CrossValidationMMRE&PRED

Songyh & AtharvaKale

3/6/2018

```
raw_data <- read.csv(file = "modelsEvaluationSlack.csv", stringsAsFactors = F)
```

```
# if there is some missing value
sort(sapply(raw_data, function(x) {
            sum(is.na(x))
}), decreasing  = T)
```

```
##                  NUM                 PROJ               Effort
##                    0                    0                    0
##           Effort_ALY          Effort_Norm          Norm_Factor
##                    0                    0                    0
##                KSLOC            UEUCW_ALY           UEXUCW_ALY
##                    0                    0                    0
##            UDUCW_ALY                  UAW                  TCF
##                    0                    0                    0
##              TCF_ALY                   EF               EF_ALY
##                    0                    0                    0
##             EUCP_ALY            EXUCP_ALY             DUCP_ALY
##                    0                    0                    0
##      Effort_Norm_UCP             Path_Num          UseCase_Num
##                    0                    0                    0
##          Diagram_Num                  INT              INT_ALY
##                    0                    0                    0
##                   DM               DM_ALY                  CTRL
##                    0                    0                    0
##             CTRL_ALY               EXTIVK           EXTIVK_ALY
##                    0                    0                    0
##               EXTCLL           EXTCLL_ALY                   NT
##                    0                    0                    0
##               NT_ALY              NWT_ALY           NWT_DE_ALY
##                    0                    0                    0
##                  DET                  RET                  ILF
##                    0                    0                    0
##                  EIF                 Type            Simple_UC
##                    0                    0                    0
##           Average_UC           Complex_UC Normalized_UC_Effort
##                    0                    0                    0
```

```
which(sapply(raw_data, function(x){sum(x == 'undefined') > 0}))
```

```
##    INT     DM   CTRL EXTIVK EXTCLL     NT
##     23     25     27     29     31     33
```

```
raw_data[which(raw_data$INT == 'undefined'),'INT'] = 0
raw_data[which(raw_data$DM == 'undefined'),'DM'] = 0
raw_data[which(raw_data$CTRL == 'undefined'),'CTRL'] = 0
raw_data[which(raw_data$EXTIVK == 'undefined'),'EXTIVK'] = 0
raw_data[which(raw_data$EXTCLL == 'undefined'),'EXTCLL'] = 0
raw_data[which(raw_data$NT == 'undefined'),'NT'] = 0
```

```
raw_data[which(raw_data$NT == 'NaN'),'NT'] = 0
```

```
# check type of each column
sapply(raw_data, mode)
```

```
##                 NUM                PROJ              Effort
##           "numeric"         "character"           "numeric"
##          Effort_ALY         Effort_Norm         Norm_Factor
##           "numeric"           "numeric"           "numeric"
##               KSLOC           UEUCW_ALY          UEXUCW_ALY
##           "numeric"           "numeric"           "numeric"
##          UDUCW_ALY                 UAW                 TCF
##           "numeric"           "numeric"           "numeric"
##            TCF_ALY                  EF              EF_ALY
##           "numeric"           "numeric"           "numeric"
##            EUCP_ALY           EXUCP_ALY            DUCP_ALY
##           "numeric"           "numeric"           "numeric"
##     Effort_Norm_UCP            Path_Num         UseCase_Num
##           "numeric"           "numeric"           "numeric"
##         Diagram_Num                 INT             INT_ALY
##           "numeric"         "character"           "numeric"
##                  DM              DM_ALY                CTRL
##         "character"           "numeric"         "character"
##             CTRL_ALY             EXTIVK           EXTIVK_ALY
##           "numeric"         "character"           "numeric"
##              EXTCLL           EXTCLL_ALY                  NT
##         "character"           "numeric"         "character"
##               NT_ALY             NWT_ALY           NWT_DE_ALY
##           "numeric"           "numeric"           "numeric"
##                 DET                 RET                 ILF
##           "numeric"           "numeric"           "numeric"
##                 EIF                Type           Simple_UC
##           "numeric"         "character"           "numeric"
##          Average_UC          Complex_UC Normalized_UC_Effort
##           "numeric"           "numeric"           "numeric"
```

```
# transfer type of columns
raw_data <- transform(raw_data, INT = as.numeric(INT),
          DM = as.numeric(DM),
          CTRL = as.numeric(CTRL),
          EXTIVK = as.numeric(EXTIVK),
          EXTCLL = as.numeric(EXTCLL),
          NT = as.numeric(NT),
          Type = as.factor(Type))
```

```
# check again
sapply(raw_data, mode)
```

```
##                 NUM                PROJ              Effort
##           "numeric"         "character"           "numeric"
##          Effort_ALY         Effort_Norm         Norm_Factor
##           "numeric"           "numeric"           "numeric"
##               KSLOC           UEUCW_ALY          UEXUCW_ALY
##           "numeric"           "numeric"           "numeric"
##          UDUCW_ALY                 UAW                 TCF
```

```
##           "numeric"           "numeric"           "numeric"
##             TCF_ALY                  EF              EF_ALY
##           "numeric"           "numeric"           "numeric"
##            EUCP_ALY           EXUCP_ALY            DUCP_ALY
##           "numeric"           "numeric"           "numeric"
##     Effort_Norm_UCP            Path_Num         UseCase_Num
##           "numeric"           "numeric"           "numeric"
##         Diagram_Num                 INT             INT_ALY
##           "numeric"           "numeric"           "numeric"
##                  DM              DM_ALY                CTRL
##           "numeric"           "numeric"           "numeric"
##            CTRL_ALY              EXTIVK           EXTIVK_ALY
##           "numeric"           "numeric"           "numeric"
##              EXTCLL           EXTCLL_ALY                  NT
##           "numeric"           "numeric"           "numeric"
##              NT_ALY             NWT_ALY           NWT_DE_ALY
##           "numeric"           "numeric"           "numeric"
##                 DET                 RET                 ILF
##           "numeric"           "numeric"           "numeric"
##                 EIF                Type           Simple_UC
##           "numeric"           "numeric"           "numeric"
##          Average_UC          Complex_UC Normalized_UC_Effort
##           "numeric"           "numeric"           "numeric"
```

```
# X_data <- subset(raw_data, select = -c(NUM, PROJ, Effort, Effort_ALY, Effort_Norm, Norm_Factor))
X_data = subset(raw_data, select = c("EF","TCF","Type","KSLOC","Normalized_UC_Effort",
                      "UAW","Average_UC","RET","EXTIVK"))
Y_data <- raw_data[,"Effort"]
```

```
X_data[which(X_data$Type == 'Mobile App' | X_data$Type == 'Mobile Game'), 'type'] = 0
X_data[which(X_data$Type == 'Web App' | X_data$Type == 'web App'), 'type'] = 1
X_data[which(X_data$Type == 'Mobile&Web App'), 'type'] = 2
```

```
X_data = subset(X_data, select = -c(Type))
```

```
# scale numberic features
myscale = function(x) sqrt(sum((x - mean(x)) ^ 2) / length(x))
sx = as.matrix(scale(X_data, scale = apply(X_data, 2, myscale)))
sy = as.vector(scale(Y_data, scale = myscale(Y_data)))
```

```
# X_data <- model.matrix(~., X_data)
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.4.3
```

```
## Loaded glmnet 2.0-13
```

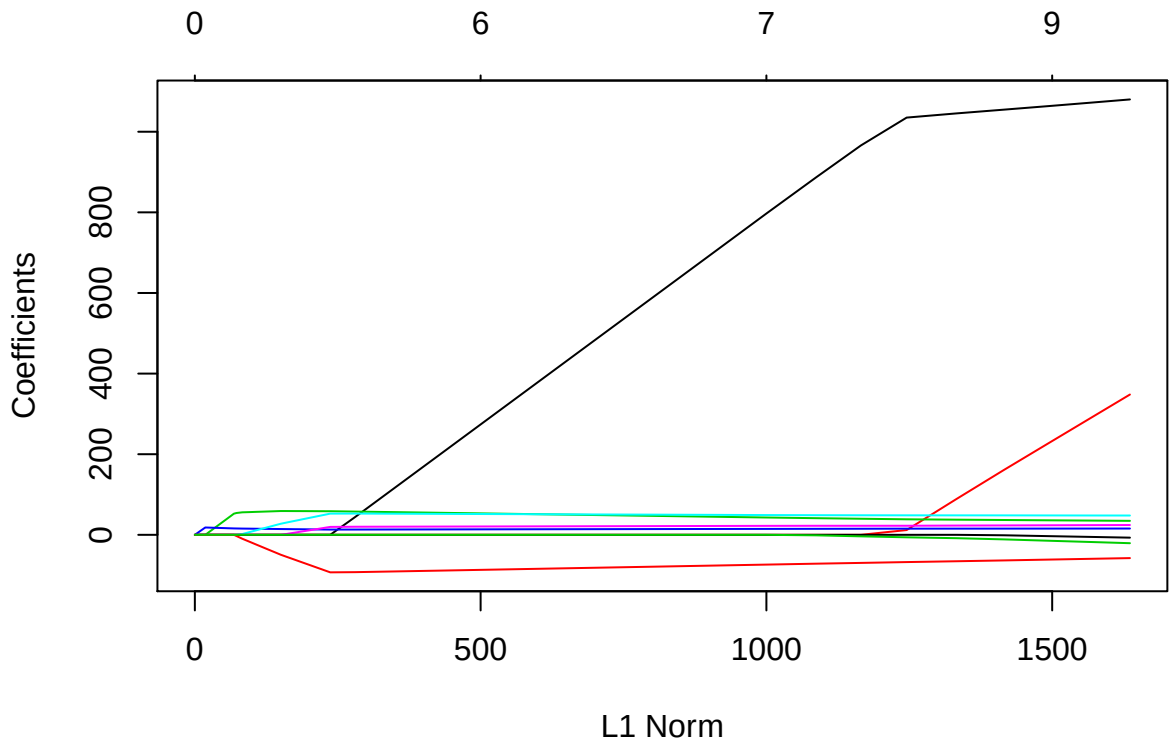
```
lasso_lm <- glmnet(x = as.matrix(X_data), y = as.vector(Y_data), alpha = 1, standardize = F)
```

```
lasso_lm$lambda
```

```
##   [1] 40228.155501 36654.397040 33398.121430 30431.124372 27727.707154
```

```
##   [6] 25264.454071 23020.029603 20974.993619 19111.632996 17413.808194
##  [11] 15866.813469 14457.249491 13172.907292 12002.662514 10936.379057
##  [16]  9964.821284  9079.574025  8272.969693  7538.021867  6868.364780
##  [21]  6258.198183  5702.237104  5195.666076  4734.097422  4313.533255
##  [26]  3930.330849  3581.171089  3263.029719  2973.151152  2709.024598
##  [31]  2468.362319  2249.079814  2049.277762  1867.225574  1701.346400
##  [36]  1550.203475  1412.487671  1287.006159  1172.672079  1068.495124
##  [41]   973.572962   887.083423   808.277377   736.472242   671.046078
##  [46]   611.432194   557.114243   507.621749   462.526032   421.436494
##  [51]   383.997237   349.883981   318.801253   290.479828   264.674401
##  [56]   241.161456   219.737337   200.216478   182.429798   166.223237
##  [61]   151.456423   138.001452   125.741783   114.571230   104.393038
##  [66]    95.119048    86.668934    78.969505    71.954071    65.561870
##  [71]    59.737534    54.430616    49.595150    45.189254    41.174765
##  [76]    37.516912    34.184013    31.147200    28.380168    25.858952
##  [81]    23.561714    21.468557    19.561349    17.823573    16.240176
##  [86]    14.797443    13.482879    12.285097    11.193723    10.199304
##  [91]     9.293226     8.467641     7.715399     7.029984     6.405460
##  [96]     5.836417     5.317925     4.845496     4.415035     4.022816
```

```
plot(lasso_lm)
```




```
#library(plotmo) # for plot_glmnet
```

```
# for 10 biggest final features
#plot_glmnet(lasso_lm)                          # default colors
#plot_glmnet(lasso_lm, label=10)
```

```r
Lasso_range = function(x, y, k){
  # inputs:
      # x, independent variables
      # y: dependent varaibles
      # k: the length of sequence
  # output:
      # seq: a sequence of lambdaa from high to low


  # define my own scale function to simulate that in glmnet
  # myscale = function(x) sqrt(sum((x - mean(x)) ^ 2) / length(x))
  #
  # # normalize x and y
  # sx = as.matrix(scale(x, scale = apply(x, 2, myscale)))
  # sy = as.vector(scale(y, scale = myscale(y)))

  sx = as.matrix(x)
  sy = as.vector(y)

  max_lambda = max(abs(colSums(sx * sy))) / dim(x)[1]
  # The default depends on the sample size nobs relative to the number of variables nvars.
  # If nobs > nvars, the default is 0.0001, close to zero.

  # If nobs < nvars, the default is 0.01.
  # A very small value of lambda.min.ratio will lead to a saturated fit in the nobs < nvars case.
  ratio = 0
  if(dim(sx)[1] > dim(sx)[2]){
    ratio = 0.0001
  }else{
    ratio = 0.01
  }

  min_lambda = max_lambda * ratio

  log_seq = seq(from  = log(min_lambda), to = log(max_lambda), length.out = k)
  seq = sort(exp(log_seq), decreasing = T)
  return(seq)
}
```

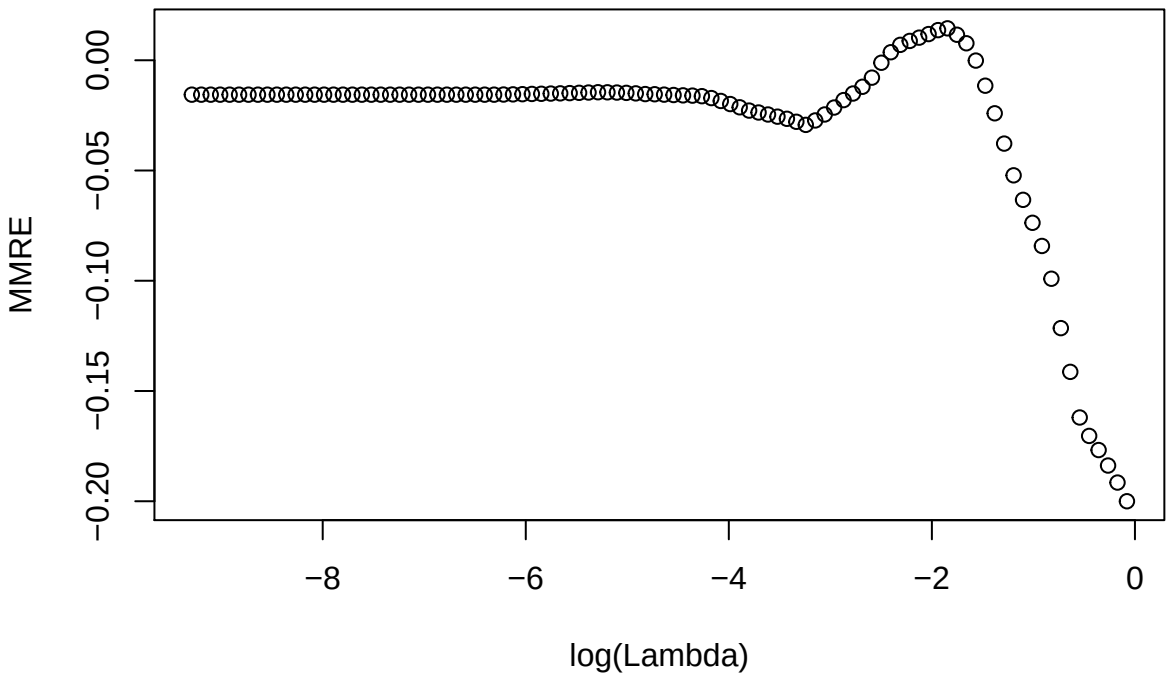
```r
Lasso_range(sx, sy, 100)
```

```
##   [1] 9.243772e-01 8.422581e-01 7.674342e-01 6.992574e-01 6.371373e-01
##   [6] 5.805358e-01 5.289626e-01 4.819710e-01 4.391540e-01 4.001408e-01
##  [11] 3.645934e-01 3.322039e-01 3.026918e-01 2.758015e-01 2.513001e-01
##  [16] 2.289753e-01 2.086338e-01 1.900993e-01 1.732114e-01 1.578238e-01
##  [21] 1.438032e-01 1.310281e-01 1.193879e-01 1.087818e-01 9.911793e-02
##  [26] 9.031257e-02 8.228945e-02 7.497908e-02 6.831815e-02 6.224895e-02
##  [31] 5.671893e-02 5.168017e-02 4.708905e-02 4.290579e-02 3.909416e-02
##  [36] 3.562114e-02 3.245665e-02 2.957330e-02 2.694609e-02 2.455227e-02
##  [41] 2.237111e-02 2.038373e-02 1.857289e-02 1.692293e-02 1.541954e-02
##  [46] 1.404971e-02 1.280157e-02 1.166432e-02 1.062809e-02 9.683921e-03
##  [51] 8.823628e-03 8.039761e-03 7.325531e-03 6.674751e-03 6.081785e-03
##  [56] 5.541496e-03 5.049204e-03 4.600647e-03 4.191938e-03 3.819538e-03
##  [61] 3.480221e-03 3.171047e-03 2.889340e-03 2.632659e-03 2.398781e-03
```

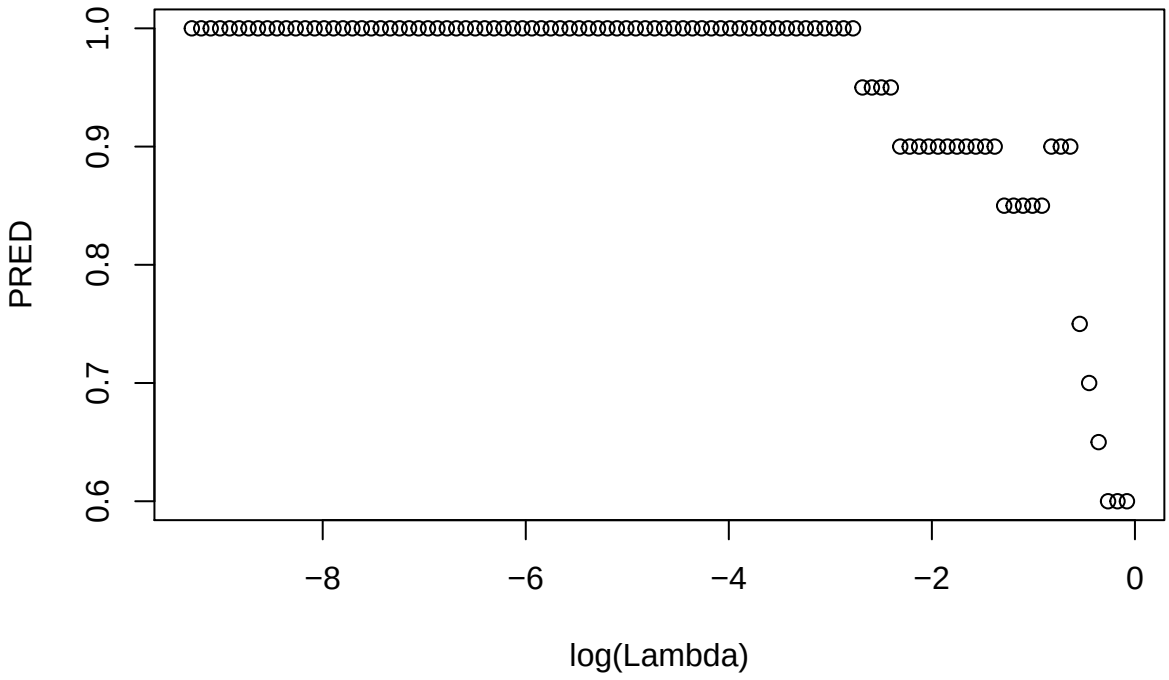
```
##  [66] 2.185680e-03 1.991510e-03 1.814590e-03 1.653387e-03 1.506504e-03
##  [71] 1.372671e-03 1.250726e-03 1.139615e-03 1.038375e-03 9.461287e-04
##  [76] 8.620772e-04 7.854927e-04 7.157117e-04 6.521298e-04 5.941964e-04
##  [81] 5.414096e-04 4.933123e-04 4.494878e-04 4.095565e-04 3.731727e-04
##  [86] 3.400210e-04 3.098145e-04 2.822914e-04 2.572134e-04 2.343633e-04
##  [91] 2.135431e-04 1.945725e-04 1.772872e-04 1.615375e-04 1.471870e-04
##  [96] 1.341113e-04 1.221972e-04 1.113416e-04 1.014503e-04 9.243772e-05
```

```
set.seed(2)
lambda_list <- Lasso_range(sx,sy,100)
percent = 50
cvfit = cv.glmnet(data.matrix(sx),sy,
                  standardize = F, type.measure = 'mse', nfolds = 5, alpha = 1)
# # 5 fold cross validation
 k <- 5
#
# function to calculate MMRE
calcMMRE <- function(testData,pred){
  mmre <- abs(testData - pred)/testData
  mean_value <- mean(mmre)
  mean_value
}
# # function to calculate PRED
calcPRED <- function(testData,pred,percent){
  value <- abs(testData - pred)/testData
  percent_value <- percent/100
  pred_value <- value <= percent_value
  mean(pred_value)
}
#
 folds <- cut(seq(1,nrow(sx)),breaks=k,labels=FALSE)
 mean_mmre <- vector("list",k)
 mean_pred <- vector("list",k)
 overall_mean_mmre <- vector("list",100)
 overall_mean_pred <- vector("list",100)
 for(iterator in seq(1,100)){
   for(i in 1:k){
     testIndexes <- which(folds==i,arr.ind=TRUE)
     testData <- sy[testIndexes]
     trainData <- sx[-testIndexes,]
     pred <- predict(cvfit,newx=data.matrix(sx),s=lambda_list[[iterator]])
     #print(paste("Iterator",iterator, i),sep=" ")
     mean_mmre[[i]] <- calcMMRE(testData,pred[testIndexes])
     mean_pred[[i]] <- calcPRED(testData,pred[testIndexes],percent)
 }
 overall_mean_mmre[[iterator]] <- mean(as.numeric(mean_mmre))
 overall_mean_pred[[iterator]] <- mean(as.numeric(mean_pred))
 #print(overall_mean_mmre[[iterator]])
 #print(overall_mean_pred[[iterator]])
 }
```

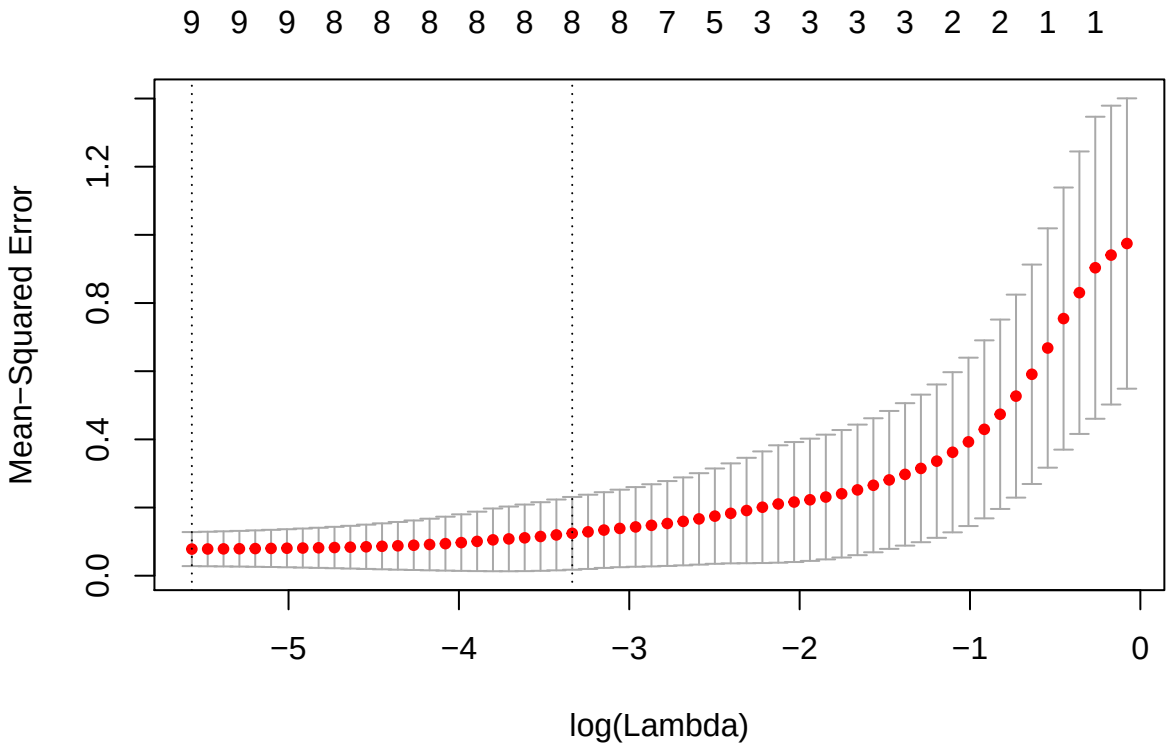
```
plot(log(lambda_list),overall_mean_mmre,xlab="log(Lambda)",ylab="MMRE")
```




```
plot(log(lambda_list),overall_mean_pred,xlab="log(Lambda)",ylab = "PRED")
```




```
plot(cvfit)
```


9 9 9 8 8 8 8 8 8 8 7 5 3 3 3 3 2 2 1 1
Mean−Squared Error
0.0
0.4
0.8
1.2
−5
−4
−3
−2
−1
0
log(Lambda)