Non Parametric Prediction Intervals For Stock Prices

Or Davidovitch

25 4 2024

```
library(glmnet)
library(ggplot2)
library(dplyr)
library(tidyr)
```

In order to write the function which returns our prediction intervals, we will need to first write a few subroutines for the whole process.

We get our data in the form of one vector which represents the closing price of the stock in each day. We need to convert it to a matrix of features, which are the stock's prices in the past m days. The next function does exactly that

```
dataCreator <- function(vec, m, T1){
  ### Returns a list containing:
  ### 1, training predictors matrix, with T1 observations and m variables
  ### 2. test predictors matrix
  ### 3. y_train
  ### 4. y_test

  K <- length(vec) - m
  dataMat <- matrix(data = 0, nrow = K, ncol = m+1)
  for(i in 1:K){
    dataMat[i, ] <- vec[i:(i+m)]
  }
  colnames(dataMat) <- c(1:m, "y")
  return(list(trainMat = dataMat[1:T1, -ncol(dataMat)],
              testMat = dataMat[(T1+1):nrow(dataMat), -ncol(dataMat)],
              y_train = dataMat[1:T1, "y"],
              y_test = dataMat[(T1+1):nrow(dataMat), "y"]))
}
```

Creating the prediction intervals requires first training a regression model on B bootstrap samples of size T. The next function returns a matrix of B rows and T columns, where each row is a random sample with replacement of 1:T:

```
bootSamplesCreator <- function(T, B){
  #### Creating a matrix where each row is a bootstrap sample on 1:T
  samplesMat <- matrix(0, nrow = B, ncol = T)
  for(b in 1:B){
    ### sample with return T observations from 1:T
    samp <- sample(1:T, size = T, replace = TRUE)
    samplesMat[b, ] <- samp
  }
  return(samplesMat)
```

```
}
```

Before we start training the models, we need to choose the regularization coefficients. We do that using 10-Fold CV.

```
chooseLambdaGamma <- function(trainMat, target){
  ### the function returns a list containing
  ### 1. lambda - the amount of regularization
  ### 2. gamma - the balance between l1 and l2 regularization

  ### Scale the data
  scaledTrainMat <- scale(trainMat)
  scaledY <- scale(target)

  cvs <- cv.glmnet(x = scaledTrainMat, y = scaledY, family = "gaussian", relax = TRUE)
  return(list(gamma = cvs$relaxed$gamma.min, lambda = cvs$relaxed$lambda.min))
}
```

Now that we have the data matrix and regularization coefficients, we can train our regression models. The next function will return a list of B trained elastic net regression models, each corresponding to a specific boostrap sample.

```
trainModels <- function(dataMat, y_train, samplesMat, opt.lambda, opt.gamma){
  ##### Returns a list of B trained elastic nets, each on a bootstrap sample of size T
  B <- dim(samplesMat)[1]
  T <- dim(samplesMat)[2]
  modelsList <- list()
  scaledDataMat <- scale(dataMat) ### scale the data first
  scaledY <- scale(y_train)
  for(b in 1:B){
    samp <- samplesMat[b, ]
    ### create bootstrap sample
    bootMat <- scaledDataMat[samp, ]
    bootY <- scaledY[samp]
    ### train the model on the bootstrap model
    modelsList[[b]] <- glmnet(x = bootMat, y = bootY, family = "gaussian",
                              lambda = opt.lambda, alpha = opt.gamma)
  }
  return(modelsList)
}
```

We can aggregate the different models in a few ways. The next function aggregates a vector of point predictions f_i by a specific given aggregation method (can be 'Mean', 'Median' or a specific quantile).

```
aggregation <- function(f_i, aggregateFunc = 'Mean'){
  #### Aggregates the results of m predictions using a specific aggregate function
  if(aggregateFunc == 'Mean'){
    p <- mean(f_i)
  } else if (aggregateFunc == 'Median'){
    p <- median(f_i)
  } else if ((aggregateFunc < 1) & (aggregateFunc > 0)){
    p <- quantile(f_i, aggregateFunc, names = FALSE)
  }
  return(p)
}
```

The next step is creating a function that will make out-of-bag predictions on the training data. At each data point i, it only looks at the models where the i'th training observation was not part of the training process (not chosen by the bootstrap sample), and aggregates them according to one of the allowed aggregation methods.

```
makePredictionsInSample <- function(modelsList, dataMat, y_train, samplesMat, aggregateFunc = 'Mean'){
  ### Creates predictions on all T training samples using out-of-bag bootstrap models
  ### returns a vector f, where f[i] is the bootsratp prediction on i'th training sample
  T <- nrow(dataMat)
  B <- length(modelsList)
  f <- rep(0, T)

  ### Models were trained with scaled Y, so we need to rescale the predictions
  yMean <- mean(y_train)
  ySd <- sd(y_train)

  ### Scale the training data
  scaledTrainMat <- scale(dataMat)
  for(i in 1:T){
    x <- scaledTrainMat[i, ]
    f_i <- c()
    for(b in 1:B){
      ### use only models that were not traind with i'th observation
      if(!(i %in% samplesMat[b, ])){
        p <- predict(modelsList[[b]], newx = x)
        f_i <- c(f_i, p)
      }
    }
    f_i <- (f_i * ySd) + yMean ### rescale the prediction
    f[i] <- aggregation(f_i, aggregateFunc) ### aggregate the predictions
  }
  return(f)
}
```

For predictions on test points, we only need to aggregate the predictions in the specific point using the last m closing prices. We need to be aware that if $s > 1$, our features might be our past predictions (until we are revealed the data from the last s days).

Now we write our main function, which will also compute the width of the prediction interval, and will return three vectors, one for the upper bound of the prediction interval, second lower bounds at each point and a third for the point predictions. It is important to notice that we can use the test data labels only after predicting the prices on s days.

```
predictionIntervals <- function(close, T, m = 5, s = 1, B = 30,
                                aggregation = 'Mean', alpha){
  ### returns a list with:
  ### 1. vector with upper part of prediction intervals on test set
  ### 2. vector with lower part of prediction intervals on test set
  ### 3. vector containing point predictions on test set

  T1 <- T - m ### number of training sample we can produce
  d <- dataCreator(close, m, T1) ### create data matrices and target vectors
  trainMat <- d$trainMat
  y_train <- d$y_train
  testMat <- d$testMat
  y_test <- d$y_test
```

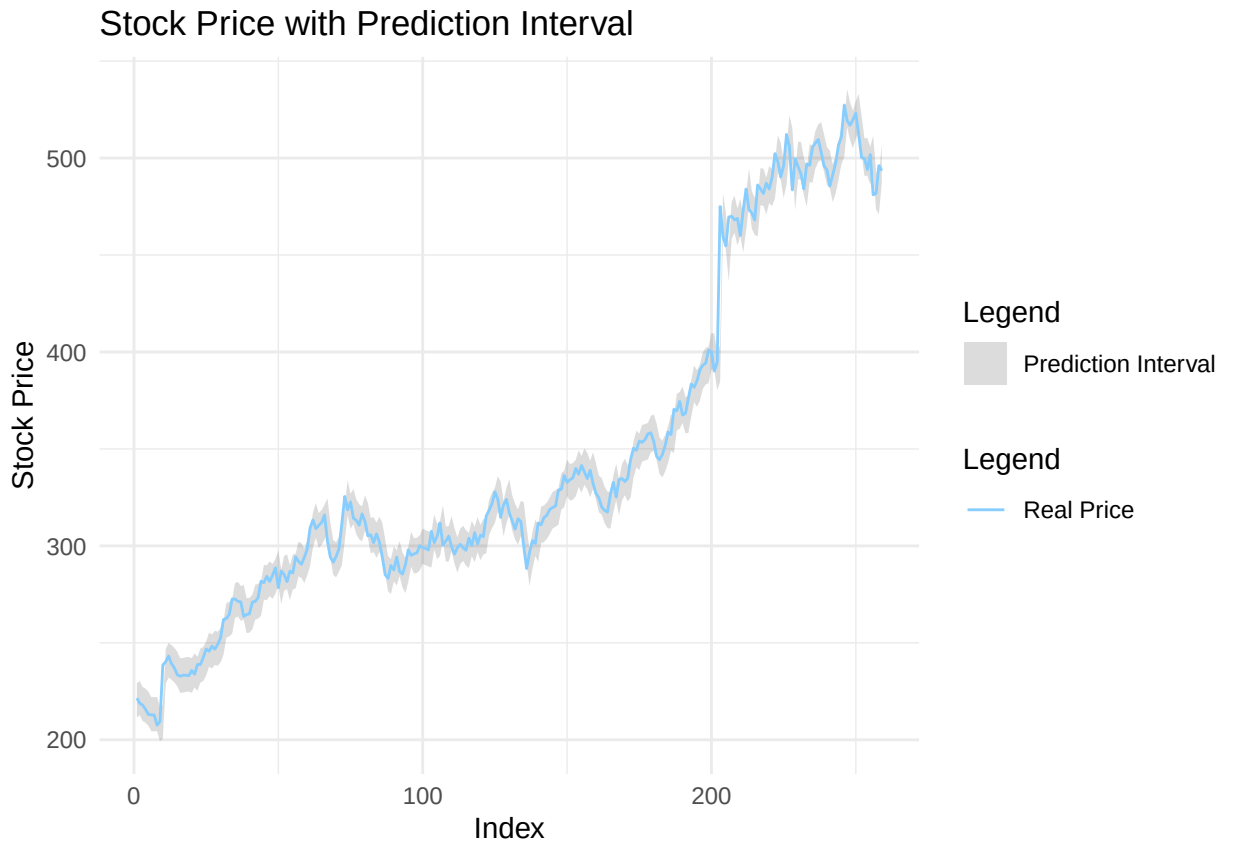
```
  e <- chooseLambdaGamma(trainMat, y_train) ### choose the regularization coefficients
  lambda <- e$lambda
  gamma <- e$gamma
  bootSampsMat <- bootSamplesCreator(T = T1, B = B) ### bootstrap matrix
  ### train the models
  modelsList <- trainModels(dataMat = trainMat, y_train = y_train,
                            samplesMat = bootSampsMat,
                            opt.lambda = lambda, opt.gamma = gamma)
  ### make predictions on the traning set to determine the width of the prediction intervals
  inSamplePredictions <- makePredictionsInSample(modelsList,
                                                 dataMat = trainMat, y_train,
                                                 samplesMat = bootSampsMat,
                                                 aggregateFunc = aggregation)
  T2 <- nrow(testMat)
  u <- rep(0, T2) ### upper bound on the prediction intervals
  l <- rep(0, T2) ### lower bound on the prediction intervals
  eps <- abs(inSamplePredictions - y_train) ### absolute values of the in sample errors
  w <- quantile(eps, 1-alpha, names = FALSE)
  pointPreds <- rep(0, T2)
  for(t in 1:T2){
    ### make prediction on test point t
    pointPreds[t] <- makePredictionsOutOfSample(modelsList, testMat,
                                                t, aggregation, trainMat, y_train)
    ### create prediction interval
    u[t] <- pointPreds[t] + w
    l[t] <- pointPreds[t] - w
    eps <- c(tail(eps, length(eps) - 1), abs(y_test[t] - pointPreds[t]))
    if (t %% s == 0){
      ### adjust width of prediction interval after we are exposed to previous observations
      w <- quantile(eps, 1-alpha, names = FALSE)
    }
  }
  return(list('Lower' = l, 'Upper' = u, 'PointPredictions' = pointPreds))
}
```

Lets try and run the function on the Meta's stock price in the past five years. We will take the first 1000 days as a training set, and try to cover the price on the next 260 days. We will use the past 5 days stock's price as the predictors, 30 bootstrap samples, and will reveal the real stock's price after every prediction (s = 1). Our aggregation strategy will be taking the mean preidction. We target achieving a coverage rate of 0.9.

```
setwd('C://Users/ordav/Desktop/ML_Projects/StockPricesPredictionIntervals/')
df <- read.csv('META.csv')
n <- 1000
m <- 7
close <- df[, 'Close']
k <- length(close)
close.diff <- tail(close, k-1) - head(close, k-1)
test <- tail(close, k-n-1)
```

```
preds <- predictionIntervals(close = close.diff, T = n, m=m, s = 1, B = 30, aggregation = "Mean", alpha

df <- data.frame(index = 1:(k-n-1), realPrice = test,
                 lower = preds$Lower + tail(head(close, k-1) , k-n-1),
                 upper = preds$Upper + tail(head(close, k-1) , k-n-1))
```

```
ggplot(df, aes(x = index)) +
  geom_ribbon(aes(ymin = lower, ymax = upper, fill = "Prediction Interval"), alpha = 0.2) +
  geom_line(aes(y = realPrice, color = "Real Price")) +
  scale_color_manual(values = c("Real Price" = "skyblue1")) +
  scale_fill_manual(values = c("Prediction Interval" = "gray32")) +
  labs(title = "Stock Price with Prediction Interval",
       x = "Index",
       y = "Stock Price",
       color = "Legend",
       fill = "Legend") +
  theme_minimal()
```




```
mean((test < df$upper) & (test > df$lower))
```

```
## [1] 0.8610039
```

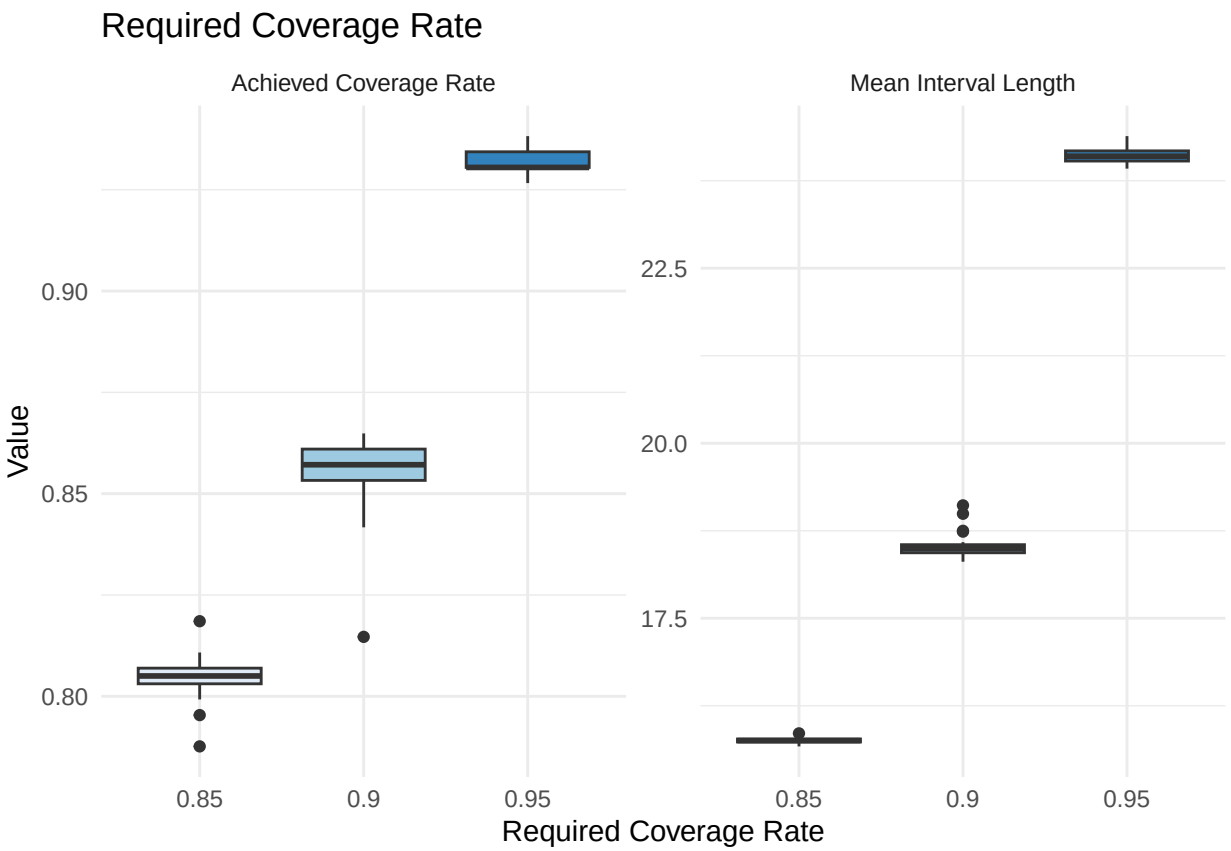
```
mean(preds$Upper - preds$Lower)
```

```
## [1] 18.49683
```

I will now try to see how the required coverage rate effects the length of the prediction interval. for each of (0.85, 0.9, 0.95) coverage rates, we will train 30 models with same parameters as before, and look at the distribution of their coverage rate on the test data, and the mean length of the prediction intervals they produce.

```
alphas <- c(0.15, 0.1, 0.05)
K <- 30
```

```r
lengths.mat <- matrix(0, ncol = length(alphas), nrow = K)
coverage.mat <- matrix(0, ncol = length(alphas), nrow = K)
colnames(lengths.mat) <- alphas
colnames(coverage.mat) <- alphas
for(a in 1:length(alphas)){
  for(i in 1:K){
    preds <- predictionIntervals(close.diff, n, m = m, alpha = alphas[a],
                                 B = 30, aggregation = 'Mean', s = 1)
    coverage.mat[i, a] <- mean((test < preds$Upper + tail(head(close, k-1) , k-n-1))
                               & (test > preds$Lower + tail(head(close, k-1) , k-n-1)))
    lengths.mat[i, a] <- mean(preds$Upper - preds$Lower)
  }
}

df <- data.frame("0.85" = coverage.mat[, 1], "0.9" = coverage.mat[, 2], "0.95" = coverage.mat[, 3])
df1 <- data.frame("0.85" = lengths.mat[, 1], "0.9" = lengths.mat[, 2], "0.95" = lengths.mat[, 3])

# Convert df to long format
df_long <- df %>%
  pivot_longer(cols = everything(), names_to = "Coverage", values_to = "Value")

# Convert df1 to long format
df1_long <- df1 %>%
  pivot_longer(cols = everything(), names_to = "Coverage", values_to = "Value")

# Fix the Coverage levels (renaming "X0.9" to "0.9", etc.)
df_long$Coverage <- factor(df_long$Coverage,
                           levels = c("X0.85", "X0.9", "X0.95"),
                           labels = c("0.85", "0.9", "0.95"))

df1_long$Coverage <- factor(df1_long$Coverage,
                            levels = c("X0.85", "X0.9", "X0.95"),
                            labels = c("0.85", "0.9", "0.95"))

# Add labels for the two metrics
df_long$Metric <- "Achieved Coverage Rate"
df1_long$Metric <- "Mean Interval Length"

# Combine both datasets
df_combined <- bind_rows(df_long, df1_long)

# Create side-by-side boxplots
ggplot(df_combined, aes(x = Coverage, y = Value, fill = Coverage)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Blues") +  # Distinct shades of blue
  facet_wrap(~ Metric, scales = "free_y") +  # Side-by-side plots
  labs(title = "Required Coverage Rate",
       x = "Required Coverage Rate",
       y = "Value") +
  theme_minimal() +
  theme(legend.position = "none")  # Remove legend
```



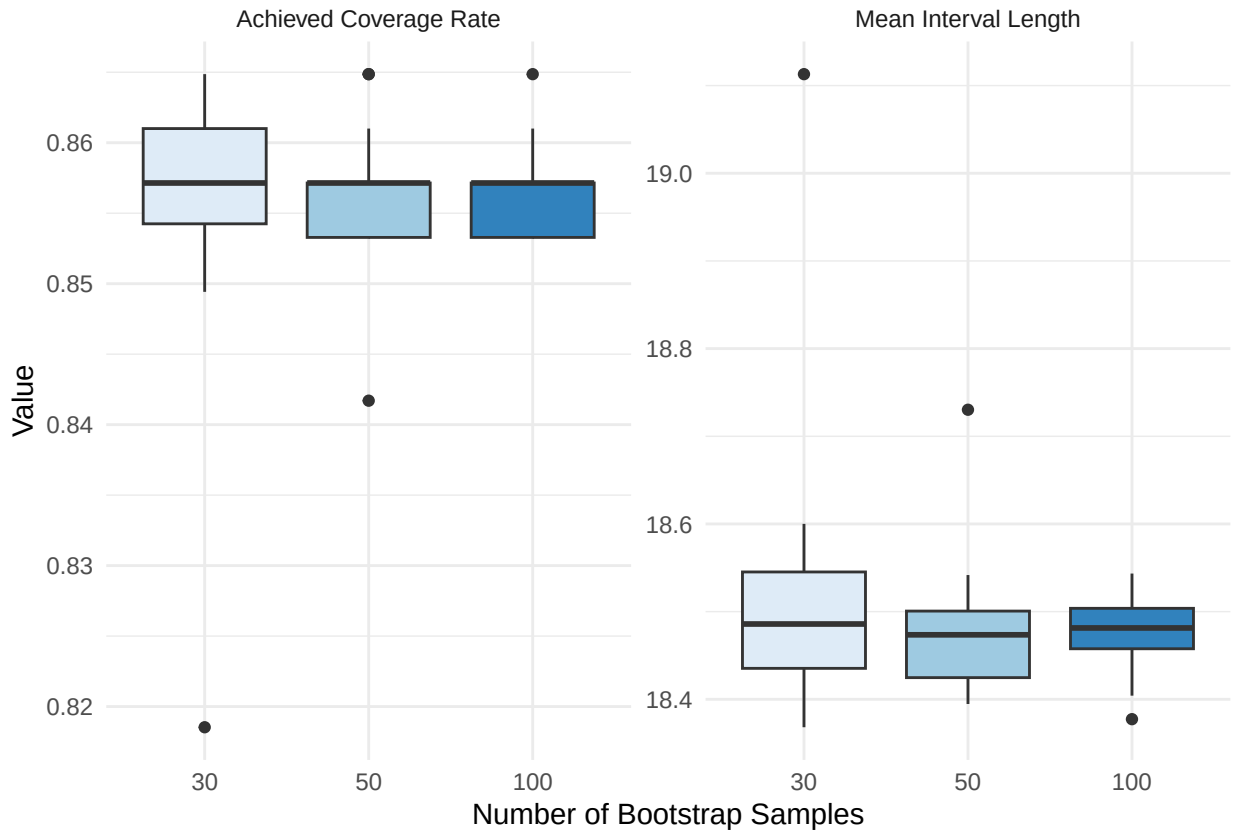
We can see that the miscoverage level we got was a bit worse than the required one for each alpha we used. This makes sense because we need to remember that we are only guaranteed an approximate level of coverage. Using a better regression model might improve our coverage level.

Not surprisingly, we see a significant increase in the mean interval length as the required miscoverage rate decreases.

We would now like to see how the number of bootstrap samples we create effect the coverage rate and the intervals' lengths. I believe that using a larger number of bootstrap samples would imply shorter prediction intervals and a higher coverage rate, since, as the theoretical proves in the article suggest, both are strictly related to the regression model's performance. Of course, increasing B comes at the cost of longer computation time.

```
B_s <- c(30, 50, 100)
lengths.mat <- matrix(0, ncol = length(B_s), nrow = K)
coverage.mat <- matrix(0, ncol = length(B_s), nrow = K)
colnames(lengths.mat) <- B_s
colnames(coverage.mat) <- B_s
for(b in 1:length(B_s)){
  for(i in 1:K){
    preds <- predictionIntervals(close.diff, n, m = m, alpha = 0.1, B = B_s[b],
                                 aggregation = 'Mean', s = 1)
    coverage.mat[i, b] <- mean((test < preds$Upper + tail(head(close, k-1) , k-n-1))
                               & (test > preds$Lower + tail(head(close, k-1) , k-n-1)))
    lengths.mat[i, b] <- mean(preds$Upper - preds$Lower)
  }
}
```

```r
df <- data.frame("30" = coverage.mat[, 1],
                 "50" = coverage.mat[, 2], "100" = coverage.mat[, 3])
df1 <- data.frame("30" = lengths.mat[, 1],
                  "50" = lengths.mat[, 2], "100" = lengths.mat[, 3])

# Convert df to long format
df_long <- df %>%
  pivot_longer(cols = everything(), names_to = "Coverage", values_to = "Value")

# Convert df1 to long format
df1_long <- df1 %>%
  pivot_longer(cols = everything(), names_to = "Coverage", values_to = "Value")

# Fix the Coverage levels (renaming "X0.9" to "0.9", etc.)
df_long$Coverage <- factor(df_long$Coverage,
                           levels = c("X30", "X50", "X100"),
                           labels = c("30", "50", "100"))

df1_long$Coverage <- factor(df1_long$Coverage,
                            levels = c("X30", "X50", "X100"),
                            labels = c("30", "50", "100"))

# Add labels for the two metrics
df_long$Metric <- "Achieved Coverage Rate"
df1_long$Metric <- "Mean Interval Length"

# Combine both datasets
df_combined <- bind_rows(df_long, df1_long)

# Create side-by-side boxplots
ggplot(df_combined, aes(x = Coverage, y = Value, fill = Coverage)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Blues") +  # Distinct shades of blue
  facet_wrap(~ Metric, scales = "free_y") +  # Side-by-side plots
  labs(x = "Number of Bootstrap Samples",
       y = "Value") +
  theme_minimal() +
  theme(legend.position = "none")  # Remove legend
```

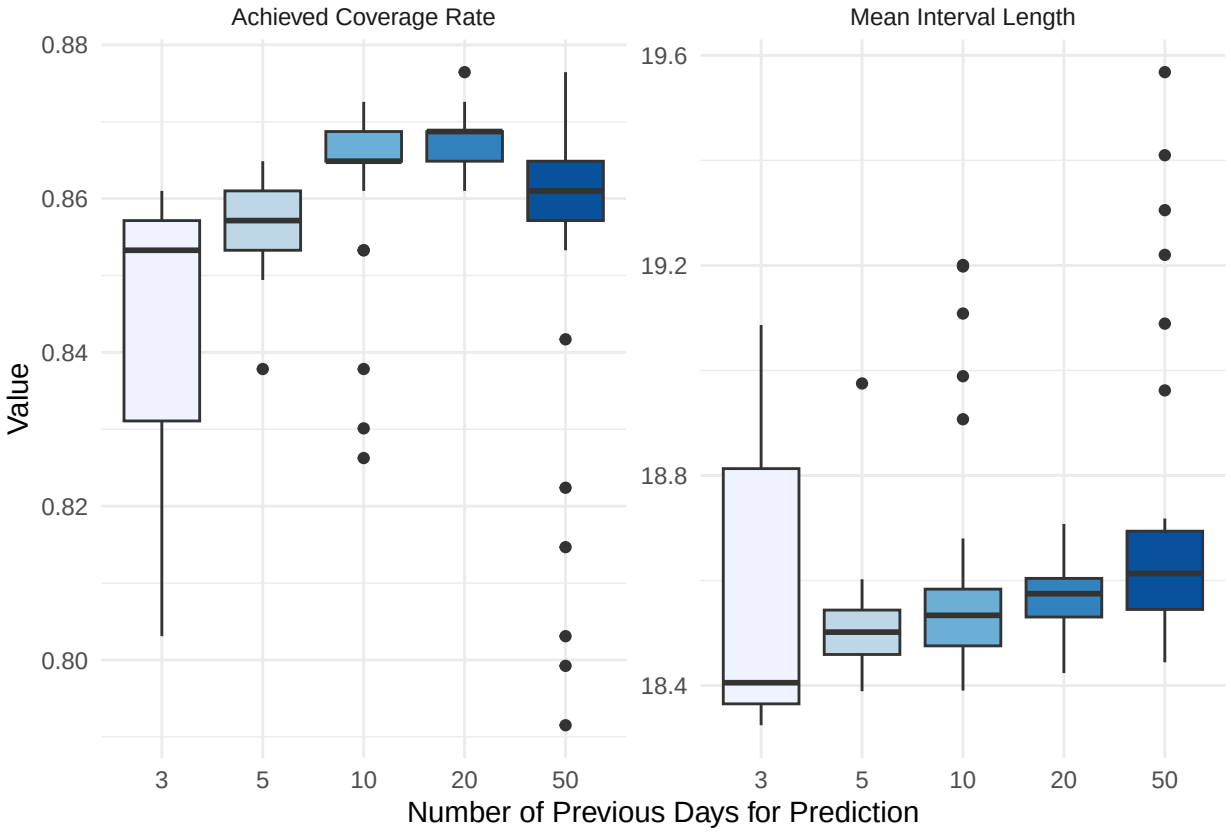


Surprisingly, we can see no actual evidence that increasing the number of bootstrap samples improves the performance of the model. The distribution of the intervals' length and coverage seems to be quite insensitive to the change in the B parameter.

Lastly, we will have a look at how the number of past days we use for prediction will effect the model's performance.

```
lengths.mat <- matrix(0, ncol = 5, nrow = K)
coverage.mat <- matrix(0, ncol = 5, nrow = K)
m.s <- c(3, 7, 10, 20, 50)
colnames(lengths.mat) <- m.s
colnames(coverage.mat) <- m.s
for(i in 1:K){
  for(j in 1:5){
    preds <- predictionIntervals(close.diff, n, m = m.s[j], alpha = 0.1, B = 30,
                                 aggregation = 'Mean', s = 1)
    coverage.mat[i, j] <- mean((test < preds$Upper + tail(head(close, k-1) , k-n-1))
                               & (test > preds$Lower + tail(head(close, k-1) , k-n-1)))
    lengths.mat[i, j] <- mean(preds$Upper - preds$Lower)
  }
}

df <- data.frame("3" = coverage.mat[, 1], "5" = coverage.mat[, 2], "10" = coverage.mat[, 3],
                 "20" = coverage.mat[, 4], "50" = coverage.mat[, 5])
df1 <- data.frame("3" = lengths.mat[, 1], "5" = lengths.mat[, 2], "10" = lengths.mat[, 3],
                  "20" = lengths.mat[, 4],  "50" = lengths.mat[, 5])
```

```r
# Convert df to long format
df_long <- df %>%
  pivot_longer(cols = everything(), names_to = "Coverage", values_to = "Value")

# Convert df1 to long format
df1_long <- df1 %>%
  pivot_longer(cols = everything(), names_to = "Coverage", values_to = "Value")

# Fix the Coverage levels (renaming "X0.9" to "0.9", etc.)
df_long$Coverage <- factor(df_long$Coverage,
                           levels = c("X3", "X5", "X10", "X20", "X50"),
                           labels = c("3", "5", "10", "20", "50"))

df1_long$Coverage <- factor(df1_long$Coverage,
                            levels = c("X3", "X5", "X10", "X20", "X50"),
                           labels = c("3", "5", "10", "20", "50"))

# Add labels for the two metrics
df_long$Metric <- "Achieved Coverage Rate"
df1_long$Metric <- "Mean Interval Length"

# Combine both datasets
df_combined <- bind_rows(df_long, df1_long)

# Create side-by-side boxplots
ggplot(df_combined, aes(x = Coverage, y = Value, fill = Coverage)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Blues") +  # Distinct shades of blue
  facet_wrap(~ Metric, scales = "free_y") +  # Side-by-side plots
  labs(x = "Number of Previous Days for Prediction",
       y = "Value") +
  theme_minimal() +
  theme(legend.position = "none")  # Remove legend
```



We can see that using the stock prices from prior 10 or 20 days gives the highest coverage rate, without increasing the intervals length significantly.