Labsheet - 2

Linear Regression and Polynomial Curve Fitting

Machine Learning

BITS F464

In this Lab-sheet, we will work on Linear Regression model and then we will see basics about polynomial regression.

1. Linear Regression:

The general mathematical equation for a linear regression model is –

```
y = ax + b
```

Here,

y is the response variable.

x is the predictor variable.

a,b are weights associated with input.

Let's take a simple example of predicting weight of a person when his height is known. We will need to establish a relationship between height and weight of a person, for which `lm()` Function is used in **R**.

Input Data

Below is the sample data representing the observations –

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

The basic syntax for lm() function in linear regression is -

lm(formula, data)

Now let's create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

print(relation)

# Get the summary of the relationship

print(summary(relation))
```

predict() Function

Its basic syntax is:

```
predict(object, newdata)
```

Object is the formula which is already created using the lm() function for us it is "relation".
Newdata is the vector containing the new value for predictor variable.

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

Visualization

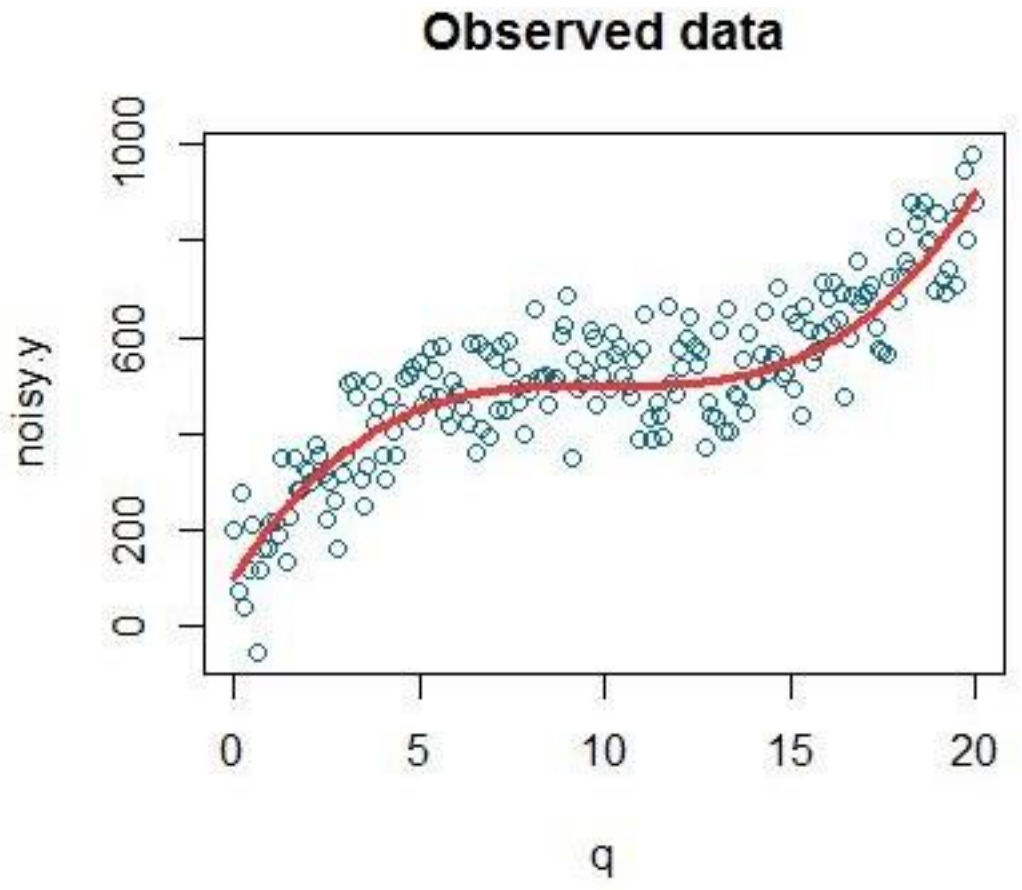
```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

#  Give  the  chart  file  a  name.
png(file = "linearregression.png")

# Plot the chart.
plot(y,x,col  =  "blue",main  =  "Height  &  Weight  Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab =
"Height in cm")

# Save the file.
dev.off()
```

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 datapoints 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))
```
Or
```
model <- lm(noisy.y ~ x + I(X^2) + I(X^3))
```

However, note that q, I(q^2) and I(q^3) will be correlated and correlated variables can cause problems. The use of poly() lets you avoid this by producing orthogonal polynomials, therefore I'm going to use the first option.

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

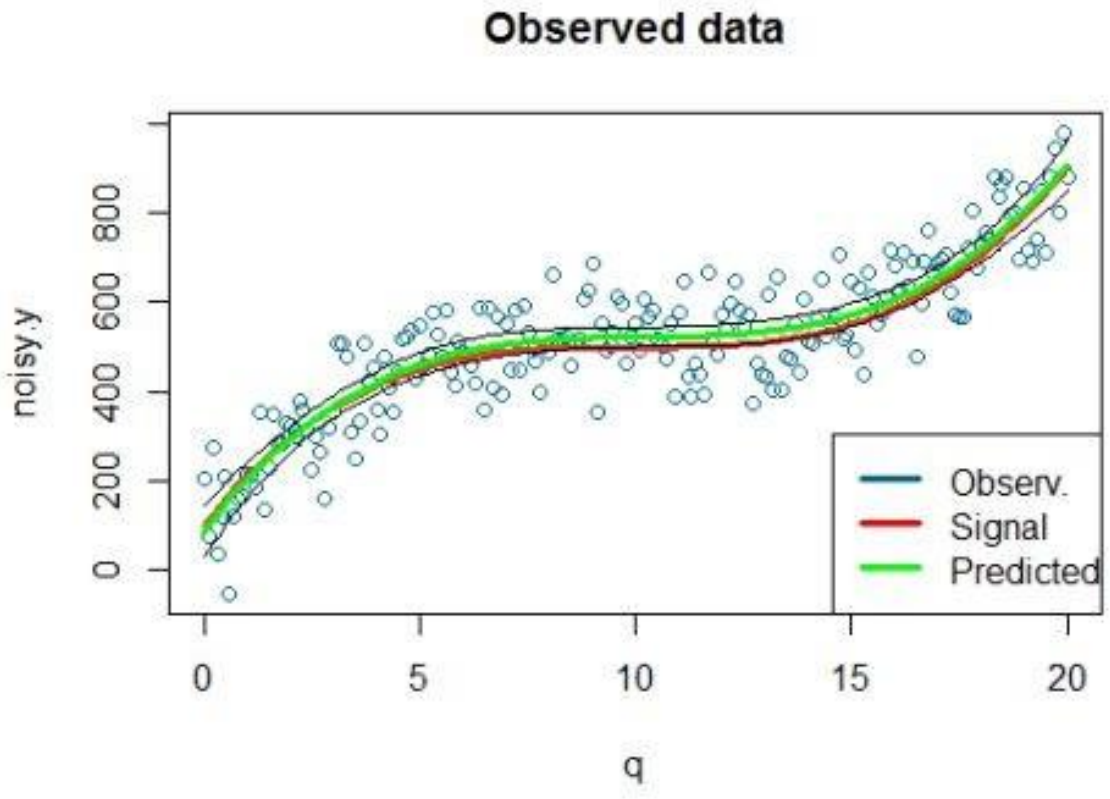
Add lines to the existing plot:

```
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)
```

Add a legend:

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

Here is the plot:



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

A word of caution: Polynomials are powerful tools but might backfire: in this case we knew that the original signal was generated using a third degree polynomial, however when analyzing real data, we usually know little about it and therefore we need to be cautious because the use of high order polynomials (n > 4) may lead to over-fitting. Over-fitting happens when your model is picking up the noise instead of the signal: even

though your model is getting better and better at fitting the existing data, this can be bad when you are trying to predict new data and lead to misleading results.

Exercise:

We can now go ahead with PCA. The base R function prcomp() is used to perform PCA. By default, it centers the variable to have mean equals to zero. With parameter `scale. = T`, we normalize the variables to have standard deviation equals to 1.

```
prin_comp <- prcomp(training_set, scale. = T)
```

As our previous data consists of only 2 dimensions, I suggest you to go to (https://archive.ics.uci.edu/ml/datasets) and download any of the Integer attribute type (because we are performing Linear Regression) dataset and split the data using methods introduced in Lab 1. Then compare the prediction accuracy with and without PCA on the training datasets.