3. Add “tuned PPR” to the CV comparison that has been used for other methods. Use the same folds as were used for the other methods. Arrange it so that in each fold, the best PPR model is chosen exactly as above and is used to produce PPR’s predictions for that fold. *This means that you will need to tune PPR exactly as in the previous problem, separately within each fold!!! (You can use 5-fold CV for tuning if you want to save a little time.* Separately save out the number of terms used in the best model for each fold.

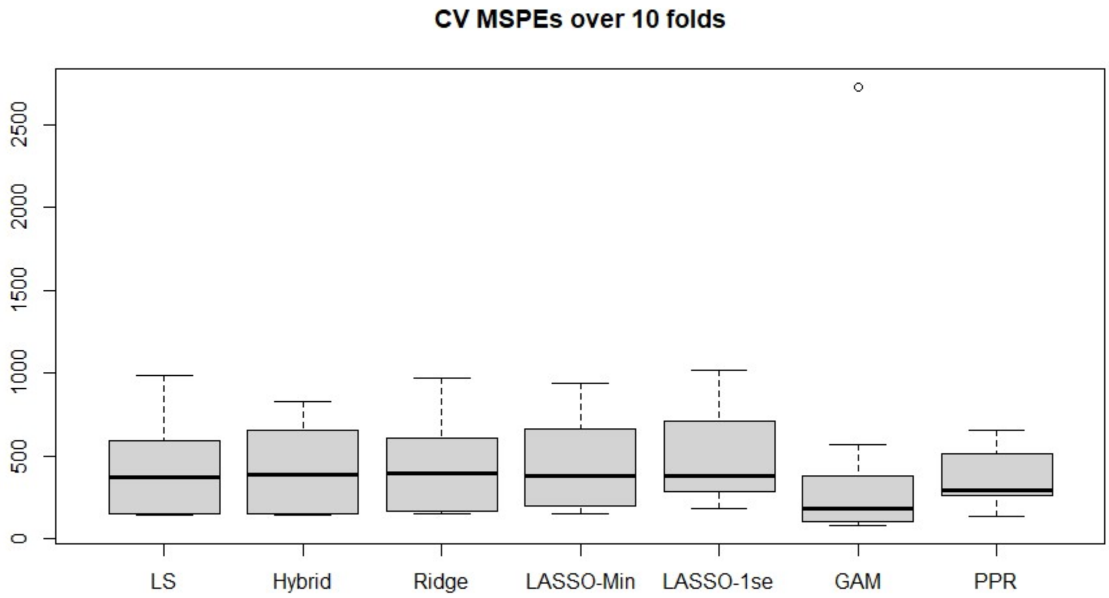
```
set.seed(50297026)
### Number of folds
K = 10

### Construct folds
n = nrow(data) # Sample size
folds = get.folds(n, K)

### Create a container for MSPEs. Let's include ordinary least-squares
### regression for reference
all.models = c("LS", "Hybrid", "Ridge", "LASSO-Min", "LASSO-1se", "GAM", "PPR")
all.MSPEs = array(0, dim = c(K, length(all.models)))
colnames(all.MSPEs) = all.models
### Begin cross-validation
for(i in 1:K){
  print(paste0(i, " of ", K))
  ### Split data
  data.train = data[folds != i,]
  data.valid = data[folds == i,]
  n.train = nrow(data.train)

  ### Get response vectors
  Y.train = data.train$Ozone
  Y.valid = data.valid$Ozone

  # LS
  fit.ls = lm(Ozone ~ ., data = data.train)
  pred.ls = predict(fit.ls, newdata = data.valid)
  MSPE.ls = get.MSPE(Y.valid, pred.ls)
  all.MSPEs[i, "LS"] = MSPE.ls

  #Hybrid Stepwise

  fit.start = lm(Ozone ~ 1, data = data.train)
  fit.end = lm(Ozone ~ ., data = data.train)

  step.BIC = step(fit.start, list(upper = fit.end), k = log(n.train),
                  trace = 0)

  pred.step.BIC = predict(step.BIC, data.valid)

  err.step.BIC = get.MSPE(Y.valid, pred.step.BIC)
```

```
  all.MSPEs[i, "Hybrid"] = err.step.BIC



  #ridge regression
  lambda.vals = seq(from = 0, to = 100, by = 0.05)

  fit.ridge = lm.ridge(Ozone ~ ., lambda = lambda.vals,
                       data = data.train)

  ind.min.GCV = which.min(fit.ridge$GCV)
  lambda.min = lambda.vals[ind.min.GCV]

  all.coefs.ridge = coef(fit.ridge)
  coef.min = all.coefs.ridge[ind.min.GCV,]

  matrix.valid.ridge = model.matrix(Ozone ~ ., data = data.valid)

  ### Now we can multiply the data by our coefficient vector. The
  ### syntax in R for matrix-vector multiplication is %*%. Note that,
  ### for this type of multiplication, order matters. That is,
  ### A %*% B != B %*% A. Make sure you do data %*% coefficients.
  ### For more information, see me in a Q&A session or, better still,
  ### take a course on linear algebra (it's really neat stuff)
  pred.ridge = matrix.valid.ridge %*% coef.min

  ### Now we just need to calculate the MSPE and store it
  MSPE.ridge = get.MSPE(Y.valid, pred.ridge)
  all.MSPEs[i, "Ridge"] = MSPE.ridge

  matrix.train.raw = model.matrix(Ozone ~ ., data = data.train)
  matrix.train = matrix.train.raw[,-1]

  ### LASSO
  all.LASSOs = cv.glmnet(x = matrix.train, y = Y.train)

  ### Get both 'best' lambda values using $lambda.min and $lambda.1se
  lambda.min = all.LASSOs$lambda.min
  lambda.1se = all.LASSOs$lambda.1se
```

```
### Get the coefficients for our two 'best' LASSO models
coef.LASSO.min = predict(all.LASSOs, s = lambda.min, type = "coef")
coef.LASSO.1se = predict(all.LASSOs, s = lambda.1se, type = "coef")

### Get which predictors are included in our models (i.e. which
### predictors have non-zero coefficients)
included.LASSO.min = predict(all.LASSOs, s = lambda.min,
                             type = "nonzero")
included.LASSO.1se = predict(all.LASSOs, s = lambda.1se,
                             type = "nonzero")

matrix.valid.LASSO.raw = model.matrix(Ozone ~ ., data = data.valid)
matrix.valid.LASSO = matrix.valid.LASSO.raw[,-1]
pred.LASSO.min = predict(all.LASSOs, newx = matrix.valid.LASSO,
                         s = lambda.min, type = "response")
pred.LASSO.1se = predict(all.LASSOs, newx = matrix.valid.LASSO,
                         s = lambda.1se, type = "response")

### Calculate MSPEs and store them
MSPE.LASSO.min = get.MSPE(Y.valid, pred.LASSO.min)
all.MSPEs[i, "LASSO-Min"] = MSPE.LASSO.min

MSPE.LASSO.1se = get.MSPE(Y.valid, pred.LASSO.1se)
all.MSPEs[i, "LASSO-1se"] = MSPE.LASSO.1se

## GAM
fit.gam = gam(Ozone ~ s(Solar.R) + s(Wind) + s(Temp) + s(TWcp) + s(TWrat),
              data = data.train)

pred.gam = predict(fit.gam, data.valid)
MSPE.gam = get.MSPE(Y.valid, pred.gam) # Our helper function
all.MSPEs[i, "GAM"] = MSPE.gam


###########
### PPR ###
###########

### To fit PPR, we need to do another round of CV. This time, do 5-fold
K.ppr = 5
n.train = nrow(data.train)
folds.ppr = get.folds(n.train, K.ppr)

### Container to store MSPEs for each number of terms on each sub-fold
MSPEs.ppr = array(0, dim = c(max.terms, K.ppr))

for(j in 1:K.ppr){
  ### Split the training data.
  ### Be careful! We are constructing an internal validation set by
  ### splitting the training set from outer CV.
  train.ppr = data.train[folds.ppr != j,]
  valid.ppr = data.train[folds.ppr == j,]
  Y.valid.ppr = valid.ppr$Ozone

  ### We need to fit several different PPR models, one for each number
  ### of terms. This means another for loop (make sure you use a different
  ### index variable for each loop).
  for(l in 1:max.terms){
    ### Fit model
    fit.ppr = ppr(Ozone ~ ., data = train.ppr,
                  max.terms = max.terms, nterms = l, sm.method = "gcvspline")

    ### Get predictions and MSPE
    pred.ppr = predict(fit.ppr, valid.ppr)
    MSPE.ppr = get.MSPE(Y.valid.ppr, pred.ppr) # Our helper function

    ### Store MSPE. Make sure the indices match for MSPEs.ppr
    MSPEs.ppr[l, j] = MSPE.ppr
  }
}

### Get average MSPE for each number of terms
ave.MSPE.ppr = apply(MSPEs.ppr, 1, mean)

### Get optimal number of terms
best.terms = which.min(ave.MSPE.ppr)

### Fit PPR on the whole CV training set using the optimal number of terms
fit.ppr.best = ppr(Ozone ~ ., data = data.train,
                   max.terms = max.terms, nterms = best.terms, sm.method = "gcvspline")
```

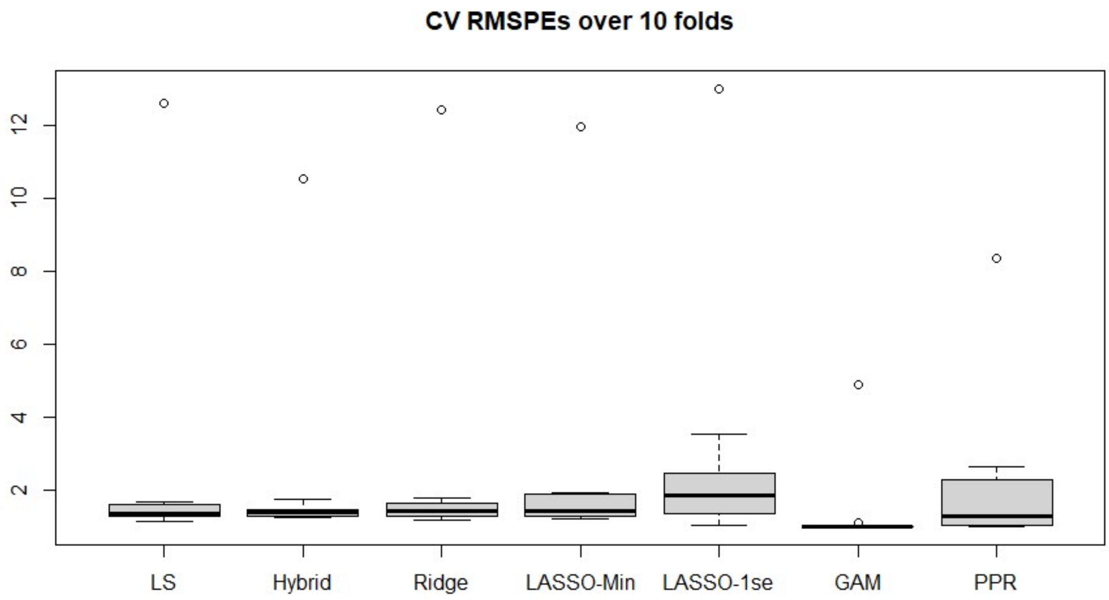
```r

  ### Get predictions, MSPE and store results
  pred.ppr.best = predict(fit.ppr.best, data.valid)
  MSPE.ppr.best = get.MSPE(Y.valid, pred.ppr.best) # Our helper function

  all.MSPEs[i, "PPR"] = MSPE.ppr.best


}

all.MSPEs

### Make a boxplot of MSPEs. I would like to include the number of folds
### in the title. This can be done by using the paste0() function,
### which concatenates strings (i.e. attaches them end-to-end), and
### can be provided numeric variables.
boxplot(all.MSPEs, main = paste0("CV MSPEs over ", K, " folds"))



### Calculate RMSPEs
all.RMSPEs = apply(all.MSPEs, 1, function(W){
  best = min(W)
  return(W / best)
})
all.RMSPEs = t(all.RMSPEs)

### Make a boxplot of RMSPEs
boxplot(all.RMSPEs, main = paste0("CV RMSPEs over ", K, " folds"))
```

(a) **Add the tuned PPR to the boxplots. Present the plots and write a sentence describing how well tuned PPR performs compared to other methods we have used thus far.**



-> It looks like PPR is better than LASSO and Hybrid.

(b) **Repeat this using relative MSPE.**



(c) **List the optimal tuning parameters that were selected for the tuned PPR in each of the 10 folds**

```
[1] "1Optimal number of terms5"
[1] "2Optimal number of terms1"
[1] "3Optimal number of terms1"
[1] "4Optimal number of terms5"
[1] "5Optimal number of terms1"
[1] "6Optimal number of terms2"
[1] "7Optimal number of terms1"
[1] "8Optimal number of terms1"
[1] "9Optimal number of terms1"
[1] "10Optimal number of terms1"
```

5, 1, 1, 5, 1, 2, 1, 1, 1, 1