homework 03

Rastko Stojsin

March 20, 2020

```
library('MASS')
library('manipulate')
library('caTools')
library('ggplot2')
library('reshape2')
library('dplyr')
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library('caret')
```

```
## Loading required package: lattice
```

```
data <- mcycle
```

1. Spliting data into training (0.75) and validation sets (0.25)

```
## seed setting for reproductability
set.seed(26)
## making group indicator
train_indicator <- sample(1:nrow(data), floor(nrow(data)) * 0.75, replace = FALSE)
train <- data[train_indicator,]
validation <- data[-train_indicator,]
```

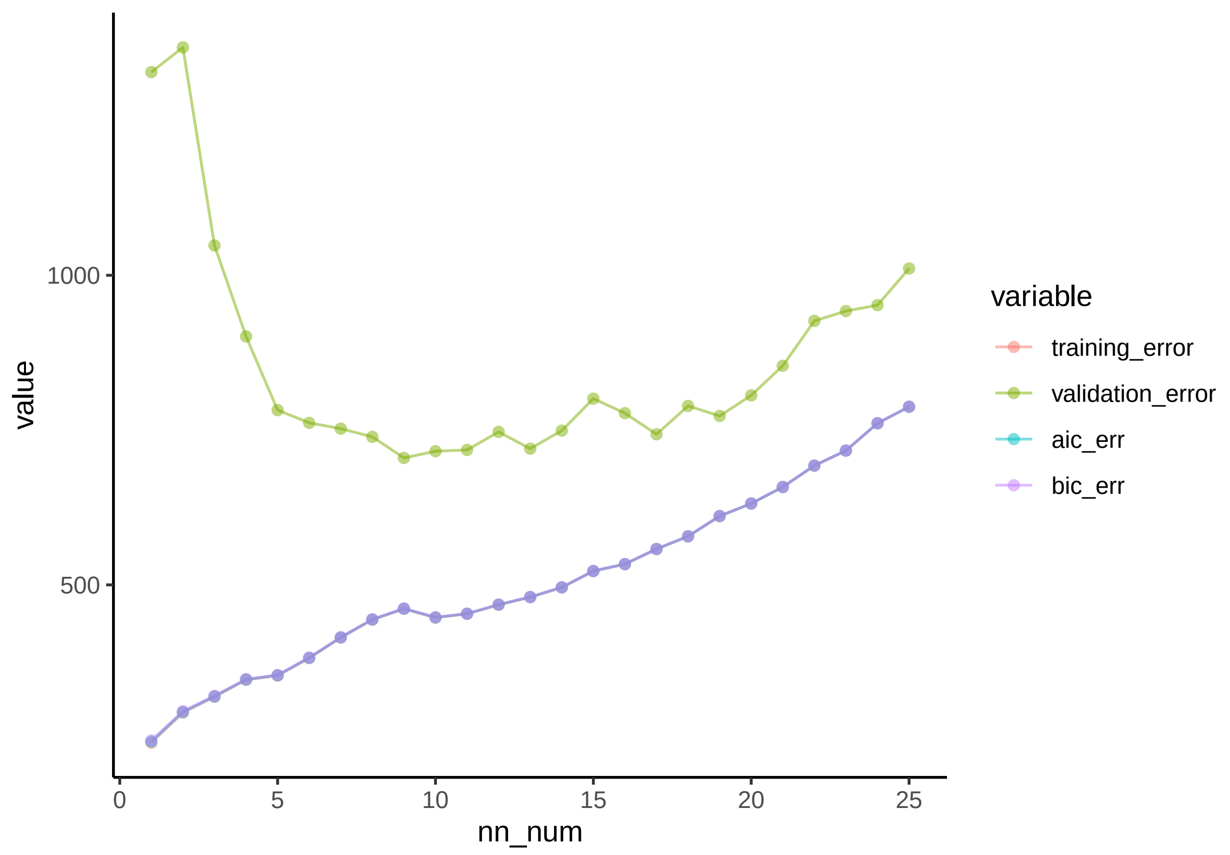
```
## k-NN kernel function
kernel_k_nearest_neighbors <- function(x, x0, k=1) {
  ## compute distance betwen each x and x0
  z <- t(t(x) - x0)
  d <- sqrt(rowSums(z*z))
```

```r
  ## initialize kernel weights to zero
  w <- rep(0, length(d))
  ## set weight to 1 for k nearest neighbors
  w[order(d)[1:k]] <- 1
  return(w) }

## NW function
nadaraya_watson <- function(y, x, x0, kern, ...) {
  k <- t(apply(x0, 1, function(x0_) {
    k_ <- kern(x, x0_, ...)
    k_/sum(k_)
  }))
  yhat <- drop(k %*% y)
  attr(yhat, 'k') <- k
  return(yhat) }

## Effective degrees of freedom function (for NW)
effective_df <- function(y, x, kern, ...) {
  y_hat <- nadaraya_watson(y, x, x,
    kern=kern, ...)
  sum(diag(attr(y_hat, 'k'))) }

## L2 loss function
loss_squared_error <- function(y, yhat) {
  (y - yhat)^2 }

## error function (L2 loss)
error <- function(y, yhat, loss=loss_squared_error) {
  mean(loss(y, yhat)) }

## AIC function
aic <- function(y, yhat, d) {
  error(y, yhat) + 2/length(y)*d }

## BIC function
bic <- function(y, yhat, d) {
  error(y, yhat) + log(length(y))/length(y)*d }
```

2 & 3. Creating prediction models at various values of k (in NW method with a k-NN kernel) and plotting training, validation, aic, and bic errors

```r
## seperating training and validation, inputs and outputs
x_train <- matrix(train$times, length(train$times), 1)
x_validation <- matrix(validation$times, length(validation$times), 1)
y_train <- matrix(train$accel, length(train$accel), 1)
y_validation <- matrix(validation$accel, length(validation$accel), 1)

## variable initilization
nn_num <- seq(1,25, by= 1)
training_error <- NA
validation_error <- NA
```

```r
aic_err <- NA
bic_err <- NA

## running models on different k values
for(i in 1:length(nn_num)) {
  ## predicting training set y values from training set x values
  y_hat_train <- nadaraya_watson(y_train, x_train, x_train,
     kern=kernel_k_nearest_neighbors, k=i)
  ## actual training set y values vs predicted ones
  training_error[i] <- error(y_train, y_hat_train)

  ## predicting validation set y values from validation set x values
  y_hat_val <- nadaraya_watson(y_train, x_train, x_validation,
     kern=kernel_k_nearest_neighbors, k=i)
  ## actual validation set y values vs predicted ones
  validation_error[i] <- error(y_validation, y_hat_val)

  ## finding dfs for models at i k
  df <- effective_df(y_train, x_train,
     kern=kernel_k_nearest_neighbors, k=i)
  ## calculating aic and bic errors
  aic_err[i] <- aic(y_train, y_hat_train, df)
  bic_err[i] <- bic(y_train, y_hat_train, df)
}

## combining outputs into table and formating for easier graphing
outputs_1 <- data.frame(nn_num, training_error, validation_error, aic_err, bic_err)
helper <- melt(outputs_1, id.vars="nn_num")

## plot of validation, training, aic, and bic errors across models with varying numbers of nearest neig
ggplot(helper, aes(x = nn_num, y = value, col = variable)) +
  geom_line(alpha = 0.5) + geom_point(alpha =0.5) + theme_classic()
```



- NOTE: the training, aic, and bic errors are so similar that they are indistingushable on the graph (print output table to confirm they are not exactly the same).

The similarity of training, aic, and bic errors indicates that aic and bic are not great actual measures of test error (super optimistic). Looking at their formulas (bic, aic) I think I see why; because we have so little data.

Also important to notice is that at this particular set seed the validation error is above the training error, but at many different seeds the validation error goes below the training error. I think (at least it makes sence to me) that this only happens if we train on hard points to predict and validate on easy ones to predict (not really sure though)?

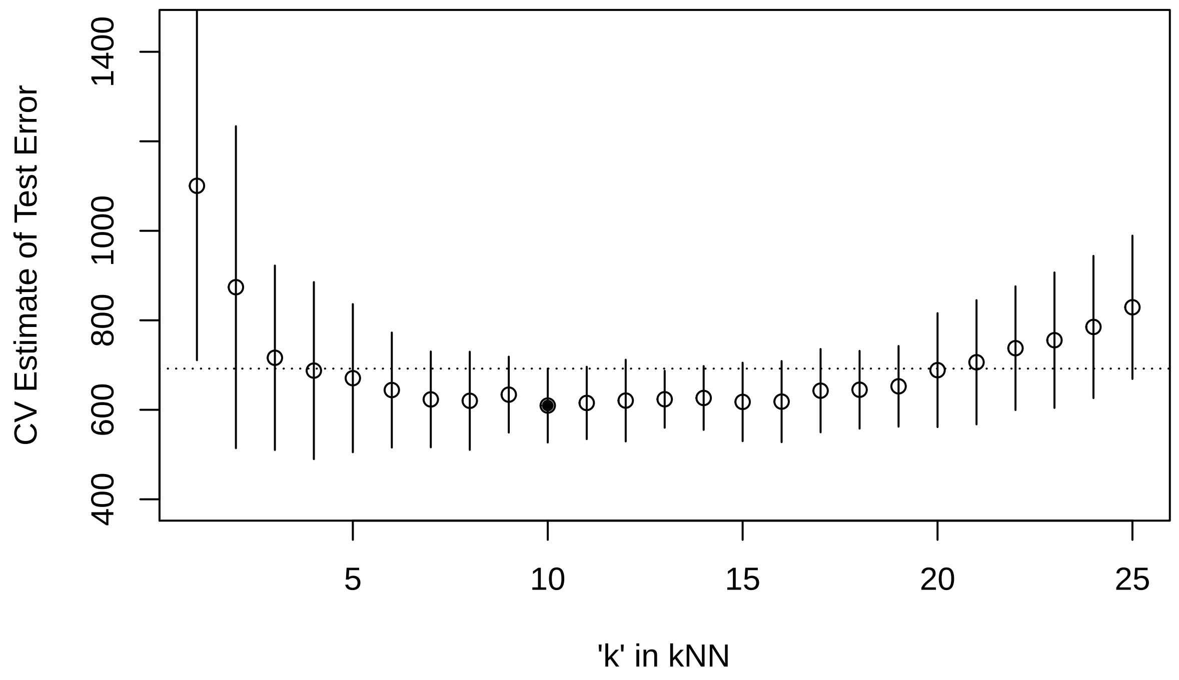
I would guess that if we had more data total that we could hedge against this.

```
cvknnreg <- function(kNN, flds=accel_folds) {
  cverr <- rep(NA, length(flds))
  for(tst_idx in 1:length(flds)) { ## for each fold

    ## get training and testing data
    inc_trn <- data[-flds[[tst_idx]],]
    inc_tst <- data[ flds[[tst_idx]],]

    ## fit kNN model to training data
    knn_fit <- knnreg(accel ~ times,
                      k=kNN, data=inc_trn)
```

```r
    ## compute test error on testing data
    pre_tst <- predict(knn_fit, inc_tst)
    cverr[tst_idx] <- mean((inc_tst$accel - pre_tst)^2)
  }
  return(cverr)
}
```

4 & 5. For each value of K preform cross fold validation (using combined test and validation data from before, as it is re-split for each fold). Then plot the cross validated error as a function of the number of K's.

```r
## create five folds using set.seed for reproducibility
set.seed(140)
accel_folds  <- createFolds(data$accel, k=5)

## Compute 5-fold CV for kNN for all values of nn_num (1 to 25)
cverrs <- sapply(nn_num, cvknnreg)
## find each K values mean and sd
cverrs_mean <- apply(cverrs, 2, mean)
cverrs_sd   <- apply(cverrs, 2, sd)

## combine outputs into table for easier graphing and sanity check
outputs_2 <- data.frame(nn_num, cverrs_mean, cverrs_sd)

## Plot the results of 5-fold CV for kNN = 1:25
plot(x = outputs_2$nn_num, y = outputs_2$cverrs_mean,
     ylim=range(cverrs),
     xlab="'k' in kNN", ylab="CV Estimate of Test Error")
segments(x0=1:max(nn_num), x1=1:max(nn_num),
         y0=cverrs_mean-cverrs_sd,
         y1=cverrs_mean+cverrs_sd)
best_idx <- which.min(cverrs_mean)
points(x=best_idx, y=cverrs_mean[best_idx], pch=20)
abline(h=cverrs_mean[best_idx] + cverrs_sd[best_idx], lty=3)
```



6. It appears that the best number of Ks to choose according to the 5 fold CV is 19. The error across the values of K is also reassuring as we would expect the actual 'test' error to be shaped like this!

It also looks like, from these graphs and for this problem, that using the a 5 fold cross validation method for parameter tuning is more appropriate than using the validation error, the aic, or the bic methods used in #2 & #3. Of course there is no way of knowing this for sure, without a final test data set for comparison.