Chapter 3 HW

Caijun Qin

10/13/2021

Install packages.

```
packages <- c('boot', 'caret', 'comprehenr', 'e1071', 'ggplot2', 'MASS')
install.packages(packages)
```

```
##
##   There is a binary version available but the source version is later:
##       binary source needs_compilation
## caret 6.0-89 6.0-90              TRUE
##
##   Binaries will be installed
## package 'boot' successfully unpacked and MD5 sums checked
## package 'caret' successfully unpacked and MD5 sums checked
## package 'comprehenr' successfully unpacked and MD5 sums checked
## package 'e1071' successfully unpacked and MD5 sums checked
## package 'ggplot2' successfully unpacked and MD5 sums checked
## package 'MASS' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\qcaij\AppData\Local\Temp\RtmpisB4V1\downloaded_packages
```

```
lapply(packages, library, character.only = TRUE)
```

```
## [[1]]
## [1] "boot"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[2]]
##  [1] "caret"     "lattice"   "ggplot2"   "boot"      "stats"     "graphics"
##  [7] "grDevices" "utils"     "datasets"  "methods"   "base"
##
## [[3]]
##  [1] "comprehenr" "caret"      "lattice"    "ggplot2"    "boot"
##  [6] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
## [11] "methods"    "base"
##
## [[4]]
##  [1] "e1071"      "comprehenr" "caret"      "lattice"    "ggplot2"
##  [6] "boot"       "stats"      "graphics"   "grDevices"  "utils"
## [11] "datasets"   "methods"    "base"
##
## [[5]]
##  [1] "e1071"      "comprehenr" "caret"      "lattice"    "ggplot2"
##  [6] "boot"       "stats"      "graphics"   "grDevices"  "utils"
## [11] "datasets"   "methods"    "base"
##
## [[6]]
##  [1] "MASS"       "e1071"      "comprehenr" "caret"      "lattice"
##  [6] "ggplot2"    "boot"       "stats"      "graphics"   "grDevices"
## [11] "utils"      "datasets"   "methods"    "base"
```

Question 1 Question 1i

Rgw dataset from `Crabs.dat` comprises of both numerical and nominal variables. The output y, color, and spine are intuitively categorical although they are presented as integers initially. These 3 variables (the last 2 being predictors) are wrapped with `factor` in the formula for statistical models. Some models such as decision trees can handle both numerical and nominal variables together naturally as it uses thresholds to split numerical variables into 2 ranges and nominal variables into 2 groups of indicator values. Other methods such as KNN classify based on a distance metric does not immediately handle nominal variables as straightforward. You can always replace nominal variables with indicator variables or one-hot encoding before running KNN.

```
# read in data
crabs <- read.table(file = 'Data/Crabs.dat', header = TRUE)
head(x = crabs)
```

```
##   y weight width color spine
## 1 1   3.05  28.3     2     3
## 2 0   1.55  22.5     3     3
## 3 1   2.30  26.0     1     1
## 4 0   2.10  24.8     3     3
## 5 1   2.60  26.0     3     3
## 6 0   2.10  23.8     2     3
```

```
length(x = crabs$y)
```

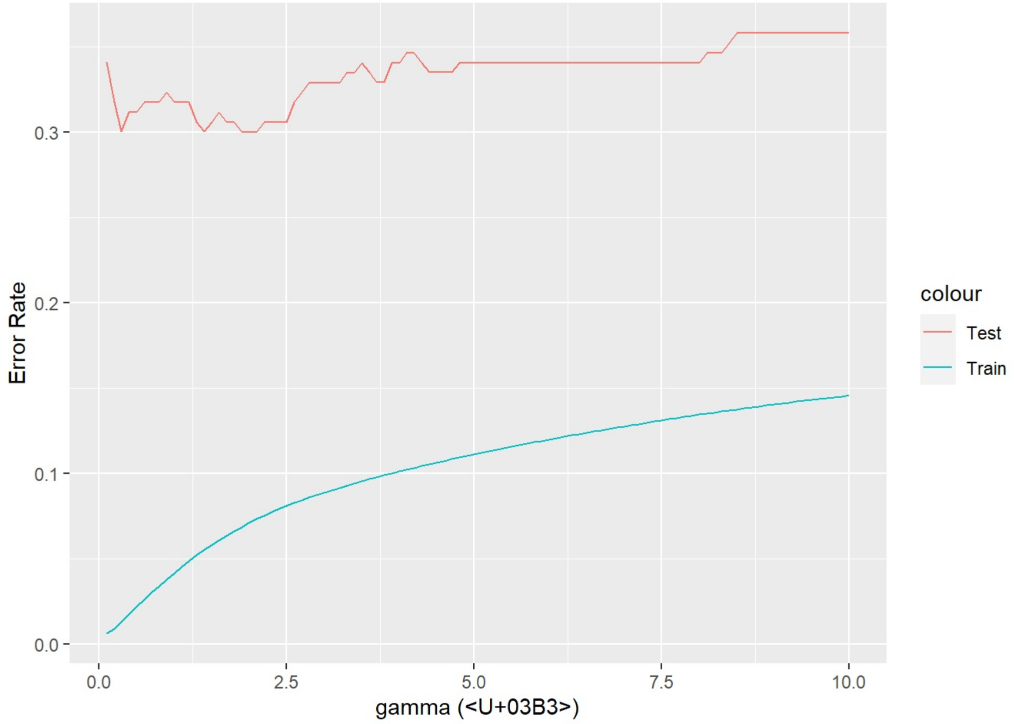
```
## [1] 173
```

```
crabs_formula <- factor(x = y) ~ factor(x = color) + factor(x = spine) + .
crabs_formula
```

```
## factor(x = y) ~ factor(x = color) + factor(x = spine) + .
```

Question 1ii

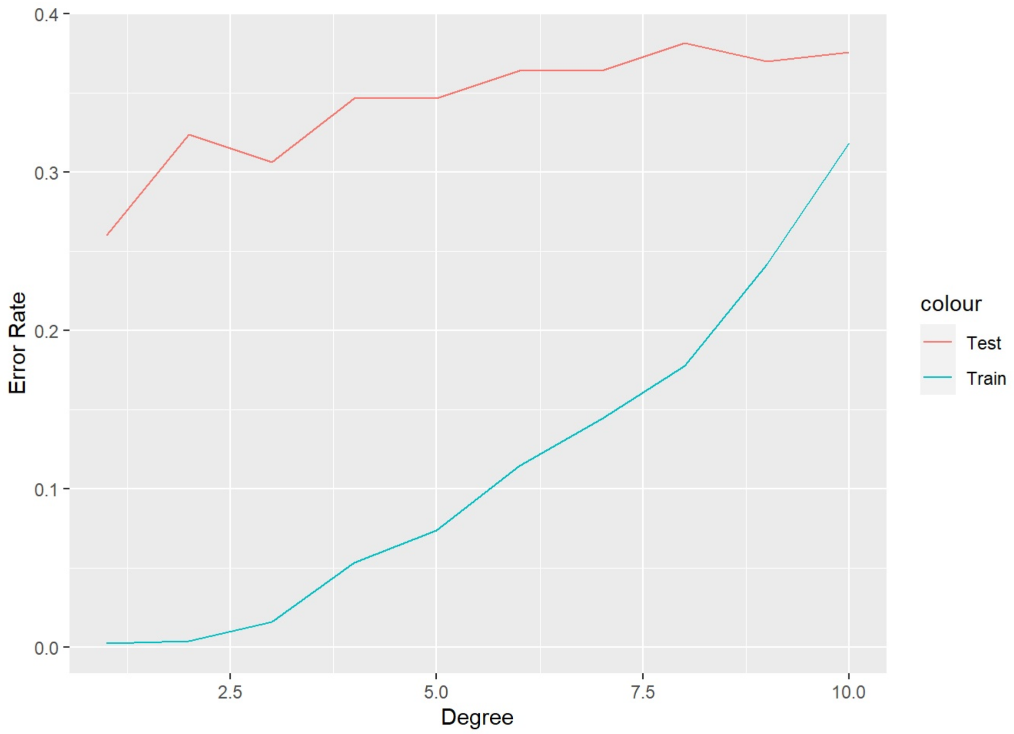
```
# radial SVM
svm_radial_tuned.test <- e1071::tune.svm(
  crabs_formula,
  data = crabs,
  kernel = 'radial',
  gamma = seq(0.1, 10.0, by = 0.1),
  tunecontrol = tune.control(cross = 2)
)

train.index <- unlist(x = svm_radial_tuned.test$train.ind$`(0.828,87]`)

svm_radial_tuned.train <- e1071::tune.svm(
  x = crabs[train.index, ],
  y = crabs[train.index, ]$y,
  crabs_formula,
  kernel = 'radial',
  gamma = seq(0.1, 10.0, by = 0.1)
)

ggplot() + geom_line(
  data = svm_radial_tuned.train$performances,
  aes(
    x = gamma,
    y = error,
    color = 'Train'
  )
) + geom_line(
  data = svm_radial_tuned.test$performances,
  aes(
    x = gamma,
    y = error,
    color = 'Test'
  )
) + xlab(label = 'gamma (γ)') + ylab(label = 'Error Rate')
```



Model starts to overfit immediately and retains a huge gap between train and test error rates with varying gamma.

Question 1iii

```
# polynomial SVM
svm_poly_tuned.test <- e1071::tune.svm(
  crabs_formula,
  data = crabs,
  kernel = 'poly',
  degree = 1:10,
  tunecontrol = tune.control(cross = 2)
)

train.index <- unlist(x = svm_poly_tuned.test$train.ind$`(0.828,87]`)

svm_poly_tuned.train <- e1071::tune.svm(
  x = crabs[train.index, ],
  y = crabs[train.index, ]$y,
  crabs_formula,
  kernel = 'poly',
  degree = 1:10
)

ggplot() + geom_line(
  data = svm_poly_tuned.train$performances,
  aes(
    x = degree,
    y = error,
    color = 'Train'
  )
) + geom_line(
  data = svm_poly_tuned.test$performances,
  aes(
    x = degree,
    y = error,
    color = 'Test'
  )
) + xlab(label = 'Degree') + ylab(label = 'Error Rate')
```



Model train and test error rates

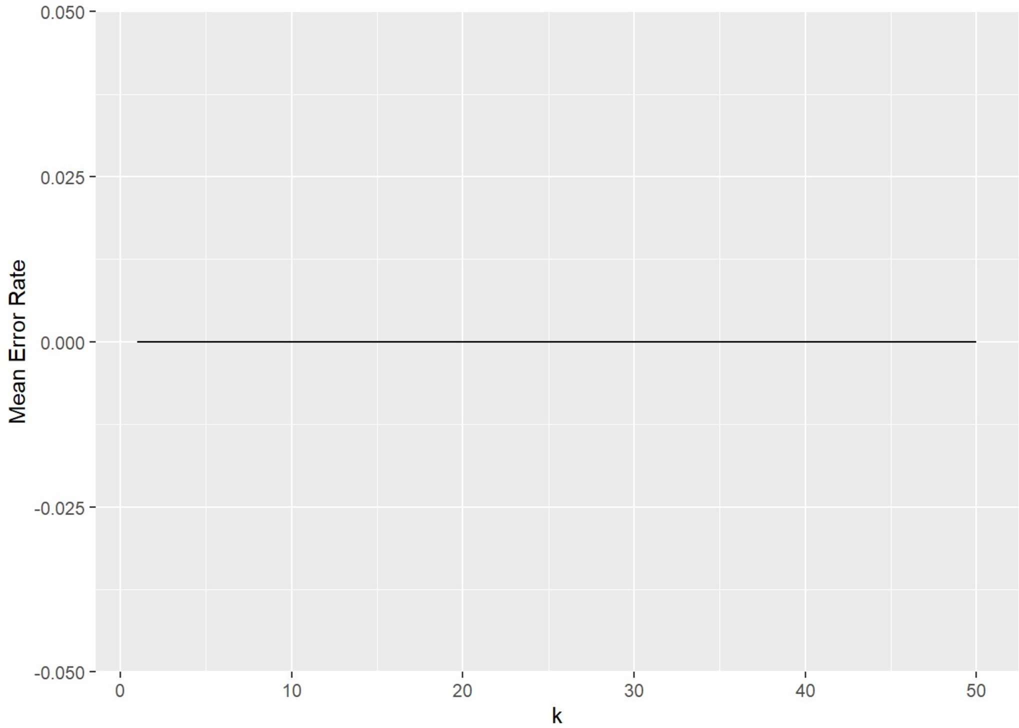
remain close together until degree >= 7. Polynomial kernel seems to be inherently unstable.

Question 1 iv

```
window <- as.integer(x = length(x = crabs$y) / 10) # sliding window to extract test indexes
k.range <- 1:50 # values of k for parameter tuning
knn.err.df <- as.data.frame(
  x = matrix(data = rep(x = 0.0, times = length(x = k.range) * 10), nrow = length(x = k.range)),
  row.names = k.range
)
colnames(knn.err.df) <- comprehenr::to_vec(for (i in 1:10) sprintf('cv%d', i))

# creates vector of error rates for a specific k for KNN with cross-validation folds
knn.cv <- function(k, cv_folds) {
  comprehenr::to_vec(
    for (cv in 1:cv_folds) {
      # print(cv)
      test.index <- c(1:window) + (cv - 1) * window
      test.data <- crabs[test.index, ]
      train.data <- crabs[-test.index, ]
      knn.model <- caret::train(
        form = crabs_formula,
        data = train.data,
        method = 'knn',
        preProcess = c("center","scale"),
        tuneGrid=data.frame(k=k)
      )
      knn.pred <- predict(object = knn.model, newdata = test.data[, -y])
      knn.pred

      cm <- caret::confusionMatrix(
        data = factor(x = knn.pred),
        reference = factor(x = test.data$y)
      )

      err.rate <- 1 - sum(... = diag(x = prop.table(cm$table)))
      err.rate
    }
  )
}

# filling up k x CV dataframe for error rates
for (k in k.range) {
  # print(k)
  knn.err.df[k, ] <- knn.cv(k = k, cv_folds = 10)
}
```

```
## Error in `[.data.frame`(test.data, , -y): object 'y' not found
```

```
knn.cv.means <- rowMeans(x = knn.err.df)
ggplot() + geom_line(aes(x = k.range, y = knn.cv.means)) + scale_x_continuous(breaks = seq(0, 100, 10)) + xlab(la
bel = 'k') + ylab(label = 'Mean Error Rate')
```



The optimal value for k appears to be 17.

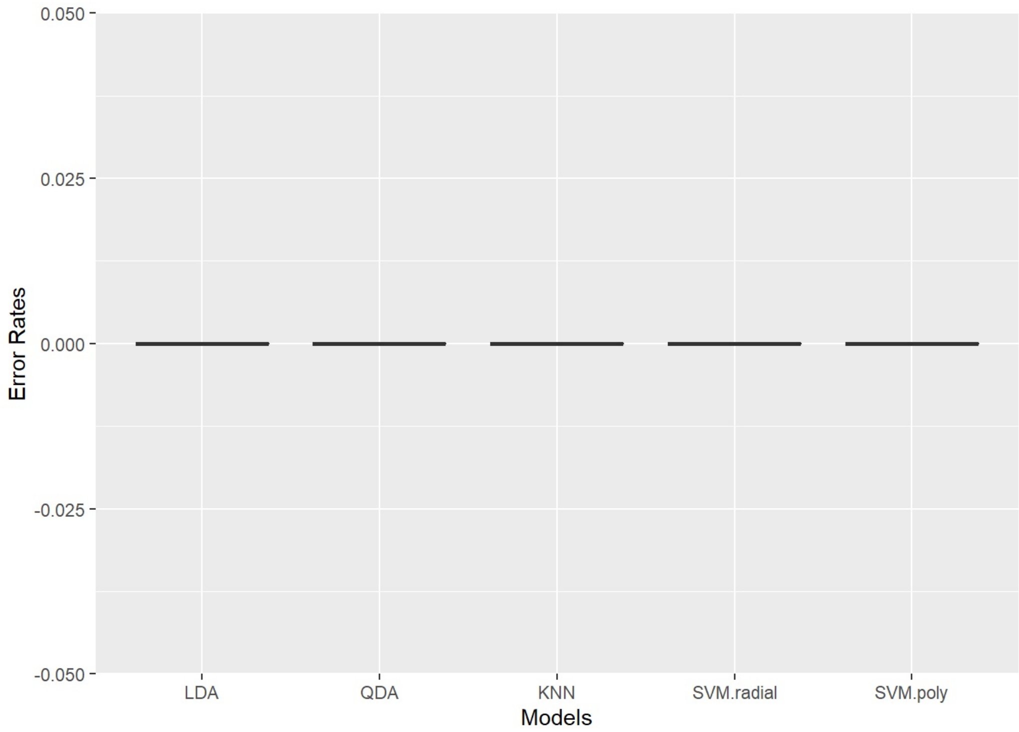
Question 1 v

```
# computes error rate
err.rate.func <- function(table) {
  1 - sum(... = diag(x = prop.table(table)))
}

model.err.df <- as.data.frame(x = matrix(data = rep(x = 0.0, times = 100 * 5), nrow = 100))
colnames(x = model.err.df) <- c('LDA', 'QDA', 'KNN', 'SVM.radial', 'SVM.poly')

# comparing multiple models
for (i in 1:100) {
  validation.index <- sample(x = c(1:length(x = crabs$y)), size = 25, replace = FALSE)
  validation.data <- crabs[validation.index, ]
  train.data <- crabs[-validation.index, ]

  # different models
  # LDA
  lda.model <- MASS::lda(formula = crabs_formula, data = train.data)

  pred <- as.numeric(
  x = predict(
    object = lda.model,
    newdata = validation.data[, -y],
    type = 'class')$class
  ) - 1

  model.err.df[i, 'LDA'] <- err.rate.func(table = caret::confusionMatrix(
        data = factor(x = pred),
        reference = factor(x = validation.data$y)
      )$table)

  # QDA
  qda.model <- MASS::qda(formula = y ~ ., data = train.data)

  pred <- as.numeric(
    x = predict(
      object = qda.model,
      newdata = validation.data[, -y],
      type = 'class')$class
    ) - 1

  model.err.df[i, 'QDA'] <- err.rate.func(table = caret::confusionMatrix(
        data = factor(x = pred),
        reference = factor(x = validation.data$y)
```

```
      )$table)

  # KNN
  knn.model <- e1071::tune.knn(x = train.data[, -y], y = as.factor(x = train.data$y), data = train.data, k = 1:50
)

  pred <- as.integer(
    x = predict(
    object = e1071::gknn(
      x = train.data[, -y],
      y = as.factor(x = train.data$y),
      k = unlist(knn.model$best.parameters)),
    newdata = validation.data[, -y],
  type = 'class')) - 1

  model.err.df[i, 'KNN'] <- err.rate.func(table = caret::confusionMatrix(
        data = factor(x = pred),
        reference = factor(x = validation.data$y)
      )$table)

  # SVM radial kernel
  svm.radial.model <- e1071::tune.svm(x = train.data[, -y], y = as.factor(x = train.data$y), kernel = 'radial', g
amma =  seq(0.1, 10.0, by = 0.1))

  pred <- as.numeric(
    x = predict(object = e1071::svm(
      x = train.data[, -y],
      y = as.factor(train.data$y),
      gamma = svm.radial.model$best.parameters
    ),
    newdata = validation.data[, -y], type = 'class')
  ) - 1

  model.err.df[i, 'SVM.radial'] <- err.rate.func(table = caret::confusionMatrix(
        data = factor(x = pred),
        reference = factor(x = validation.data$y)
      )$table)

  # SVM polynomial kernel
  svm.poly.model <- e1071::tune.svm(x = train.data[, -y], y = as.factor(x = train.data$y), kernel = 'poly', degre
e = 1:10)

  pred <- as.numeric(
    x = predict(object = e1071::svm(
      x = train.data[, -y],
      y = as.factor(train.data$y),
      degree = svm.poly.model$best.parameters
    ),
    newdata = validation.data[, -y], type = 'class')
  ) - 1

  model.err.df[i, 'SVM.poly'] <- err.rate.func(table = caret::confusionMatrix(
        data = factor(x = pred),
        reference = factor(x = validation.data$y)
      )$table)
}
```

```
## Error in `[.data.frame`(validation.data, , -y): object 'y' not found
```

```
# model.err.df

library(ggplot2)
ggplot(stack(model.err.df), aes(x = ind, y = values)) +
  geom_boxplot() + xlab(label = 'Models') + ylab(label = 'Error Rates')
```



Question 2 i

Since we know the distribution of X_i follows a uniform distribution exactly and the domain being integers confined to [0, 10], q_0.8, or the 80th percentile, is simply the CDF of the uniform distribution taken at 0.80. The value comes out to be 8.

Question 2 ii

A good estimator for q_0.8 would be the corresponding statistic, in this case 80th percentile, of the sample of X_i. Following convention, this would usually be denoted as q_0.8 ^ hat.

Question 2 iii

```
R <- 10 # number of resamples

# computes whether number is inside interval
inside.interval <- function(x, interval, inclusive = TRUE) {
  if (inclusive) {
    return (x >= min(interval) & x <= max(interval))
  }
  x > min(interval) & x < max(interval)
}

# function to compute statistic i.e. q_0.8
q_eighty <- function(data, indices) {
  bootstrap.sample <- data[indices]
  quantile(x = bootstrap.sample, probs = c(0.80))
}

# bootstrapping
boot.data <- sort(x = rep(x = c(1:10), times = 10))
boot.result <- boot::boot(data = boot.data, statistic = q_eighty, R = R)
boot.result
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot::boot(data = boot.data, statistic = q_eighty, R = R)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*      8.2     0.1   0.6749486
```

```
# plot(boot.result)

# confidence intervals
boot.conf.intervals <- boot::boot.ci(boot.out = boot.result, conf = 0.95, type = c('perc', 'norm'))
boot.conf.intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = boot.result, conf = 0.95, type = c("perc",
##     "norm"))
##
## Intervals :
## Level      Normal             Percentile
## 95%   ( 6.777,  9.423 )   ( 7.000,  9.000 )
## Calculations and Intervals on Original Scale
## Warning : Percentile Intervals used Extreme Quantiles
## Some percentile intervals may be unstable
```

Question 2 iv

```
parallel.packages <- c('parallel', 'doParallel', 'foreach', 'snow')
install.packages(parallel.packages)
```

```
## package 'doParallel' successfully unpacked and MD5 sums checked
## package 'foreach' successfully unpacked and MD5 sums checked
## package 'snow' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\qcaij\AppData\Local\Temp\RtmpisB4V1\downloaded_packages
```

```
lapply(parallel.packages, library, character.only = TRUE)
```

```
## [[1]]
##  [1] "parallel"   "MASS"       "e1071"      "comprehenr" "caret"
##  [6] "lattice"    "ggplot2"    "boot"       "stats"      "graphics"
## [11] "grDevices"  "utils"      "datasets"   "methods"    "base"
##
## [[2]]
##  [1] "doParallel" "iterators"  "foreach"    "parallel"   "MASS"
##  [6] "e1071"      "comprehenr" "caret"      "lattice"    "ggplot2"
## [11] "boot"       "stats"      "graphics"   "grDevices"  "utils"
## [16] "datasets"   "methods"    "base"
##
## [[3]]
##  [1] "doParallel" "iterators"  "foreach"    "parallel"   "MASS"
##  [6] "e1071"      "comprehenr" "caret"      "lattice"    "ggplot2"
## [11] "boot"       "stats"      "graphics"   "grDevices"  "utils"
## [16] "datasets"   "methods"    "base"
##
## [[4]]
##  [1] "snow"       "doParallel" "iterators"  "foreach"    "parallel"
##  [6] "MASS"       "e1071"      "comprehenr" "caret"      "lattice"
## [11] "ggplot2"    "boot"       "stats"      "graphics"   "grDevices"
## [16] "utils"      "datasets"   "methods"    "base"
```

```r
# TRUE or FALSE? normal CI for each specific resample covers true parameter
# boot.coverage <- function(parameter, z, std.err) {
#   comprehenr::to_vec(
#     for (t in as.vector(x = boot.result$t)) {
#       ci <- t + z * std.err * c(-1, 1)
#       inside.interval(x = parameter, interval = ci)
#     }
#   )
# }

# multiprocessing setup
ncores <- parallel::detectCores()
cluster <- parallel::makeCluster(spec = ncores - 1, type = 'SOCK', methods = FALSE)
parallel::setDefaultCluster(cl = cluster)
doParallel::registerDoParallel(cl = cluster)

nsim <- 1000 # nubmer of simulation trials

# compute proportion of percentile CIs that cover true parameter
x <- quantile(x = boot.data, probs = c(0.80))
boot.coverage.perc <- unlist(x = foreach::foreach (i = 1:nsim) %dopar% {

  # bootstrap resampling
  boot.result <- boot::boot(data = boot.data, statistic = q_eighty, R = R)

  # CIs
  boot.conf.intervals <- boot::boot.ci(boot.out = boot.result, conf = 0.95, type = c('perc', 'norm'))

  # recording whether CI for specific resampling covers true parameter
  inside.interval(
    x = x,
    interval = c(boot.conf.intervals$percent[4:5]),
    inclusive = TRUE
  )
})

# compute proportion of normal CIs that cover true parameter
boot.coverage.norm.95 <- unlist(x = foreach::foreach (i = 1:nsim) %dopar% {

  # bootstrap resampling
  boot.result <- boot::boot(data = boot.data, statistic = q_eighty, R = R)

  # CIs
  boot.conf.intervals <- boot::boot.ci(boot.out = boot.result, conf = 0.95, type = c('perc', 'norm'))

  # recording whether CI for specific resampling covers true parameter
  norm.coverage <- inside.interval(
    x = x,
    interval = c(boot.conf.intervals$normal[2:3]),
    inclusive = TRUE
  )
})

parallel::stopCluster(cl = cluster)
```

```r
# coverage of  individual CIs over true parameter
table(boot.coverage.perc)
```

```
## boot.coverage.perc
## TRUE
## 1000
```

```r
table(boot.coverage.norm.95)
```

```
## boot.coverage.norm.95
## FALSE  TRUE
##     5   995
```

Out of 1000 simulated trials, the percentile CI covers the true parameter 999 times (99.9%) while the normal CI covers the true parameter 994 times (99.4%).

Question 2 v

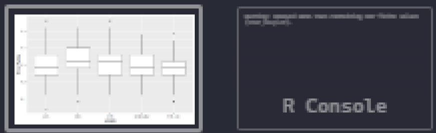
```r
# multiprocessing setup
cluster <- parallel::makeCluster(spec = ncores - 1, type = 'SOCK', methods = FALSE)
parallel::setDefaultCluster(cl = cluster)
doParallel::registerDoParallel(cl = cluster)

# function to compute statistic i.e. q_0.99
q_ninty_nine <- function(data, indices) {
  bootstrap.sample <- data[indices]
  quantile(x = bootstrap.sample, probs = c(0.99))
}

x <- quantile(x = boot.data, probs = c(0.99))
boot.coverage.norm.99 <- unlist(foreach (i = 1:nsim) %dopar% {

  # bootstrap resampling
  boot.result <- boot::boot(data = boot.data, statistic = q_ninty_nine, R = R)

  # CIs
  boot.conf.intervals <- boot::boot.ci(boot.out = boot.result, conf = 0.95, type = c('perc', 'norm'))

  # recording whether CI for specific resampling covers true parameter
  boot.coverage.perc[i] <- inside.interval(
    x = x,
    interval = c(boot.conf.intervals$percent[4:5]),
    inclusive = TRUE
  )
  boot.coverage.norm.95[i] <- inside.interval(
    x = x,
    interval = c(boot.conf.intervals$normal[2:3]),
    inclusive = TRUE
  )
})

parallel::stopCluster(cl = cluster)
```

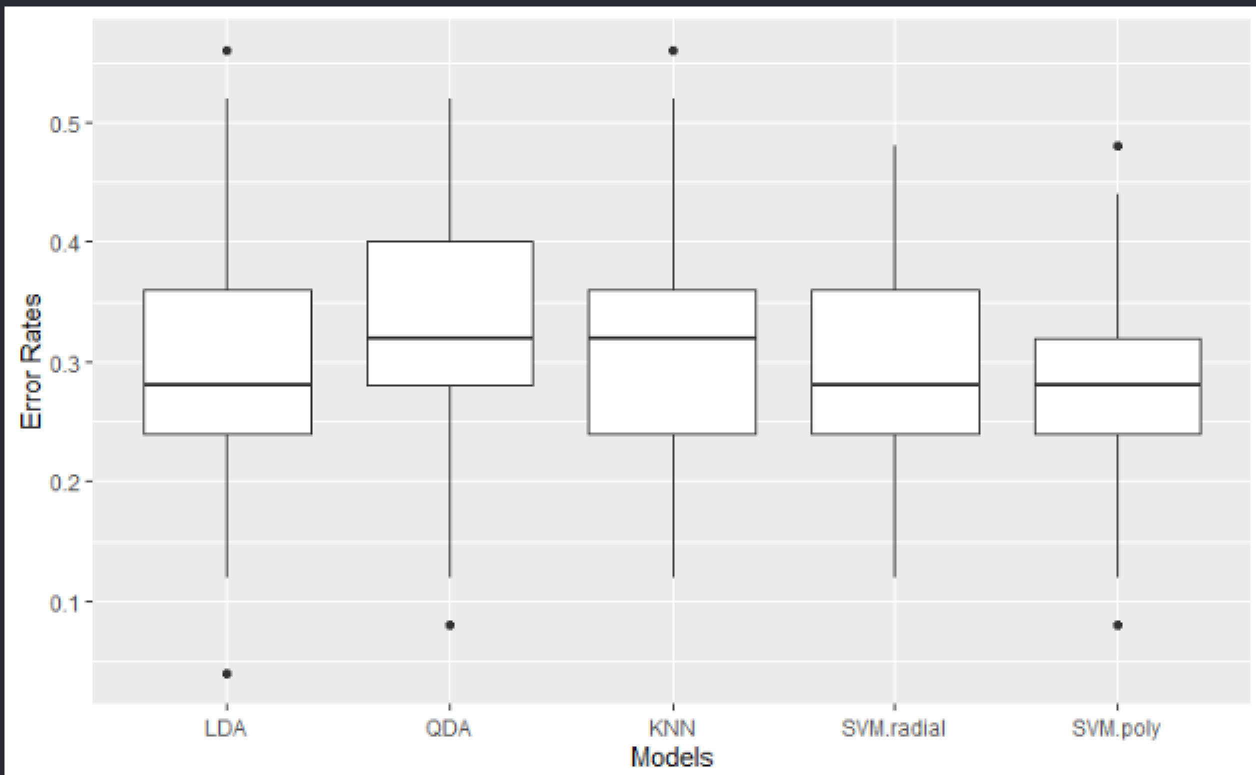
```r
# coverage of individual CIs over true parameter
table(boot.coverage.norm.99)
```

```
## boot.coverage.norm.99
## FALSE  TRUE
##   997     3
```

Only 2 out of 1000 trials (0.2%) successfully covered for q_0.99 with a CI. This drastic flip in the cover to no cover ratio compared to part iv. is due to the value of the parameter being at the very edge of the domain. Since the top 10% of the initial sample would already be X = 10, which includes the top 1% aligned with q_0.99, the histogram of parameter estimates from bootstrap resampling would look like a vertical bar concentrated at 10. The CI would essentially be collapsed into a single number. But from intuition, we understand that almost every bootstrap resampling would have a 99th percentile at 10 regardless of the code output that fooled the computer.

```{r echo = TRUE}

# model.err.df

library(ggplot2)
ggplot(stack(model.err.df), aes(x = ind, y = values)) +
  geom_boxplot() + xlab(label = 'Models') + ylab(label = 'Error Rates')

```





Given $\theta - \hat{\theta} \approx \hat{\theta} - \hat{\theta}^b$
$\Rightarrow \theta \approx 2\hat{\theta} - \hat{\theta}^b$ with some simple algebra
From percentile bootstrap, $\hat{\theta}^b$ with a 1-a confidence level has a CI of$\left(q_{\frac{\alpha}{2}}, q_{1-\frac{\alpha}{2}}\right)$.
In the 2nd line, $\hat{\theta}$ from the initial sample is fixed, and variation in $\hat{\theta}^b$ can be represented by the CI. Ensuring LB and UB are the min and max elements of the CI, $2\hat{\theta} - \hat{\theta}^b$ has a CI of $\left(2\hat{\theta} - q_{1-\frac{\alpha}{2}}, 2\hat{\theta} - q_{\frac{\alpha}{2}}\right)$.