Ridge Regression

Marcel Pons Cloquells

November 22, 2020

1 Choice of λ based on $MPSE_{val}(\lambda)$

We created the function `PMSE.VAL()` that has as inputs a matrix x and a vector y corresponding to the training sample; a matrix x_{val} and vector y_{val} corresponding to the validation set (previously centered and standardized), and a vector of lambdas of canditate values for λ. Moreover, the function has the argument *plottype*, with the options "loglambda" and "dflambda".

The main core of the function consists on a *for* loop in which for every λ of the vector of lambdas the estimation of the coefficients beta ($\hat{\beta}$) are obtained (using the training set), which are used to obtain the predicted values of the response variable for the validation set. Finally, the PMSE is obtained by computing the average square distance between the true values of the validation set and the predicted ones. Here it is a snippet of the main code of the function:

```
p <- dim(X)[2]
for (l in 1:length(lambdas)){
    lambda <- lambdas[l]
    PMSE.VAL[l] <- 0
    h.lambda.aux <- t(solve(t(X)%*%X + lambda*diag(1,p))) %*% t(X)
    betas <- h.lambda.aux %*% Y
    hat.y <- X_val %*% betas
    PMSE.VAL[l] <- sum((Y_val-hat.y)^2)/length(Y_val)
```

The function returns a vector of PMSE values for each λ in the input vector of lambdas. If the *plottype* is "loglambda", a plot representing the different PMSE values with respect to the values of *log(1+λ)* is returned. Otherwise, if the *plottype* hyperparameter is "dflambda", a plot representing the different PMSE values with respect the effective degrees of freedom (df(λ)) is returned (this option also returns the df for the λ that gives the minimum PPMSE).

As an example, a call of the function with *plottype* "loglambda" is:

```
PMSE_val <- PMSE.VAL(X,Y,Xval,Yval, lambda.v, plottype = "loglambda")
```

2 Choice of λ based on $MPSE_{k-cv}(\lambda)$

We created the function `PMSE.KCV()` that has different inputs than the previous created function. With this function, validation data is not needed since now we use different folds, which are used in each iteration as validation data or training data. It is important that the data in the input has all the variables that we want to consider and excludes the ones not needed (like *train* in Prostate).

Moreover, the input paremeter *y* is needed for specifying in which column is the response variable located. The data is centered and standardized inside the function. The other parameters for the functions are the vector of lambdas, the *plottype* and *k*, used to specify into how many folds we want to split the data (*e.g.*, 5 for 5-fold CV).

The main core of the function consists on first centering and standardizing the data. Then we randomly shuffle the data with sample() and afterwards we cut into k-folds. Then, *k* iterations are performed, where on each one a different fold is used as validation data and the others as training data. In each *k* iteration, in the inner *for* loop the PMSE is computed for each lambda (like in the previous function). When the outer *for* loop finishes, we have a matrix in which there are the values of PMSE for each lambda (columns) in k rows (for every iteration). We obtain the final PMSE vector computing the mean of each lambda for all the folds.

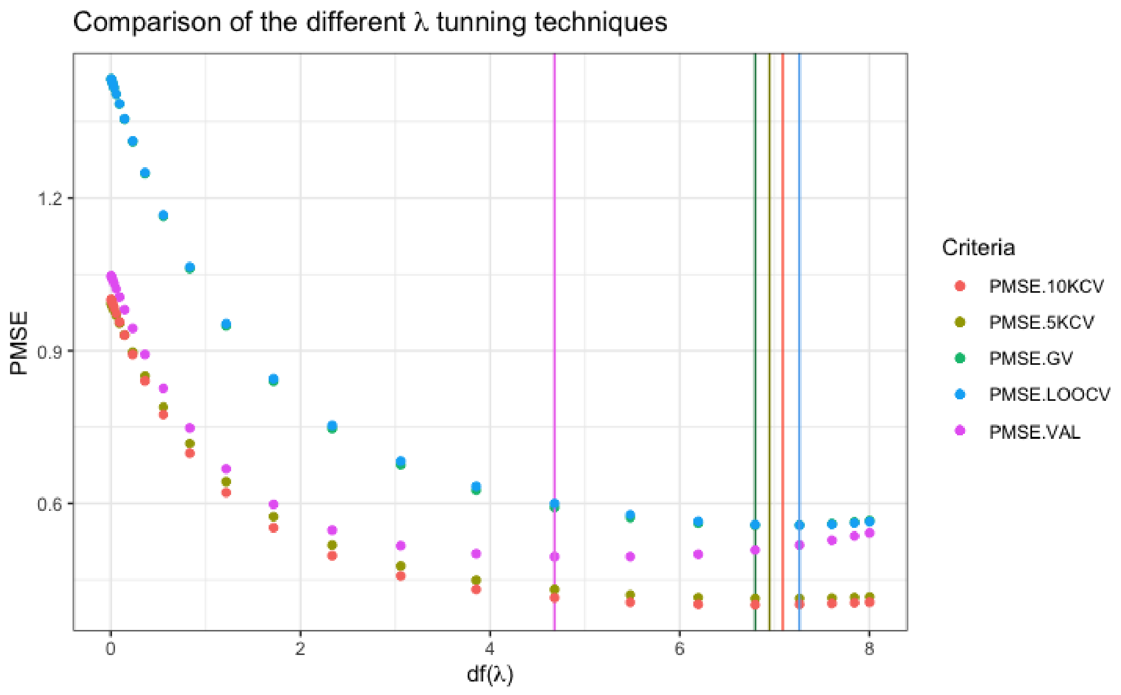
```
data[,-c(y)] <- scale(as.matrix(data[,-c(y)],center=TRUE,␣
↪scale=TRUE))
data[,y] <- scale(as.matrix(data[,y],center=TRUE, scale=FALSE))
# Shuffle data
data <- data[sample(nrow(data)),]
# Create K equally size folds
folds <- cut(seq(1,nrow(data)), breaks=K, labels=FALSE)
PMSE <- matrix(0, nrow = K, ncol = length(lambdas))

# Perform k fold cross validation
for (i in 1:K){
    # Segment your data by fold using the which() function
    testIndexes <- which(folds==i, arr.ind = TRUE)
    X <- as.matrix(data[-testIndexes, -c(y)])
    Y <- as.matrix(data[-testIndexes, y])
    X_val <- as.matrix(data[testIndexes, -c(y)])
    Y_val <- as.matrix(data[testIndexes, y])

    # Perform Regressions
    p <- dim(X)[2]
    for (l in 1:length(lambdas)){
      lambda <- lambdas[l]
      h.lambda.aux <- t(solve(t(X)%*%X + lambda*diag(1,p))) %*% t(X)
      betas <- h.lambda.aux %*% Y
      hat.y <- X_val %*% betas # predicted values y.hat
      PMSE[i,l] <- sum((Y_val-hat.y)^2)/length(Y_val)
    }
    #Mean of the folds /K
  }
  PMSE.KCV <- colMeans(PMSE)
```

The output of this PMSE.KCV function is the same as the PMSE.VAL() function (both for “loglambda” or “dflambda”). As an example, we call the function for k=5 and K=10 with *plot-type* “loglambda”:

```
PMSE.5KCV <- PMSE.KCV(prostate[,-10],K = 5, lambdas = lambda.v, y=9,␣
 ↪plottype = "loglambda")
PMSE.10KCV <- PMSE.KCV(prostate[,-10],K = 10, lambdas = lambda.v,␣
 ↪y=9, plottype = "dflambda")
```

2.1 Comparison of the different tunning criteria

With the two functions that we implemented and the routines for Leave-one-out CV and Generalized CV already available by the lecturer, we can visualize and compare the different performance of each one. To that end, we are going to plot the PMSE values for each implementation with respect to the *Effective Degrees of Freedom* (df(λ)).



From the plot it can be appreciated that the effective degrees of freedom for *10KCV*, *5KCV*, *GCV* and *LOOCV* are quite similar. On the other hand, the df(λ) of *PMSE.VAL* is the most different among all the criteria. Since df(λ) *PMSE.VAL* is smaller than the others, and knowing that the effective number of parameters (df) is a decreasing function of penalizing parameter λ, we conclude that the regression estimators following this criteria are more complex and flexible (having more possibility to over-fitting).

The difference of *PMSE.VAL* with respect to the others is possibly due to the fact that the λ that minimizes PMSE is highly conditioned on the validation set. Furthermore, this fact is accentuated even more in small datasets, like the *Prostate*, where we have 97 observations, 30 of which represent the validation set. Since the other techniques, like *LOOCV* and *KCV* use more observations as validation data, this problem is minimized (at the expanse of more computational complexity which increases the larger the dataset is).