Homework 09

STAT 430, Fall 2017

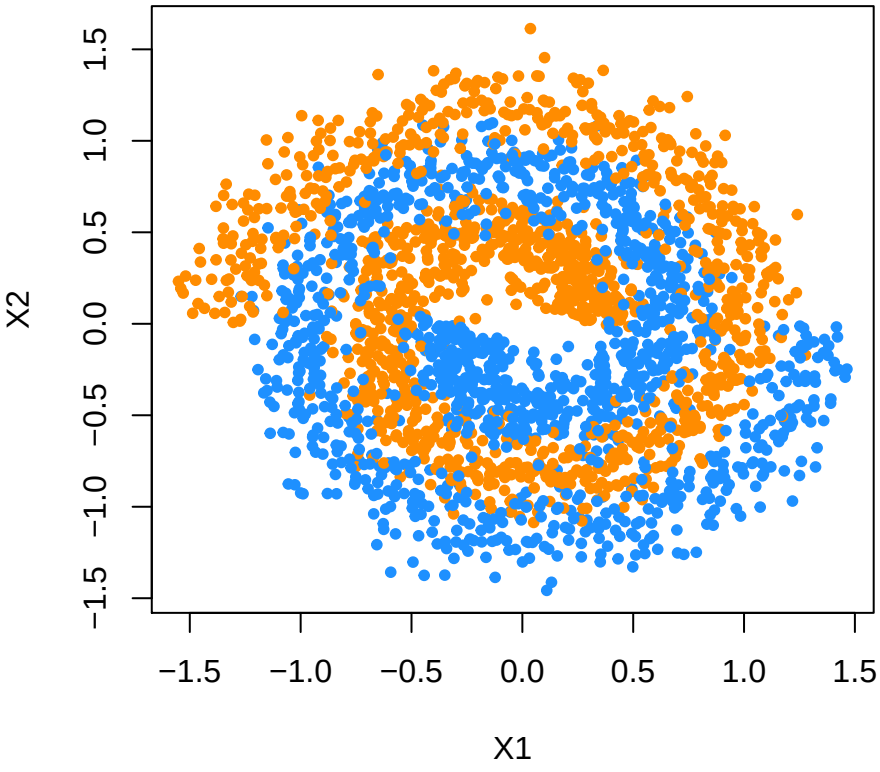
Due: Monday, November 20, 11:59 PM

Exercise 1 (Computation Time)

[8 points] For this exercise we will create data via simulation, then assess how well certain methods perform. Use the code below to create a train and test dataset.

```
library(mlbench)
set.seed(42)
sim_trn = mlbench.spirals(n = 2500, cycles = 1.5, sd = 0.125)
sim_trn = data.frame(sim_trn$x, class = as.factor(sim_trn$classes))
sim_tst = mlbench.spirals(n = 10000, cycles = 1.5, sd = 0.125)
sim_tst = data.frame(sim_tst$x, class = as.factor(sim_tst$classes))
```

The training data is plotted below, with colors indicating the `class` variable, which is the response.



Before proceeding further, set a seed equal to your UIN.

```
uin = 123456789
set.seed(uin)
```

We'll use the following to define 5-fold cross-validation for use with `train()` from `caret`.

```
library(caret)
cv_5 = trainControl(method = "cv", number = 5)
```

We now tune two models with `train()`. First, a logistic regression using `glm`. (This actually isn't "tuned" as there are not parameters to be tuned, but we use `train()` to perform cross-validation.) Second we tune a single decision tree using `rpart`.

We store the results in `sim_glm_cv` and `sim_tree_cv` respectively, but we also wrap both function calls with `system.time()` in order to record how long the tuning process takes for each method.

```
glm_cv_time = system.time({
  sim_glm_cv  = train(
    class ~ .,
    data = sim_trn,
    trControl = cv_5,
    method = "glm")
})

tree_cv_time = system.time({
  sim_tree_cv = train(
    class ~ .,
    data = sim_trn,
    trControl = cv_5,
    method = "rpart")
})
```

We see that both methods are tuned via cross-validation in a similar amount of time.

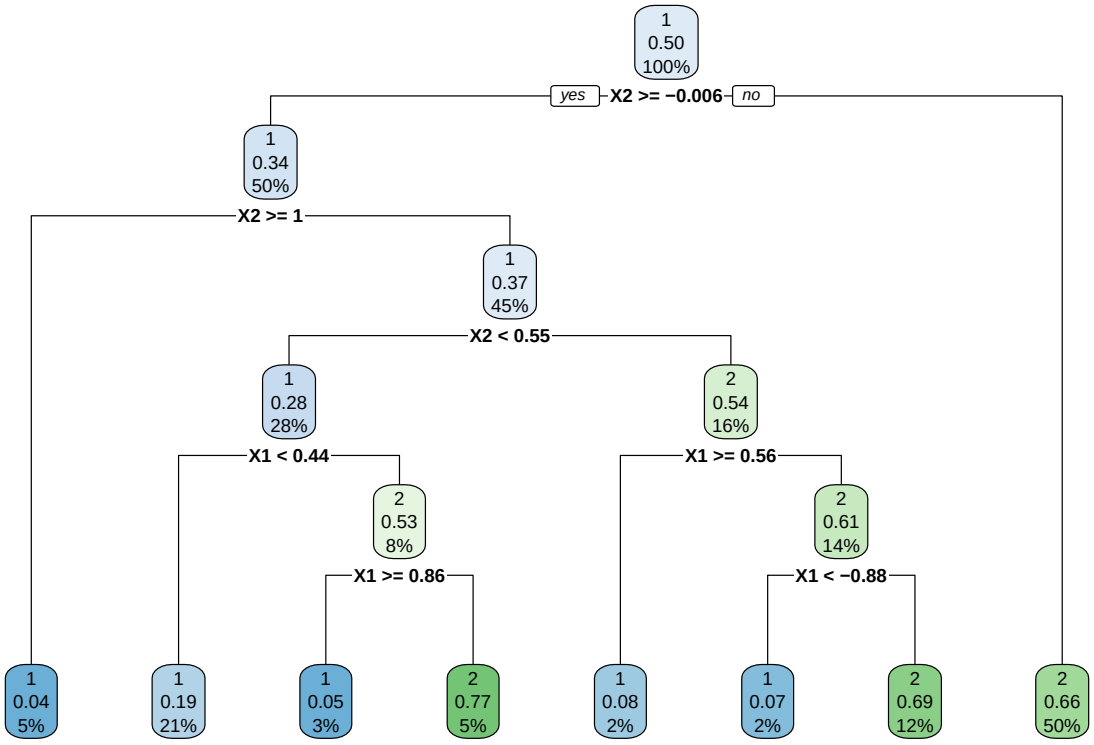
```
glm_cv_time["elapsed"]
```

```
## elapsed
##   0.763
```

```
tree_cv_time["elapsed"]
```

```
## elapsed
##   0.691
```

```
library(rpart.plot)
rpart.plot(sim_tree_cv$finalModel)
```



Repeat the above analysis using a random forest, twice. The first time use 5-fold cross-validation. (This is how we had been using random forests before we understood random forests.) The second time, tune the model using OOB samples. We only have two predictors here, so, for both, use the following tuning grid.

```
rf_grid = expand.grid(mtry = c(1, 2))
```

```
oob  = trainControl(method = "oob")
rf_oob_time = system.time({
  sim_rf_oob = train(
    class ~ .,
    data = sim_trn,
    trControl = oob,
    tuneGrid = rf_grid)
})

rf_cv_time = system.time({
  sim_rf_cv = train(
    class ~ .,
    data = sim_trn,
    trControl = cv_5,
    tuneGrid = rf_grid)
})
```

Create a table summarizing the results of these four models. (Logistic with CV, Tree with CV, RF with OOB, RF with CV). Report:

- Chosen value of tuning parameter (If applicable)
- Elapsed tuning time
- Resampled (CV or OOB) Accuracy
- Test Accuracy

Method	Best Tune	Elapsed	Resampled Accuracy	Test Accuracy
Logistic, CV	NA	0.763	0.6580	0.6606
Tree, CV	0.0196	0.691	0.7528	0.7233
RF, OOB	1.0000	2.274	0.8532	0.8500
RF, CV	1.0000	5.448	0.8500	0.8506

Exercise 2 (Predicting Baseball Salaries)

[**7 points**] For this question we will predict the `Salary` of `Hitters`. (`Hitters` is also the name of the dataset.) We first remove the missing data:

```
library(ISLR)
Hitters = na.omit(Hitters)
```

After changing `uin` to your UIN, use the following code to test-train split the data.

```
uin = 123456789
set.seed(uin)
hit_idx = createDataPartition(Hitters$Salary, p = 0.6, list = FALSE)
hit_trn = Hitters[hit_idx,]
hit_tst = Hitters[-hit_idx,]
```

Do the following:

- Tune a boosted tree model using the following tuning grid and 5-fold cross-validation.

```
gbm_grid = expand.grid(interaction.depth = c(1, 2),
                       n.trees = c(500, 1000, 1500),
                       shrinkage = c(0.001, 0.01, 0.1),
                       n.minobsinnode = 10)
```

- Tune a random forest using OOB resampling and **all** possible values of `mtry`.

Create a table summarizing the results of three models:

- Tuned boosted tree model
- Tuned random forest model
- Bagged tree model

For each, report:

- Resampled RMSE
- Test RMSE

```
hit_gbm = train(Salary ~ ., data = hit_trn,
                method = "gbm",
                trControl = cv_5,
                verbose = FALSE,
                tuneGrid = gbm_grid)

rf_grid = rf_grid = expand.grid(mtry = 1:(ncol(hit_trn) - 1))
hit_rf  = train(Salary ~ ., data = hit_trn,
                method = "rf",
                trControl = oob,
                tuneGrid = rf_grid)

# storing the bagged model for making predictions
```

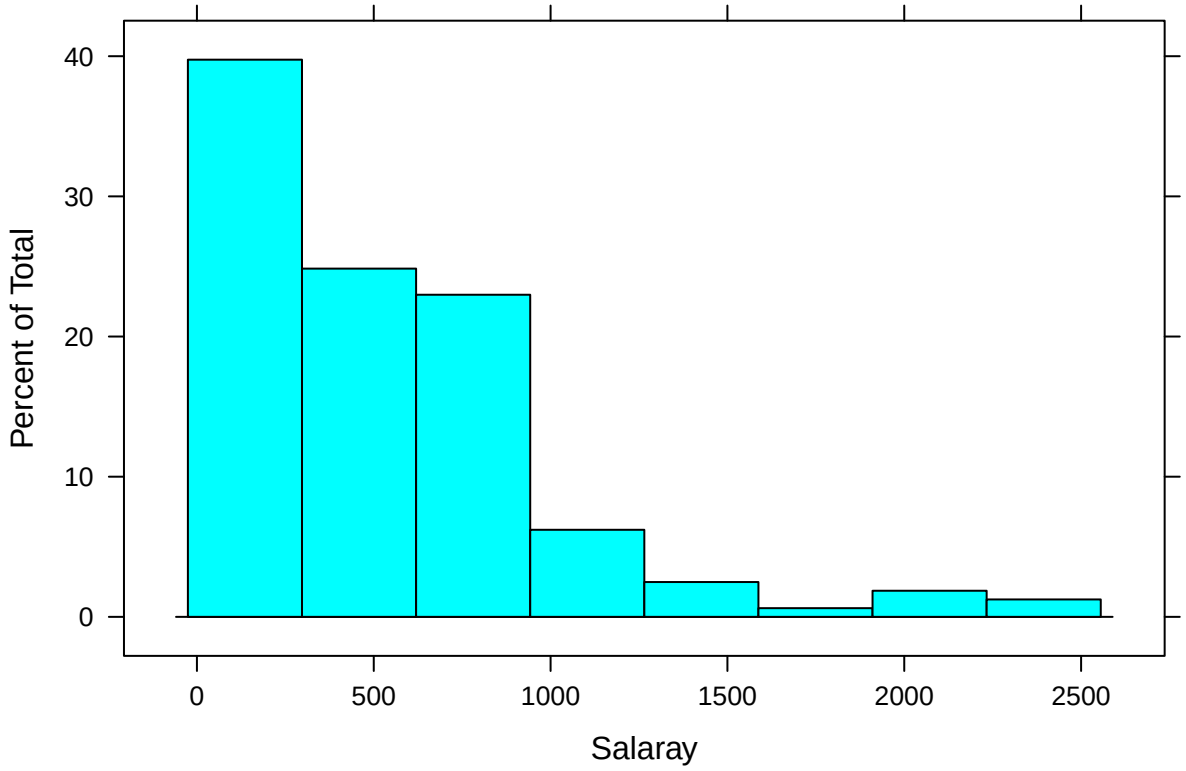
```
hit_bag = train(Salary ~ ., data = hit_trn,
                method = "rf",
                trControl = oob,
                tuneGrid = data.frame(mtry = (ncol(hit_trn) - 1)))
```

```
calc_rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

gbm_tst_rmse = calc_rmse(predicted = predict(hit_gbm, hit_tst),
                         actual    = hit_tst$Salary)

rf_tst_rmse = calc_rmse(predicted = predict(hit_rf, hit_tst),
                        actual    = hit_tst$Salary)

bag_tst_rmse = calc_rmse(predicted = predict(hit_bag, hit_tst),
                         actual    = hit_tst$Salary)


hitters_results = data.frame(
  c("Boosting", "Random Forest", "Bagging"),
  c(min(hit_gbm$results$RMSE), min(hit_rf$results$RMSE), min(hit_bag$results$RMSE)),
  c(gbm_tst_rmse, rf_tst_rmse, bag_tst_rmse)
)
colnames(hitters_results) = c("Method", "Resampled RMSE", "Test RMSE")
knitr::kable(hitters_results, digits = 4)
```

Method	Resampled RMSE	Test RMSE
Boosting	323.9203	257.8148
Random Forest	298.7520	255.5084
Bagging	304.8423	280.6918

Exercise 3 (Transforming the Response)

[**5 points**] Continue with the data from Exercise 2. The book, ISL, suggests log transforming the response, `Salary`, before fitting a random forest. Is this necessary? Re-tune a random forest as you did in Exercise 2, except with a log transformed response. Report test RMSE for both the untransformed and transformed model on the original scale of the response variable.




```
hit_rf_log  = train(log(Salary) ~ ., data = hit_trn,
                    method = "rf",
                    trControl = oob,
                    tuneGrid = rf_grid)
```

```
# without transformation
calc_rmse(predicted = predict(hit_rf, hit_tst),
          actual    = hit_tst$Salary)
```

```
## [1] 255.5084
```

```
# with log transformation
calc_rmse(predicted = exp(predict(hit_rf_log, hit_tst)),
          actual = hit_tst$Salary)
```

```
## [1] 272.3838
```

Exercise 4 (Concept Checks)

[**1 point each**] Answer the following questions based on your results from the three exercises.

Timing

(a) Compare the time taken to tune each model. Is the difference between the OOB and CV result for the random forest similar to what you would have expected?

Solution: The speed-up for OOB is only about three times that of 5-fold CV, instead of the five times that would have been expected. There appears to be some additional overhead in using OOB.

```
rf_cv_time["elapsed"] / rf_oob_time["elapsed"]
```

```
##  elapsed
## 2.395778
```

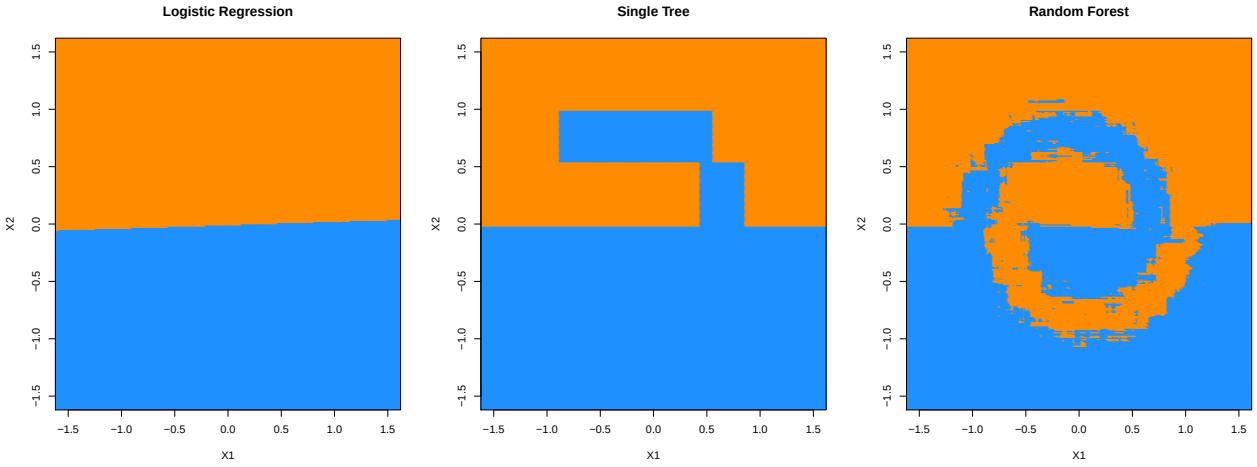
(b) Compare the tuned value of `mtry` for each of the random forests tuned. Do they choose the same model?

Solution: They choose the same model, although, there were only two to choose from, and they are not very different. In practice, the two methods may differ more.

(c) Compare the test accuracy of each of the four procedures considered. Briefly explain these results.

Solution:

- Logistic: Performs the worst. This is expected as clearly a non-linear decision boundary is needed.
- Single Tree: Better than logistic, but not the best seen here. We see above that this is not a very deep tree. It will have non-linear boundaries, but since it uses binary splits, they will be rectangular regions.
- Random Forest: First note that both essentially fit the same model. (The exact forests will be different due to randomization.) By using many trees (500) the boundaries will become less rectangular than the single tree, and will better match the spiral data in the data.
- See below for plots of decision boundaries created by making predictions from the different models.



Salary

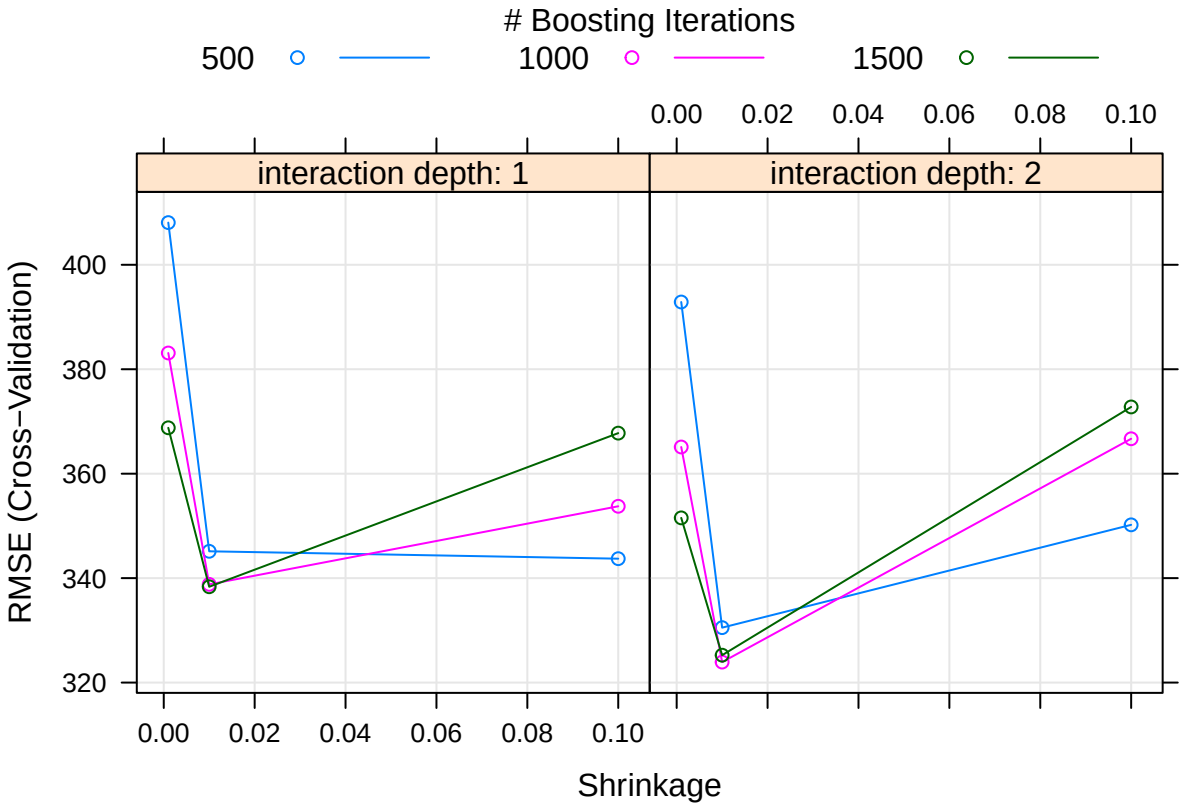
(d) Report the tuned value of `mtry` for the random forest.

```
hit_rf$bestTune
```

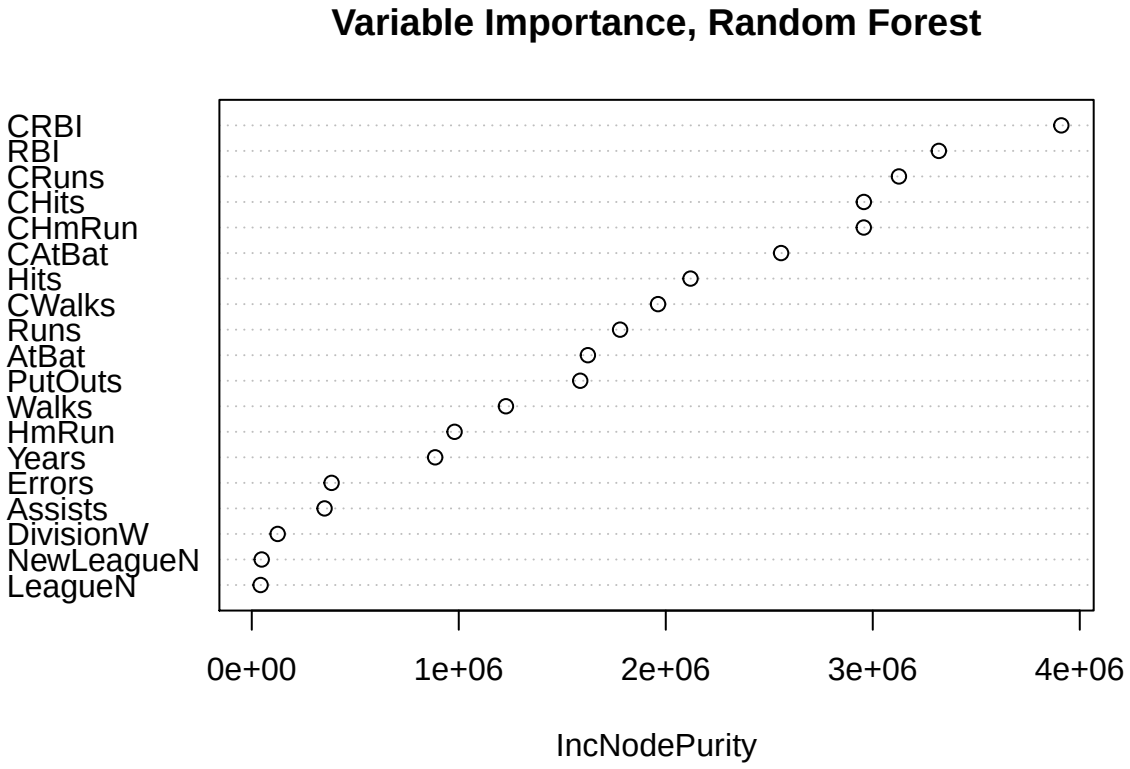
```
##   mtry
## 4    4
```

(e) Create a plot that shows the tuning results for the tuning of the boosted tree model.

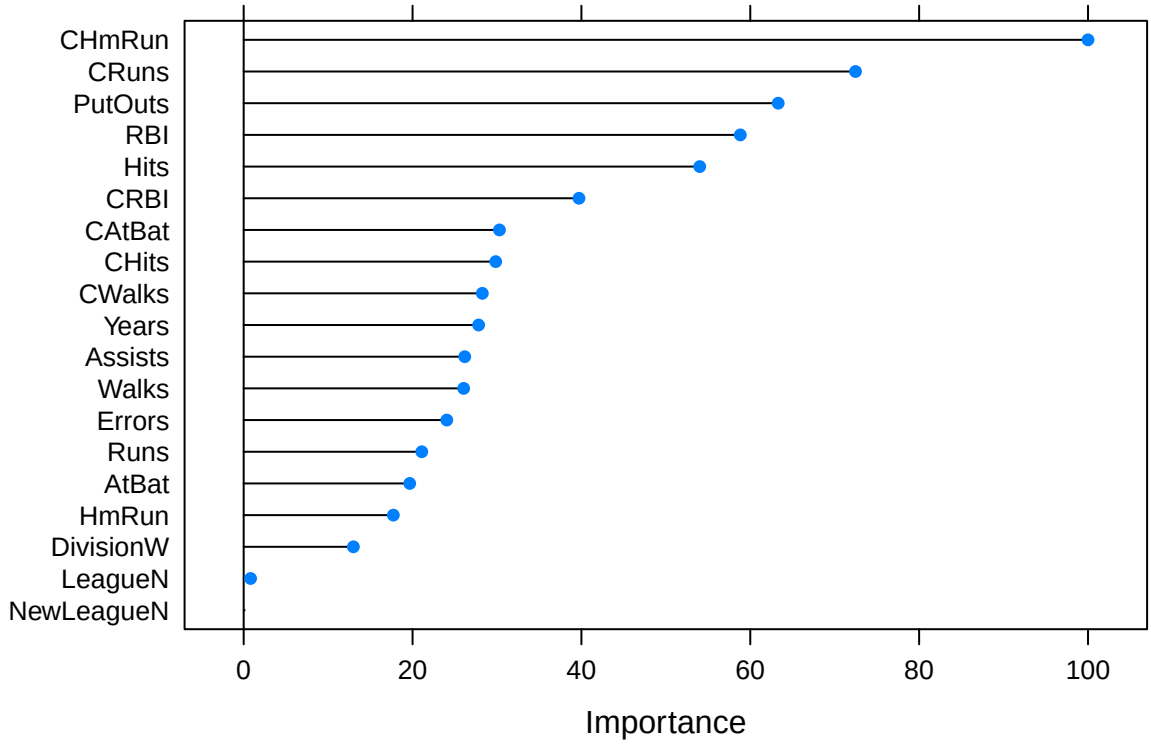
```
plot(hit_gbm)
```



(f) Create a plot of the variable importance for the tuned random forest.



(g) Create a plot of the variable importance for the tuned boosted tree model.



(h) According to the random forest, what are the three most important predictors?

```
## [1] "CRBI"  "RBI"   "CRuns"
```

(i) According to the boosted model, what are the three most important predictors?

```
## [1] "CHmRun"  "CRuns"   "PutOuts"
```

Transformation

(j) Based on these results, do you think the transformation was necessary?

Solution: Here we see that the untransformed model actually performs better. However, they are relatively close, so either could be acceptable. Note that a random forest can model a non-linear relationship, which is why the transformation is not necessary.