Birla Institute of Technology and Science, Pilani
Machine Learning (SS ZG565)
Lab Sheet #2

Linear Regression

Linear Regression is a supervised modelling technique for continuous data. The model fits a line that is closest to all observation in the dataset. The basic assumption here is that functional form is the line and it is possible to fit the line that will be closest to all observation in the dataset. Please note that if the basic assumption about the linearity of the model is away from reality then there is bound to have an error (bias towards linearity) in the model however best one will try to fit the model.

The basic equation for any supervised learning algorithm

$Y = F(x) + \varepsilon$

The above equation has three terms:

Y – define the response variable.

F(X) – defines the function that is dependent on the set of input features.

ε – defines the random error. For ideal model, this should be random and should not be dependent on any input.

In linear regression, we assume that functional form, F(X) is linear and hence we can write the equation as below. Next step will be to find the coefficients (β0, β1..) for below model.

$Y = \beta_0 + \beta_1 X + \varepsilon$ (for simple regression)

$Y = \beta_0 + \beta_1 X1 + \beta_2 X2 + \beta_3 X3 + \ldots + \beta_p Xp + \varepsilon$ (for multiple regression)

Application of Linear Regression

The coefficient for linear regression is calculated based on the sample data. The basic assumption here is that the sample is not biased. This assumption makes sure that the sample does not necessarily always overestimate or underestimate the coefficients. The idea is that a particular sample may overestimate or underestimate but if one takes multiple samples and try to estimate the coefficient multiple times, then the average of coefficient from multiple samples will be spot on.

Extract the data and create the training and testing sample

For the current model, let's take the Boston dataset that is part of the MASS library in R Studio. Following are the features available in Boston dataset. The problem statement is to predict 'medv' based on the set of input features.

```
library(MASS)
library(ggplot2)
attach(Boston)
```

Split the sample data and make the model

Split the input data into training and evaluation set and make the model for the training dataset. It can be seen that training dataset has 404 observations and testing dataset has 102 observations based on 80-20 split.

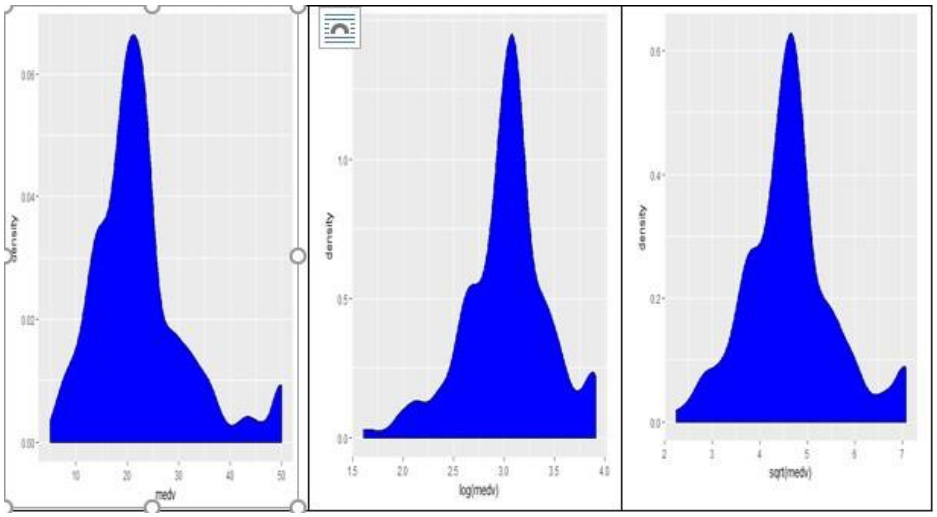
```
##Sample the dataset. The return for this is row nos.
set.seed(1)
row.number <- sample(1:nrow(Boston), 0.8*nrow(Boston))
train = Boston[row.number,]
test = Boston[-row.number,]
dim(train)
dim(test)
```

Explore the response variable

Let's check for the distribution of response variable 'medv'. The following figure shows the three distributions of 'medv' original, log transformation and square root transformation. We can see that both 'log' and 'sqrt' does a decent job to transform 'medv' distribution closer to normal. In the following model, I have selected 'log' transformation but it is also possible to try out 'sqrt' transformation.

```
##Explore the data.
ggplot(train, aes(medv)) + geom_density(fill="blue")
ggplot(train, aes(log(medv))) + geom_density(fill="blue")
ggplot(train, aes(sqrt(medv))) + geom_density(fill="blue")
```

Gives this plot:

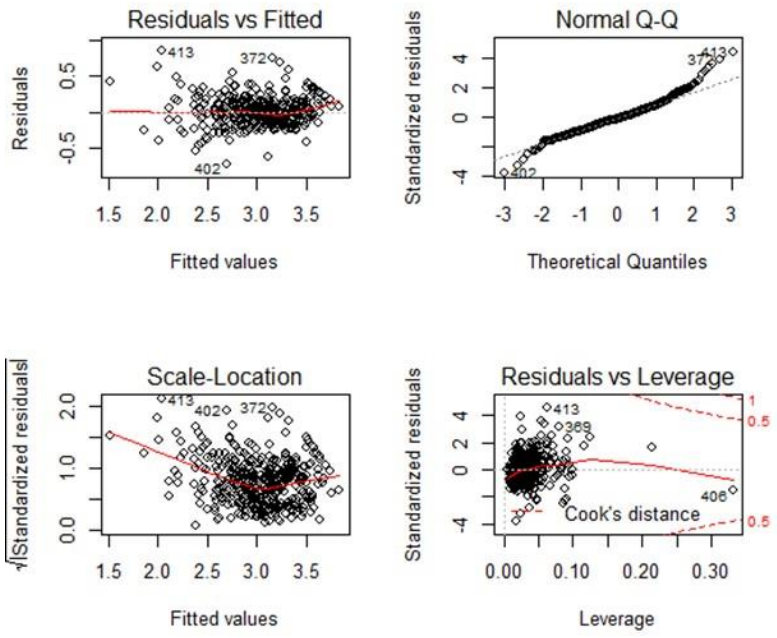


Model Building – Model 1

Now as a first step we will fit the multiple regression models. We will start by taking all input variables in the multiple regression.

```
#Let's make default model.
model1 = lm(log(medv)~., data=train)
summary(model1)
par(mfrow=c(2,2))
plot(model1)
```

Gives this plot



Observation from summary (model1)

1. *Is there a relationship between predictor and response variables?*

   We can answer this using F stats. This defines the collective effect of all predictor variables on the response variable. In this model, F=102.3 is far greater than 1, and so it can be concluded that there is a relationship between predictor and response variable.
2. *Which of the predictor variables are significant?*

   Based on the ‘p-value’ we can conclude on this. The lesser the ‘p’ value the more significant is the variable. From the ‘summary’ dump we can see that ‘zn’, ‘age’ and ‘indus’ are less significant features as the ‘p’ value is large for them. In next model, we can remove these variables from the model.
3. *Is this model fit?*

   We can answer this based on R2 (multiple-R-squared) value as it indicates how much variation is captured by the model. R2 closer to 1 indicates that the model explains the large value of the variance of the model and hence a good fit. In this case, the value is 0.7733 (closer to 1) and hence the model is a good fit.

Observation from the plot

1. *Fitted vs Residual graph*

   Residuals plots should be random in nature and there should not be any pattern in the graph. The average of the residual plot should be close to zero. From the above plot, we can see that the red trend line is almost at zero except at the starting location.
2. *Normal Q-Q Plot*

   Q-Q plot shows whether the residuals are normally distributed. Ideally, the plot should be on the dotted line. If the Q-Q plot is not on the line then models need to be reworked to make the residual normal. In the above plot, we see that most of the plots are on the line except at towards the end.
3. *Scale-Location*

   This shows how the residuals are spread and whether the residuals have an equal variance or not.
4. *Residuals vs Leverage*

   The plot helps to find influential observations. Here we need to check for points that are outside the dashed line. A point outside the dashed line will be influential point and removal of that will affect the regression coefficients.

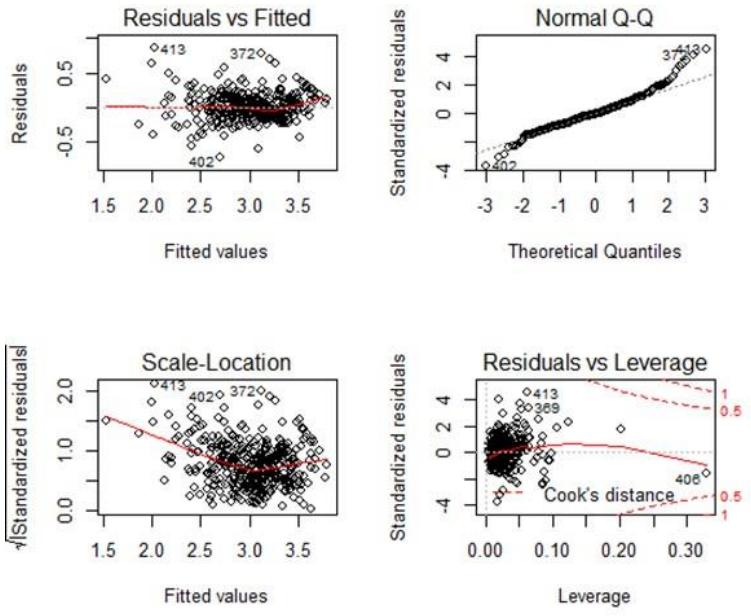
Model Building – Model 2

As the next step, we can remove the four lesser significant features (‘zn’, age’ and ‘indus’) and check the model again.

```
# remove the less significant feature model2
= update(model1, ~.-zn-indus-age)
summary(model2)
```

```
par(mfrow=c(2,2))
plot(model2)
```

Gives this plot



Observation from summary (model2)

1. *Is there a relationship between predictor and response variable?*

   F=131.2 is far greater than 1 and this value is more than the F value of the previous model. It can be concluded that there is a relationship between predictor and response variable.

2. *Which of the variable are significant?*

   Now in this model, all the predictors are significant.

3. *Is this model fit?*

   R2 =0.7696 is closer to 1 and so this model is a good fit. Please note that this value has decreased a little from the first model but this should be fine as removing three predictors caused a drop from 0.7733 to 0.7696 and this is a small drop. In other words, the contribution of three predictors towards explaining the variance is an only small value(0.0037) and hence it is better to drop the predictor.

Observation of the plot

All the four plots look similar to the previous model and we don't see any major effect.

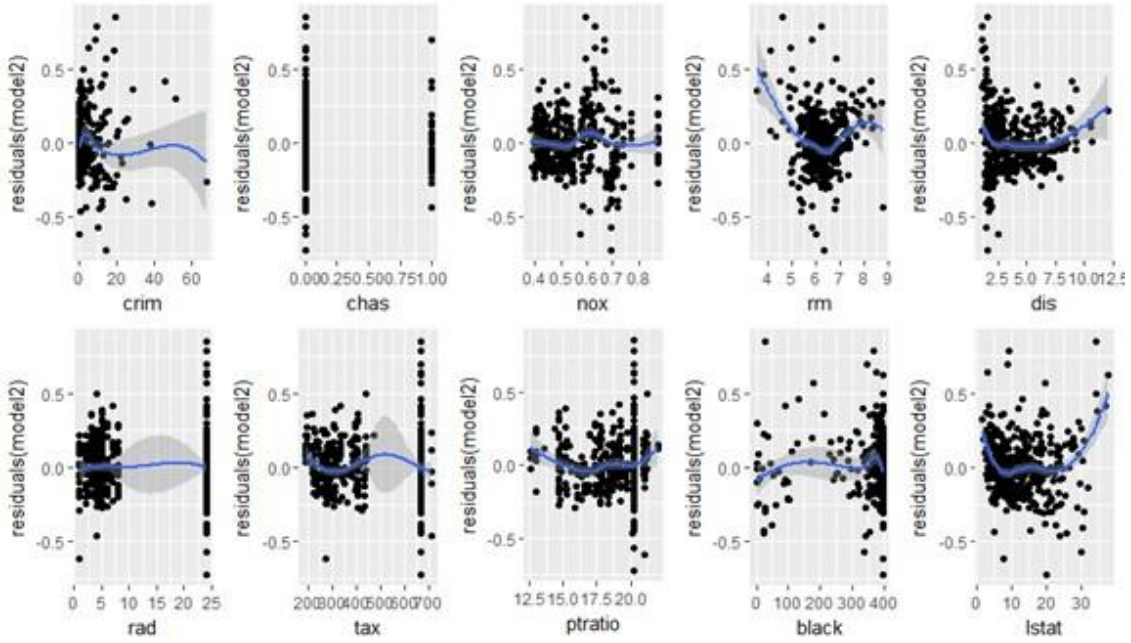
Check for predictor vs Residual Plot

In the next step, we will check the residual graph for all significant features from Model 2. We need to check if we see any pattern in the residual plot. Ideally, the residual plot should be random plot and we should not see a pattern. In the following plots, we can see some non-linear pattern for features like 'crim', 'rm', 'nox' etc.

```
##Plot the residual plot with all predictors.
attach(train)
require(gridExtra)
plot1 = ggplot(train, aes(crim, residuals(model2))) + geom_point() +
geom_smooth()
plot2=ggplot(train, aes(chas, residuals(model2))) + geom_point() +
geom_smooth()
plot3=ggplot(train, aes(nox, residuals(model2))) + geom_point() +
geom_smooth()
plot4=ggplot(train, aes(rm, residuals(model2))) + geom_point() +
geom_smooth()
plot5=ggplot(train, aes(dis, residuals(model2))) + geom_point() +
geom_smooth()
plot6=ggplot(train, aes(rad, residuals(model2))) + geom_point() +
geom_smooth()
plot7=ggplot(train, aes(tax, residuals(model2))) + geom_point() +
geom_smooth()
plot8=ggplot(train, aes(ptratio, residuals(model2))) + geom_point() +
geom_smooth()
```

```
plot9=ggplot(train, aes(black, residuals(model2))) + geom_point() +
geom_smooth()
plot10=ggplot(train, aes(lstat, residuals(model2))) + geom_point() +
geom_smooth()
grid.arrange(plot1,plot2,plot3,plot4,plot5,plot6,plot7,plot8,plot9,plot
10,ncol=5,nrow=2)
```

Gives this plot



Model Building – Model 3 & Model 4

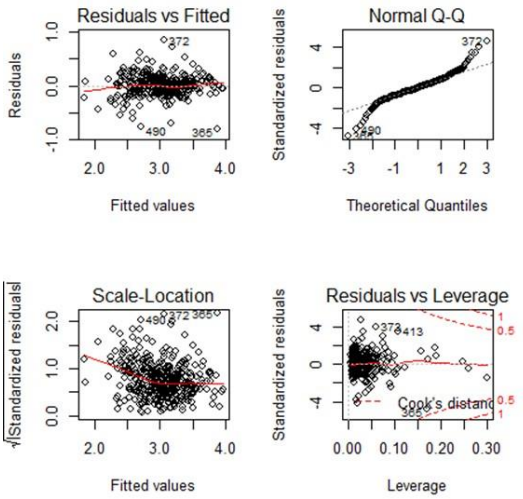
We can now enhance the model by adding a square term to check for non-linearity. We can first try model3 by introducing square terms for all features (from model 2). And in the next iteration, we can remove the insignificant feature from the model.

```
#Lets  make default model and add square term in the model. model3 =
lm(log(medv)~crim+chas+nox+rm+dis+rad+tax+ptratio+ black+lstat+
I(crim^2)+ I(chas^2)+I(nox^2)+ I(rm^2)+ I(dis^2)+ I(rad^2)+ I(tax^2)+
I(ptratio^2)+ I(black^2)+ I(lstat^2), data=train) summary(model3)
```

```
##Removing the insignificant variables.
model4=update(model3,   ~.-nox-rad-tax-I(crim^2)-I(chas^2)- I(rad^2)-
I(tax^2)-I(ptratio^2)-I(black^2))
summary(model4)
```

```
par(mfrow=c(2,2))
plot(model4)
```

Gives this plot



Observation from summary (model4)

1. *Is there a relationship between predictor and response variables?*
   F-Stat is 137.9 and it is far greater than 1. So there is a relationship between predictor and response variable.
2. *Which of the predictor variable are significant?*
   All predictor variables are significant.
3. *Is this model fit?*
   R2 is 0.7946 and this is more (and better) than our first and second model.

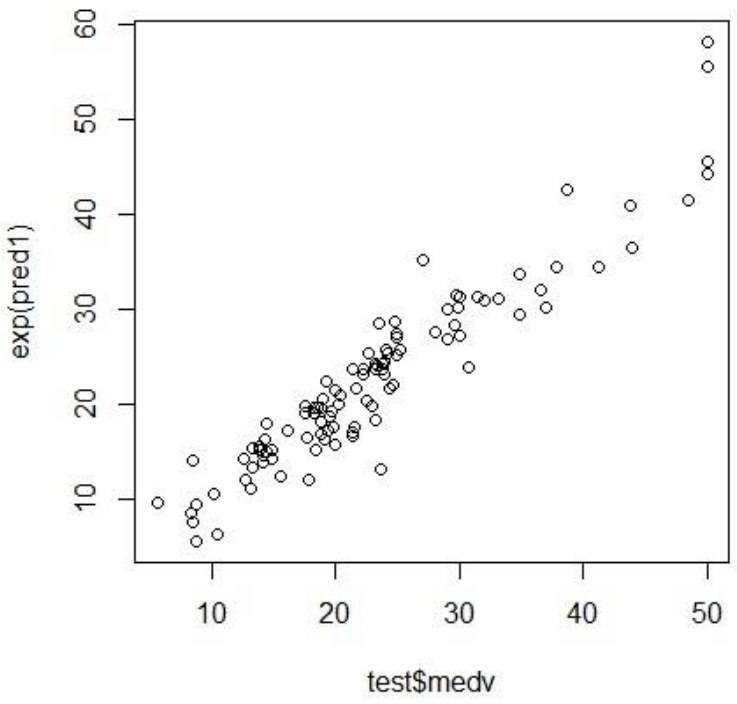
Prediction

Till now we were checking training-error but the real goal of the model is to reduce the testing error. As we already split the sample dataset into training and testing dataset, we will use test dataset to evaluate the model that we have arrived upon. We will make a prediction based on ‘Model 4’ and will evaluate the model. As the last step, we will predict the ‘test’ observation and will see the comparison between predicted response and actual response value. RMSE explains on an average how much of the predicted value will be from the actual value. Based on RMSE = 3.278, we can conclude that on an average predicted value will be off by 3.278 from the actual value.

```
pred1 <- predict(model4, newdata = test)
rmse <- sqrt(sum((exp(pred1) - test$medv)^2)/length(test$medv))
c(RMSE = rmse, R2=summary(model4)$r.squared)
```

```
par(mfrow=c(1,1))
plot(test$medv, exp(pred1))
```

Gives this plot



Conclusion

The example shows how to approach linear regression modelling. The model that is created still has scope for improvement as we can apply techniques like Outlier detection, Correlation detection to further improve the accuracy of more accurate prediction. A piece of warning is that we should refrain from overfitting the model for training data as the test accuracy of the model will reduce for test data in case of overfitting.

Polynomial Regression

With polynomial regression we can fit models of order n > 1 to the data and try to model nonlinear relationships.

How to fit a polynomial regression

First, always remember use to **set.seed(n)** when generating pseudo random numbers. By doing this, the random number generator generates always the same numbers.

```
set.seed(20)
```

Predictor (q). Use seq for generating equally spaced sequences fast

```
q <- seq(from=0, to=20, by=0.1)
```

Value to predict (y):

```
y <- 500 + 0.4 * (q-10)^3
```

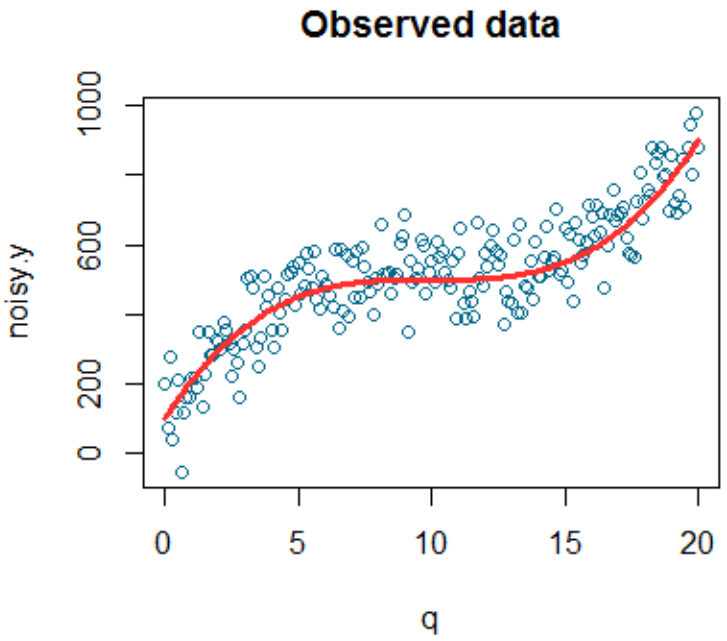
Some noise is generated and added to the real signal (y):

```
noise <- rnorm(length(q), mean=10, sd=80)
noisy.y <- y + noise
```

Plot of the noisy signal:

```
plot(q, noisy.y, col='deepskyblue4',xlab='q',main='Observed data')
lines(q, y, col='firebrick1', lwd=3)
```

This is the plot of our simulated observed data. The simulated data points are the blue dots while the red line is the signal (signal is a technical term that is often used to indicate the general trend we are interested in detecting).



Our model should be something like this: $y = a*q + b*q^2 + c*q^3 + cost$

Let's fit it using R. When fitting polynomials you can either use

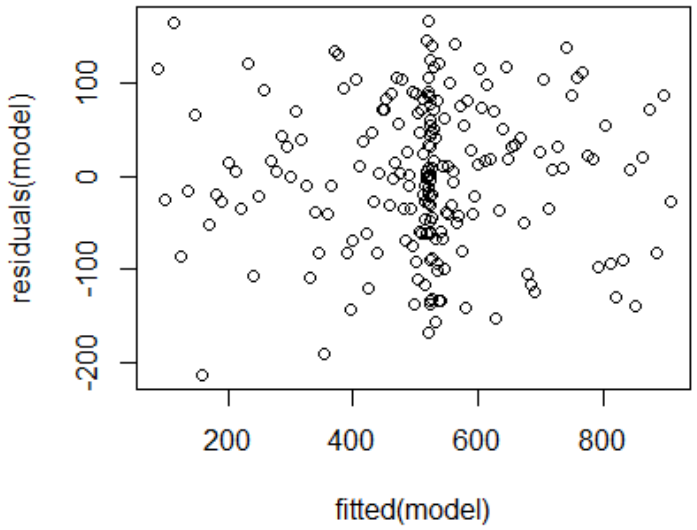
```
model <- lm(noisy.y ~ poly(q,3))
summary(model)
```

By using the confint() function we can obtain the confidence intervals of the parameters of our model.

Confidence intervals for model parameters:

```
confint(model, level=0.95)
plot(fitted(model),residuals(model))
```

Plot of fitted vs residuals. No clear pattern should show in the residual plot if the model is a good fit.



Overall the model seems a good fit as the R squared of 0.8 indicates. The coefficients of the first and third order terms are statistically significant as we expected. Now we can use the **predict()** function to get the fitted values and the confidence intervals in order to plot everything against our data.

Predicted values and confidence intervals:

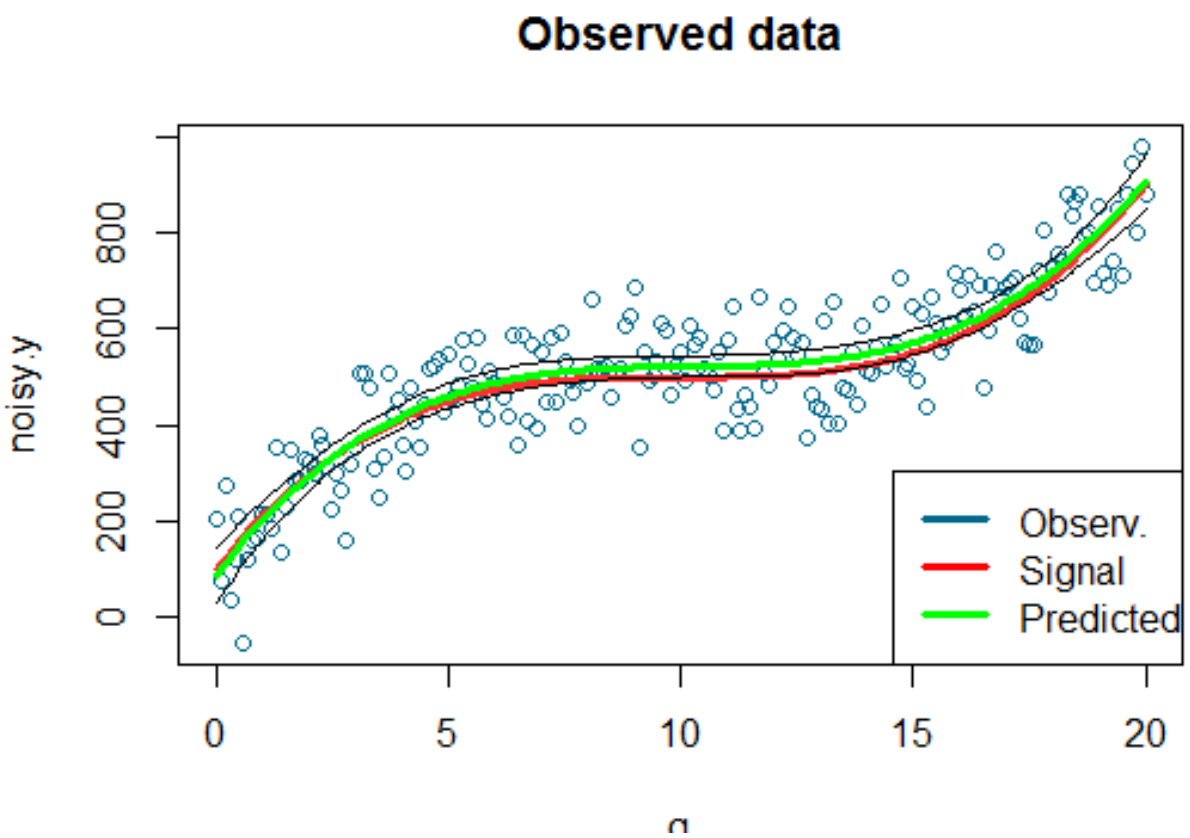
```
predicted.intervals <-predict (model, data.frame (x=q) ,
interval='confidence', level=0.99)
```

Add lines to the existing plot:

```
lines(q,          predicted.intervals[,1],col='green',lwd=3)
lines(q,          predicted.intervals[,2],col='black',lwd=1)
lines(q, predicted.intervals[,3],col='black',lwd=1)

legend("bottomright",c("Observ.","Signal","Predicted"),
col=c("deepskyblue4","red","green"), lwd=3)
```

Final Plot



We can see that our model did a decent job at fitting the data and therefore we can be satisfied with it.