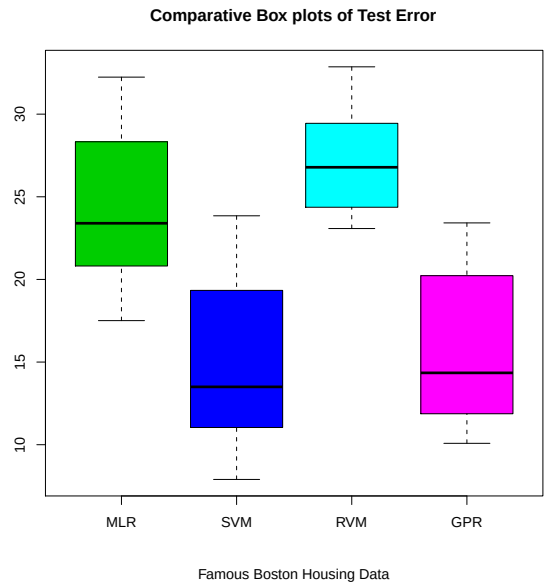
Housing data for 506 census tracts of Boston from the 1970 census. The dataframe BostonHousing contains the original data by Harrison and Rubinfeld (1979), the dataframe BostonHousing2 the corrected version with additional spatial information (see references below). This data frame contains the following columns:

`crim`	per capita crime rate by town
`zn`	proportion of residential land zoned for lots over 25,000 sq.ft
`indus`	proportion of non-retail business acres per town
`chas`	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
`nox`	nitric oxides concentration (parts per 10 million)
`rm`	average number of rooms per dwelling
`age`	proportion of owner-occupied units built prior to 1940
`dis`	weighted distances to five Boston employment centres
`rad`	index of accessibility to radial highways
`tax`	full-value property-tax rate per USD 10,000
`ptratio`	pupil-teacher ratio by town
`b`	$1000(B - 0.63)^2$ where B is the proportion of blacks by town
`lstat`	percentage of lower status of the population
`medv`	median value of owner-occupied homes in USD 1000's

The response variable for this problem is `medv`, the median value of owner-occupied homes in USD 1000's. The data is contained in the dataframe `BostonHousing` which can be obtained from the `R` package `mlbench`. Below are the performance comparison table and boxplots.

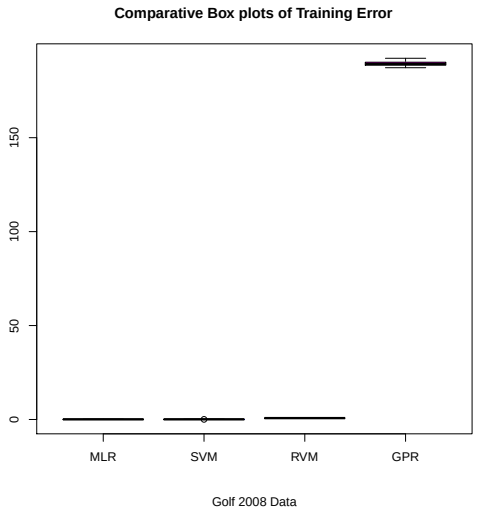
	MLR	SVM	RVM	GPR
Training Error	22.25	11.14	13.26	580.87
Test Error	24.23	14.68	26.95	15.61

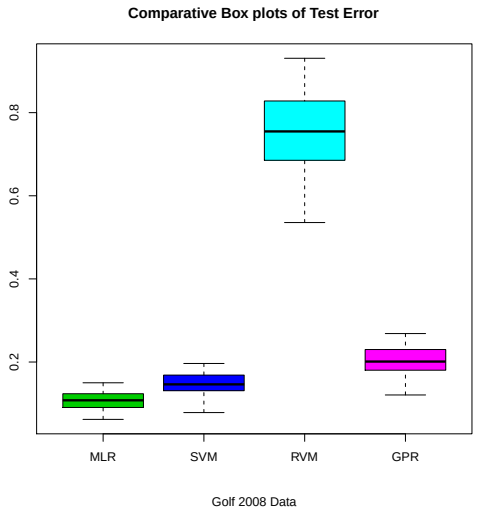




1.5.7 Golf 2008 Benchmark task

	MLR	SVM	RVM	GPR
Training Error	0.0808	0.08433	0.7330	189.3802
Test Error	0.1078	0.14814	0.7547	0.2044





1.6 Cross Comparison of Methods on Various Datasets

The following table shows both the training error and the test error achieved by each method on different benchmark datasets. The lowest test error is marked in red for clarity.

Note: Except for the linear model for which stepwise was used to isolate the best model, None of

Age	Age of the golfer
Yards.Drive	Number of yards per drive
Driving.Acc	Driving Accuracy
Drive.Total	Total number of driving plays
Greens.Reg	Number of greens in regulation
Putting.Avg	Putting average
Save.Pct	proportion of saves made
Scrambling.Pct	Percentage of scrambling situations saved
Putts.Round	Number of putts per round
Birdie.Conversions	Number of birdies converted
Par.5.Birdies	Number of Par 5 birdies
Par.4.Birdies	Number of Par 4 birdies
Par.3.Birdies	Number of Par 3 birdies
Avg.Score	Average score
Avg.Score.Before.Cut	Average Score before cut
Avg.Finish	Average Finish
Events	Number of events entered
Rounds	Number of rounds played
Cuts.Made	Number of cuts made
Top.10	Number of times at the top 10
FedEx.Points	Total number FedEx Championship points
Earnings	Total Amount of money earn on tour in 2008

the techniques is tuned perfectly. For instance, the Support Vector Machine has many different tuning parameters whose adjustments have the potential of substantially improving their performance. In this case, we have used default tuning parameter values.

	Poly		SVM		RVM		GPR	
	Training	Test	Training	Test	Training	Test	Training	Test
`Fokoue #1`	0.0969	0.1014	0.1735	0.6016	0.0105	0.8472	20.028	1.593
`Fokoue #2`	0.0998	0.3866	0.0335	0.1465	0.0295	0.1481	0.1861	0.1556
`Friedman #1`	6.601	6.291	2.312	3.929	1.421	3.921	225.449	4.814
`Friedman #2`	34015	19967	13757	5744	119052	122787	384062	6777
`Friedman #3`	0.0508	0.0420	0.0235	0.0212	0.0766	0.0817	1.9554	0.0212
`Boston Housing`	22.25	24.23	11.14	14.68	13.26	26.95	580.87	15.61
`Golf 2008`	0.0808	0.1078	0.0843	0.1481	0.7330	0.7547	189.3802	0.2044

表 2: Comparison of various techniques on benchmark datasets. Both the average training error and the average test error are shown over many replications.