Labsheet - 2

Linear Regression and Polynomial Curve Fitting

Machine Learning

BITS F464

I Semester 2021-22

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

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 sine wave for which lm() function is used in R.

Input Data:

Generate some data by using the function sin(2πx) in [0,1]

```
x = seq(0,1,length=11)
y = sin(2*pi*x)
Some noise is generated and added to the real signal (y):
noise = y + rnorm(11, sd=0.3)
#we will make y the response variable and x the predictor
#the response variable is usually on the y-axis
plot(x,noise,pch=19)
```

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

lm(formula,data)

```
#fit first degree polynomial equation:
lm1  <- lm(y~x)
#second degree
```

```
lm2 <- lm(y~poly(x,2)
```

Now try the model it with degree varying from 3 to 9 and plot the fits.
Write down your observations as degree increases from 1 to 9.

Now increase the number of training points, N as mentioned below for M=9
N=10
N=15
N=30
N=50
N=100
Write down your observations as N increases.

Overfitting Problem:

1) **Plot a graph of training error and testing error**

```
#getting the test and training errors
lms = list(lm1,lm2,lm3,lm4,lm5,lm6,lm7,lm8,lm9)
# set up two null vectors for the errors
train = test = rep(0,10) # train = vector(length=10) also works
# this is SSE/10
for(i in 1:10){train[i]=var(lms[i][[1]]$residuals)}
# now compute similarly scaled prediction error for test data
newy = sin(2*pi*x)+rnorm(11,sd=.3) # note we used the same x's
for(j in 1:10){test[j] = (1/10)*sum((newy-predict(lms[j]

[[1]],newdata=data.frame(x)))^2)}
lines (1:10,train, ylim=c(0,.5), col='black',lwd=3)
lines(1:10,test,pch="X", col='red',lwd=3)
cbind(train,test)
```

2) **Show all coefficients of the polynomial in a tabular form. And write down your observations.**

Coefficients are the intercept and the slope of the regression line, but more informative results about the model can find by:

```
#summary
summary(lm1)
#coefficient
C1 <- coef(lm1)
```

Use data frame for table generation

```
max_length <- max(c(length(C1), length(C2))) # Find out maximum
length for unequal table
```

```
#Now, we can use this information to create a data frame with NA
values at the bottom of each column that is shorter as the
maximum length.
data <- data.frame(col1 = c(C1,rep(NA, max_length - length(C1))),
                   col2 = c(C2,rep(NA, max_length - length(C2))))
```

Regularization:

Linear regression algorithm works by selecting coefficients for each independent variable that minimizes a loss function. However, if the coefficients are large, they can lead to over-fitting on the training dataset, and such a model will not generalize well on the unseen test data. To overcome this shortcoming, we'll do regularization, which penalizes large coefficients.

We will be using the glmnet() package to build the regularized regression models. The glmnet function does not work with dataframes, so we need to create a numeric matrix for the training features and a vector of target values.

```
lambdas <- 10^seq(3, -2, by = -.1)
fit <- glmnet(x, y, alpha = 0, lambda = lambdas)
```

Exercise

- Load “mtcar” dataset and split it into training and testing part. Plot a graph with various polynomial degree on training dataset. Then choose suitable degree for testing data.
- Apply ridge regularization by using glmnet function and plot the graph between lambda & error.