Non Parametric Prediction Intervals For Stock Prices

Or Davidovitch

25 4 2024

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-7
```

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(train, m){
  T <- length(train) - m
  dataMat <- matrix(data = 0, nrow = T, ncol = m)
  for(i in 1:T){
    dataMat[i, ] <- train[i:(i+m-1)]
  }
  return(dataMat)
}
```

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(X, y, 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){
    samp <- sample(1:T, size = T, replace = TRUE)
    samplesMat[b, ] <- samp
  }
  return(samplesMat)
}
```

We need to train our regression model on each of the B bootstrap samples. First, we need to choose the appropriate regularization coefficient to use. Our next function returns a regularization coefficient which is chosen using cross validation.

```
chooseLambda <- function(X, y){
  ### Uses 5-fold CV to choose optimal regularizaion coefficient for ridge regression
  a <- cv.glmnet(X, y)
  return(a$lambda.min)
}
```

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

```
trainModels <- function(X, y, samplesMat, T, B, lambda){
  ##### Returns a list of B trained ridge regression, each on a bootstrap sample of size T
  modelsList <- list()
  for(b in 1:B){
    samp <- samplesMat[b, ]
    bootMat <- X[samp, ]
    boot_y <- y[samp]
    modelsList[[b]] <- glmnet(x = bootMat,  y = boot_y, alpha = 0, lambda = lambda)
  }
  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, X, samplesMat, T, B, aggregateFunc = 'Mean'){
  ### Creates predictions on all T training samples using out-of-bag bootstrap models
  f <- rep(0, T)
  for(i in 1:T){
    f_i <- c()
    for(b in 1:B){
      if(!(i %in% samplesMat[b, ])){
        x <- X[i, ]
        p <- predict(modelsList[[b]], newx = t(x))
        f_i <- c(f_i, p)
      }
    }
    f[i] <- aggregation(f_i, aggregateFunc)
  }
  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).

```
makePredictionsOutOfSample <- function(modelsList, samplesMat, x,
                                       aggregateFunc, B){
  #### returns point prediction for the test points using the appropriate
  #### aggregate function on the B predictions for each bootstrap model.
  f <- rep(0, T)
  for(i in 1:T){
    f_i <- c()
    for(b in 1:B){
      if(!(i %in% samplesMat[b, ])){
        p <- predict(modelsList[[b]], newx = t(x))
        f_i <- c(f_i, p)
      }
    }
    f[i] <- aggregation(f_i, aggregateFunc)
  }
  return(aggregation(f, 0.95))

}
```

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.

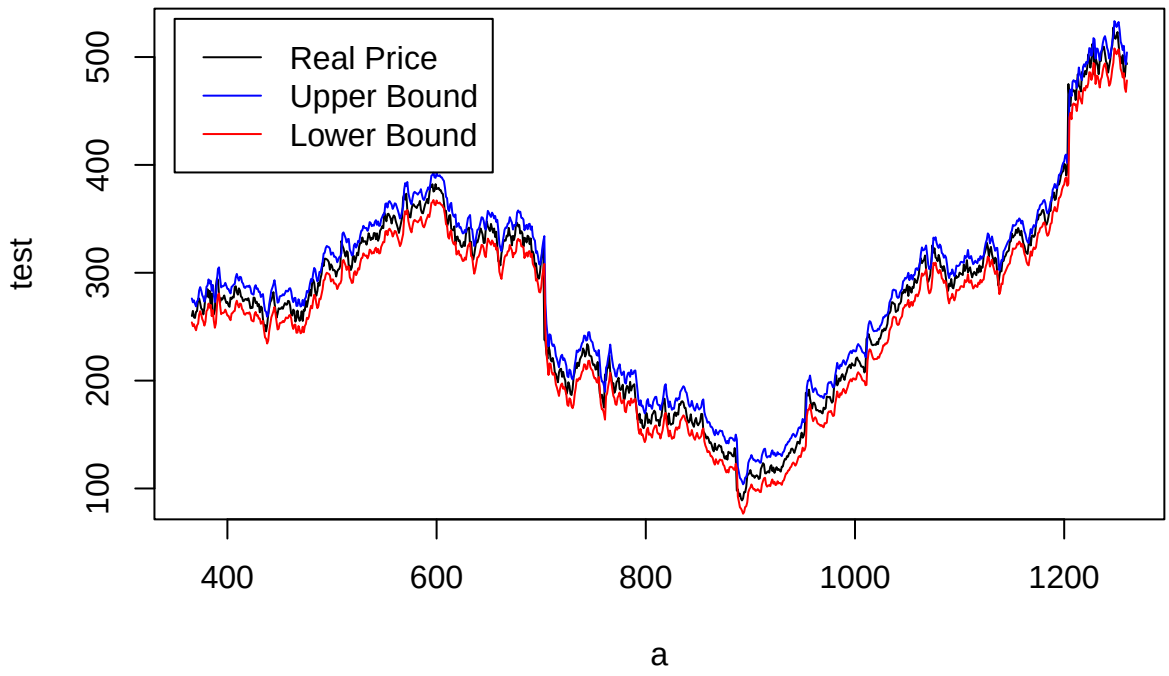
```
predictionIntervals <- function(train, test, m = 5, s = 1, B = 30,
                                aggregation = 'Mean', alpha){
  T1 <- length(test)
  y_train <- train[(m+1):length(train)]
  u <- rep(0, T1)
  l <- rep(0, T1)
  T <- length(train) - m
  eps <- c()
  dataMat <- dataCreator(train, m)
  lambda <- chooseLambda(dataMat, y_train)
  bootSampsMat <- bootSamplesCreator(dataMat, y_train, T, B)
  modelsList <- trainModels(dataMat, y_train, bootSampsMat, T, B, lambda)
  inSamplePredictions <- makePredictionsInSample(modelsList, dataMat, bootSampsMat, T, B, aggregation)
  eps <- abs(inSamplePredictions - y_train)
  w <- quantile(eps, 1-alpha, names = FALSE)
  pointPreds <- rep(0, T1)
  train.test <- c(train, test)
  start <- T+1
  x <- train.test[start:(start+m-1)]
  for(t in 1:T1){
    pointPreds[t] <- makePredictionsOutOfSample(modelsList, bootSampsMat, x,
                                                aggregation, B)
    u[t] <- pointPreds[t] + w
    l[t] <- pointPreds[t] - w
    x <- c(x[2:m], pointPreds[t])
    eps <- c(eps[2:T], abs(test[t] - pointPreds[t]))
    if (t %% s == 0){
      start <- start + s
      x <- train.test[start:(start+m-1)]
      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 365 days as a training set, and try to cover the price on the next 895 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 median preidction. We target achieving a coverage rate of 0.95.

```
setwd('C://Users/ordav/Desktop/ML_Projects/StockPricesPredictionIntervals/')
df <- read.csv('META.csv')
train <- df[1:365, 'Close']
a <- (length(train) + 1):1260
test <- df[a, 'Close']
preds <- predictionIntervals(train, test, m = 5, alpha = 0.05, B = 30,
                             aggregation = 'Median', s = 1)
```

```
plot(x = a, test, type = 'l')
lines(x = a, preds$Upper, col = 'blue')
lines(x = a, preds$Lower, col = 'red')
legend(x='topleft', inset = 0.02, legend = c('Real Price', 'Upper Bound', 'Lower Bound'),
       col = c('black', 'blue', 'red'), lty = 1)
```

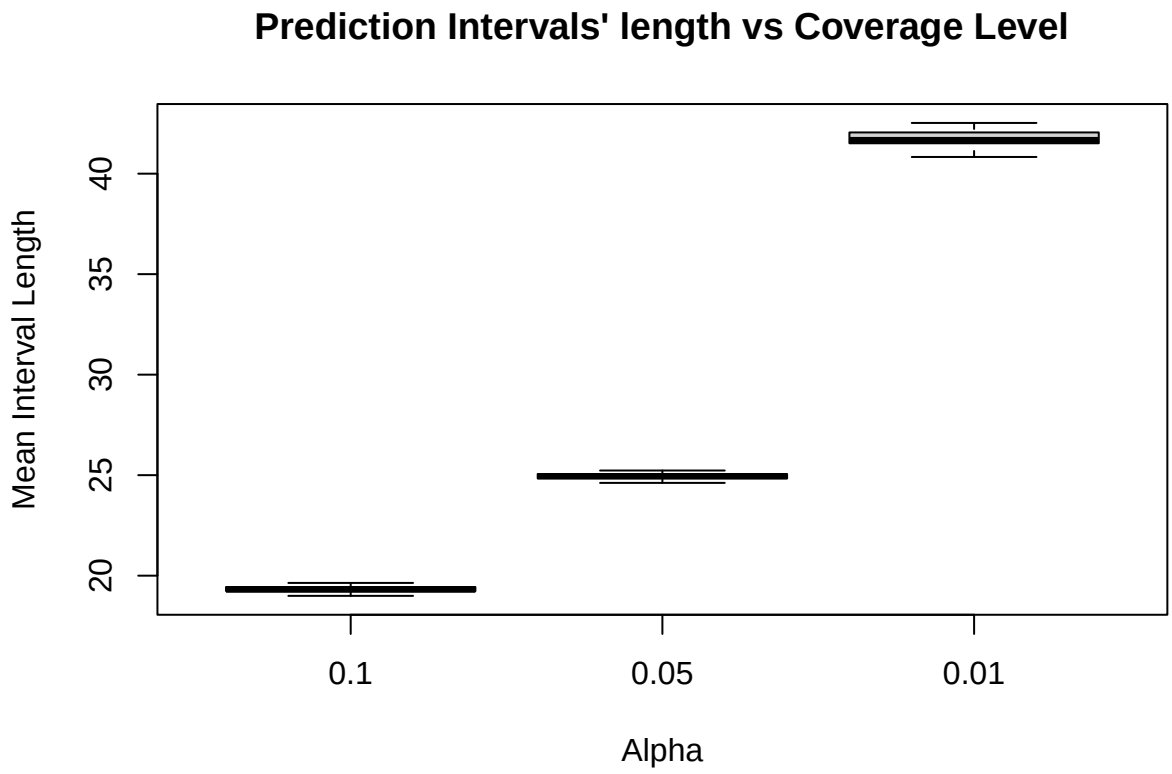


I will now try to see how the required coverage rate effects the length of the prediction interval. for each of (0.9, 0.95, 0.99) coverage rates, we will train 20 models with same parameters as before, and look at the distribution of their coverage rate on the test data, and the mean length of the prediction intervals they

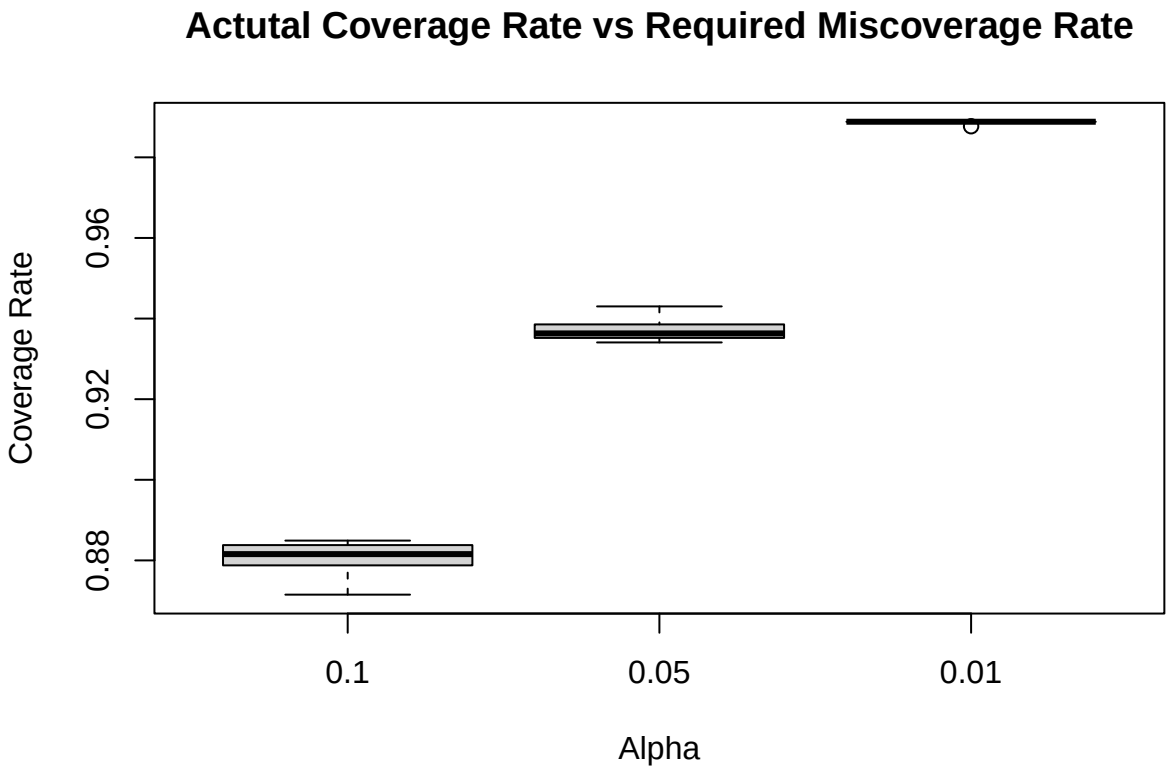
produce.

```
alphas <- c(0.1, 0.05, 0.01)
lengths.mat <- matrix(0, ncol = 3, nrow = 20)
coverage.mat <- matrix(0, ncol = 3, nrow = 20)
colnames(lengths.mat) <- alphas
colnames(coverage.mat) <- alphas
for(a in 1:length(alphas)){
  for(i in 1:20){
    preds <- predictionIntervals(train, test, m = 5, alpha = alphas[a],
                                 B = 30, aggregation = 'Median', s = 1)
    coverage.mat[i, a] <- mean((test < preds$Upper) & (test > preds$Lower))
    lengths.mat[i, a] <- mean(preds$Upper - preds$Lower)
  }
}

boxplot(lengths.mat, ylab = 'Mean Interval Length', xlab = 'Alpha')
title(main = "Prediction Intervals' length vs Coverage Level")
```



```
boxplot(coverage.mat, ylab = 'Coverage Rate', xlab = 'Alpha')
title(main = "Actutal Coverage Rate vs Required Miscoverage Rate")
```



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.

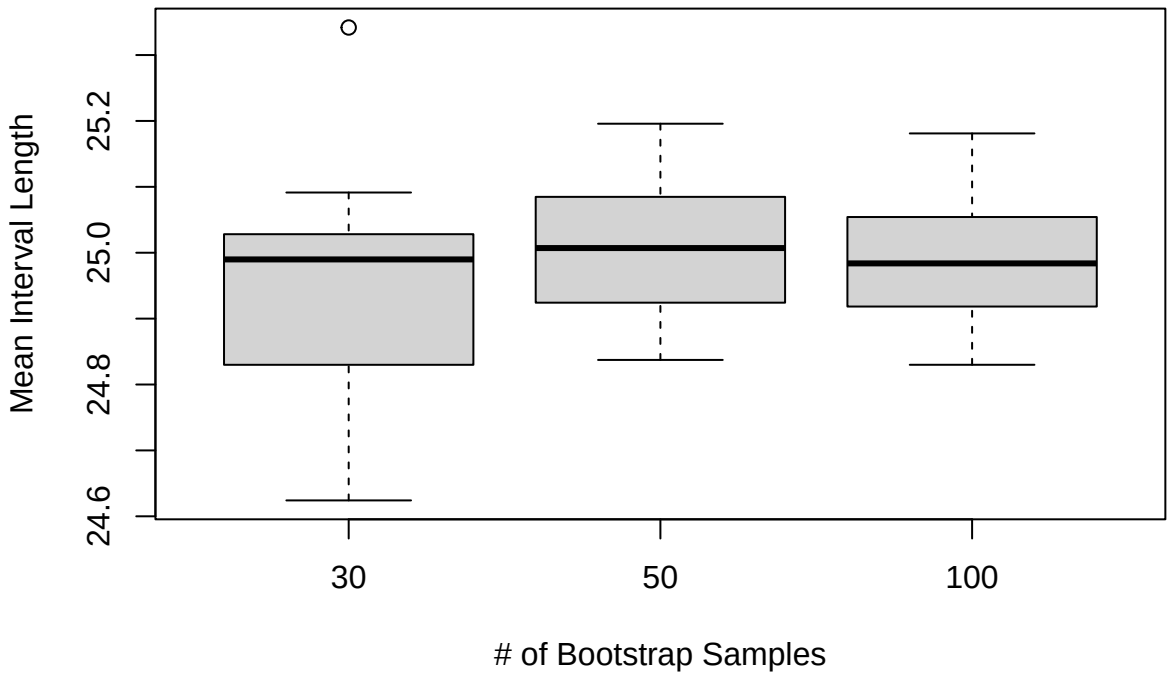
Not surprisingly, we see a significant increase in the mean interval length as the required miscoverage rate decreases.

We would now like to see how the number of bootstrap samples we create effect the coverage rate and the intervals' lengths. We would 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.

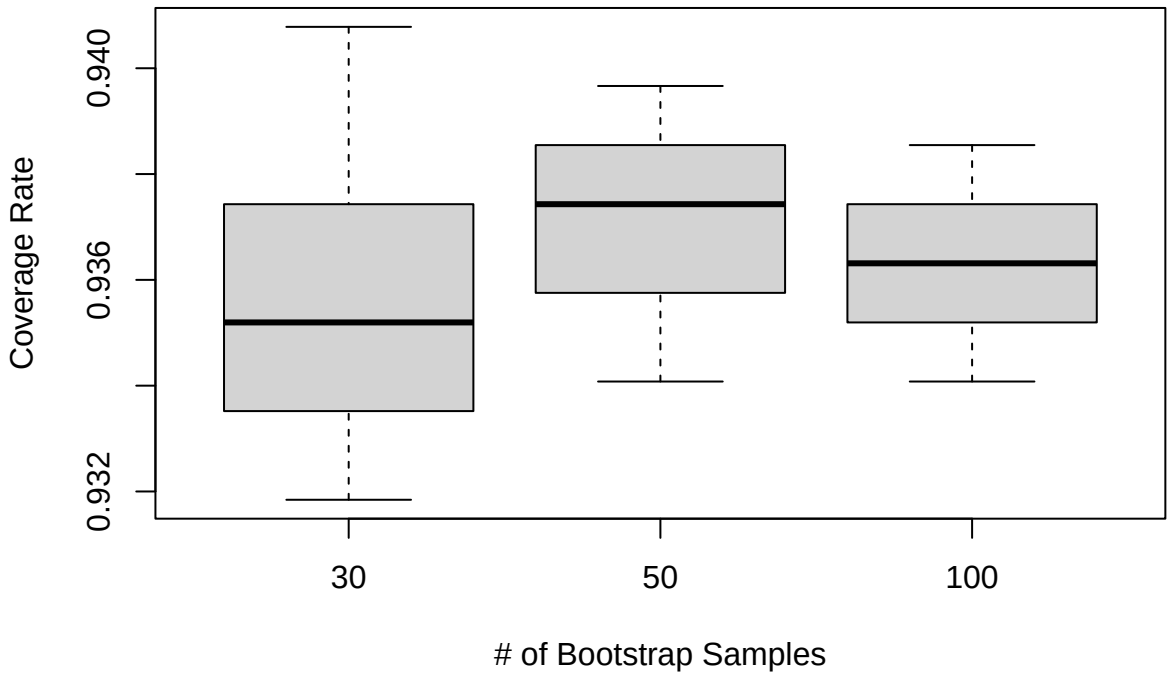
```
B_s <- c(30, 50, 100)
lengths.mat <- matrix(0, ncol = 3, nrow = 20)
coverage.mat <- matrix(0, ncol = 3, nrow = 20)
colnames(lengths.mat) <- B_s
colnames(coverage.mat) <- B_s
for(b in 1:length(B_s)){
  for(i in 1:20){
    preds <- predictionIntervals(train, test, m = 5, alpha = 0.05, B = B_s[b],
                                 aggregation = 'Median', s = 1)
    coverage.mat[i, b] <- mean((test < preds$Upper) & (test > preds$Lower))
    lengths.mat[i, b] <- mean(preds$Upper - preds$Lower)
```

```
  }
}

boxplot(lengths.mat, ylab = 'Mean Interval Length', xlab = '# of Bootstrap Samples')
```



```
boxplot(coverage.mat, ylab = 'Coverage Rate', xlab = '# of Bootstrap Samples')
```

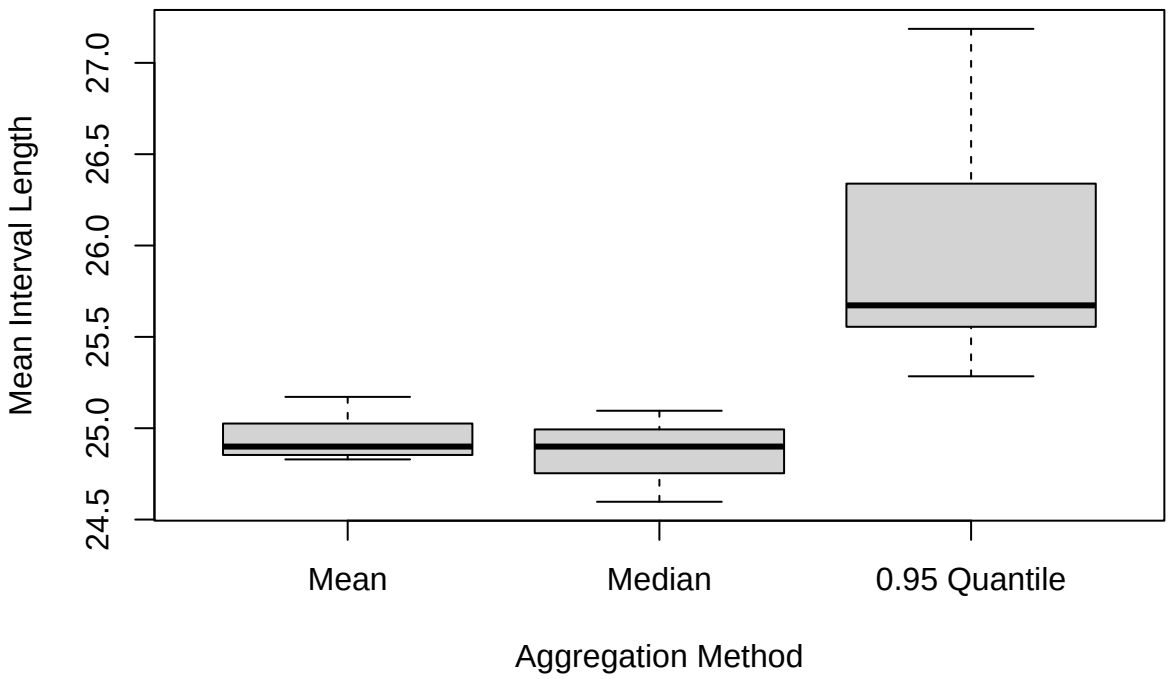


Surprisingly, we can see no actual evidence that increasing the number of bootstrap samples improves the performance of the model. The distribution of the intervals' length and coverage seems to be quite insensitive to the change in the B parameter.

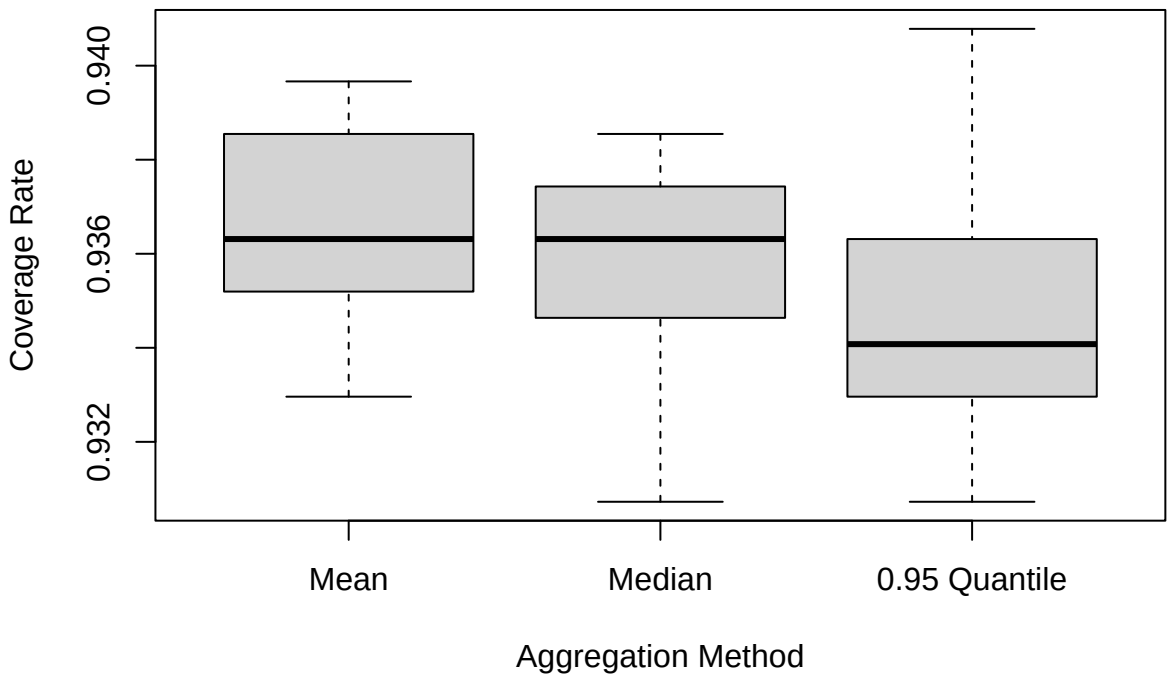
Lastly, we will have a look at how the aggregation strategy we choose will effect the model's performance.

```
lengths.mat <- matrix(0, ncol = 3, nrow = 20)
coverage.mat <- matrix(0, ncol = 3, nrow = 20)
colnames(lengths.mat) <- c('Mean', 'Median', '0.95 Quantile')
colnames(coverage.mat) <- c('Mean', 'Median', '0.95 Quantile')
for(i in 1:20){
    preds <- predictionIntervals(train, test, m = 5, alpha = 0.05, B = 30,
                                 aggregation = 'Mean', s = 1)
    coverage.mat[i, 1] <- mean((test < preds$Upper) & (test > preds$Lower))
    lengths.mat[i, 1] <- mean(preds$Upper - preds$Lower)
    preds <- predictionIntervals(train, test, m = 5, alpha = 0.05, B = 30,
                                 aggregation = 'Median', s = 1)
    coverage.mat[i, 2] <- mean((test < preds$Upper) & (test > preds$Lower))
    lengths.mat[i, 2] <- mean(preds$Upper - preds$Lower)
    preds <- predictionIntervals(train, test, m = 5, alpha = 0.05, B = 30,
                                 aggregation = 0.95, s = 1)
    coverage.mat[i, 3] <- mean((test < preds$Upper) & (test > preds$Lower))
    lengths.mat[i, 3] <- mean(preds$Upper - preds$Lower)
}
```

```
boxplot(lengths.mat, ylab = 'Mean Interval Length', xlab = 'Aggregation Method')
```



```
boxplot(coverage.mat,  ylab = 'Coverage Rate', xlab = 'Aggregation Method')
```



We can see that all aggregation strategies give about the same coverage rate, but using the 95 percentile gives slightly wider prediction intervals.