Chapter 8 Exercises: Tree-Based Methods

Statistical Learning with R

Jon Geiger

March 20, 2022

Conceptual Exercise 5

Suppose we produce ten bootstrapped samples from a data set containing red and gree classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of $P(\text{ClassisRed}|X)$:

$$0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75.$$

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the classification under each of these two approaches?

Solution

Because each of these values represents the probability that the class is red, there is a simple, logical cutoff value for how each of these estimates would classify Y.

The first approach is the majority vote, which involves looking at how each observation would classify the sample, and taking the majority. This is demonstrated here:

```r
probs <- c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
classes_ind <- factor(ifelse(probs >= 0.5, "Red", "Green"))
classes_ind
```

```
##  [1] Green Green Green Green Red   Red   Red   Red   Red   Red
## Levels: Green Red
```

```r
table(classes_ind)
```

```
## classes_ind
## Green   Red
##     4     6
```

We can see that the majority of the observations classify this X as Red, so the majority vote approach classifies it as red.

The second approach is to take the average probability and classify X according to the mean of the sampled probability. This is demonstrated here:

```r
result <- ifelse(mean(probs) >= 0.5, "Red", "Green")
cat("Mean of probabilities: ", mean(probs), "\n",
    "Classification: ", ifelse(mean(probs) >= 0.5, "Red", "Green"), sep = "")
```

```
## Mean of probabilities: 0.45
## Classification: Green
```

This approach has the opposite classification as the majority vote approach. This is because taking the average probability uses the mean, which is sensitive to outliers and skew, whereas the majority vote approach is closer to using the median on a skewed dataset. Even if the mean is below 50%, the median of the data set still falls above that value at 0.575.

Applied Exercise 10

We now use boosting to predict `Salary` in the `Hitters` data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

(c) Perform boosting on the trianing set with 1,0000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

(d) Produce a plot with the different shrinkage values on the x-axis and the corresponding test set MSE on the y axis.

(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

(f) Which variables appear to be the most important predictors in the boosted model?

(g) Now apply bagging to the training set. What is the test MSE for this approach?

Solution

(a)
```
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

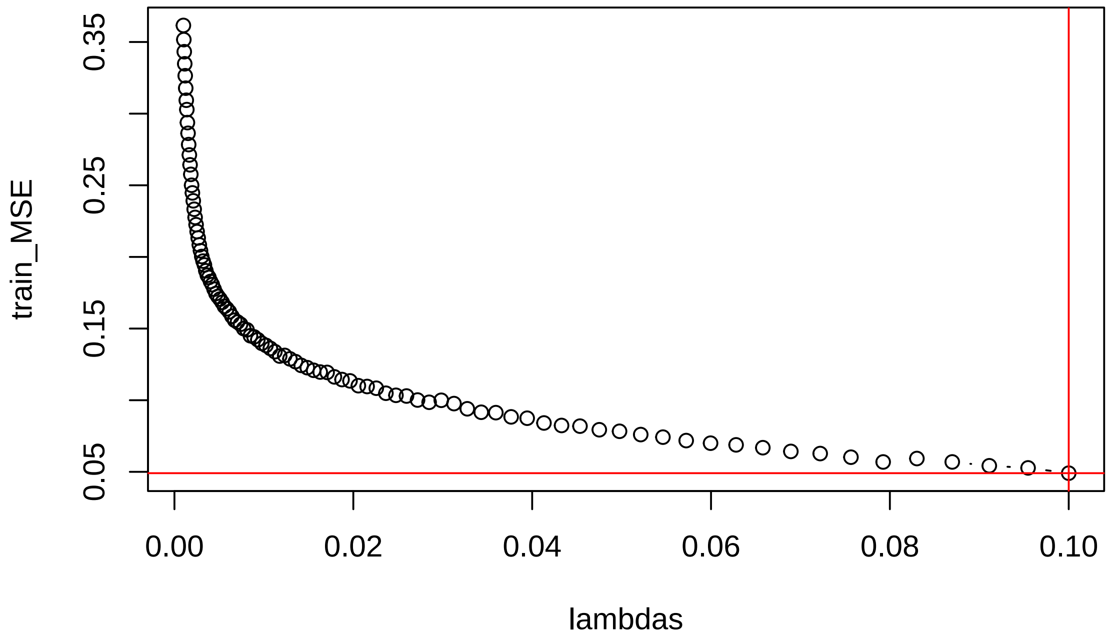
```
Hitters_noNA <- Hitters %>%
    filter(!is.na(Salary)) %>%
    mutate("logSalary" = log(Salary)) %>%
    select(-Salary)
```

(b)
```
train <- 1:200
hitters_train <- Hitters_noNA[train, ]
hitters_test  <- Hitters_noNA[-train, ]
```

(c)
```
lambdas <- 10^seq(-3, -1, length.out = 100)

train_MSE <- c()
test_MSE  <- c()
```

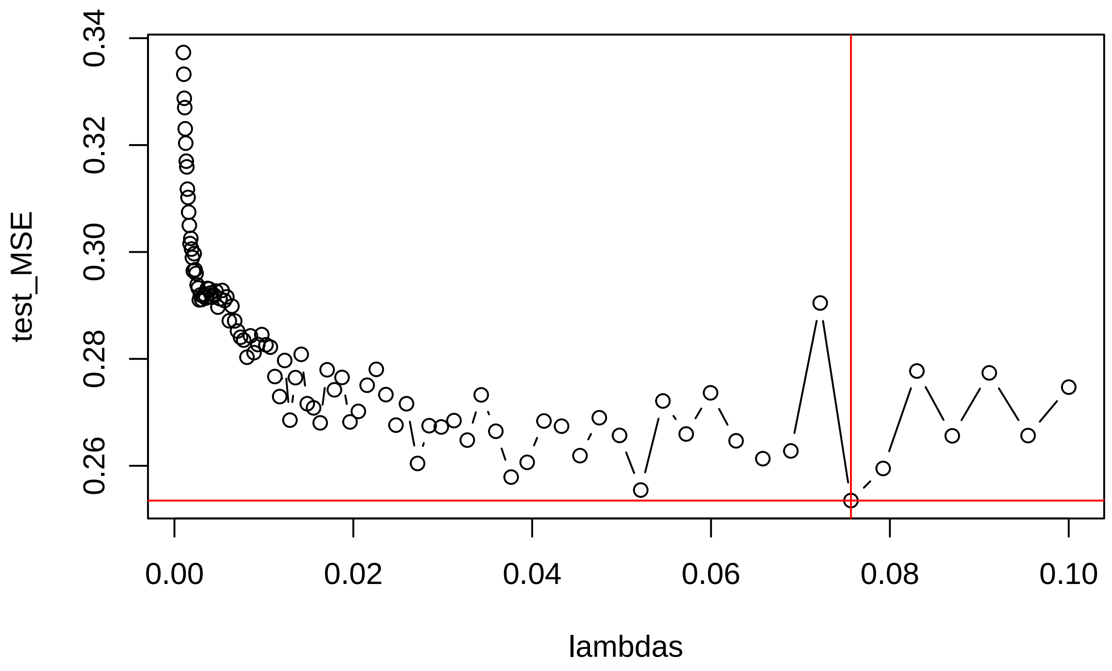
```
for(lambda in lambdas) {
    boost <- gbm(logSalary ~ .,
                 data = hitters_train,
                 distribution = "gaussian",
                 n.trees = 1000,
                 shrinkage = lambda,
                 interaction.depth = 1)
    train_MSE <- c(
        train_MSE,
        mean((predict(boost, hitters_train, n.trees = 1000) -
                  hitters_train$logSalary)^2)
    )
    test_MSE <- c(
        test_MSE,
        mean((predict(boost, hitters_test, n.trees = 1000) -
                  hitters_test$logSalary)^2)
    )
}
plot(x = lambdas, y = train_MSE,
     type = "b")
abline(v = lambdas[which.min(train_MSE)], col = "red")
abline(h = min(train_MSE), col = "red")
```



As expected, the training MSE decreases with a higher shrinkage value. If we increase the number of trees, this should have the same effect.

(d) We can now plot the test MSE versus λ to reveal the best value:

```r
bestlambda <- lambdas[which.min(test_MSE)]
plot(x = lambdas, y = test_MSE,
         type = "b")
abline(v = bestlambda, col = "red")
abline(h = min(test_MSE), col = "red")
```

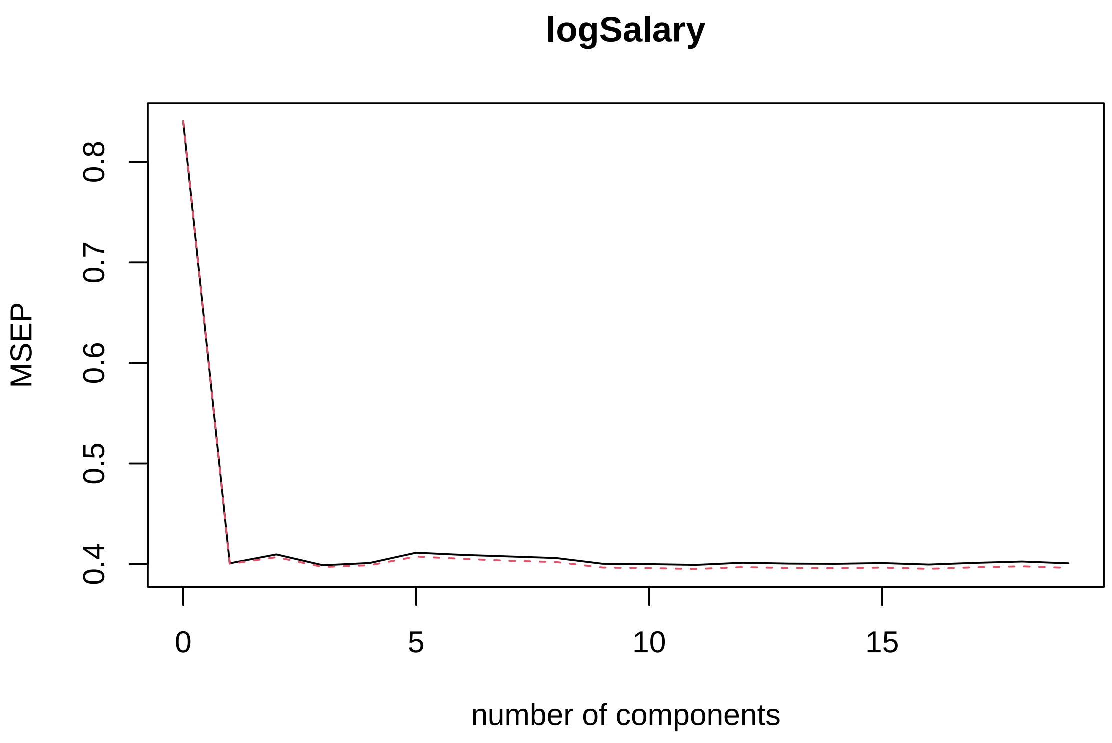


```r
cat("Best lambda = ", bestlambda, "\n",
    "MSE at best lambda = ", min(test_MSE),
    sep = "")
```

```
## Best lambda = 0.07564633
## MSE at best lambda = 0.2534905
```

(e) We'll compare this boosted MSE to a test mse from a Lasso model as well as a Partial Least Squares model. We'll start with PLS:

```r
pls_fit <- plsr(logSalary ~ .,
                data = hitters_train,
                scale = TRUE,
                validation = "CV")
validationplot(pls_fit, val.type = "MSEP")
```



```
pls_pred <- predict(pls_fit,
                    hitters_test,
                    ncomp = 1)
cat("MSE for PLS model with M=1: ",
    mean((pls_pred - hitters_test$logSalary)^2),
    sep = "")
```

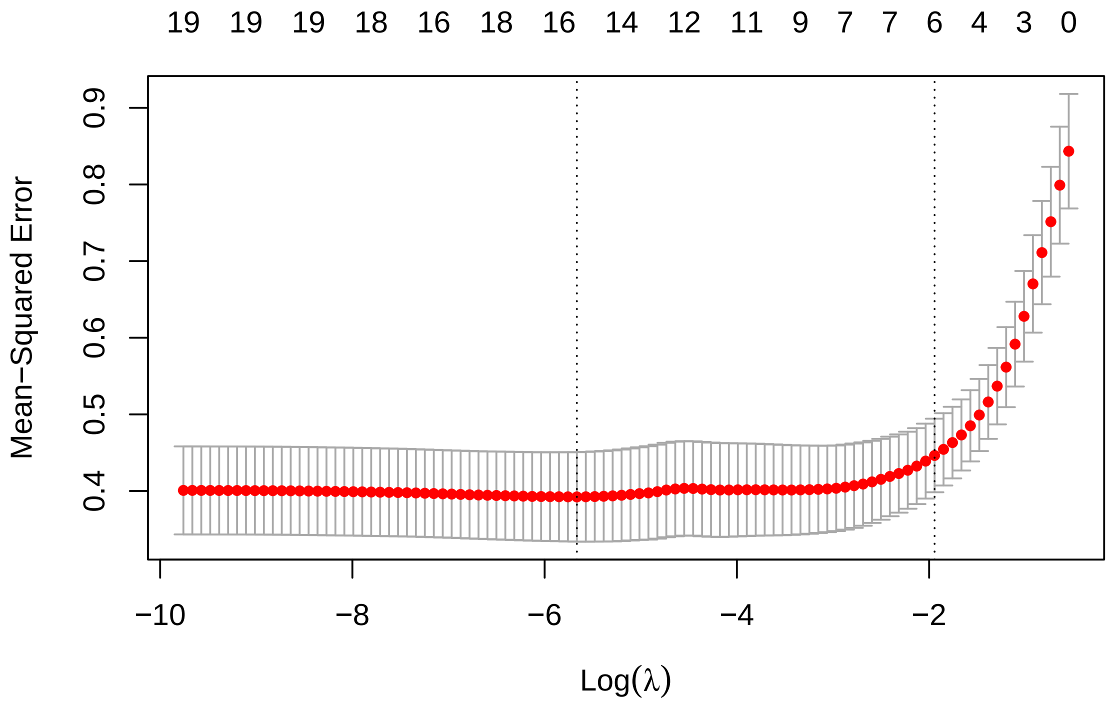
```
## MSE for PLS model with M=1: 0.4659979
```

This is a higher MSE than any of the possible MSEs from the boosting model, even with very low values of λ. We'll now try a lasso model:

```
grid <- 10^seq(10, -2, length = 100)


Xtrain <- model.matrix(logSalary ~ .,
                       hitters_train)
Xtest  <- model.matrix(logSalary ~ .,
                       hitters_test)
ytrain <- hitters_train$logSalary
ytest <- hitters_test$logSalary

lasso_fit <- glmnet(Xtrain, ytrain,
                    alpha = 1,
                    lambda = grid)
lasso_cv <- cv.glmnet(Xtrain, ytrain, alpha = 1)
plot(lasso_cv)
```

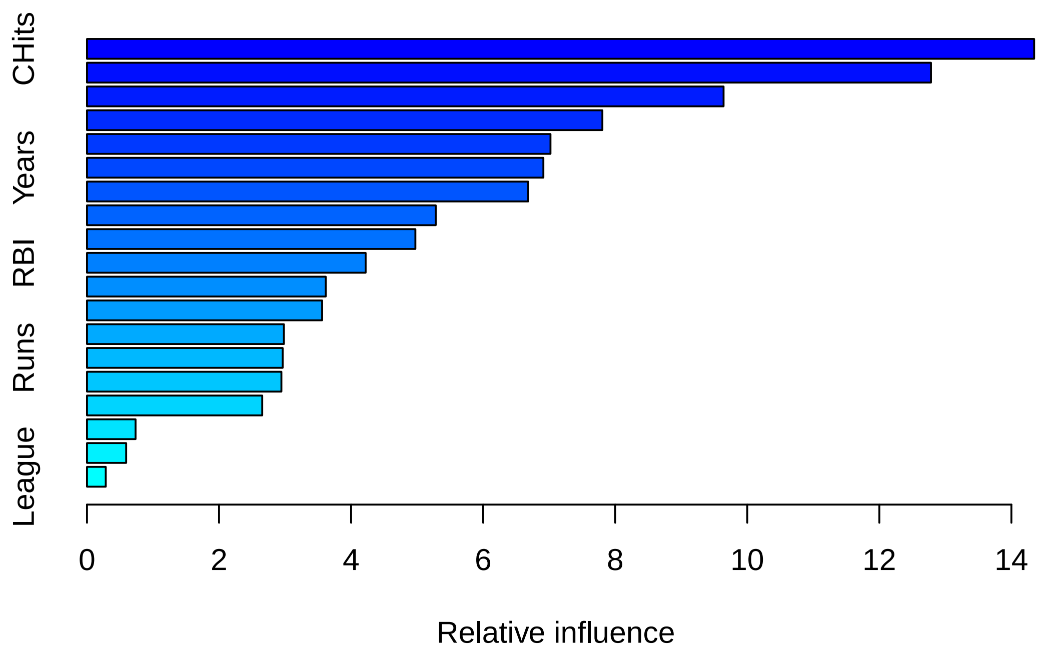


```r
bestlam <- lasso_cv$lambda.min
lasso_pred <- predict(lasso_fit, s = bestlam,
                      newx = Xtest)
mean((lasso_pred - ytest)^2)
```

```
## [1] 0.470371
```

This MSE is comparable to that from PLS, but still significantly higher than that from the boosting model.

(f)
```r
boost <- gbm(logSalary ~ .,
             data = hitters_train,
             distribution = "gaussian",
             n.trees = 1000,
             shrinkage = bestlambda,
             interaction.depth = 1)
summary(boost)
```



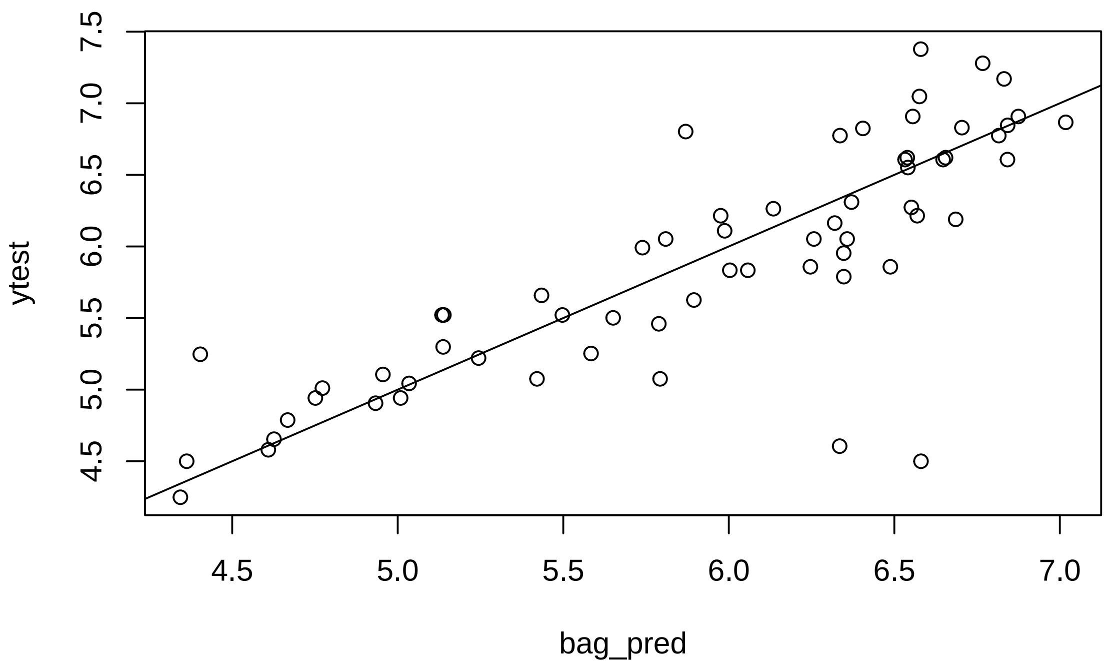
```
##                 var    rel.inf
## CHits         CHits 14.3448469
## CAtBat       CAtBat 12.7824166
## CWalks       CWalks  9.6413323
## PutOuts     PutOuts  7.8056132
## CRBI           CRBI  7.0194766
## Years         Years  6.9119466
## Walks         Walks  6.6827397
## CHmRun       CHmRun  5.2819223
## Assists     Assists  4.9736441
## RBI             RBI  4.2204697
## AtBat         AtBat  3.6160953
## Hits           Hits  3.5617023
## CRuns         CRuns  2.9820721
## Runs           Runs  2.9638414
## HmRun         HmRun  2.9446785
## Errors       Errors  2.6542196
## Division   Division  0.7354821
## NewLeague NewLeague  0.5933369
## League       League  0.2841638
```

From the relative influence table above, we can see that the number of Career at-bats and the Career Hits have the most influence/importance in the boosting model. This makes good sense, because if we are to interpret this as a sort of inference problem, it would make sense that the salary of a baseball player would depend on their career, because generally, players with very long careers have performed very well and typically have high salaries.

(g) We will now create a bagged model on the same data, using the `randomForest()` function with $m = p$, which in this case should be 19.

```r
p = length(names(hitters_train))-1

bag_fit <- randomForest(logSalary ~ .,
                        data = hitters_train,
                        mtry = p,
                        importance = TRUE)
bag_pred <- predict(bag_fit,
                    newdata = hitters_test)
plot(bag_pred, ytest)
abline(0, 1)
```

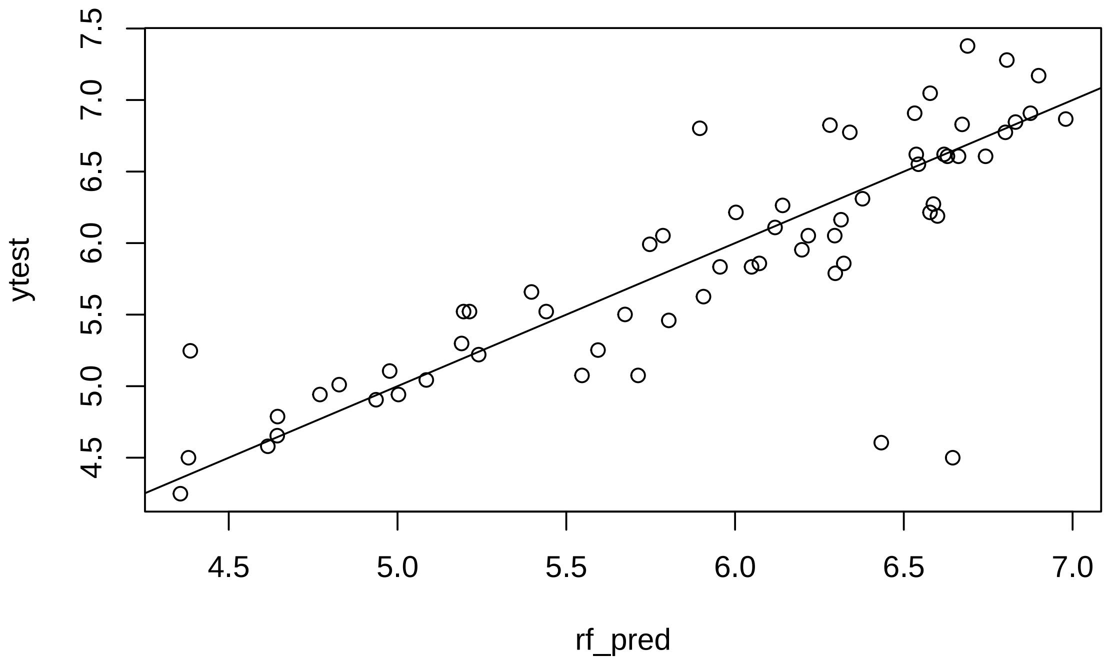


```r
mean((bag_pred - ytest)^2)
```

```
## [1] 0.224522
```

This MSE is lower than that from the boosted model. Now, for fun, let's try a random forest with $m = 6$ predictors:

```r
rf_fit <- randomForest(logSalary ~ .,
                        data = hitters_train,
                        mtry = 6,
                        importance = TRUE)
rf_pred <- predict(rf_fit,
                               newdata = hitters_test)
plot(rf_pred, ytest)
abline(0, 1)
```



```
mean((rf_pred - ytest)^2)
```

```
## [1] 0.2211228
```

The random forest model yields the lowest MSE out of any model we've tried thus far. This makes sense, because there are two variables which have high importance, as we saw in part (f), so random forests are able to decorrelate the trees from always starting with the same two splits.