Project Report

ID: 20620838

Gurpreet Gosal

0.1 Data Description

The Computer Activity databases are a collection of computer systems activity measures. The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department. Users would typically be doing a large variety of tasks ranging from accessing the internet, editing files or running heavy cpu-oriented programs. The data was collected continuously which was system activity gathered every 5 seconds.

System measures (variates) used:

1. lread - Reads (transfers per second) between system memory and user memory.
2. lwrite - writes (transfers per second) between system memory and user memory.
3. scall - Number of system calls of all types per second.
4. sread - Number of system read calls per second.
5. swrite - Number of system write calls per second .
6. fork - Number of system fork calls per second.
7. exec - Number of system exec calls per second.
8. rchar - Number of characters transferred per second by system read calls.
9. wchar - Number of characters transfreed per second by system write calls.
10. runqsz - Process run queue size.
11. freemem - Number of memory pages available to user processes.
12. freeswap - Number of disk blocks available for page swapping.
13. usr - Portion of time (%) that cpus run in user mode.

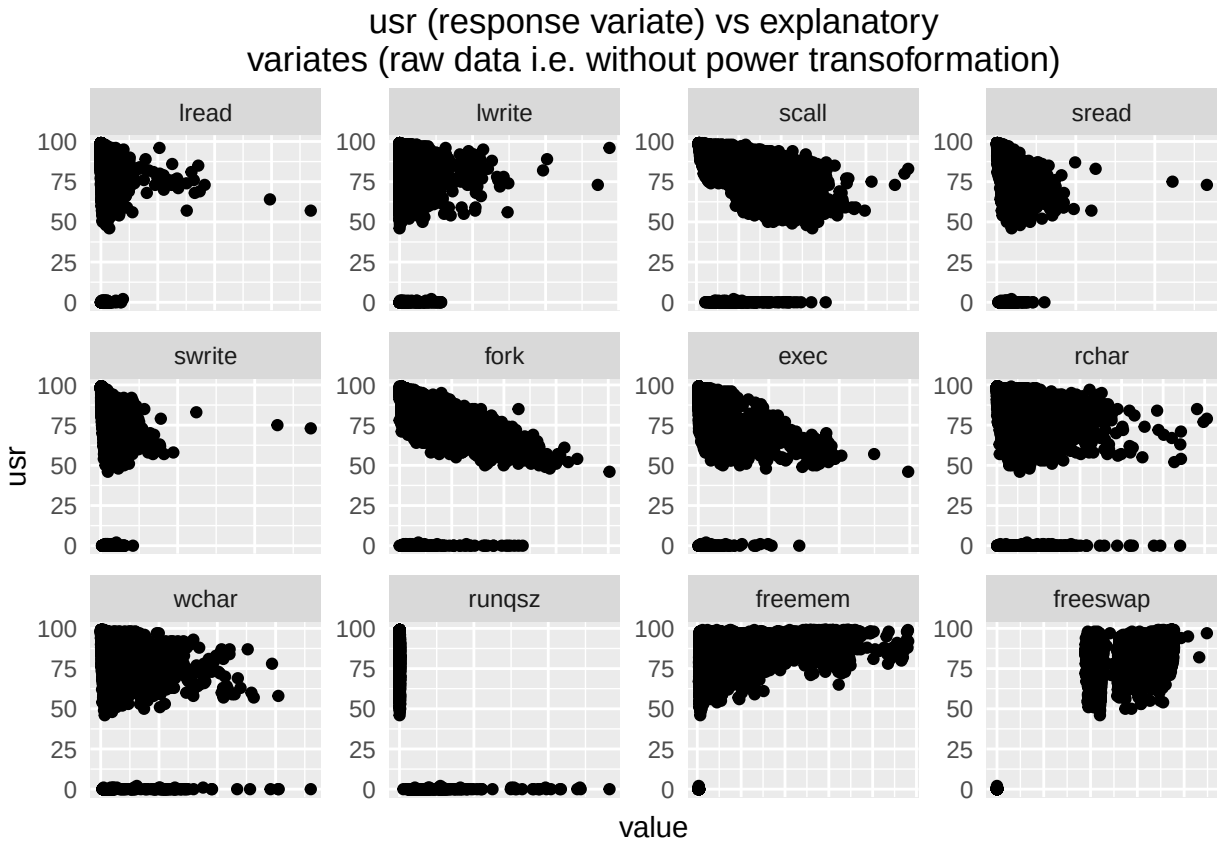
The goal of regression/ function estimation is to predict the attribute **usr** i.e. the portion of time that CPU runs in user mode using attributes 1-12.

0.2 Exploration of data

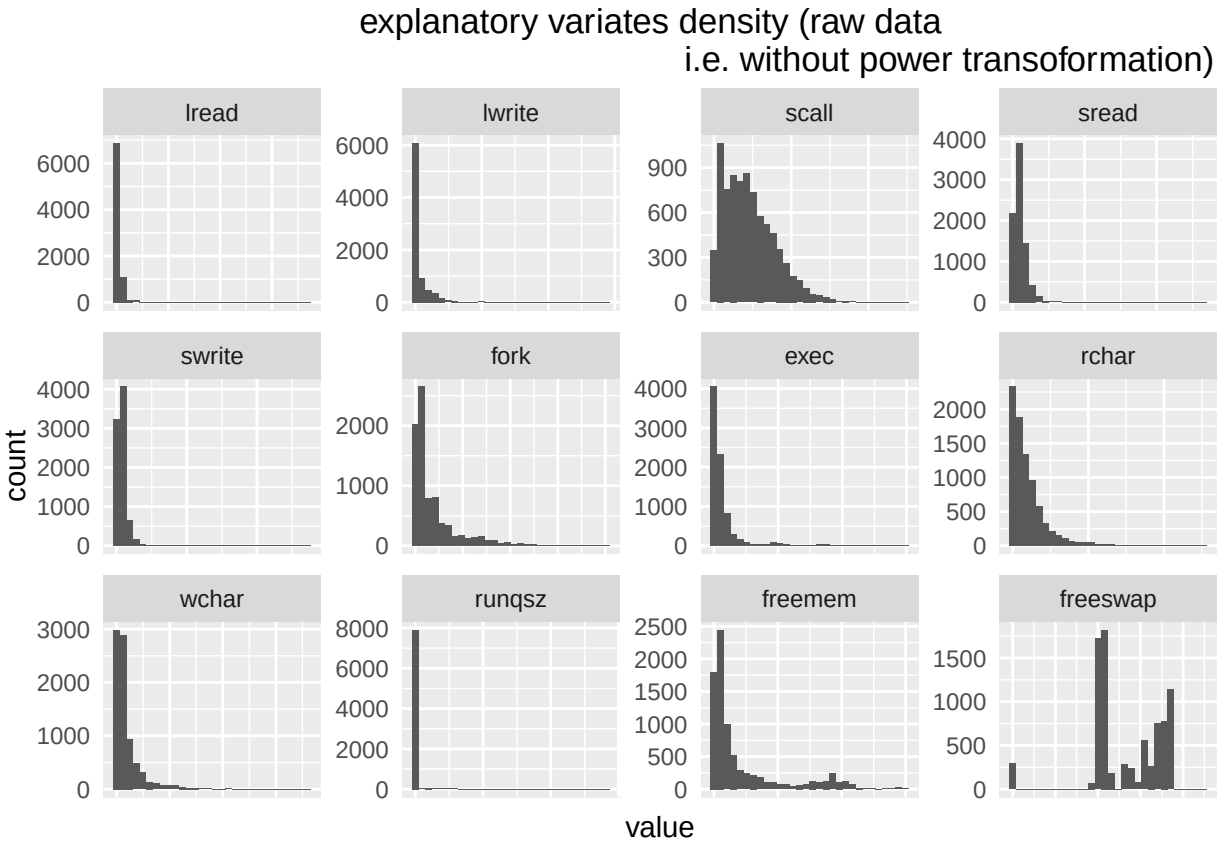
It was observed that data is not normally distributed for most of the explanatory variates as well the response variable. It is shown in the following figures. Therefore, in order to obtain corresponding normalized variates to facilitate statistical analysis, Box-Cox power transformation was used.

To choose lambda, interactive loon plots were utilized which will be discussed next.

```
setwd("~/Dropbox/UW_Coursework/Winter-2016/CM764/Project")
library(ggplot2)
library(reshape2)
load("cpu_small.RData")
df_melt <- melt(cpu_small,"usr")

ggplot(df_melt, aes(value, usr)) +  geom_point() + facet_wrap(~variable,scales="free")+theme(axis.ticks
= element_blank(), axis.text.x = element_blank()) + ggtitle("usr (response variate) vs explanatory
variates (raw data i.e. without power transoformation)")
```




```
ggplot(df_melt, aes(value)) +  geom_histogram() + facet_wrap(~variable,scales="free")+theme(axis.ticks
  = element_blank(), axis.text.x = element_blank())+ggtitle("explanatory variates density (raw data
                                                          i.e. without power transoformation)")
```



0.3 Power Transformation using interactive loon plots

For each variate it's relationship with response variate usr is plotted using l_plot along with histograms of each explanatory variate and usr using l_hist. Then values of lambda_x and lambda_y are so chosen that data is normalized along each direction. The selection process is done interactively using loon library. Since, response variable is **usr** for all variates, it is chosen once, and the process is repeated for all explanatory variates. Power transformation is depicted graphically for two variates, usr and swrite. It can be compared to figures shown previosly which are without power transformation to witness the normalization of data after transformation.

Table of lambda values is provided for rest of the variates.

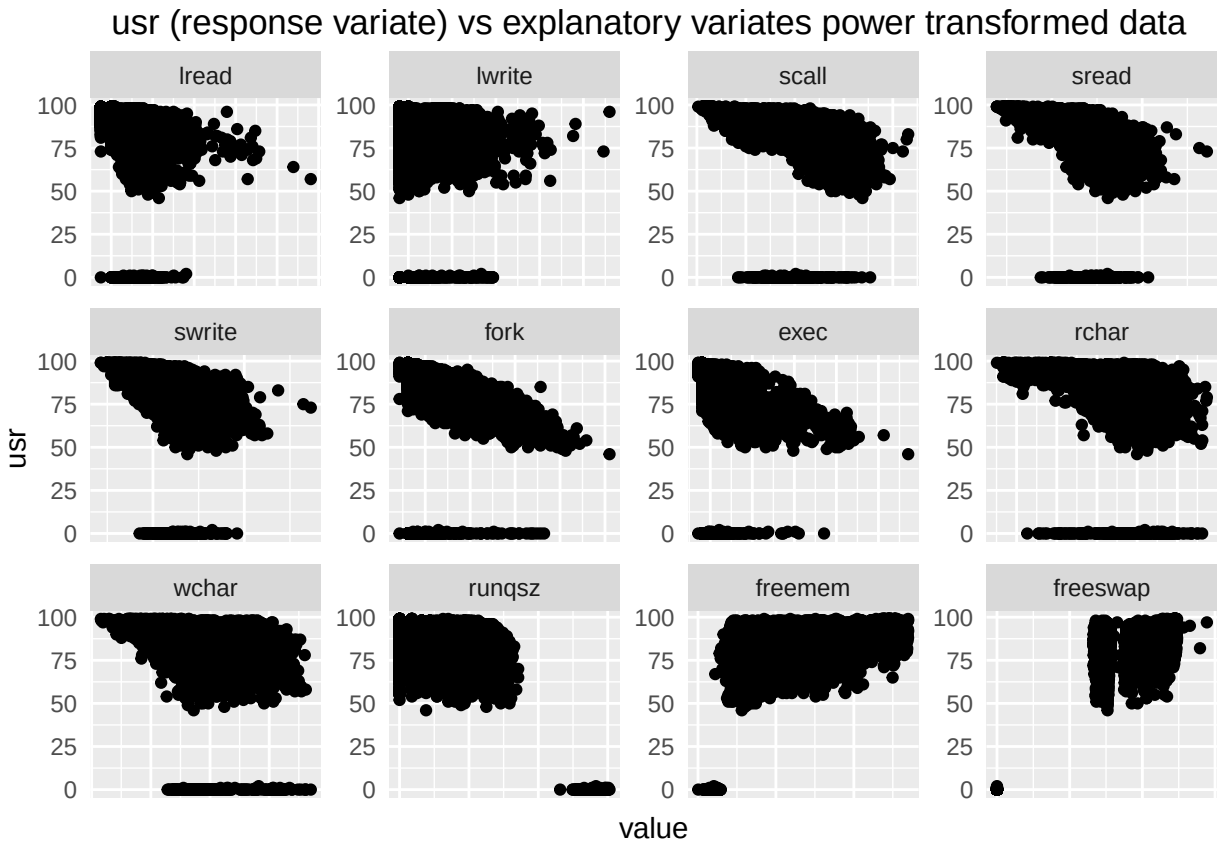
Variate | lambda

lread | 0.4 lwrite | 0.5 scall | 0.3 sread | 0.1 swrite | 0.1 fork | 0.7 exec | 0.7 rchar | 0.1 wchar | 0.0 runqsz | -0.3 freemem | 0.0 freeswap | 0.9 usr | 1

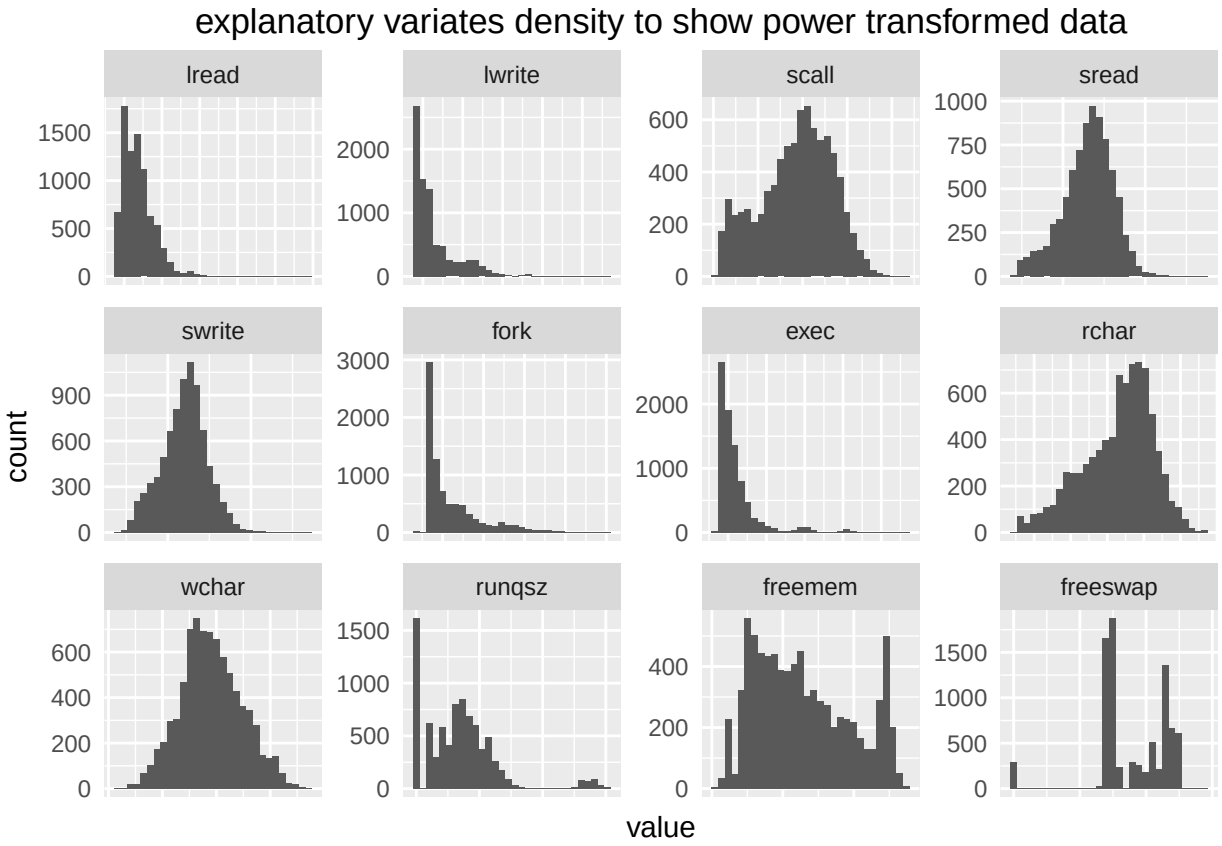
After Power transformation, we get following data graphs for variates.

```
knitr::opts_chunk$set(warning = FALSE, message = FALSE)

powers_1<-c(0.4,  0.5, 0.3 , 0.1,  0.1, 0.7, 0.7, 0.1, 0.0, -0.3, 0.0, 0.9,1)
power_trans <- data.frame(colnames(cpu_small),powers_1)
cpu_trans <-  list()

for (i in 1:length(powers_1)){
  x<- cpu_small[,i]
  lambda <- powers_1[i]
```

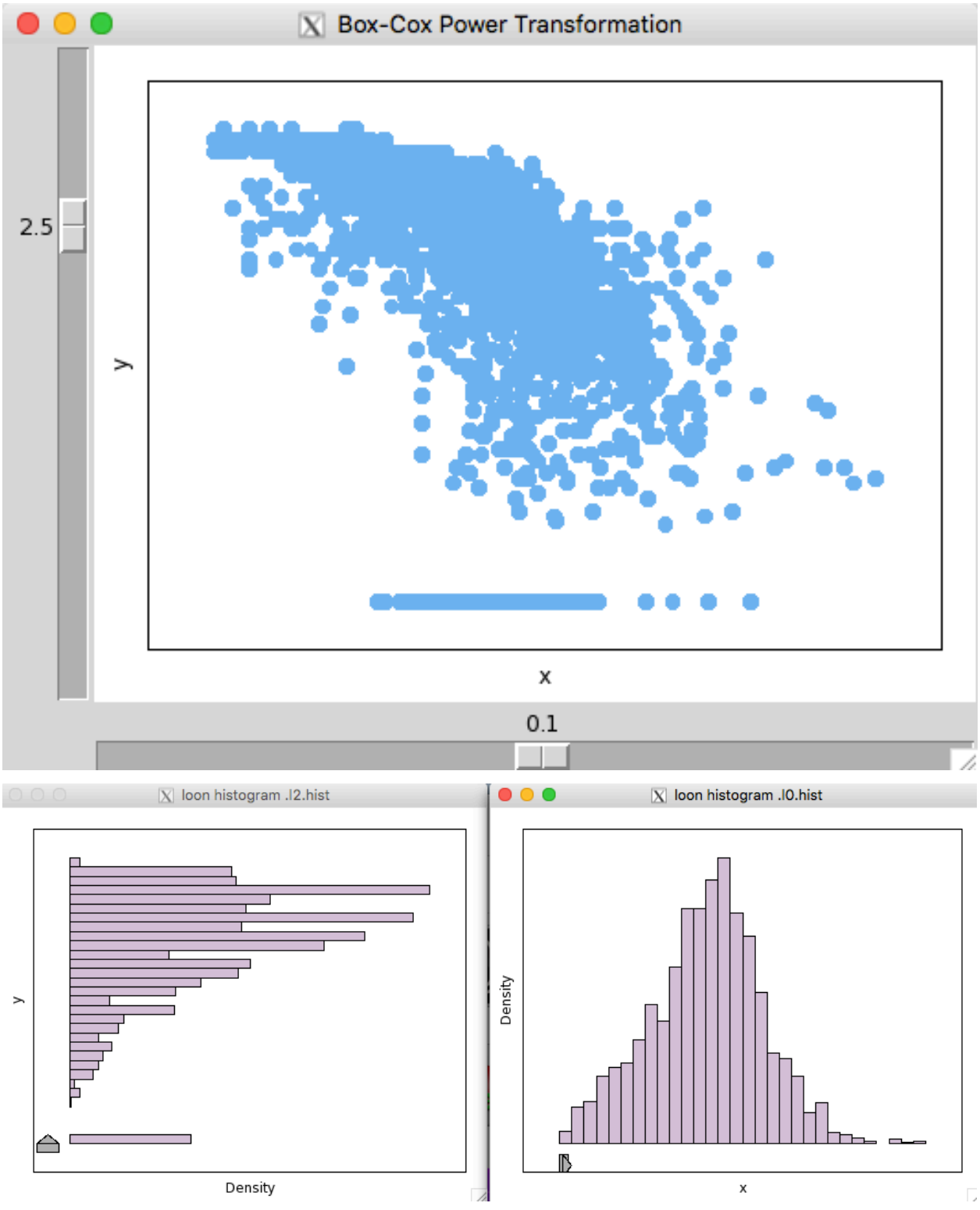
```
  if (lambda== 0){
    cpu_trans[[i]] <-log(x)}
  else if(lambda == 1){
    cpu_trans[[i]] <- x
  }
  else{
    cpu_trans[[i]] <- (x^lambda-1)/lambda
  }
}

cpu_trans <- as.data.frame(cpu_trans)
colnames(cpu_trans) <- colnames(cpu_small)
df_melt2 <- melt(cpu_trans,"usr")

ggplot(df_melt2, aes(value, usr)) +  geom_point() + facet_wrap(~variable,scales="free")+
  theme(axis.ticks =element_blank(), axis.text.x = element_blank()) +
  ggtitle("usr (response variate) vs explanatory variates power transformed data ")
```




```
ggplot(df_melt2, aes(value)) +  geom_histogram() + facet_wrap(~variable,scales="free")+
  theme(axis.ticks = element_blank(), axis.text.x = element_blank())+
  ggtitle("explanatory variates density to show power transformed data")
```



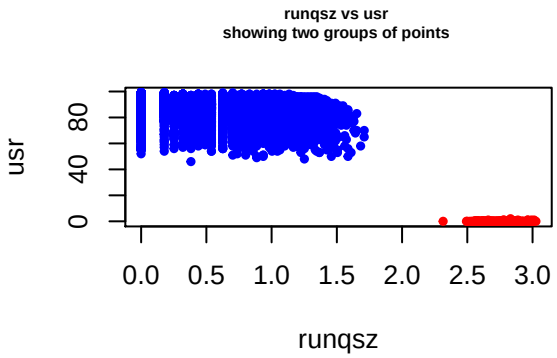
Follwoing figure represents the interactive procedure utilized for power transformation. In the figures below x represents variate swrite, and y represents dependent variate usr.

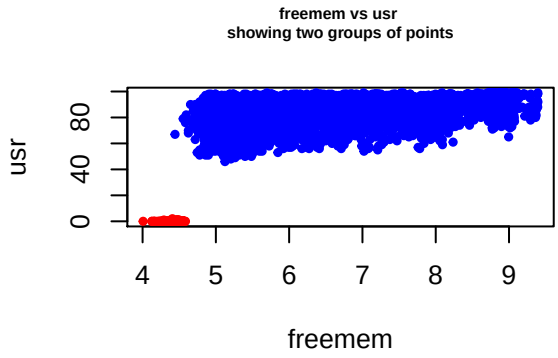

Box-Cox Power Transformation
2.5
y
x
0.1
loon histogram .l2.hist
y
Density
loon histogram .l0.hist
Density
x

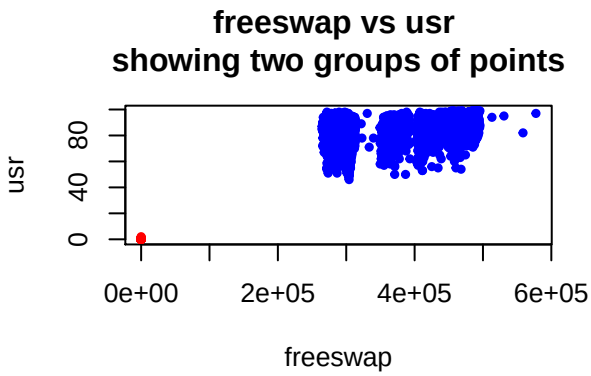
0.4 Classification of data into two groups

Observing the power transformed data, it is clear that the respnse variate (usr) is either above 50 or at 0 level. Therefore, it is a good idea to separate the data into two classes. One with cpu usage in user mode (usr) at the zero level and one above let's say 10% level. And looking at response variate versus explanatory variates plots it can inferred that variables such as runqsz (i.e. number of processes in queue), freeswap (which represents number of disk blocks available for page swapping) and freemem (which represents number of memory pages available to user processes.)can be used for this classification task. First let's have a look at runqsz versus usr and freeswap versus usr plots.

```
# here we plot these variates versus usr top demonstrate the possibility of creating two classes based
#on these variates i.e. class 1 if usr>10 and class 2 if usr <10

cpu_class_vars <- cbind(cpu_trans$runqsz,cpu_trans$freemem,cpu_trans$freeswap,cpu_trans$usr)
cpu_class_vars<- as.data.frame(cpu_class_vars)
colnames(cpu_class_vars)<- c("runqsz","freemem","freeswap","usr")

par(mfrow=c(2,2))
plot(cpu_class_vars$runqsz, cpu_class_vars$usr,
     main = "runqsz vs usr \nshowing two groups of points ",
     xlab = "runqsz",
     ylab = "usr",
     pch=19, col= "white",cex.main =0.6
)
points(cpu_class_vars$runqsz[which(cpu_class_vars$usr<10)],
       cpu_class_vars$usr[which(cpu_class_vars$usr<10)],col="red", pch=19,cex=.6)
points(cpu_class_vars$runqsz[which(cpu_class_vars$usr>10)],
       cpu_class_vars$usr[which(cpu_class_vars$usr>10)],col="blue", pch=19,cex=.6)

plot(cpu_class_vars$freemem, cpu_class_vars$usr,
     main = "freemem vs usr \nshowing two groups of points",
     xlab = "freemem",
     ylab = "usr",
     pch=19, col= "white",cex.main =0.6
)
points(cpu_class_vars$freemem[which(cpu_class_vars$usr<10)],
       cpu_class_vars$usr[which(cpu_class_vars$usr<10)],col="red", pch=19,cex=.6)
points(cpu_class_vars$freemem[which(cpu_class_vars$usr>10)],
       cpu_class_vars$usr[which(cpu_class_vars$usr>10)],col="blue", pch=19,cex=.6)

plot(cpu_class_vars$freeswap, cpu_class_vars$usr,
     main = "freeswap vs usr \nshowing two groups of points",
     xlab = "freeswap",
     ylab = "usr",
     pch=19, col= "white"
)
points(cpu_class_vars$freeswap[which(cpu_class_vars$usr<10)],
       cpu_class_vars$usr[which(cpu_class_vars$usr<10)],col="red", pch=19,cex=.6)
points(cpu_class_vars$freeswap[which(cpu_class_vars$usr>10)],
       cpu_class_vars$usr[which(cpu_class_vars$usr>10)],col="blue", pch=19,cex=.6)
```







Observing above plots which show response variate versus three other variates, we can say that for very large values of runqsz (which represents number of processes in queue), neglible time is spent in user mode by cpu i.e. usr is negligible. This means if the process queue size is very large it can be said that cpu is working in kernel mode and not accepting any new user processes and therefore usr is approx. 0.

Now, consider freemem variate which represents number of memory pages available to user processes, if we look at the above plot for usr versus freemem, for very low freemem values usr is almost 0 which means very low value of freemem is indicative of the fact that cpu is not executing user initiated processes and that it is runnning in kernel mode.

Finally, looking at freeswap versus usr, it can be said that when number of disk blocks available for page swapping (freeswap) is very low, then cpu is not in user mode or that usr is almost 0.

Now, we will generate a naive bayes classifier based on the above three variates and fit response models to the two groups separately. The idea is to fit a response model to each group (i.e. group with usr>10 and group with usr <10). Then for the test data we use a naive bayes classifier to determine if the new point lies in group-1 or group-2 and predict the value of usr accordingly.

There are 8192 observations in cpu_trans dataframe (which contains power transformed data) the sample at
hand. Lets call this S therefore, $N_S = 8192$. We will further state that this is our study population and we partion it into into training set (S_o) and test set (τ_o) such that training set (S_o) contains 70% of the observations and test set (τ_o) contains 30% of the observtions. Sampling is done without replacement.

After partitioning S into training set (S_o) and test set (τ_o), we have,

$$N_{S_o} = 5734$$

and

$$N_{\tau_o} = 2458$$

We will use observations in S_o to generate classifier and test it on observations in τ_o.

Let,

$$X1 = runqsz$$
$$X2 = freemem$$
$$X3 = freeswap$$

and let G1 be the group with observations having usr > 10, and G2 be the group with observations having usr <= 10. So we can write the naive Bayes' classifier as,

$$ratio = \frac{Pr(G = 1|X1 = x1, X2 = x2, X3 = x3)}{Pr(G = 2|X1 = x1, X2 = x2, X3 = x3)} \;=\; \frac{\pi_1 f_1(x1, x2, x3)}{\pi_2 f_2(x1, x2, x3)}$$

Assuming that variates X, Y are independently distributed, we can write

$$f_1(x1, x2, x3) \;=\; f_{11}(x1)\ x\ f_{12}(x2)\ x\ f_{13}(x3)$$

, and

$$f_2(x1, x2, x3) \;=\; f_{21}(x1)\ x\ f_{22}(x2)\ x\ f_{23}(x3)$$
$$ratio \;=\; \frac{\pi_1 f_{11}(x1) f_{12}(x2) f_{13}(x3)}{\pi_2 f_{21}(x) f_{22}(x2) f_{23}(x3)}$$

where,

$$f_1(x1, x2, x3) = f(X1 = x1, X2 = x2, X3 = x3|G = 1)$$
$$f_2(x1, x2, x3) = f(X1 = x1, X2 = x2, X3 = x3|G = 2)$$

Since, we have to compute density estimates for S_o, above equations are applied to training data.

This classifier gives approximately 99% prediction accuracy on the test data τ_o. This classifier will be used to identify the cluster which a new point in validation set belongs to during the generation of response models.

0.5 Response Models

We have 12 explanatory variates and 1 response variate, and we will use non parametric Generalized Additive Models (gam) . where the assumption will be that the response model is additive in each explanatory variate such that the dependance of the response on each variate can be described by a single smoothing spline, cubic regression spline or loess assuming no interaction terms are present. Although in the end we will add interaction terms using some selection criteria.

Generalized additive model of non parametric form can be expressed as:

$$\widehat{\mu}(x1, x2, ...x12) \;=\; \widehat{\beta_o} \;+\; \widehat{\beta_1} f_1(x1) \;+\; ..... + \;\widehat{\beta_1 2} f_1 2(x12)$$

where,
x1 = lread
x2 = lwrite x3 = scall
x4 = sread
x5 = swrite x6 = fork x7 = exec
x8 = rchar x9 = wchar x10 = runqsz
x11 = freemem x12 = freeswap

Now, let's look at a generalized additive model(gam) generated by assuming three interaction terms and smoothing terms for all 12 explanatory variates as follows:

```
library("mgcv", lib.loc="/Library/Frameworks/R.framework/Versions/3.2/Resources/library")

cpu_data_group1 <- cpu_trans[which(cpu_trans$usr>=10),]
# gam model with interaction terms
cpu_gam1 <- mgcv::gam(usr~s(lread,lwrite)+s(exec,rchar)+s(freemem,freeswap)+s(scall)+s(sread)+
s(swrite)+ s(fork)+s(exec)+s(rchar)+s(wchar)+s(runqsz)+s(freemem)+s(freeswap)+s(lread),
data=cpu_data_group1)

# gam model with no interaction term
cpu_gam2 <- mgcv::gam(usr~s(lread)+s(lwrite)+s(scall)+s(sread)+s(swrite)+s(fork)+s(exec)+
                        s(rchar)+s(wchar)+s(runqsz)+s(freemem)+s(freeswap),data=cpu_data_group1)

# Lets look at their
```

Let's look at their GCV score (Genralized Cross Validation Score).

```
summary(cpu_gam1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## usr ~ s(lread, lwrite) + s(exec, rchar) + s(freemem, freeswap) +
##     s(scall) + s(sread) + s(swrite) + s(fork) + s(exec) + s(rchar) +
##     s(wchar) + s(runqsz) + s(freemem) + s(freeswap) + s(lread)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 87.09306    0.03165    2752   <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(lread,lwrite)      21.858 25.688   8.261  < 2e-16 ***
## s(exec,rchar)        13.796 27.000   2.872 5.39e-14 ***
## s(freemem,freeswap) 27.000 27.000  19.336  < 2e-16 ***
## s(scall)              8.691  8.971 185.248  < 2e-16 ***
## s(sread)              8.396  8.687   4.125 3.36e-05 ***
## s(swrite)             8.122  8.573  13.612  < 2e-16 ***
## s(fork)               6.971  7.813 253.412  < 2e-16 ***
## s(exec)               7.104  7.956   8.189 4.12e-11 ***
## s(rchar)              9.000  9.000   4.119 2.63e-05 ***
## s(wchar)              8.441  8.914  27.484  < 2e-16 ***
## s(runqsz)             5.897  6.907   7.816 2.69e-09 ***
## s(freemem)            8.472  8.913  31.666  < 2e-16 ***
## s(freeswap)           8.842  8.986   8.789 1.09e-12 ***
## s(lread)              1.000  1.000  51.988 6.09e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =    0.9   Deviance explained = 90.2%
```

```
## GCV = 8.0586  Scale est. = 7.9111    n = 7898
```

```
summary(cpu_gam2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## usr ~ s(lread) + s(lwrite) + s(scall) + s(sread) + s(swrite) +
##     s(fork) + s(exec) + s(rchar) + s(wchar) + s(runqsz) + s(freemem) +
##     s(freeswap)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 87.09306    0.03286    2651   <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(lread)    8.670  8.944  44.791  < 2e-16 ***
## s(lwrite)   7.346  8.166   5.352 8.92e-07 ***
## s(scall)    8.785  8.986 185.654  < 2e-16 ***
## s(sread)    8.490  8.751   5.973 3.84e-08 ***
## s(swrite)   8.120  8.586  15.777  < 2e-16 ***
## s(fork)     7.145  7.953 252.662  < 2e-16 ***
## s(exec)     8.065  8.681  55.608  < 2e-16 ***
## s(rchar)    8.778  8.986  48.519  < 2e-16 ***
## s(wchar)    8.688  8.972  27.905  < 2e-16 ***
## s(runqsz)   6.307  7.272  14.093  < 2e-16 ***
## s(freemem)  8.660  8.963  33.895  < 2e-16 ***
## s(freeswap) 8.940  8.999  58.454  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.892   Deviance explained = 89.4%
## GCV = 8.6354  Scale est. = 8.5272    n = 7898
```

As we can see there is not significant difference between two gam models int terms of GCV score, we can neglect the interaction terms.

Next, we look at the effective degrees of freedom(edf) i.e. trace of smoother matrix of the smoothing splines used for each variate. Note that in mgcv package edf is determined by using GCV/UBRE scores. But
the maximum value of edf is limited by default k= 10 or user supplied valaue k. Let's try different values of k for different variates. Consider the model cpu_gam2 (i.e. no consideration of interaction
terms) for now. As we can see in the summary of cpu_gam2 that edf values for smooths of some variates is close to default value 10 such as for exec, freemem and freeswap.

So, it is recommended in the literature to test with higher values of k for those smooths. So we set k= 20 for these cases.

```r
# set higher k
cpu_gam3 <- mgcv::gam(usr~s(lread)+s(lwrite)+s(scall)+s(sread)+s(swrite)+s(fork)+s(exec,k=20)+s(rchar)+
                        s(wchar)+s(runqsz)+s(freemem,k=20)+s(freeswap,k=20),data=cpu_data_group1)
summary(cpu_gam3)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## usr ~ s(lread) + s(lwrite) + s(scall) + s(sread) + s(swrite) +
##     s(fork) + s(exec, k = 20) + s(rchar) + s(wchar) + s(runqsz) +
##     s(freemem, k = 20) + s(freeswap, k = 20)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  87.0931     0.0323    2696   <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(lread)     8.677  8.944  48.404  < 2e-16 ***
## s(lwrite)    6.855  7.750   5.945 1.77e-07 ***
## s(scall)     8.775  8.980 183.121  < 2e-16 ***
## s(sread)     6.073  7.221   7.886 1.07e-09 ***
## s(swrite)    1.000  1.000 118.540  < 2e-16 ***
## s(fork)      6.576  7.468 292.184  < 2e-16 ***
## s(exec)     11.479 13.600  36.749  < 2e-16 ***
## s(rchar)     8.784  8.986  47.454  < 2e-16 ***
## s(wchar)     8.607  8.956  26.166  < 2e-16 ***
## s(runqsz)    6.171  7.155  13.663  < 2e-16 ***
## s(freemem)  14.966 17.083  17.812  < 2e-16 ***
## s(freeswap) 18.767 18.990  42.277  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.896   Deviance explained = 89.7%
## GCV = 8.3532  Scale est. = 8.2393    n = 7898
```

```r
#detach("package:mgcv", unload=TRUE)
```

edf is still close to 20 for some freemem freeswap.

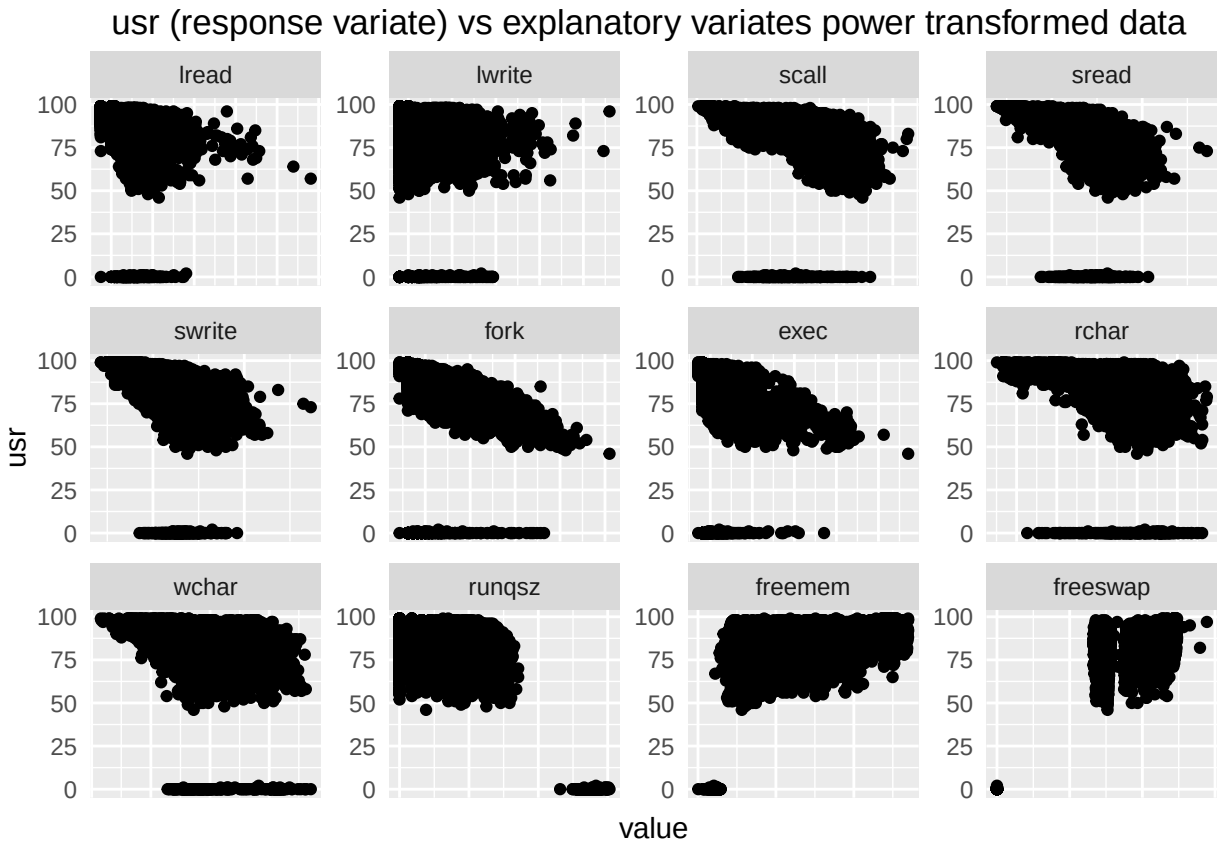
Another approach to find optimum edf is to calculate GCV score independently for a smooth for each variate. But why only test with spline? We can also use combination of smoothing splines, natural splines or local polynomial regression splines. To do this let's consider response model for each variate independently and find an optimal model which can be any form of spline with best GCV score. This way we can also detect outliers and thus adjust the response model accordingly.

Finally, when we have optimal df for all variates we can use gam to those specific values and generate our final response model which will consider all the variates and we can also add interaction terms if needed. loess is not available in 'mgcv' package, so 'gam' package was used. In gam package degree of freedom is not optimized, and has to be manually supplied.

0.6 Selection of response model for each variate

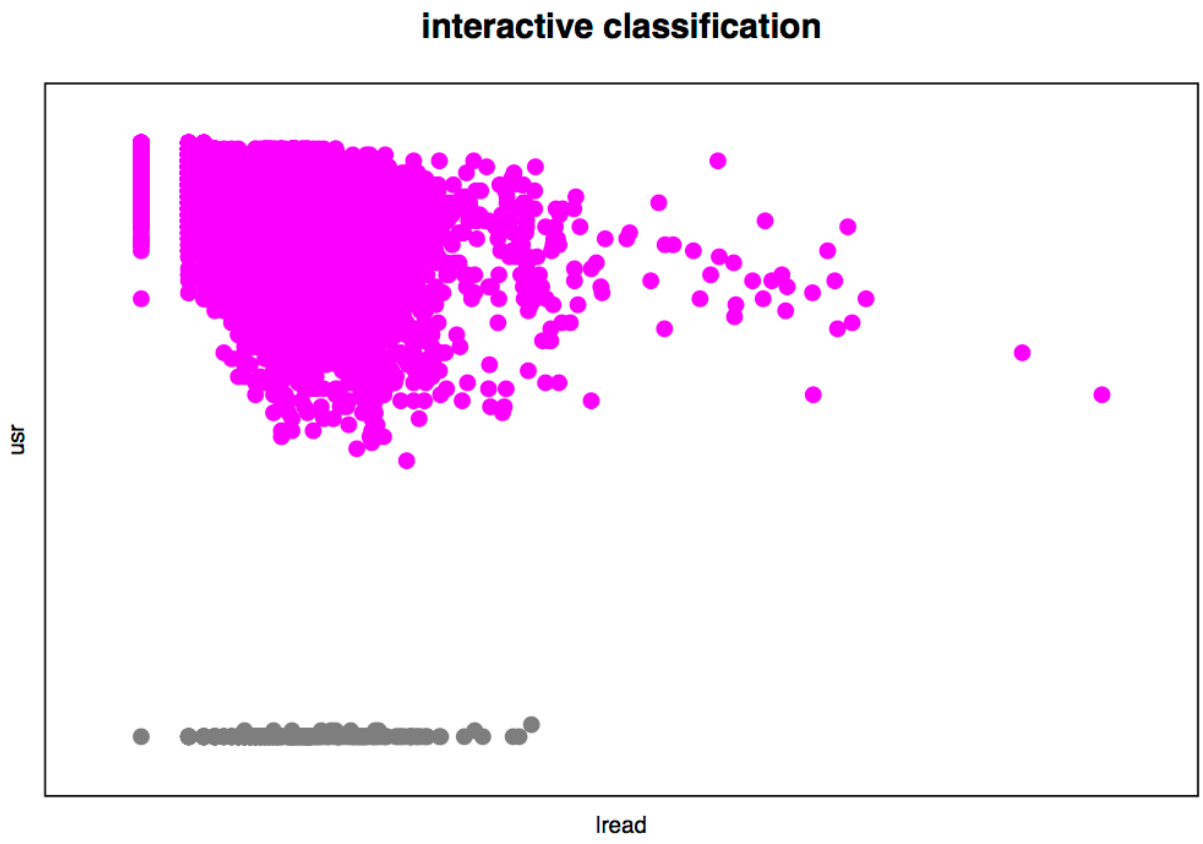
Starting with first independent variate x1 = lread and response y = usr. If we look at pairs plot specifically for variate runqsz, there is certainly two groups of observations, which can be analyzed independently.

```
ggplot(df_melt2, aes(value, usr)) +  geom_point() + facet_wrap(~variable,scales="free")+
  theme(axis.ticks =element_blank(), axis.text.x = element_blank()) +
  ggtitle("usr (response variate) vs explanatory variates power transformed data ")
```



Actually these are the same observations which we classified for earlier in the naive bayes classifier. But we can also interactively classify these two groups and fit response models to the group for which usr value is high and ignore th group for which usr is approximately 0. That is what we will do for usr vs lread relationship. Let's fit smoothers and locally wighted sum of squares (loess) models.

First we use loon to select the group of points to which we fit models. Selected points are in pink color

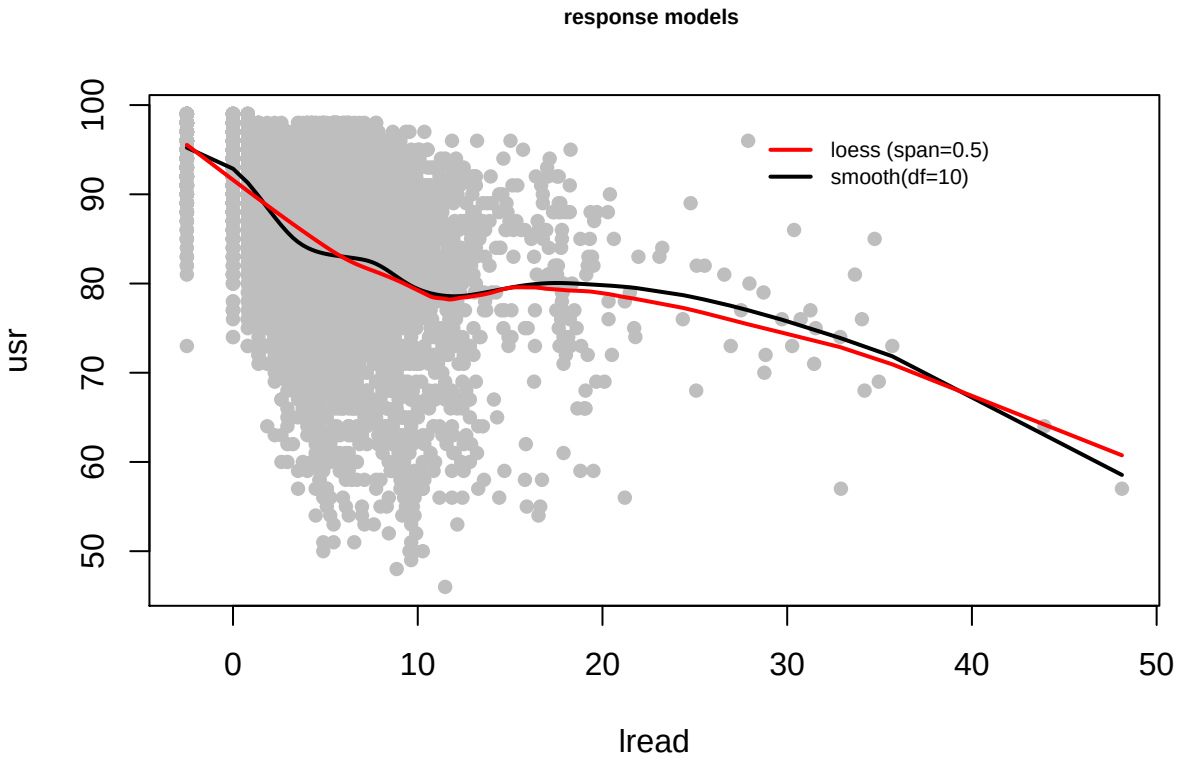


Now, we use these selected points as follows:

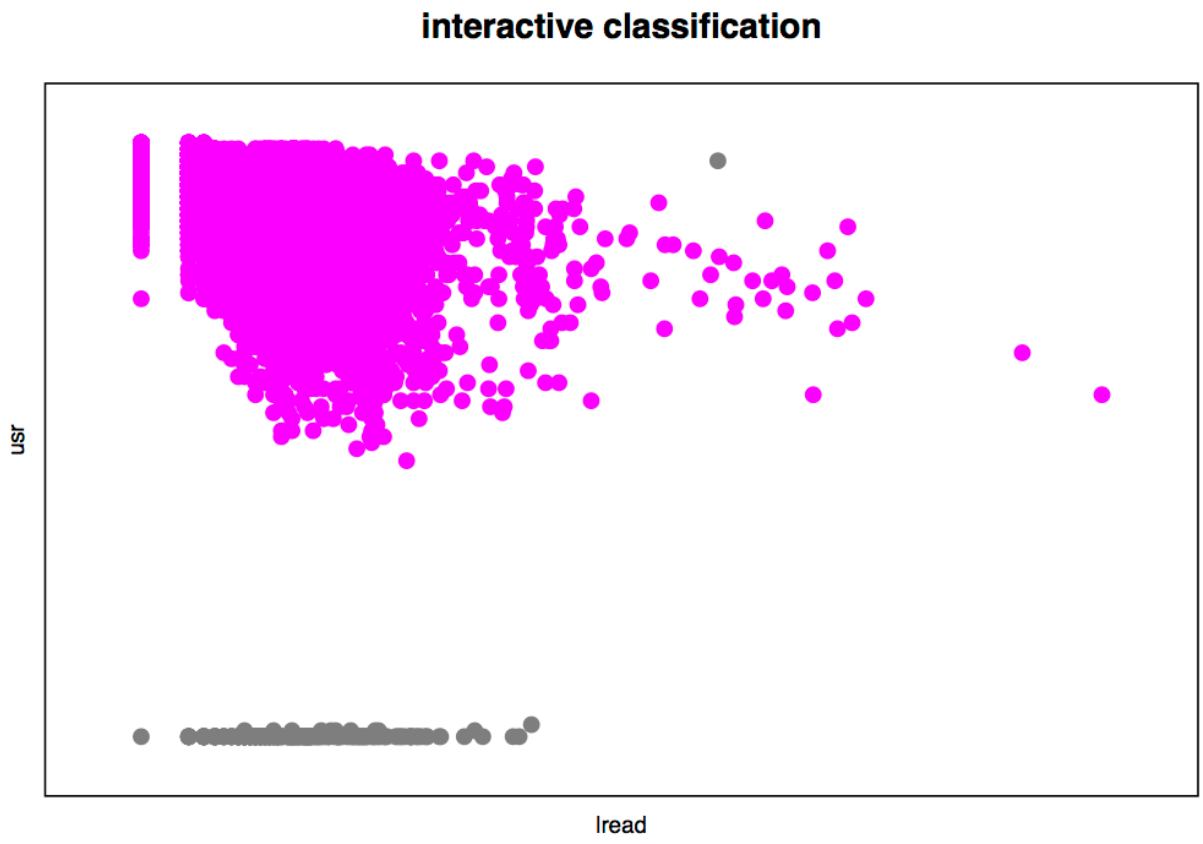
First create a function that stores selected points in a data.frame

```
select_points <- function(p) {
  ## For x
  xnew <- p['xTemp']
  if (length(xnew) == 0) {xnew <- p['x']}

  ## For y
  ynew <- p['yTemp']
  if (length(ynew) == 0) {ynew <- p['y']}

  ## use only the selected points for the fits
  sel <- p['selected']
  xnew <- xnew[sel]
  ynew <- ynew[sel]
  Nsel <- sum(sel)

  data_sel <- as.data.frame(cbind(xnew,ynew))
  colnames(data_sel) <- c("x","y")
  data_sel
}
data_sel <- select_points(p)
data_sel_outlier <- select_points(p2)
```

```r
library(gam)

#data_sel <- data_sel_lread1
#data_sel_outlier <- data_sel_lread2

#load("data_sel.RData")
#load("data_sel_outlier.RData")

smp_size <- floor(0.7 * nrow(data_sel))

# first we sample points to be used to make predictions from among the selected points
## set the seed to make the partition reproductible
set.seed(123)
pred_ind <- sample(seq_len(nrow(data_sel)), size = smp_size)

data_sel_train <- data_sel[pred_ind, ] # points used for fit
data_sel_pred  <- data_sel[-pred_ind, ] # points to be used for making prediction
x_ext <- append(data_sel_pred$x,min(data_sel$x),after=0)
x_ext <- append(x_ext,max(data_sel$x),after=length(data_sel_pred$x))

fit_smooth <- smooth.spline(data_sel_train$x,data_sel_train$y,df=10)
fit_loess <- gam::gam(y~lo(x,span=0.5), data=data_sel_train)

plot(data_sel$x,data_sel$y,
     main="response models ",
     xlab="lread",
     ylab="usr",
     col="grey",pch=16,cex.main =0.7)

Xorder <- order(x_ext)
pred_sm <- predict(fit_smooth, x = x_ext[Xorder])
pred_lo <- gam::predict.gam(fit_loess, data.frame(x=x_ext[Xorder]))

lines( pred_sm$x,pred_sm$y,lwd = 2,col="black")
lines(x_ext[Xorder],pred_lo,col="red",lwd = 2)
legend(28,98,legend = c("loess (span=0.5)","smooth(df=10)"),lwd =2,col=c("red","black"),
       cex=0.7,bty = "n")
```

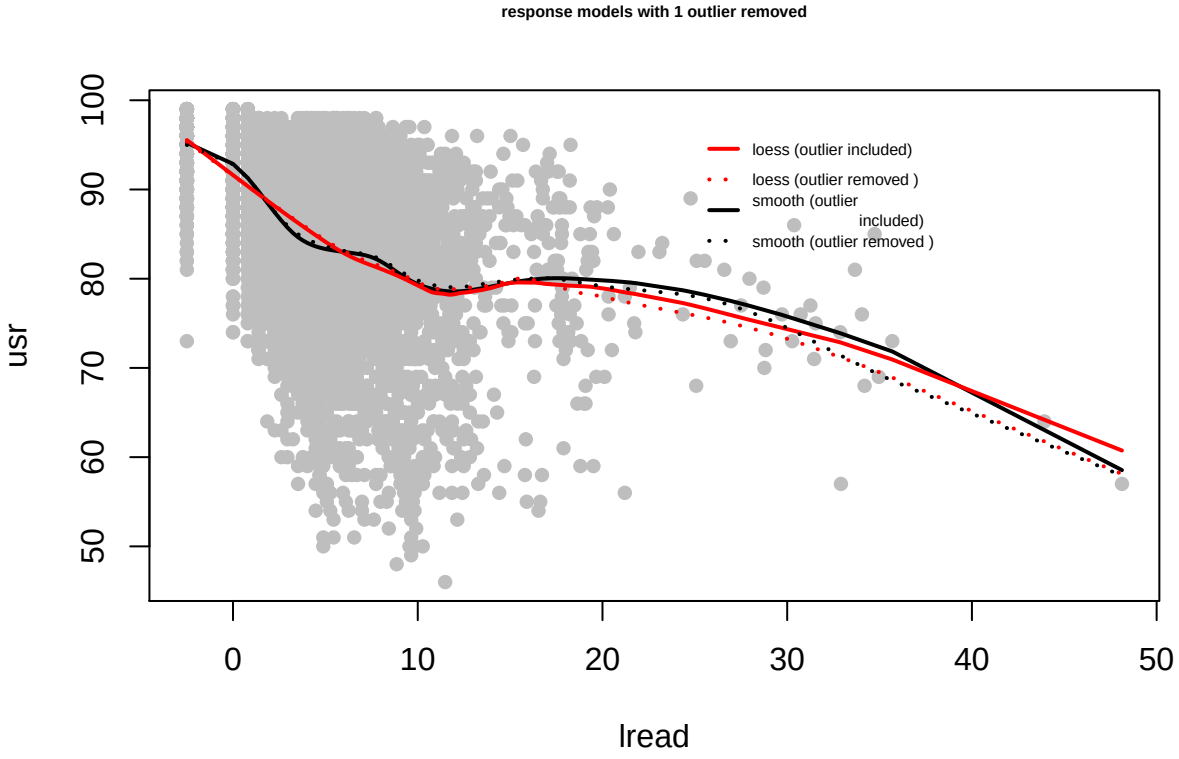


Above plots show smooth and loess fit for the selected points. Let's investigate the influence of outliers on these curves. We remove one outliers from the selected points and again plot the fitting curves. Outlier is in grey color located above the top group.




```
smp_size <- floor(0.7 * nrow(data_sel_outlier))

# first we sample points to be used to make predictions from among the selected points
## set the seed to make the partition reproductible
set.seed(123)
pred_ind <- sample(seq_len(nrow(data_sel_outlier)), size = smp_size)

data_sel_train <- data_sel_outlier[pred_ind, ] # points used for fit
data_sel_pred  <- data_sel_outlier[-pred_ind, ] # points to be used for making prediction
x_ext2 <- append(data_sel_pred$x,min(data_sel_outlier$x),after=0)
x_ext2 <- append(x_ext2,max(data_sel_outlier$x),after=length(data_sel_pred$x))

fit_smooth2 <- smooth.spline(data_sel_train$x,data_sel_train$y,df=10)
fit_loess2 <- gam::gam(y~lo(x,span=0.5), data=data_sel_train)

plot(data_sel_outlier$x,data_sel_outlier$y,
     main="response models with 1 outlier removed",
     xlab="lread",
     ylab="usr",
     col="grey",pch=16,cex.main =0.5)

Xorder2 <- order(x_ext2)
pred_sm2 <- predict(fit_smooth2, x = x_ext2[Xorder2])
pred_lo2 <- gam::predict.gam(fit_loess2, data.frame(x=x_ext2[Xorder2]))
```

```
lines( pred_sm$x,pred_sm$y,lwd = 2,col="black")
lines(x_ext[Xorder],pred_lo,col="red",lwd = 2)
lines( pred_sm2$x,pred_sm2$y,lwd = 2,col="black",lty=3)
lines(x_ext2[Xorder2],pred_lo2,col="red",lwd = 2,lty=3)

legend(25,98,legend = c("loess (outlier included)","loess (outlier removed )","smooth (outlier
                        included)","smooth (outlier removed )"),lty=c(1,3,1,3),lwd
       =2,col=c("red","red","black","black"),cex=0.5,bty = "n")
```



AS we can see both of the curves are affected by outlier's inclusion and exclusion but loess is closer to the original curve compared to smooth. To futher improve selection criteria, we can look at GCV scores for varying degrees of freedom for loess and smooth and select the one with the lowest GCV.

Since we are working with P_o, we can write the estimate of APSE as,

$$\widehat{APSE}(P, \tilde{mu}) \;=\; APSE(P_o, \tilde{mu})$$

such that

$$APSE(P_o, \tilde{mu}) \;=\; \frac{1}{N_S}\sum_{j=1}^{N_S} APSE(S_j, \widehat{mu}_{S_j})$$

This is computed as follows:

```
# First define a function to generate samples for k fold cross validation
kfold <- function(N, k=N, indices=NULL){
  # get the parameters right:
  if (is.null(indices)) {
    # Randomize if the index order is not supplied
    indices <- sample(1:N, N, replace=FALSE)
  } else {
```

```
    # else if supplied, force N to match its length
    N <- length(indices)
  }
  # Check that the k value makes sense.
  if (k > N) stop("k must not exceed N")
  #

  # How big is each group?
  gsize <- rep(round(N/k), k)

  # For how many groups do we need odjust the size?
  extra <- N - sum(gsize)

  # Do we have too few in some groups?
  if (extra > 0) {
    for (i in 1:extra) {
      gsize[i] <- gsize[i] +1
    }
  }
  # Or do we have too many in some groups?
  if (extra < 0) {
    for (i in 1:abs(extra)) {
      gsize[i] <- gsize[i] - 1
    }
  }

  running_total <- c(0,cumsum(gsize))

  # Return the list of k groups of indices
  lapply(1:k,
         FUN=function(i) {
           indices[seq(from = 1 + running_total[i],
                       to = running_total[i+1],
                       by = 1)
                   ]
         }
  )
}

getKfoldSamples <- function (x, y, k, indices=NULL){
  groups <- kfold(length(x), k, indices)
  Ssamples <- lapply(groups,
                     FUN=function(group) {
                       list(x=x[-group], y=y[-group])
                     })
  Tsamples <- lapply(groups,
                     FUN=function(group) {
                       list(x=x[group], y=y[group])
                     })
  list(Ssamples = Ssamples, Tsamples = Tsamples)
}
```

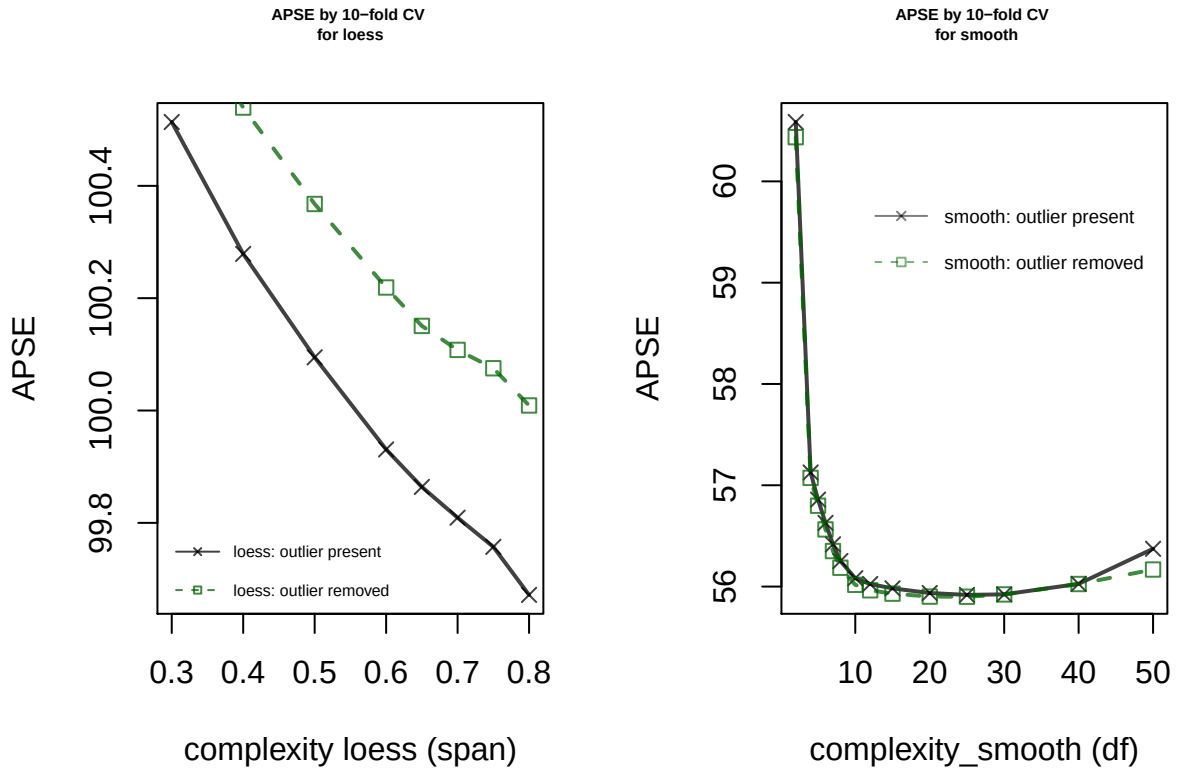
```
getmuhat <- function(sample, df) {

  if (df == 2) {
    fit <- lm(y ~ x, data=sample)
    muhat <- function(x){
      predict(fit,newdata=data.frame(x=x))
    }
  } else {
    fit <- smooth.spline(sample$x,
                         sample$y,
                         df = df)
    muhat <- function(x){predict(fit, x=x)$y}
  }
  # muhat is the function that we need to
  # calculate values at any x, so we return this function
  # as the value of getmuhat
  muhat
}

getmuhat_loess <- function(sample, span) {

  fit <- gam::gam(y~lo(x,span=span),data=sample )
  muhat_loess <- function(x){ Xorder3 <- order(x)
  gam::predict.gam(fit, data.frame(x=x[Xorder3]))}
  # fit<- loess(y~x,data=sample,span=span)
  # muhat_loess <- function(x){
  #    Xorder3 <- order(x)
  #    predict(fit, data.frame(x=x[Xorder3]))}
  # muhat is the function that we need to
  # calculate values at any x, so we return this function
  # as the value of getmuhat
  muhat_loess
}

ave_y_mu_sq <- function(sample, predfun){
  mean(abs(sample$y - predfun(sample$x))^2)
}

apse <- function(Ssamples, Tsamples, df){
  # average over the samples S
  #
  N_S <- length(Ssamples)
  mean(sapply(1:N_S,
              FUN=function(j){
                S_j <- Ssamples[[j]]
                # get the muhat function based on
                # the sample S_j
                muhat <- getmuhat(S_j, df=df)
                # average over (x_i,y_i) in a
                # single sample T_j the squares
                # (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_y_mu_sq(T_j,muhat)
```

```
            }
  )
  )
}

apse_loess <- function(Ssamples, Tsamples, span){
  # average over the samples S
  #
  N_S <- length(Ssamples)
  mean(sapply(1:N_S,
              FUN=function(j){
                S_j <- Ssamples[[j]]
                # get the muhat function based on
                # the sample S_j
                muhat_loess <- getmuhat_loess(S_j, span=span)
                # average over (x_i,y_i) in a
                # single sample T_j the squares
                # (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_y_mu_sq(T_j,muhat_loess)
              }
  )
  )
}
```

Now, call these functions as required

```
complexity_smooth <- c(2,4,5,6,7,8,10,12,15,20,25,30,40,50)
complexity_loess <- c(0.3,0.4,0.5,0.6,0.65,0.7,0.75,0.8)
# samples from data with outlier included
samples_10fold_1 <-  getKfoldSamples(data_sel$x, data_sel$y, k=10)
# samples from data with outlier excluded
samples_10fold_2 <-  getKfoldSamples(data_sel_outlier$x, data_sel_outlier$y, k=10)

Ssamples_1 <- samples_10fold_1$Ssamples
Ssamples_2 <- samples_10fold_2$Ssamples
Tsamples_1<- samples_10fold_1$Tsamples
Tsamples_2<- samples_10fold_2$Tsamples

apsehat_10fold_1_smooth <-  sapply(complexity_smooth,
                                   FUN = function(df){
                                     apse(Ssamples_1, Tsamples_1, df=df)
                                   }
)
apsehat_10fold_1_loess <- sapply(complexity_loess,
                                 FUN = function(span){
                                   apse_loess(Ssamples_1, Tsamples_1, span=span)
                                 }
)

apsehat_10fold_2_smooth<-  sapply(complexity_smooth,
                                  FUN = function(df){
                                    apse(Ssamples_2, Tsamples_2, df=df)
```

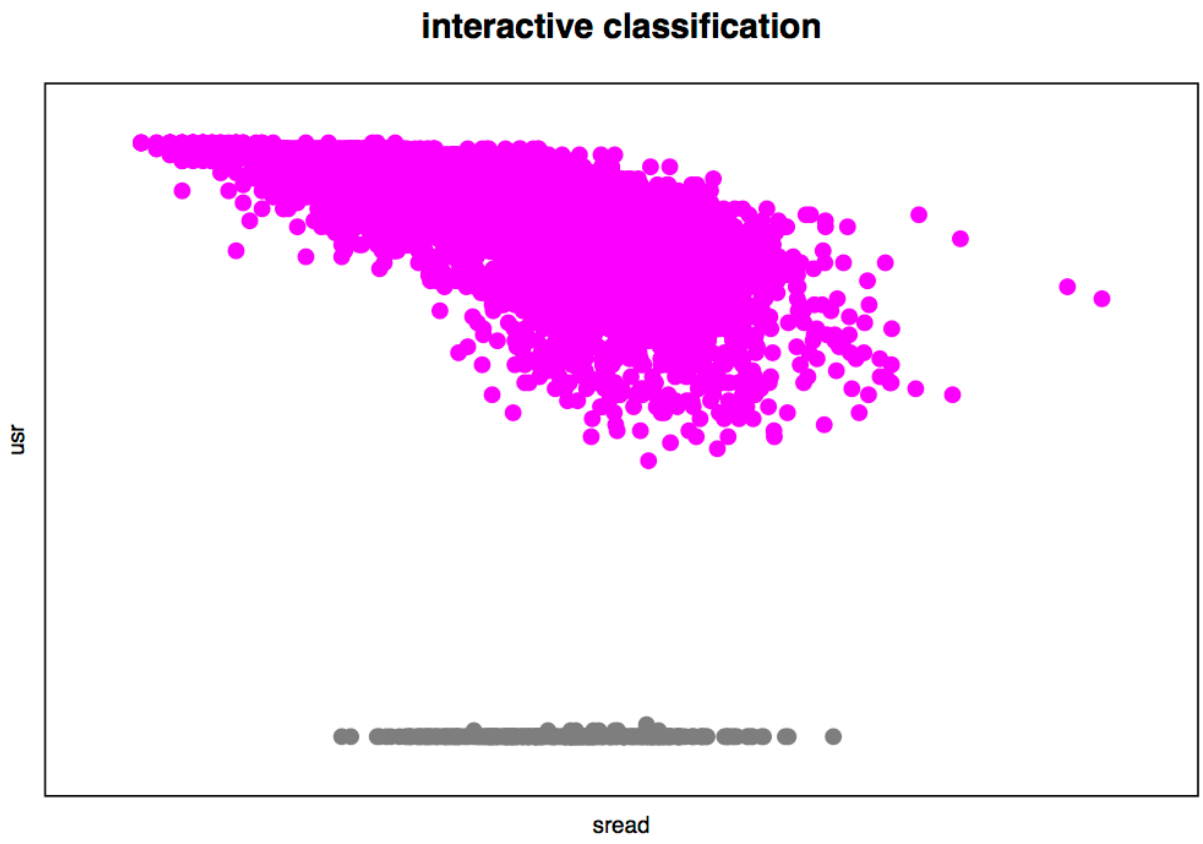
```
                                  }
)
apsehat_10fold_2_loess<- sapply(complexity_loess,
                                FUN = function(span){
                                  apse_loess(Ssamples_2, Tsamples_2, span=span)
                                }
)

par(mfrow=c(1,2))
plot(complexity_loess, apsehat_10fold_1_loess,
     main="APSE by 10-fold CV \nfor loess",
     ylab="APSE",
     xlab = "complexity loess (span)",
     pch=4, col= adjustcolor("black", 0.75),cex.main =0.5)
lines(complexity_loess, apsehat_10fold_1_loess,
      col=adjustcolor("black", 0.75), lwd=2)

lines(complexity_loess, apsehat_10fold_2_loess,
      col=adjustcolor("darkgreen", 0.75), lwd=2, lty=2)
points(complexity_loess, apsehat_10fold_2_loess,
       col=adjustcolor("darkgreen", 0.75), pch=0)
legend(0.3, 99.8,
       legend=c("loess: outlier present", "loess: outlier removed"),
       lty =1:2, pch=c(4,0),
       col=adjustcolor(c("black", "darkgreen"), 0.75),
       cex=0.5, y.intersp=2, bty="n", seg.len=3)

plot(complexity_smooth, apsehat_10fold_1_smooth,
     main="APSE by 10-fold CV \n for smooth",
     ylab="APSE",
     xlab="complexity_smooth (df)",
     pch=4, col= adjustcolor("black", 0.75),cex.main =0.5)
lines(complexity_smooth, apsehat_10fold_1_smooth,
      col=adjustcolor("black", 0.75), lwd=2)

lines(complexity_smooth, apsehat_10fold_2_smooth,
      col=adjustcolor("darkgreen", 0.75), lwd=2, lty=2)
points(complexity_smooth, apsehat_10fold_2_smooth,
       col=adjustcolor("darkgreen", 0.75), pch=0)

legend(12, 60,
       legend=c("smooth: outlier present", "smooth: outlier removed"),
       lty =1:2, pch=c(4,0),
       col=adjustcolor(c("black", "darkgreen"), 0.5),
       cex=0.6, y.intersp=2, bty="n", seg.len=3)
```




```
par(mfrow=c(1,1))
```

As we can see in the plots loess and smooths with outlier removed perform better. Also note that APSE for smooth is significantly better as compared to loess and the minimumum occurs at df = 25. So, in our gam we will use a smoothing spline with df = 25 for 'usr ~ lread' relationship.

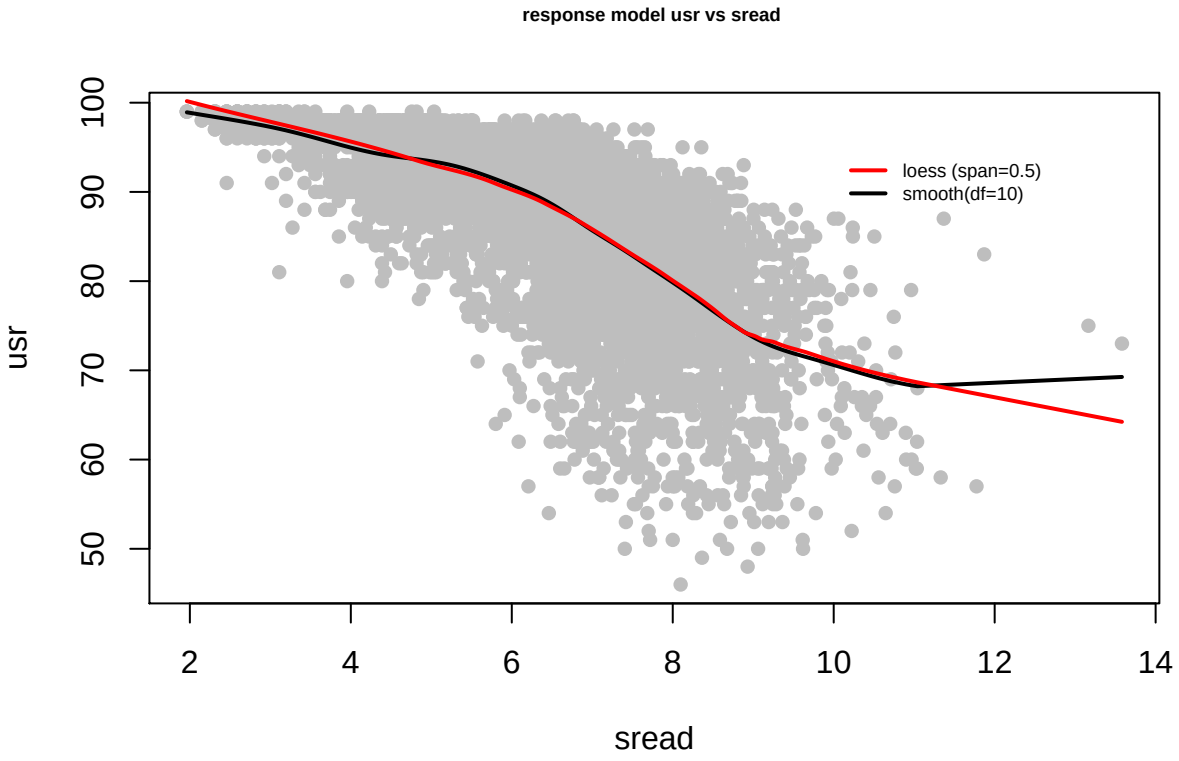
We repeat this process for all the explanatory variates. Next variate is sread and we consider its relationship with usr. First select the group of points we are interested in and do interactive classification and then fit response models to that data.



Again, use selectpoints() function which was created before.

```
load("data_sel_sread.RData")
smp_size <- floor(0.7 * nrow(data_sel_sread))

## set the seed to make the partition reproductible
set.seed(123)
pred_ind <- sample(seq_len(nrow(data_sel_sread)), size = smp_size)

data_sel_train <- data_sel_sread[pred_ind, ]
data_sel_pred  <- data_sel_sread[-pred_ind, ]
x_ext <- append(data_sel_pred$x,min(data_sel_sread$x),after=0)
x_ext <- append(x_ext,max(data_sel_sread$x),after=length(data_sel_pred$x))

fit_smooth <- smooth.spline(data_sel_train$x,data_sel_train$y,df=10)
fit_loess <- gam::gam(y~lo(x,span=0.5), data=data_sel_train)
par(mfrow=c(1,1))

plot(data_sel_sread$x,data_sel_sread$y,
     main="response model usr vs sread ",
     xlab="sread",
     ylab="usr",
     col="grey",pch=16,cex.main =0.6)

Xorder <- order(x_ext)
```

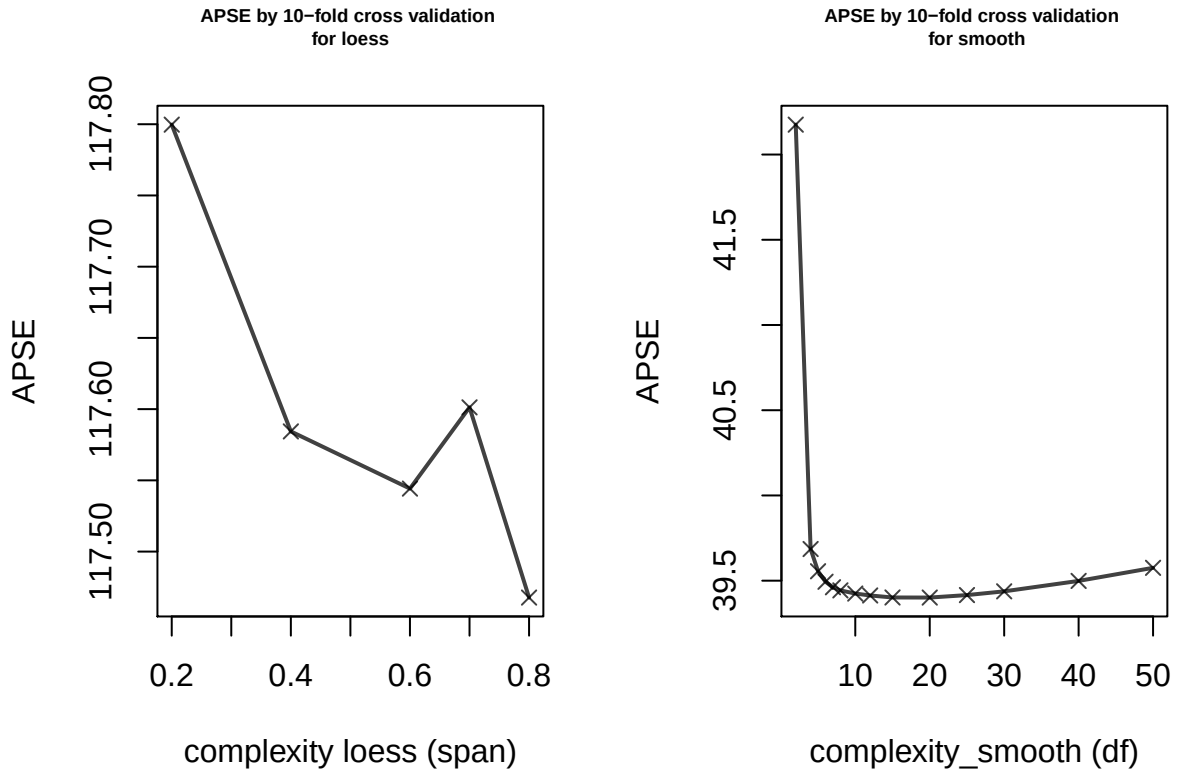
```
pred_sm <- predict(fit_smooth, x = x_ext[Xorder])
pred_lo <- gam::predict.gam(fit_loess, data.frame(x=x_ext[Xorder]))

lines( pred_sm$x,pred_sm$y,lwd = 2,col="black")
lines(x_ext[Xorder],pred_lo,col="red",lwd = 2)
legend(10,95,legend = c("loess (span=0.5)","smooth(df=10)"),lwd =2,col=c("red","black"),
       cex=0.6,bty = "n")
```



We can see loess is relatively less influenced by boundary points compared to smooth, otherwise both are overlapping for other values of sread. Let's look at their APSE to check which performs better and that if this wiggly nature at the boundary has any drastic effect on performance for smooth. Same procedure is followed as in the case of lread.

```
getmuhat_loess <- function(sample, span) {

  # fit <- gam(y~lo(x,span=span),data=sample )
  # muhat_loess <- function(x){ Xorder3 <- order(x)
  #   predict.gam(fit, data.frame(x=x[Xorder3]))}
  fit<- loess(y~x,data=sample,span=span,control=loess.control(surface="direct"))
  muhat_loess <- function(x){
    Xorder3 <- order(x)
    predict(fit, data.frame(x=x[Xorder3]))}
  # muhat is the function that we need to
  # calculate values at any x, so we return this function
  # as the value of getmuhat
  muhat_loess
}
complexity_loess <- c(.2,0.4,0.6,0.7,0.8)
```

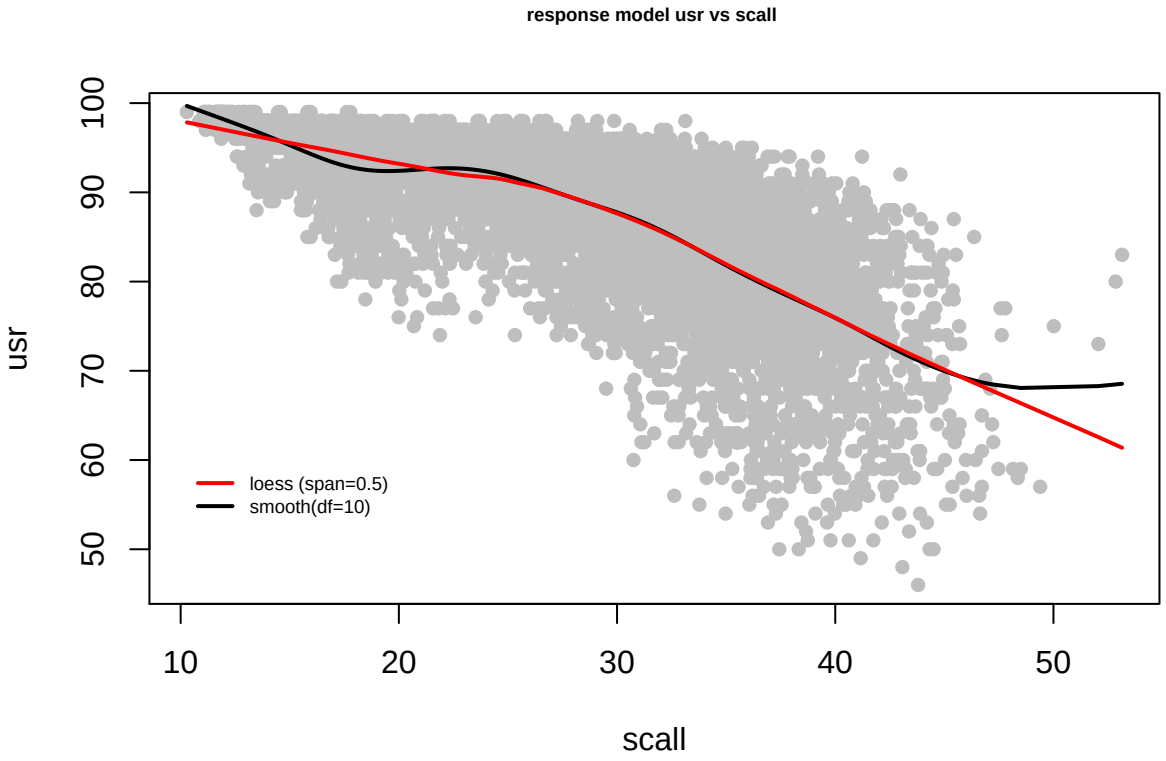
```
samples_10fold_1 <-  getKfoldSamples(data_sel_sread$x, data_sel_sread$y, k=10)

Ssamples_1 <- samples_10fold_1$Ssamples
Tsamples_1<- samples_10fold_1$Tsamples

apsehat_10fold_smooth_sread <-  sapply(complexity_smooth,
                                       FUN = function(df){
                                         apse(Ssamples_1, Tsamples_1, df=df)
                                       }
)
apsehat_10fold_loess_sread <- sapply(complexity_loess,
                                     FUN = function(span){
                                       apse_loess(Ssamples_1, Tsamples_1, span=span)
                                     }
)

par(mfrow=c(1,2))
plot(complexity_loess, apsehat_10fold_loess_sread,
     main="APSE by 10-fold cross validation \nfor loess",
     ylab="APSE",
     xlab = "complexity loess (span)",
     pch=4, col= adjustcolor("black", 0.75),cex.main =0.6)
lines(complexity_loess, apsehat_10fold_loess_sread,
      col=adjustcolor("black", 0.75), lwd=2)

legend(0.6, 100.5,
       legend=c("loess"),
       lty =1, pch=c(4),
       col=adjustcolor(c("black"), 0.75),
       cex=0.6, y.intersp=2, bty="n", seg.len=3)

plot(complexity_smooth, apsehat_10fold_smooth_sread,
     main="APSE by 10-fold cross validation \n for smooth",
     ylab="APSE",
     xlab="complexity_smooth (df)",
     pch=4, col= adjustcolor("black", 0.75),cex.main =0.6)
lines(complexity_smooth, apsehat_10fold_smooth_sread,
      col=adjustcolor("black", 0.75), lwd=2)

legend(0.6, 100.5,
       legend=c("smooth"),
       lty =1, pch=c(4),
       col=adjustcolor(c("black"), 0.75),
       cex=0.6, y.intersp=2, bty="n", seg.len=3)
```



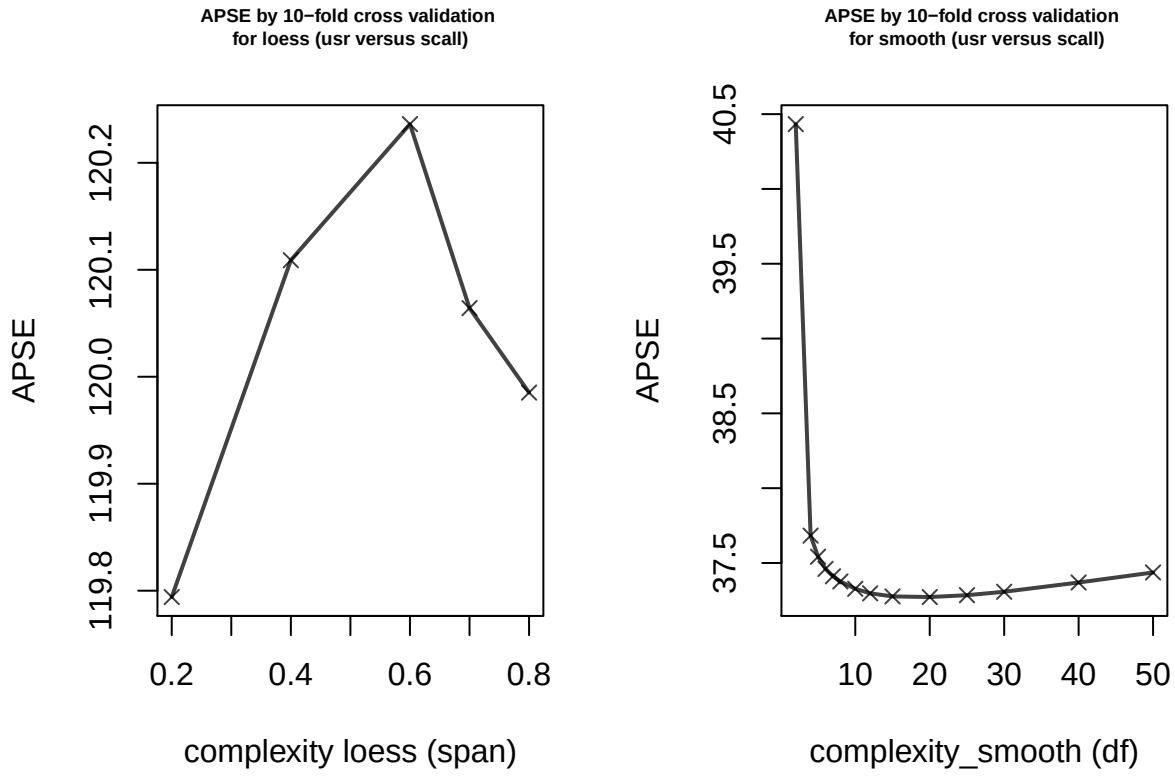

```
par(mfrow=c(1,1))
```

In this case as well smooth performs better then loess with best performance in terms of APSE at df = 20. So, we can conclude that smooth performed better despite its wigglier nature at the boundary points.

Now, we consider x = scall. Using the same analysis for this variate:

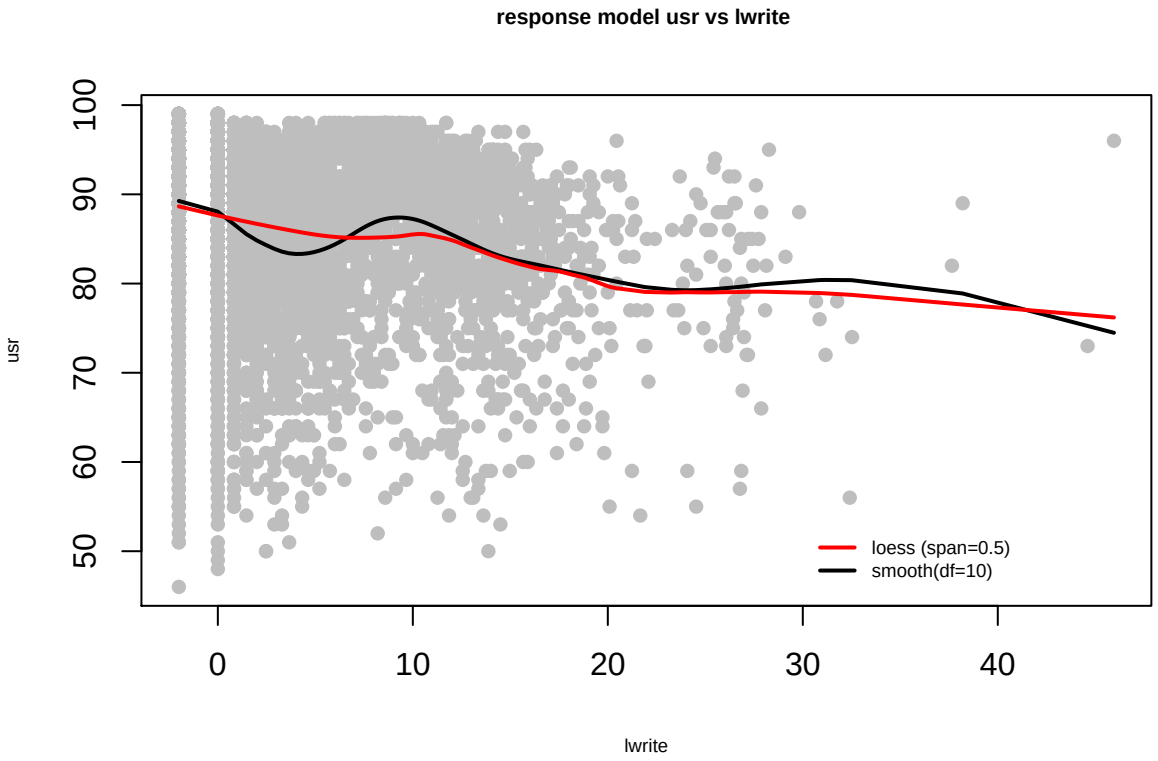


Above plot shows, usr vs scall with loess and smooth fits. In this case loess is more wigglier at the right boundary. Again looking at their APSE score,

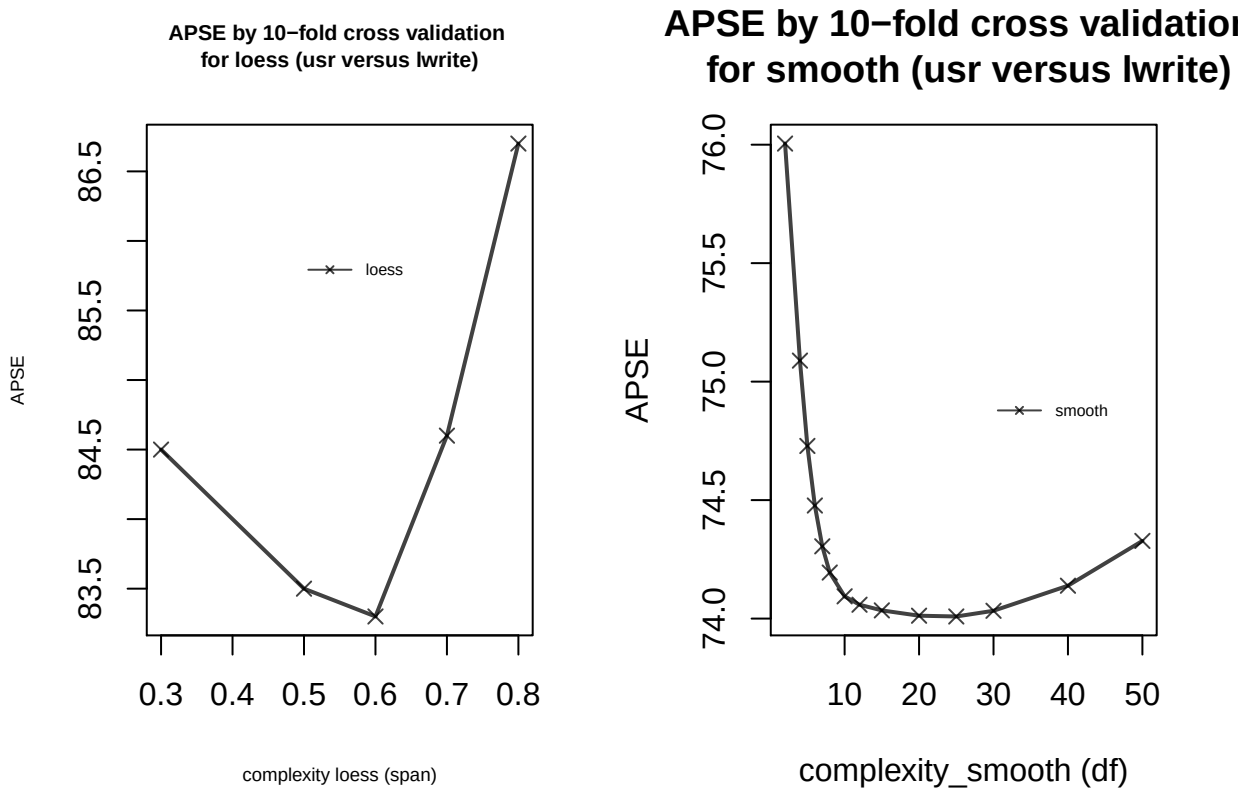


loess doesn't perform well even in this case, and again we select smooth with df = 20.

Now, we consider x = lwrite. Using the same analysis for this variate:

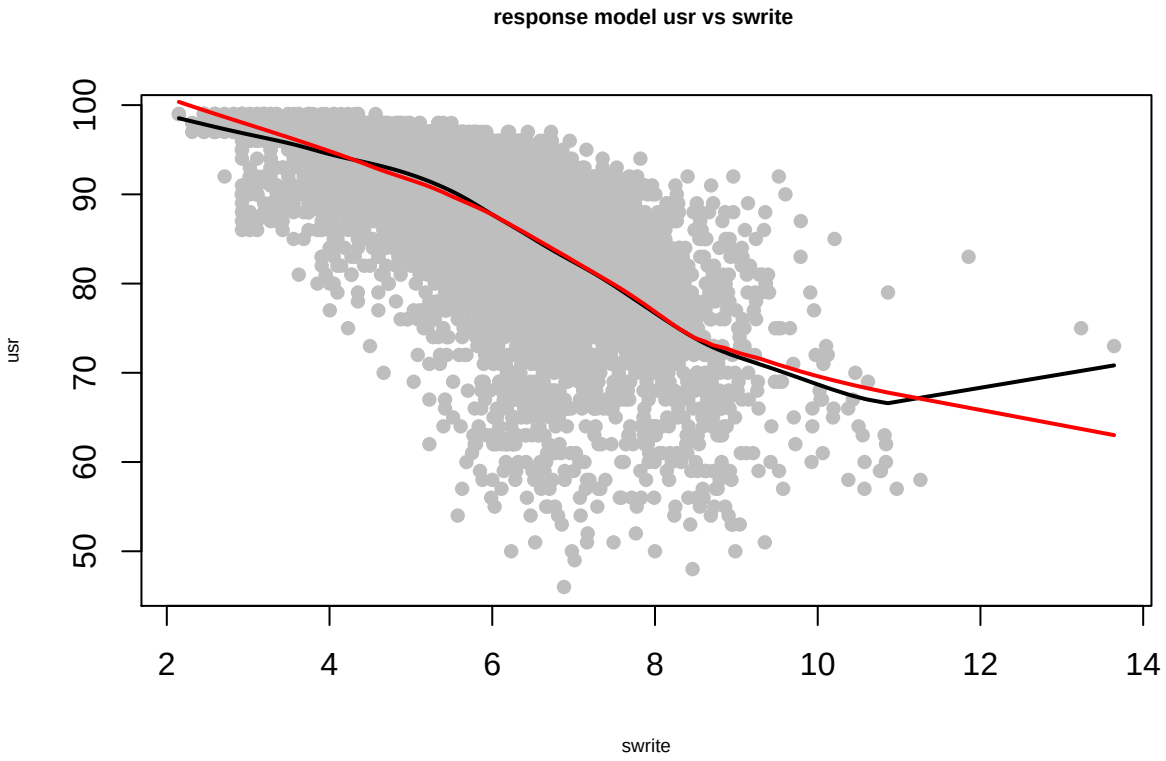


Above plot shows, usr vs lwrite with loess and smooth fits. In this case loess very oscilatory therefore only smooth is considered for this case and next we determine APSE score for smooth,

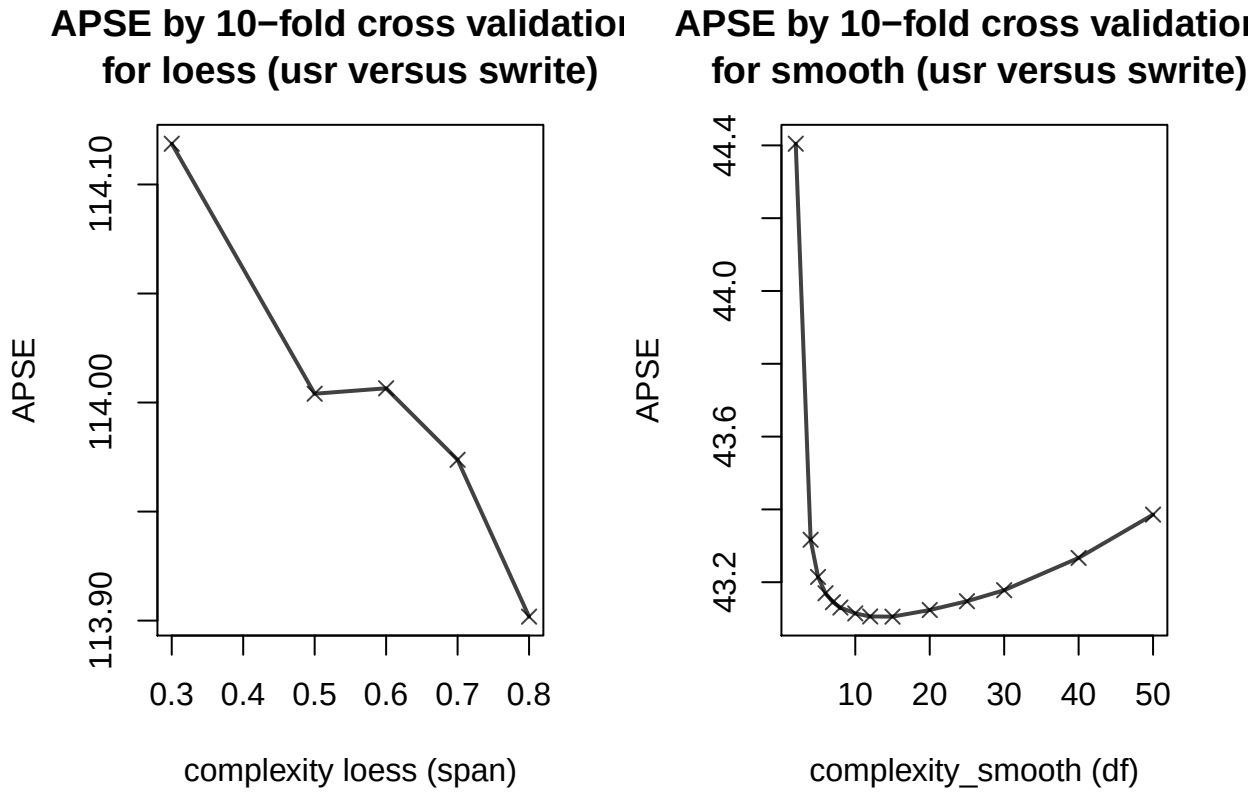


smooth with df = 25 performs better then any loess for lwrite.

Now, we consider x = swrite. Using the same analysis for this variate:



Above plot shows, usr vs swrite with loess and smooth fits. In this case loess and smooth appear almost identical except at the boundary.

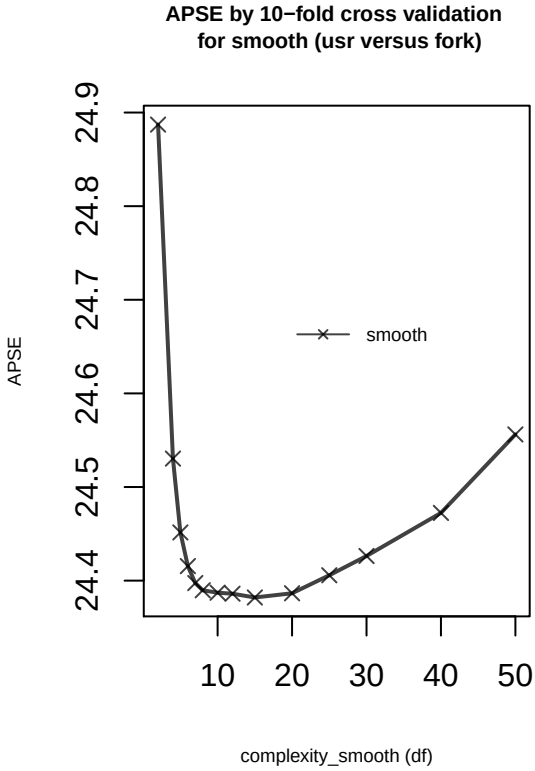


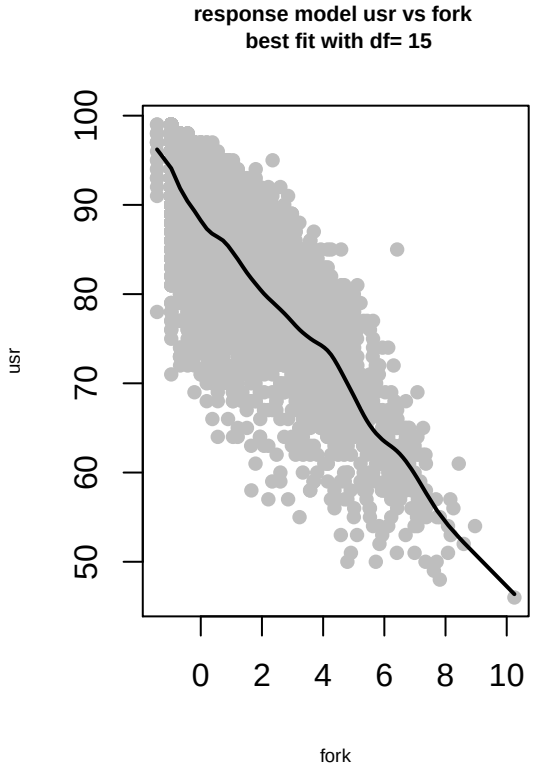
Smooth performs better in this case as well with best APSE at df = 15.

For the rest of the variates, only smooth (loess is dropped since for none of the above variates it

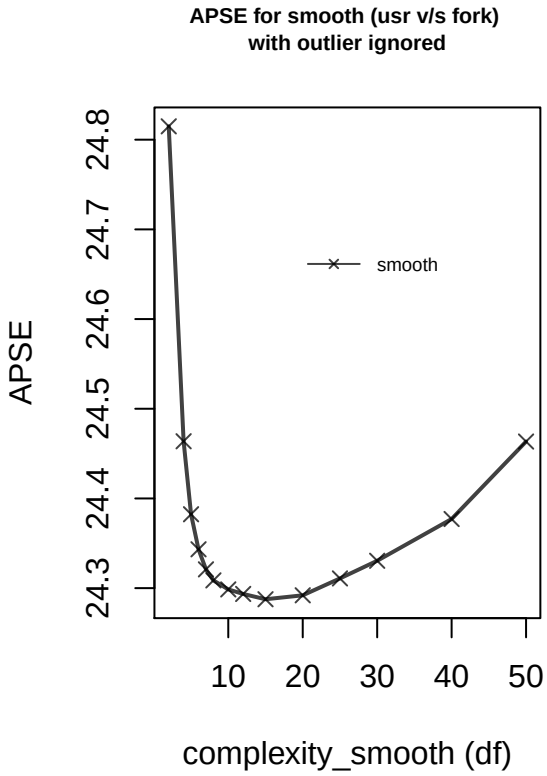
betters smooth) is considered and APSE is computed to find the optimum df for smooth.

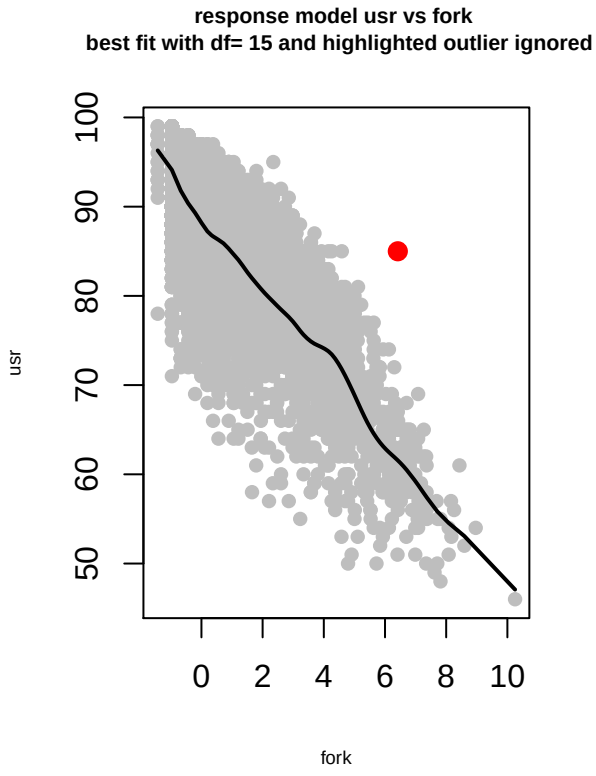
Next consider x = fork, for this variate usr decreases sharply as can be seen below.





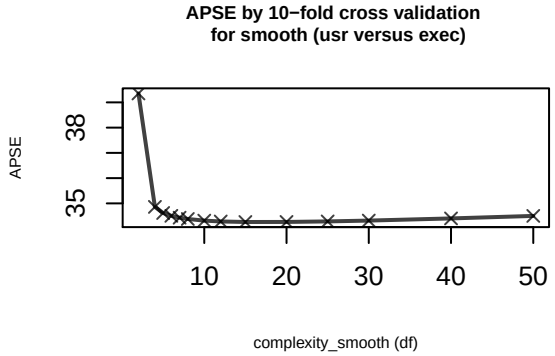
As we can see there is an outlier in the above usr ~ fork plot, so let's explore it's impact on the smooth.

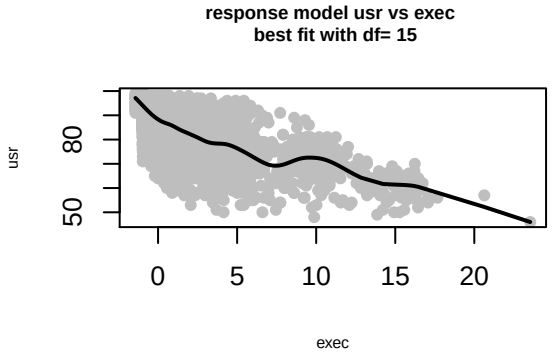


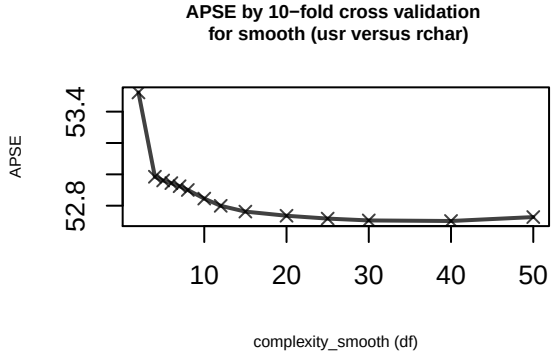


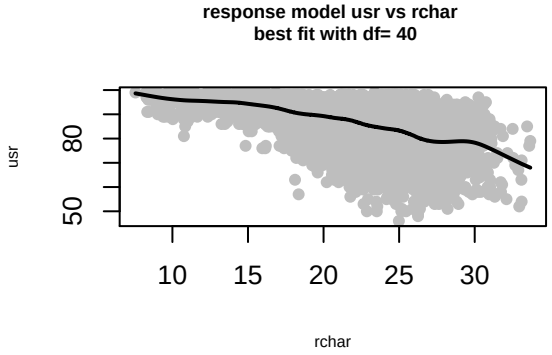
There is very insignificant difference between two smooths i.e. when highlighted outlier is considered vs when it is ignored. Also optimal df remains 15 in both cases.

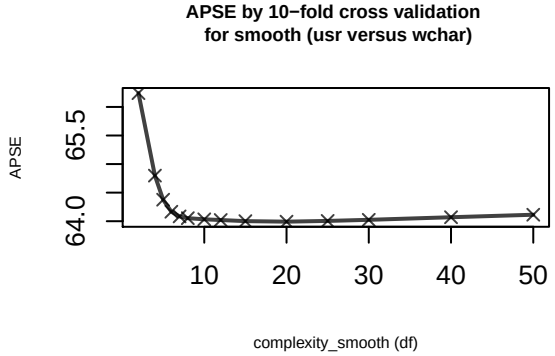
For rest of the six variables df was computed and APSE plots used in the process are shown along with the best fit smooth.

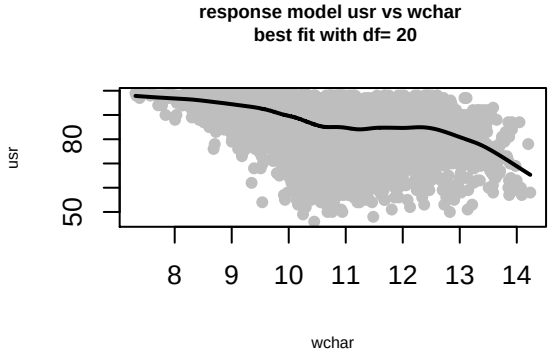


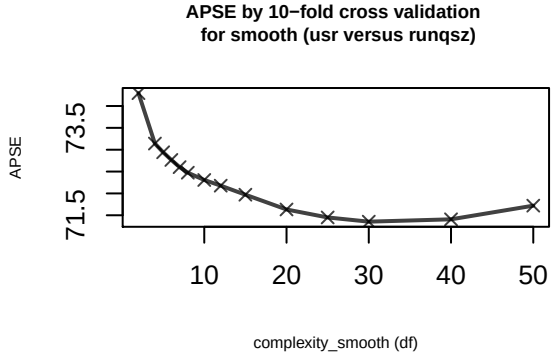


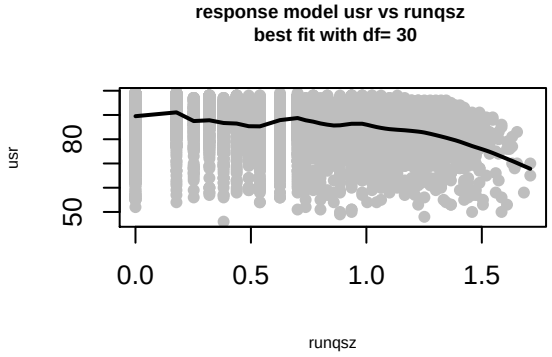


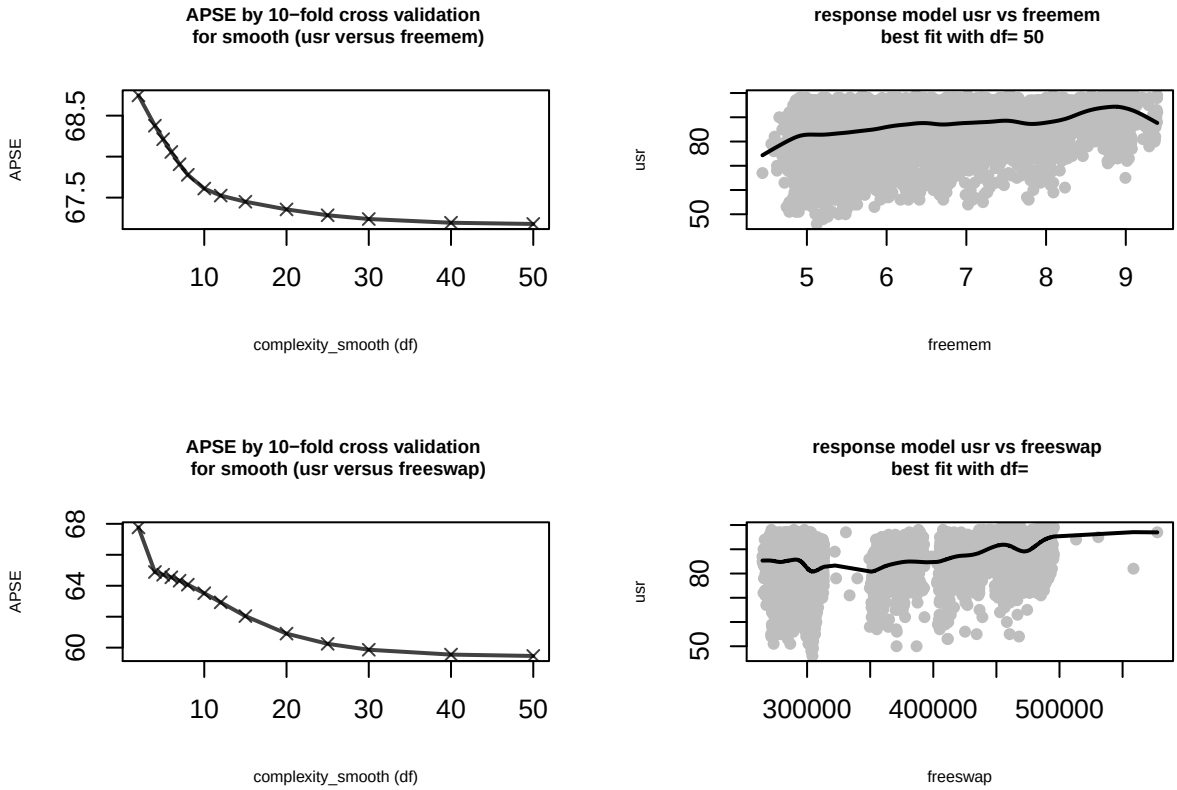












Finally, we got optimal degrees of freedom for smooth of each variate. Noticeably, for none of the variates loess did better than smoothing splines. Let's list down the final df's,

$$
\begin{aligned}
df(lread) &= 25 \\
df(lwrite) &= 25 \\
df(scall) &= 20 \\
df(sread) &= 20 \\
df(swrite) &= 25 \\
df(fork) &= 15 \\
df(exec) &= 15 \\
df(rchar) &= 40 \\
df(wchar) &= 20 \\
df(runqsz) &= 30 \\
df(freemem) &= 50 \\
df(freeswap) &= 50
\end{aligned}
$$