PRESIDENCY UNIVERSITY

NAME-SOUVIK SAHA

CLASS-MSc SEMESTER 4

CLASS ROLL NO-STAT 06

REGISTRATION NO-18414110005

EXAM ROLL NO-18414005

SUBJECT-DATA MINING

INTRODUCTION

We are working with the Auto dataset in R.The objective is to perform regression on mpg on horsepower using non-parametric procedures only.At the end we have tried to fit regression models by including other variables as covariates in a non-parametric way.

METHODOLOGY

We apply the following non-parametric regression procedures here:-

- KNN Regression
- Kernel Regression
- Local Linear Regression
- Polynomial Fitting
- Piecewise Polynomials
- Splines
- Generalized Additive Models

KNN Regression

KNN or K Nearest Neighbor regression is a special form of Linear Smoothers.A linear smoother is given by:-

$\hat{f(x)} = \sum_i y_i \widehat{w}(x_i, x)$.

In k-Nearest Neighbor regression :-

$\hat{w}(x_i, x)$={ $\frac{1}{k}$, if x_i is one of the closest neighbors of x

=0, otherwise

Here x=horsepower and y=mpg.So in KNN Regression first we find the k closest values of horsepower (x_i, i=1(1)k) with respect to a new value of horsepower (x).

Then we obtain the mean of the mpg values (y_i, i=1(1)k) corresponding to the k closest horsepower points and use it as an predicted mpg corresponding to the new value of horsepower(x) viz

$\hat{f(x)} = \frac{1}{k} \sum\limits_{i \epsilon N_k(x)} y_i$

Here $N_k(x)$ denotes the set of the k values nearest to the new horsepower (x).

This serves as a good estimator and achieves great flexibility with varying k. Since flexibility increases as we decrease k and vice versa and a small value of k makes the estimator more sensitive to the distance between a neighbor and the new value (x) an idea of an optimum value of k is necessary.We use n fold cross-validation to find the optimum value of k by minimising the test error.

The problem with KNN is k serves as the tuning parameter here and controls the degree of smoothness which is absurd as the degree of smoothness is expressed in terms of number of data points when it should be based on the range of the independent variable and use the entire training dataset.

Kernel Regression

The Kernel Regression is also known as Nadaraya-Watson regression.We start with a kernel function which satisfies the following properties:-

1) $K(x_i, x) > 0$

2) $K(x_i, x)$ depends on $(x_i - x)$ rather than x_i or x.In other words it depends on the distance between a horsepower value(x_i) and a new horsepower value (x) i.e it is also based on how close a horsepower value is with respect to the new horsepower value.

3)$\int xK(0, x)dx$=0

4)0<$\int x^2 K(0, x)dx$<∞

This implies if the distance between any horsepower value (x_i) and the new horsepower value (x) is very large then the kernel becomes close to 0 viz,

$K(x_i, x) \to 0$ as $|x_i - x| \to \infty$.

We can write the kernel as the joint function of $x_i$and x like $K(x_i, x) = K(x - x_i)$.Then the estimated wights are given by:-

$\hat{w}(x_i, x) = \frac{K(\frac{x-x_i}{h})}{\sum K(\frac{x-x_j}{h})}$ where h is the bandwidth.

We define the Nadaraya-Watson estimate of the regression function as:-

$\hat{f(x)} = \sum y_i \frac{K(\frac{x-x_i}{h})}{\sum K(\frac{x-x_j}{h})}$ i.e it is expressed as a weighted average of the mpg values corresponding to each horsepower value in the training set.

The Kernels we have used are:-

- The Boxchar Kernel : $K(x) = \frac{1}{2}I(x)$ where $I(x) = \begin{cases} 1 & ,|x| \leq 1 \\ 0 & ,|x| > 1 \end{cases}$
- The Gaussian Kernel: $K(x) = \frac{1}{\sqrt{2\pi}}^{e-\frac{x^2}{2}}$

Now for finding the optimum bandwidth we start with a grid of bandwidths and perform n-fold cross-validation to check how each generalises and choose the one with the lowest error.

We see unlike in KNN regression the Kernel regression uses the entire training set.But the variance increases as n increases and bias increases as any horsepower value(x_i) gets further away from x.

Local Linear Regression

We have a value for the horsepower of a car,say x_0 for which we want to predict the mpg.Then the local regression at X=$x_0$is obtained by:-

1)We gather the fraction s=$\frac{k}{n}$ of the training points whose horsepower values(x_i) are closest to x_0.

2)We assign a weight $K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.

3)Fit a weighted least squares regression of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta_0}$ and $\hat{\beta_1}$ that minimize $\sum_{i=1}^{n} K(x_i, x_0)(y_i - \beta_0 - \beta_1 x_i)^2$

4)The fitted value at x_0 is given by $f(\hat{x_0}) = \hat{\beta_0} + \hat{\beta_1} x_0$

This is similar to the way of prediction in a simple linear regression.Moreover the weights used are those of a Kernel i.e similar to Kernel regression as well.This serves better than dividing the range of horsepower values into bins and fitting linear regressions for each bin.

Polynomial Fitting

We have a value for the horsepower of a car,say x_0 for which we want to predict the mpg.We extend from linear to non-linear models and try to predict using polynomials.

The model is an extension to linear regression and looks like this:-

$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + . . . + \beta_d x_i^d + \epsilon_i,$

where $\epsilon_i$denotes the random error.

The fitted values are given by:-

$y_i = \hat{\beta_0} + \hat{\beta_1} x_i + \hat{\beta_2} x_i^2 + \hat{\beta_3} x_i^3 + . . . + \hat{\beta_d} x_i^d$

For large regression enough degree d, a polynomial regression allows us to produce an extremely non-linear curve.it is unusual to use d greater than 3 or 4 because for large values of d, the polynomial curve can become overly flexible and can take on some very strange shapes. This is especially true near the boundary of the horsepower variable.So this is not a good method.

Piecewise Polynomial Fitting

A piecewise polynomial function f(x) is obtained by dividing the domain of horsepower variable into contiguous intervals, and representing f by a separate polynomial in each interval i.e instead of fitting a high-degree polynomial over the entire range of horsepower, piecewise polynomial regression involves fitting separate low-degree polynomials over different regions of horsepower.

First we use a very special case of Piecewise Polynomials.They are known as Piecewise Constant Models.

We know horsepower is a continuous variable.We break the range of the horsepower into bins and fit constants in each bin.

This does the following:-

1)It helps to convert horsepower into some ordered categories.

2)It gets rid of the global structure imposed by polynomial regression.

The process is done in the following way:-

We create cutpoints in the range of the horsepower variable as $c_1, c_2, ..., c_k$and create (k+1) variables as follows:-

$C_0(X) = I(X < c_1)$

$C_1(X) = I(c_1 \leq X < c_2)$

$C_2(X) = I(c_2 \leq X < c_3)$

.

.

.

$C_{k-1}(X) = I(c_{k-1} \leq X < c_k)$

$C_k(X) = I(X \geq c_k)$

where I(.) is the indicator variable which takes value 1 if the condition is satisfied and 0 otherwise.

These variables are called Dummy Variables.An interesting fact which is observed is that for any value of horsepower $\sum\limits_{i=0}^{k} C_i(X) = 1$,since any value of horsepower must be in exactly one of the k + 1 intervals. We then use least squares to fit a linear model using $C_1(X)$, $C_2(X)$, . . ., $C_k(X)$ as predictors as,

$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + . . . + \beta_k C_k(x_i) + \epsilon_i$

For a given value of horsepower(X), at most one of C_1, C_2, . . ., C_k can be non-zero.Note that when X $<$ c_1 , all of the predictors in the above equation are zero, so β_0 can be interpreted as the mean value of mpg for the values of horsepower $<$ c_1 . By comparison, the equation predicts a response of $\beta_0 + \beta_j$ for c_j $\leq$X $<$$c_j$ +1, so β_j represents the average increase in the response for values of horsepower in c_j $\leq$X $<$$c_j$+1 relative to the values of horsepower when X $<$$c_1$.

Splines

A q^{th}-order spline f is a piecewise polynomial function of degree q that is continuous and has continuous derivatives of orders 1,2,..., q-1; at its knot points. Specifically, there are knots t_1 $<$ t_2 $<$ ...$<$ t_k such that f is a polynomial of degree q on each of the intervals (1, t_1]; [t_1,t_2],...., [t_k, ∞) and $f^{(i)}$ is continuous at t_1,t_2,...,$t_k$for each j = 0,1,....,q-1.

We use the following splines to regress mpg on horsepower.

1)Splines with a plus function basis

2)Regression Splines

3)Natural Splines

4)B-Splines

5)Smoothing Splines

Splines with a plus function basis

This is a way of parametrization of splines.It is defined as:-

$$x_+^q = \begin{cases} x^q & x > 0 \\ 0 & otherwise \end{cases}$$

Then

$f(x) = a_0 + a_1x + a_2x^2 .... + a_qx^q + \sum_{l=1}^{k} a_{q+1}(x - t_l)_+^q$

Here the terms are polynomous and there are q + 1 + k independent parameters.Hence estimation will not involve constraints.

Regression Splines

Here we consider the values of horsepower viz x_1, x_2, ..., x_n and the values of mpg y_1,y_2,...,$y_n$we fit functions f which are $(q+1)^{th}$order splines with knots at some chosen locations t_1,t_2,...,t_k.

We express f as:-

$f(x) = \sum_{j=1}^{q+k+1} \beta_j g_j(x)$

where β_1, β_2,...β_{q+k+1} are the coefficients and g_1, g_2,......,g_{q+k+1} are the basis functions for order (q+1) splines over the knots t_1,t_2,...,t_k.

Let y=(y_1, y_2, ...,y_n)$\epsilon\mathbb{R}^n$ and the basis function matrix $G\epsilon\mathbb{R}_{n\times q+k+1}$by G=(($G_{ij}$)) defined by $G_{ij} = g_j(x_i)$.Then we can just use the least square method to determine the optimal coefficients $\hat{\beta} = (\hat{\beta_1}, \hat{\beta_2}, ..., \hat{\beta_{q+k+1}})$ as

$\hat{\beta}= \underset{\beta\epsilon\mathbb{R}^{\prime}}{argmin} \; ||y - G\beta||_2^2$

This gives us the fitted regression spline as $\hat{f(x)} = \sum_{j=1}^{q+k+1} \hat{\beta}_j g_j$.

Natural Splines

Regression Splines may result in high variance at the boundaries and pose many problems with increase in q. Natural splines are a natural remedy to this problem. They force the piecewise polynomial function to have a lower degree to the left of the leftmost knot, and to the right of the rightmost knot.A natural spline of order (q+1) with knots at $t_1 < t_2 <$ ...$< t_k$ is a piecewise polynomial function f such that:-

1)f is a polynomial of degree q at each $[t_1, t_2]$,.....,$[t_{k-1}, t_k]$.

2)f is a polynomial of degree $\frac{(q-1)}{2}$ on (-∞, t_1] and [t_k, ∞).

3)f is continuous and has continuous derivatives of order 1,2,....,q-1 at t_1,t_2,...,t_k.

B-Splines

B-spline basis representation tends to be the most numerically stable representation of a spline but intuition is lost.

It is defined as;-

$$t_l = \begin{cases} a & l < 1 \\ b & l > k \end{cases}$$

$$\text{Set } B_{l1}(x) = \begin{cases} I(t_l \leq x < t_{l+1} & l = 0(1)k \\ 0 & l > 0 or l < k \end{cases}$$

Now $\sum_{l=0}^{k} \beta_l B_{l1}(x)$ is the B-spline of 1st order.B-splines of higher order are defined through a recursive relationship.

$B_{lr}(x) = \frac{(x-t_l)}{(t_{l+r+1}-t_l)} B_{l(r-1)}(x) + \frac{(t_{l+r}-x)}{(t_{l+r}-t_{l+1})} B_{(l+1)(r-1)}(x)$

where l=1-r,....,k and r=q+1 is the order of the spline.

The r^{th} order spline is given by:-

$f(x) = \sum_{l=1-r}^{k} \beta_l B_{lr}(x)$

Smoothing Splines

This is a direct method of smoothing the approximation $\hat{f}$ of the regression function of the data points of mpg and horsepower.

It is done by minimising the spline objective function:-

$\mathcal{L}(m,\lambda) = \frac{1}{n}\sum_{i=1}^{n}(y_i - m(x_i))^2 + \lambda \int (m''(x))^2 dx$

The first term is the MSE of m(x) to predict y and the second term denotes the curvature of m at x.It is like adding a penalty to the MSE criterion and choosing the one with less curvature when two have same MSE.

The smoothing spline is the curve which minimises the objective function.

$\hat{f_\lambda} = \underset{m}{argmin} \mathcal{L}(m,\lambda)$

Generalized Additive Models

Suppose we need to include any other variable as covariate along with horsepower and perform regression in a non-parametric way.Then we can use Additive models for the purpose.

Instead of considering a full p-dimensional function of the form:-

$f(x) = f(x_1, x_2, ..., x_p)$

we restrict ourselves to functions of the form

$f(x) = f_1(x_1) + f_2(x_2) + ... + f_p(x_p)$

Instead of fitting the full non-parametric model we fit the additive version where each $f_j$is an univariate fitting an additive model.

This is useful since we can get a good approximation and capture interesting features in high dimensions.

Now the various functions are estimated by Backfitting Algorithm by taking the partial residuals into consideration.

This is an iterative procedure where we choose our favourite smoother and cycle through estimating each function.

This means once our univariate smoother has been chosen, we initialize $\hat{f}_1, \hat{f}_2, ..., \hat{f}_p$ (say, to all to zero) and cycle over the following steps:-

1)For j=1,2,....,p

a)define $r_i = y_i - \sum_{k \neq j} f_k(\hat{x}_{ik)}$

b)Smooth $\hat{f}_j$= Smooth(x_j, r)

c)Center $\hat{f}_j = \hat{f}_j - \frac{1}{n}\sum_{i=1}^{n} \hat{f_j}(x_{ij})$

In last step above, we are removing the mean from each fitted function to make the model identifiable. Then the fitted model is

$f(\hat{x}) = \bar{y} + \hat{f}_1(x_1) + ... + \hat{f}_p(x_p)$

OUTPUT

First let us see the description of the dataset Auto.

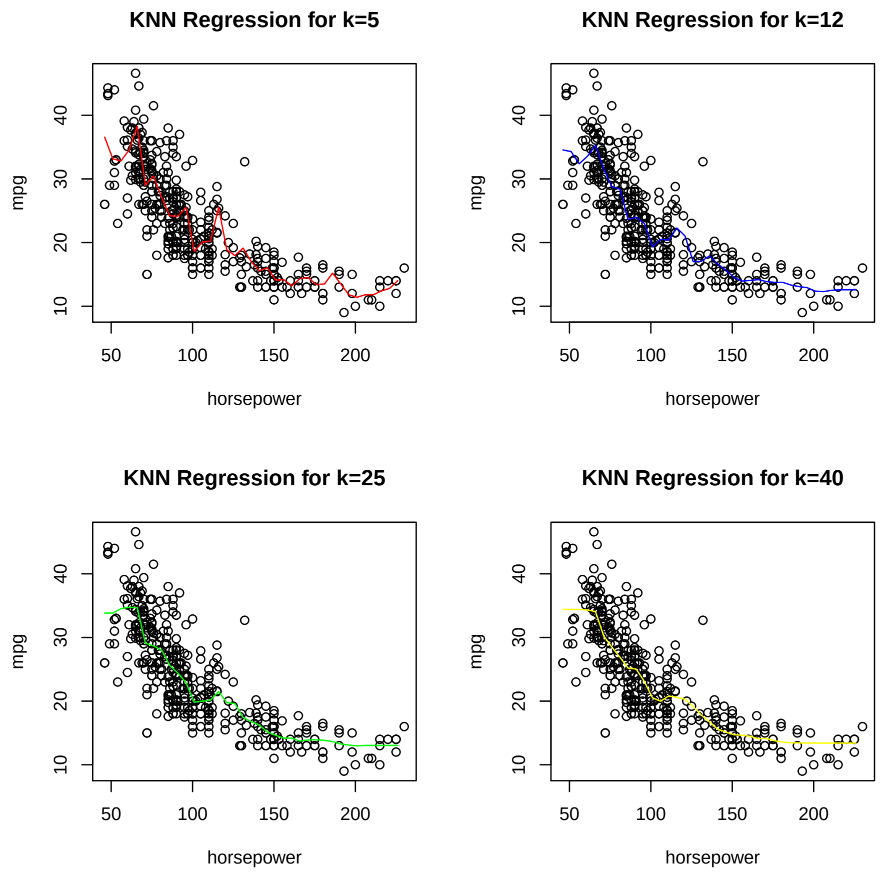
```
'data.frame': 392 obs. of  9 variables:
 $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
$ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
$ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
$ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
$ weight      : num  3504 3693 3436 3433 3449 ...
$ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
$ year        : num  70 70 70 70 70 70 70 70 70 70 ...
$ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
$ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54
```

Comments:-

The dataset consists of 392 observations out of 9 variables.There are 8 quantitative and one qualitative variable.

Lets work with KNN regression.

```
Warning:  package 'FNN' was built under R version 4.0.2
```

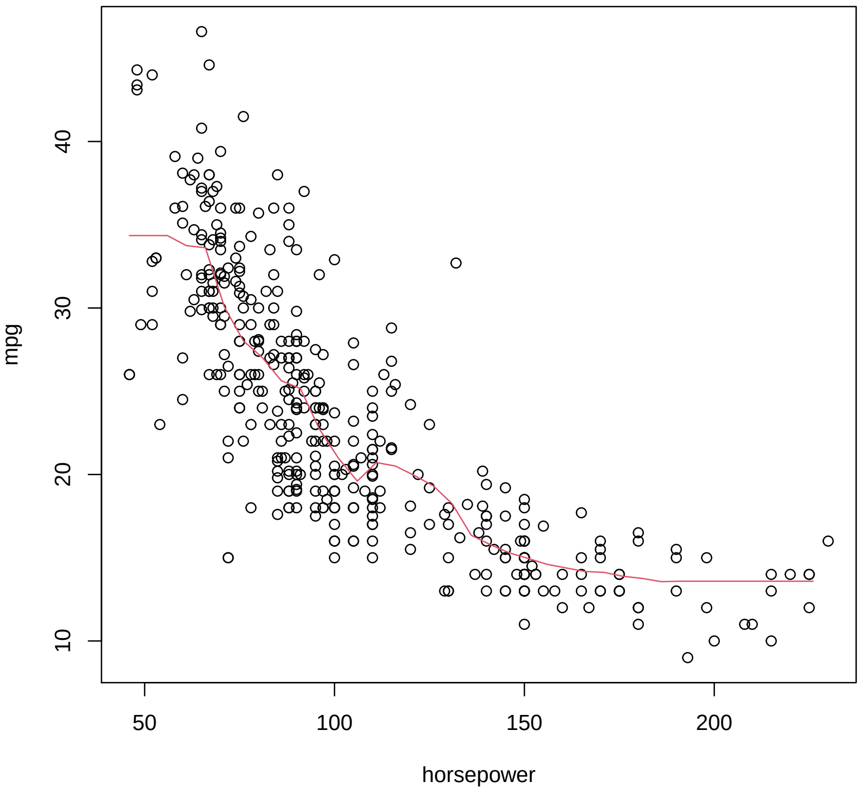


Comments:-

We fit KNN for k=5,12,25,40. We see the degree of smoothness increases with increase in k.

Now to choose an optimum value of k we perform 10 fold cross-validation and perform a KNN with the optimum value of k.

```
[1] 47
```



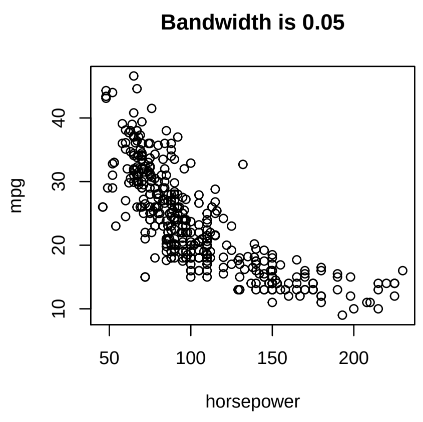
Comments:-

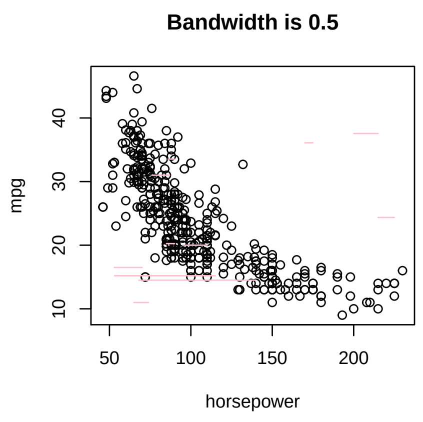
We see that the graph shows considerable smoothing at the optimum value of k.

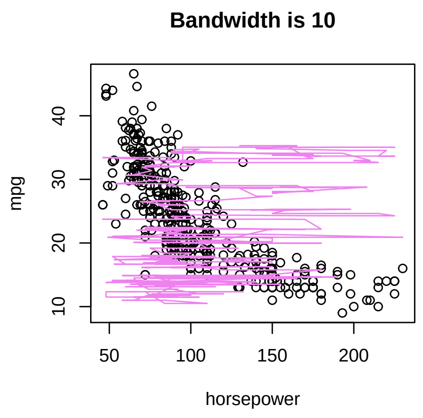
This method is not satisfactory since with optimum k the smoothing is not perfect and fails to catch the entire data.

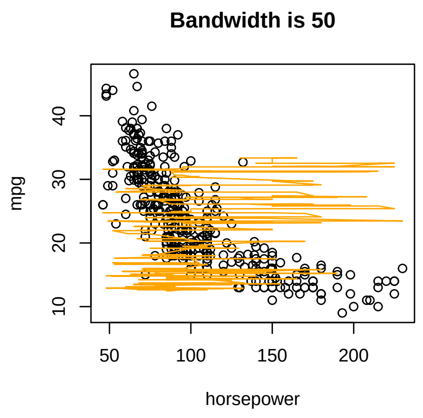
So lets see how Kernel regression performs.

```
[1] "x" "y"
```





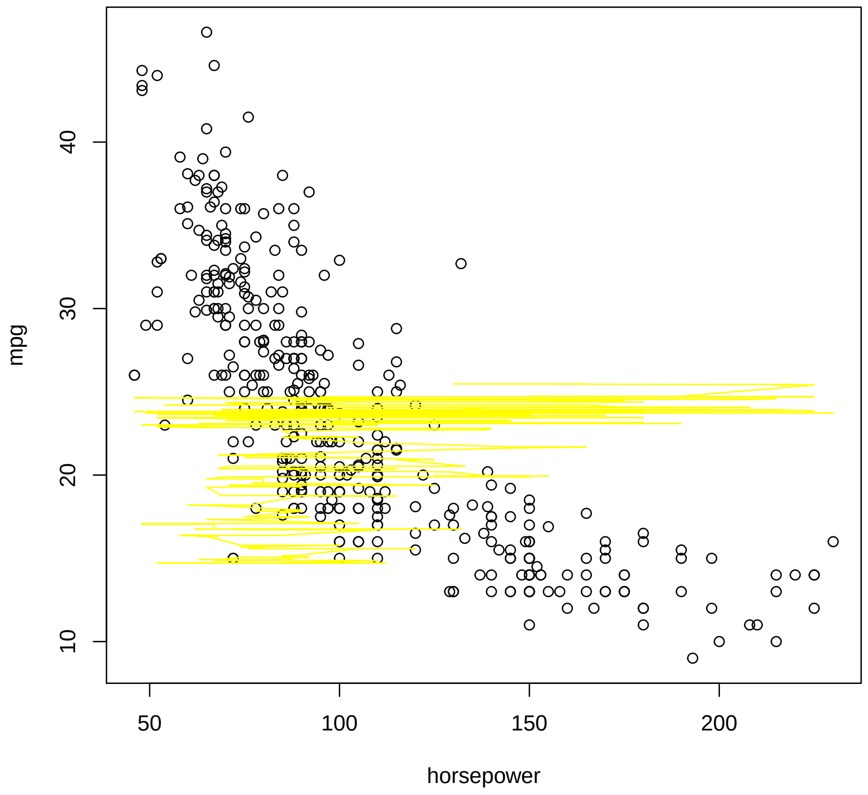




Comment:-

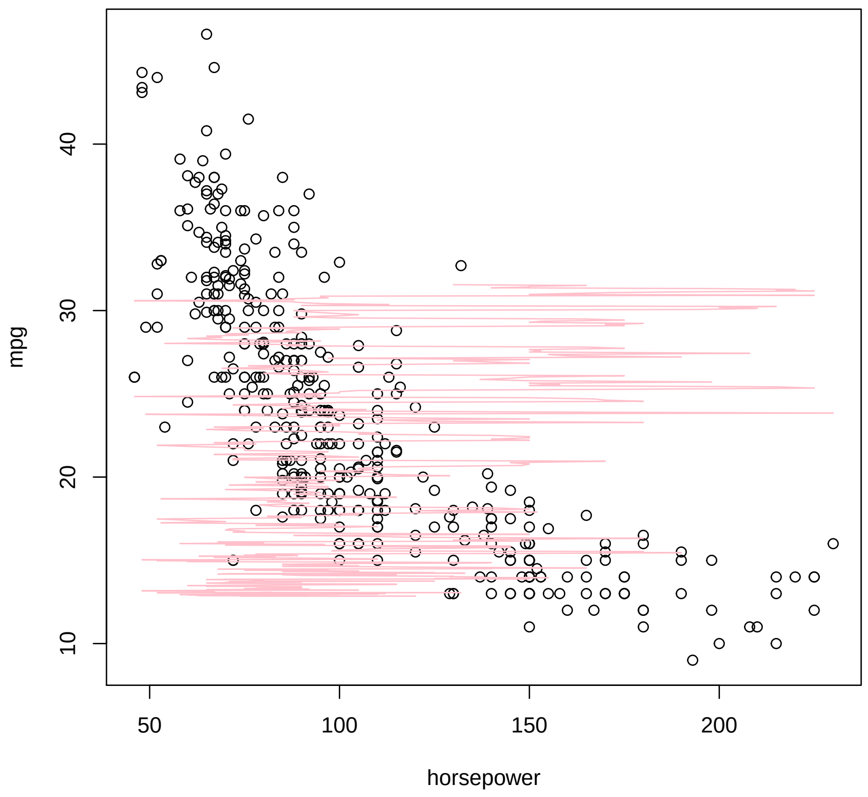
As bandwidth increases the smoothing increases.
Hence we find an optimum value of bandwidth by 10 fold cross-validation.

```
[1] 200
```

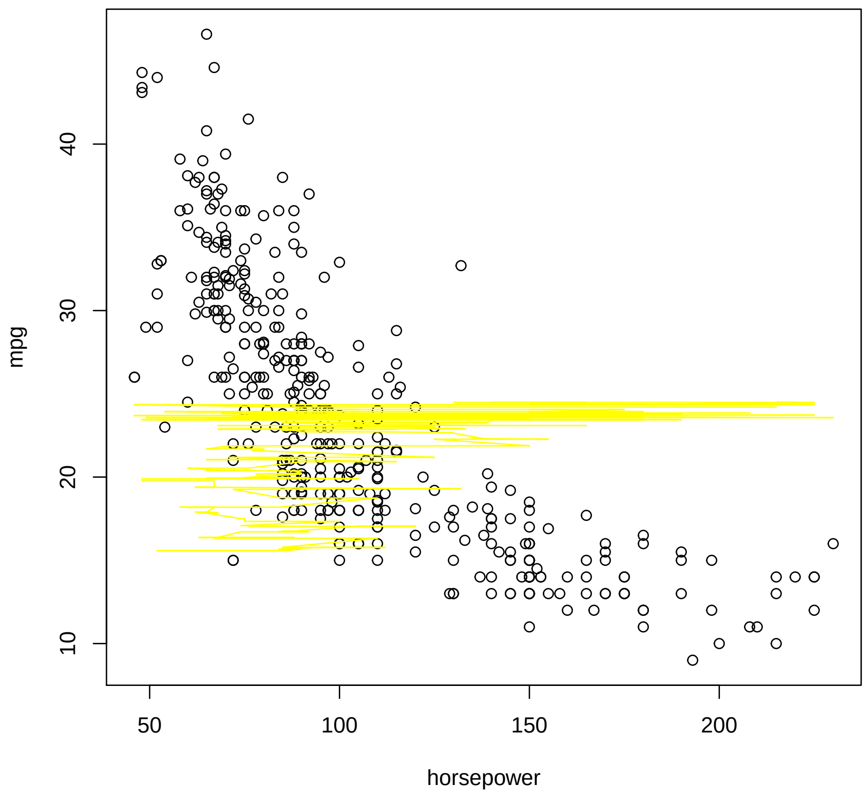


Comments:-

We see the data is smoothed at the optimum value of the bandwidth.
Now we see for the normal kernel.



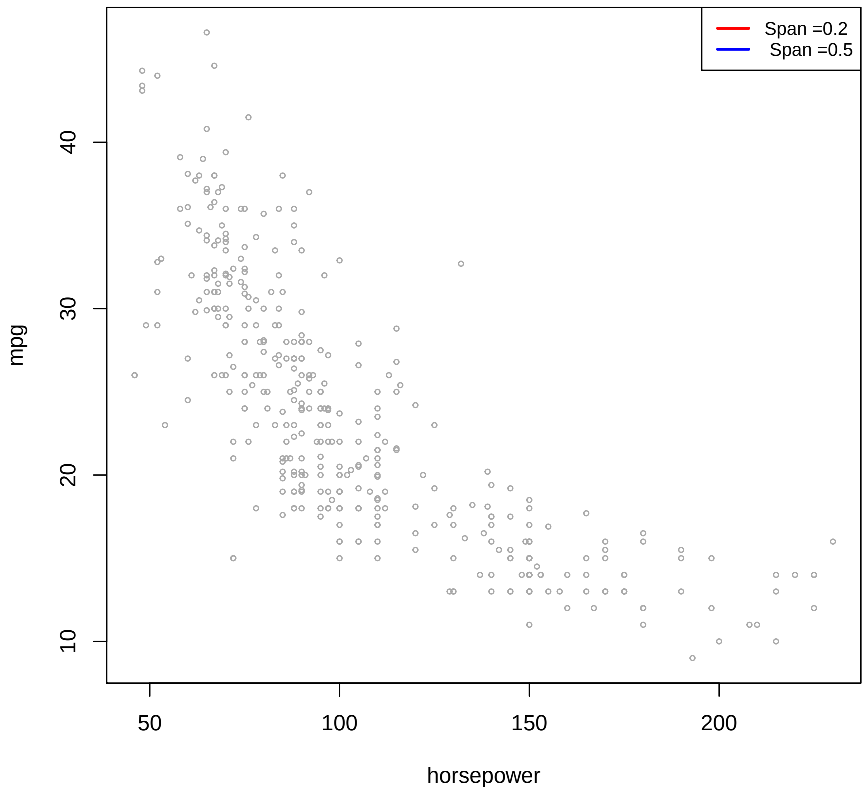
```
[1] 230
```



<u>Comments:-</u>

We see there is considerable amount of smoothing but still it is not perfect although there is a slight improvement over KNN.

So we proceed to Local Linear Regression.



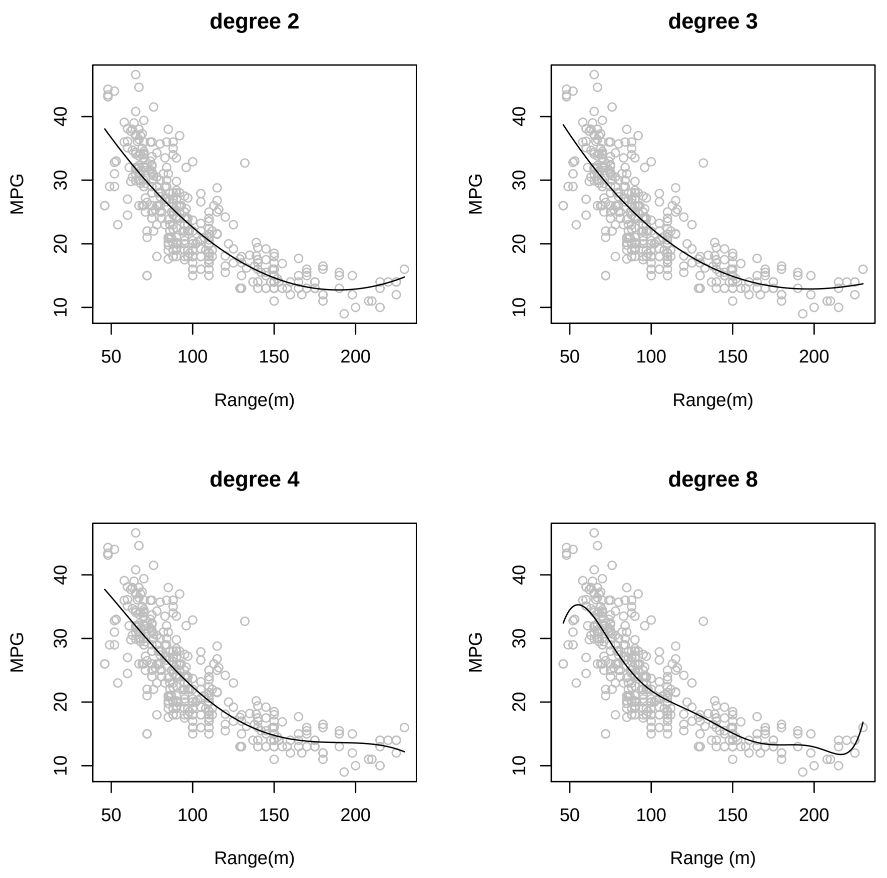
Comments:-

We can see a considerable improvement and the span with 0.2 fits better than 0.5

Lets try polynomial fitting with the Auto data.

```
The following objects are masked from Auto (pos = 4):
       acceleration, cylinders, displacement, horsepower, mpg, name,
origin, weight, year

'data.frame': 392 obs. of  9 variables:
 $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
 $ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
 $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
 $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
 $ weight      : num  3504 3693 3436 3433 3449 ...
 $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
 $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
$ origin       : num  1 1 1 1 1 1 1 1 1 1 ...
$ name         : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54
```



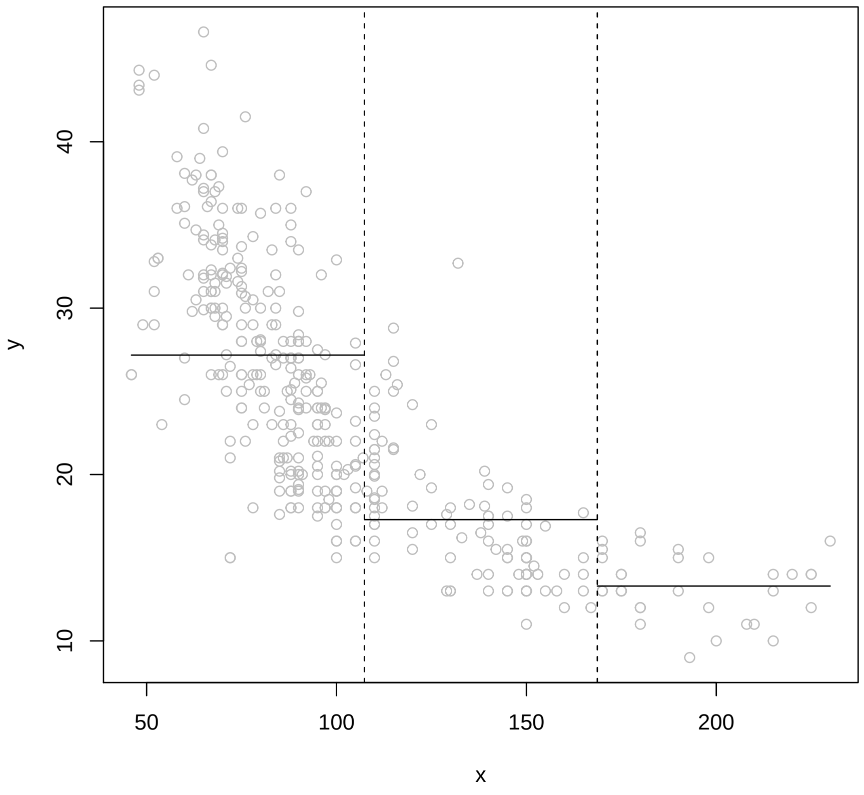
Comments:-

We have fitted four polynomials of degrees 2,3,4,8.The quadratic and cubic polynomials fitted very badly while the degree 8 polynomial had improved but still not satisfactory as it had wiggles at the high end of the horsepower variable .

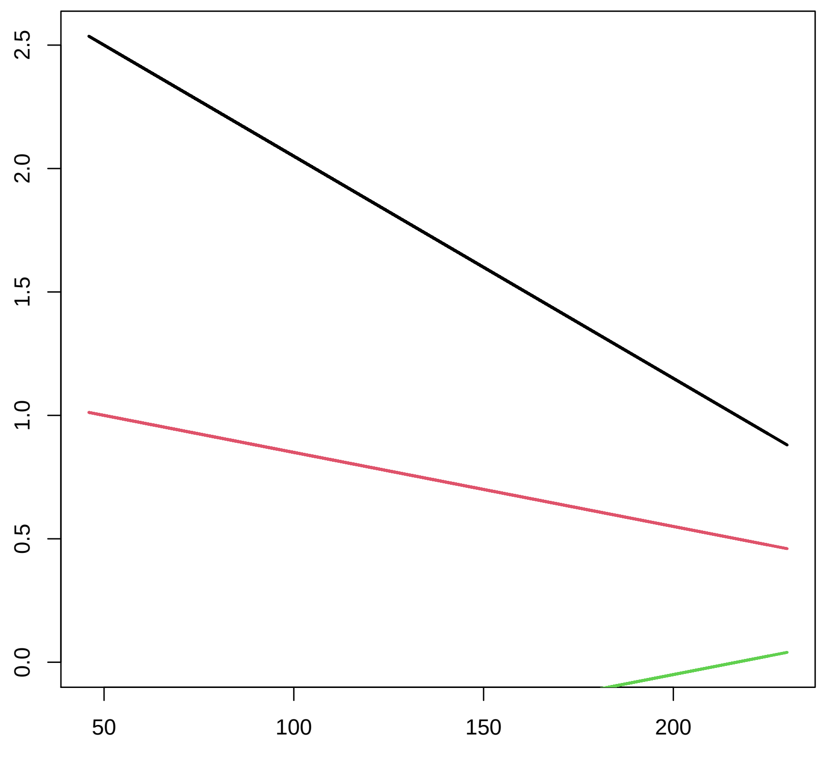
Lets try piecewise polynomial models.We fit piecewise constant models using least squares with three pieces in each case.

```r
y <-mpg
x <-horsepower
xi1 <- min(x) + (range(x)[2]-range(x)[1])/3
xi2 <- min(x) + 2*(range(x)[2]-range(x)[1])/3
xtruncflat1 <- ifelse(x>xi1,1,0)
xtruncflat2 <- ifelse(x>xi2,1,0)
```

```
pieceflat <- lm(y~1+xtruncflat1+xtruncflat2)
plot(x,y,ylim=c(min(y),max(y)),ylab="y",type="n")
points(x,y,col="grey")
segments(x0=min(x),x1=xi1,y0=pieceflat$coeff[1],y1=pieceflat$coeff[1])
segments(x0=xi1,x1=xi2,y0=pieceflat$coeff[1]+pieceflat$coeff[2],
         y1=pieceflat$coeff[1]+pieceflat$coeff[2])
segments(x0=xi2,x1=max(x),y0=pieceflat$coeff[1]+pieceflat$coeff[2]+
           pieceflat$coeff[3],y1=pieceflat$coeff[1]+pieceflat$coeff[2]+pieceflat$coeff[3])
abline(v=xi1,lty=2)
abline(v=xi2,lty=2)
```

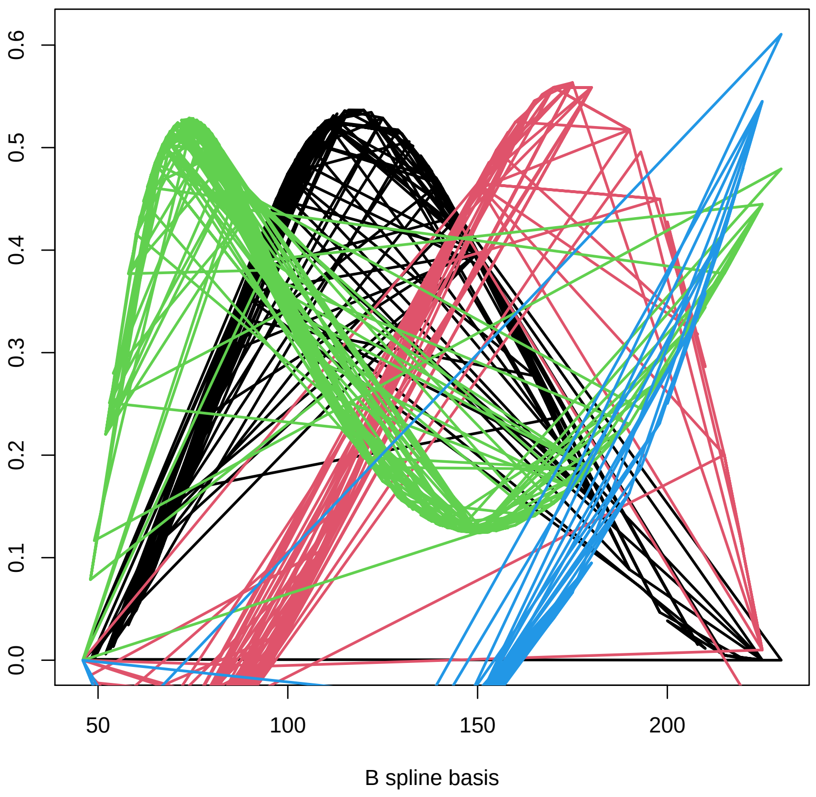


Comments:-
This was a good fit and a motivation towards spline regression.
We proceed to fit natural splines.



Natural Cubic spline basis

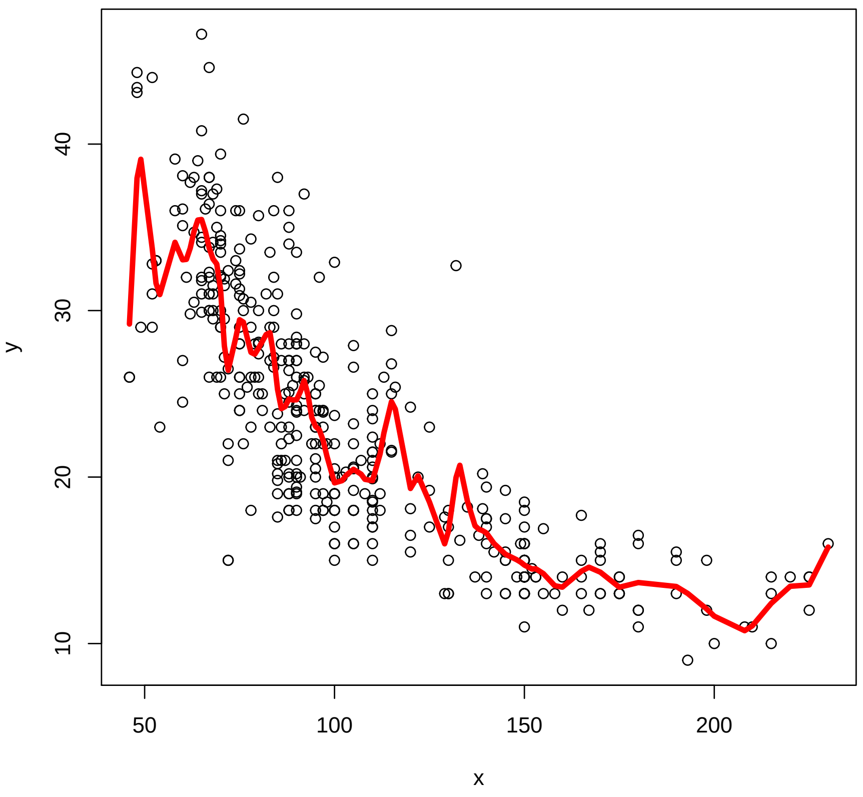
Lets try B-splines.



We proceed with smoothing splines.

```
Call:
smooth.spline(x = horsepower, y = mpg)

Smoothing Parameter  spar= 0.2648644  lambda= 5.101567e-07 (12 iterations)
Equivalent Degrees of Freedom (Df): 42.14987
Penalized Criterion (RSS): 1117.657
GCV: 18.64132
```



For modifying regression models to include other co-variates in non-parametric way we work with additive models now.

```
This is mgcv 1.8-31.  For overview type 'help("mgcv-package")'.
Error in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):
A term has fewer unique covariate combinations than specified maximum
degrees of freedom
```

Comments:-

We see that there is an error.So we have to take a subset and then work with additive models.

```
Family: gaussian
Link function: identity

Formula:
```

```
mpg ~ s(displacement) + s(horsepower) + s(weight)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  23.4459     0.1918   122.2   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                  edf Ref.df      F  p-value
s(displacement) 4.201  5.236  4.438 0.000509 ***
s(horsepower)   5.888  7.044 10.771 1.41e-12 ***
s(weight)       1.000  1.000 20.194 9.22e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.763   Deviance explained =   77%
GCV = 14.884  Scale est. = 14.425    n = 392

Family: gaussian
Link function: identity

Formula:
mpg ~ s(displacement) + s(horsepower) + s(weight) + s(acceleration)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  23.4459     0.1908   122.9   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                  edf Ref.df      F  p-value
s(displacement) 4.165  5.191  4.646 0.000338 ***
s(horsepower)   5.510  6.671 10.353 1.83e-11 ***
s(weight)       1.000  1.000  5.188 0.023284 *
s(acceleration) 1.000  1.000  6.274 0.012664 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.766   Deviance explained = 77.3%
GCV =  14.74  Scale est. = 14.264    n = 392
```

Comments:-

Both the models perform well but the second models performs slightly better than the former.

CODES:-

The R Codes are:-

```
library(ISLR)
attach(Auto)
str(Auto)
#KNN
require(FNN,quiet=T)
l=seq(min(horsepower),max(horsepower),5)
a=data.frame(l)
knn1=knn.reg(train=horsepower,test=a,y=mpg,k=5)
knn2=knn.reg(train=horsepower,test=a,y=mpg,k=12)
knn3=knn.reg(train=horsepower,test=a,y=mpg,k=25)
knn4=knn.reg(train=horsepower,test=a,y=mpg,k=40)
par(mfrow=c(2,2))
plot(horsepower,mpg,main="KNN Regression for k=5")
lines(l,knn1$pred,col="red")
plot(horsepower,mpg,main="KNN Regression for k=12")
lines(l,knn2$pred,col="blue")
plot(horsepower,mpg,main="KNN Regression for k=25")
lines(l,knn3$pred,col="green")
plot(horsepower,mpg,main="KNN Regression for k=40")
lines(l,knn4$pred,col="yellow")
#Finding k by cross validation
cvknn<-function(k,n){
error=NULL
for(i in 1:length(k))
{
test.error=NULL
folds=split(sample(1:n),ceiling(seq_along(sample(1:n))/(n/10)))
for(j in 1:10)
{
model=knn.reg(train=horsepower[-folds[[j]]],test=data.frame(horsepower[folds[[j]]]),y=mpg[-f
ter=sum((mpg[folds[[j]]]-model$pred)^2)
test.error=c(test.error,ter)
}
error=c(error,mean(test.error))
}
a=k[which(error==min(error))]
return(a)
}
k=seq(2,200,by=5)
kopt=cvknn(k,nrow(Auto))
kopt
#Fitting with the optimum k
knnopt=knn.reg(train=horsepower,test=a,y=mpg,k=kopt)
```

```r
plot(horsepower,mpg)
lines(l,knnopt$pred,col=578,main="KNN with optimum k by Cross-Validation")
#Kernel Smoothing
ks1=ksmooth(horsepower,mpg,"box",bandwidth=0.05)
names(ks1)
ks2=ksmooth(horsepower,mpg,"box",bandwidth=0.5)
ks3=ksmooth(horsepower,mpg,"box",bandwidth=10)
ks4=ksmooth(horsepower,mpg,"box",bandwidth=50)
par(mfrow=c(2,2))
plot(horsepower,mpg,main="Bandwidth is 0.05")
lines(horsepower,ks1$y,col="blue")
plot(horsepower,mpg,main="Bandwidth is 0.5")
lines(horsepower,ks2$y,col="pink")
plot(horsepower,mpg,main="Bandwidth is 10")
lines(horsepower,ks3$y,col="violet")
plot(horsepower,mpg,main="Bandwidth is 50")
lines(horsepower,ks4$y,col="orange")
#Finding the bandwidth h by cross validation
cvkernel<-function(h,n){
error2=NULL
for(i in 1:length(h))
{
folds=split(sample(1:n),ceiling(seq_along(sample(1:n))/(n/10)))
test.error2=NULL
for(j in 1:10)
{
model=ksmooth(horsepower[-folds[[j]]],mpg[-folds[[j]]],"box",bandwidth=h[i],x.points=horsepo
ter=mean((mpg[folds[[j]]]-model$y)^2)
test.error2=c(test.error2,ter)
}
error2=c(error2,mean(test.error2))
}
return(h[which.min(error2)])
}
h=seq(10,300,by=10)
hopt=cvkernel(h,nrow(Auto))
hopt
#Fitting with the optimum bandwidth
ksmod=ksmooth(horsepower,mpg,"box",bandwidth=hopt)
plot(horsepower,mpg)
lines(horsepower,ks$y,col="yellow")
#For normal
ks2=ksmooth(horsepower,mpg,"normal",bandwidth=50)
plot(horsepower,mpg)
lines(horsepower,ks2$y,col="pink")
```

```
#Finding the bandwidth h by cross validation
cvkernel2<-function(h,n){
error3=NULL
for(i in 1:length(h))
{
folds=split(sample(1:n),ceiling(seq_along(sample(1:n))/(n/10)))
test.error3=NULL
for(j in 1:10)
{
model=ksmooth(horsepower[-folds[[j]]],mpg[-folds[[j]]],"normal",bandwidth=h[i],x.points=hors
ter3=mean((mpg[folds[[j]]]-model$y)^2)
test.error3=c(test.error3,ter3)
}
error3=c(error3,mean(test.error3))
}
b=h[which(error3==min(error3))]
return(b)
}
h=seq(10,300,by=10)
hopt2=cvkernel2(h,nrow(Auto))
hopt2
#Kernel Smoothing
ksmod=ksmooth(horsepower,mpg,"box",bandwidth=hopt2)
plot(horsepower,mpg)
lines(horsepower,ks$y,col="yellow")
#Local Linear Regression
hlims =range(horsepower)
l=seq(from=hlims[1],to=hlims[2],length=392)
a=data.frame(l)
plot(horsepower,mpg,xlim=hlims,cex =.5,col =" darkgrey ")
fit1=loess(horsepower~mpg,span=0.2,data=Auto)
fit2=loess(horsepower~mpg,span=0.5,data=Auto)
lines(l,predict(fit1,a),col ="red ",lwd =2)
lines(l,predict(fit2,a),col =" blue",lwd =2)
legend("topright",legend =c("Span =0.2" ," Span =0.5") ,
        col=c("red "," blue "),lty =1, lwd =2, cex =.8)
#Polynomial Fitting
par(mfrow=c(2,2))
xvals <- seq(min(horsepower),max(horsepower),.1)
l2 <- lm(mpg~poly(horsepower,deg=2,raw=T))
l3 <- lm(mpg~poly(horsepower,deg=3,raw=T))
l4 <- lm(mpg~poly(horsepower,deg=4,raw=T))
l8 <- lm(mpg~poly(horsepower,deg=8,raw=T))
plot(mpg~horsepower,xlab="Range(m)",ylab="MPG",col="grey",main="degree 2")
lines(xvals,l2$coeff[1]+l2$coeff[2]*xvals+l2$coeff[3]*xvals*xvals)
```

```r
plot(mpg~horsepower,xlab="Range(m)",ylab="MPG",col="grey",main="degree 3")
lines(xvals,l3$coeff[1]+l3$coeff[2]*xvals+l3$coeff[3]*xvals*xvals+
        l3$coeff[4]*xvals*xvals*xvals,lty=1)
plot(mpg~horsepower,xlab="Range(m)",ylab="MPG",col="grey",main="degree 4")
lines(xvals,l4$coeff[1]+l4$coeff[2]*xvals+l4$coeff[3]*xvals*xvals+
        l4$coeff[4]*xvals*xvals*xvals+l4$coeff[5]*xvals*xvals*xvals*xvals,lty=1)
plot(mpg~horsepower,xlab="Range (m)",ylab="MPG",col="grey",main="degree 8")
lines(xvals,l8$coeff[1]+l8$coeff[2]*xvals+l8$coeff[3]*xvals^2+
        l8$coeff[4]*xvals^3+l8$coeff[5]*xvals^4+l8$coeff[6]*xvals^5+
        l8$coeff[7]*xvals^6+l8$coeff[8]*xvals^7+l8$coeff[9]*xvals^8,lty=1)
#Piecewise Constant Models
y <-mpg
x <-horsepower
xi1 <- min(x) + (range(x)[2]-range(x)[1])/3
xi2 <- min(x) + 2*(range(x)[2]-range(x)[1])/3
xtruncflat1 <- ifelse(x>xi1,1,0)
xtruncflat2 <- ifelse(x>xi2,1,0)
pieceflat <- lm(y~1+xtruncflat1+xtruncflat2)
plot(x,y,ylim=c(min(y),max(y)),ylab="y",type="n")
points(x,y,col="grey")
segments(x0=min(x),x1=xi1,y0=pieceflat$coeff[1],y1=pieceflat$coeff[1])
segments(x0=xi1,x1=xi2,y0=pieceflat$coeff[1]+pieceflat$coeff[2],
         y1=pieceflat$coeff[1]+pieceflat$coeff[2])
segments(x0=xi2,x1=max(x),y0=pieceflat$coeff[1]+pieceflat$coeff[2]+
           pieceflat$coeff[3],y1=pieceflat$coeff[1]+pieceflat$coeff[2]+pieceflat$coeff[3])
abline(v=xi1,lty=2)
abline(v=xi2,lty=2)
#Splines
#Natural Splines
require(splines,quiet=T)
not=c(50,100,200)
spl=ns(horsepower,knots=not,Boundary.knots=c(250,350))
plot(spl[,1]~horsepower, ylim=c(0,max(spl)), type='l',
     lwd=2, col=1,xlab="Natural Cubic spline basis", ylab="")
for (j in 2:ncol(spl)) lines(spl[,j]~horsepower, lwd=2, col=j)
#Basis Splines
not=c(50,100,200)
spl=ns(horsepower,knots=not)
plot(spl[,1]~horsepower, ylim=c(0,max(spl)), type='l',
     lwd=2, col=1,xlab="B spline basis", ylab="")
for (j in 2:ncol(spl)) lines(spl[,j]~horsepower, lwd=2, col=j)
#Smoothing Splines
model=smooth.spline(horsepower,mpg)
model
plot(x,y)
```

```
lines(model,col="red",lwd=4)
#Generalized Additive Models
require(mgcv,quiet=T)
calif.auto<-gam(mpg~s(displacement)+s(horsepower)+ s(weight)+s(acceleration)+s(cylinders)+s(
calif.auto1<-gam(mpg~s(displacement)+s(horsepower)+ s(weight), data=Auto)
summary(calif.auto1)
calif.auto2<-gam(mpg~s(displacement)+s(horsepower)+ s(weight)+s(acceleration), data=Auto)
summary(calif.auto2)
```