Inducing Uncorrelated Component Models; Random Forests

Introduction

Running Data Set Example

Boston housing prices; predicting median value.

```r
library(readr)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(purrr)
library(glmnet)
library(caret)
library(rpart)

# read in data
Boston <- read_csv("http://www.evanlray.com/data/mass/Boston.csv")

# Initial train/test split ("estimation"/test) and cross-validation folds
set.seed(63770)
tt_inds <- caret::createDataPartition(Boston$medv, p = 0.8)
train_set <- Boston %>% slice(tt_inds[[1]])
test_set <- Boston %>% slice(-tt_inds[[1]])
```

Strategy 1: Bagging

Algorithm:

1. Allocate space to save test set predictions from B component models (often B is in the range of 500 or 1000)
2. For $b = 1, \ldots, B$
    a. Draw a bootstrap sample (i.e., a sample of n rows/observations, drawn with replacement) from the original data set.
    b. Fit the model to the bootstrap sample from step a.
    c. Obtain test set predictions and save them
3. Ensemble prediction combines predictions for the B models obtained in step 1 (most commonly, simple average for regression or majority vote for classification)

I would never implement this by hand, code just for illustration of the idea!

```
B <- 500

component_test_mses <- rep(NA, B)
component_test_predictions <- matrix(NA, nrow = nrow(test_set), ncol = B)

for(b in seq_len(B)) {
  n <- nrow(train_set)

  bootstrap_resampled_train <- train_set %>%
    dplyr::sample_n(size = n, replace = TRUE)

  tree_fit <- train(medv ~ .,
    data = bootstrap_resampled_train,
    method = "rpart")

  test_predictions_b <- predict(tree_fit, newdata = test_set)

  component_test_mses[b] <- mean((test_predictions_b - test_set$medv)^2)
  component_test_predictions[, b] <- test_predictions_b
}

ensemble_test_predictions <- apply(component_test_predictions, 1, mean)
ensemble_test_mse <- mean((ensemble_test_predictions - test_set$medv)^2)

single_tree <- train(medv ~ .,
    data = bootstrap_resampled_train,
    method = "rpart")
single_tree_test_predictions <- predict(single_tree, newdata = test_set)
single_tree_test_mse <- mean((single_tree_test_predictions - test_set$medv)^2)
```

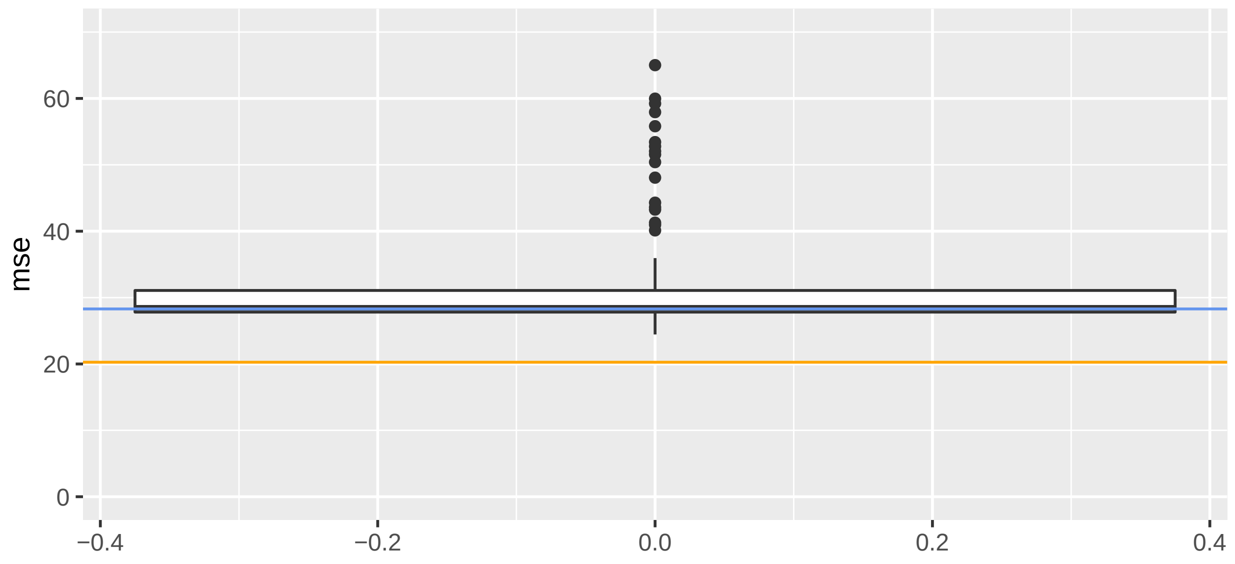
```
single_tree_test_mse
```

```
## [1] 28.29364
```

```
ensemble_test_mse
```

```
## [1] 20.26028
```

```r
ggplot() +
  geom_boxplot(
    data = data.frame(mse = component_test_mses),
    mapping = aes(y = mse)) +
  geom_hline(
    yintercept = ensemble_test_mse,
    color = "orange") +
  geom_hline(
    yintercept = single_tree_test_mse,
    color = "cornflowerblue") +
  ylim(c(0, 70))
```



Strategy 2: Feature Subsets

Similar to above, but different subsets of the features (explanatory variables) are considered for each model, or at different stages within estimation for each model.

- We could divide the explanatory variables into different groups, and train different models on different subsets of the available explanatory variables.
  - Only effective if there are lots of explanatory variables available.

```r
names(train_set)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"    
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```r
B <- 500

component_test_mses <- rep(NA, B)
component_test_predictions <- matrix(NA, nrow = nrow(test_set), ncol = B)

for(b in seq_len(B)) {
  features_subset_train <- train_set %>%
    dplyr::select(c(sample(13, size = 6, replace = FALSE), 14))

  tree_fit <- train(medv ~ .,
    data = features_subset_train,
    method = "rpart")

  test_predictions_b <- predict(tree_fit, newdata = test_set)

  component_test_mses <- mean((test_predictions_b - test_set$medv)^2)
  component_test_predictions[, b] <- test_predictions_b
}

ensemble_test_predictions <- apply(component_test_predictions, 1, mean)
ensemble_test_mse <- mean((ensemble_test_predictions - test_set$medv)^2)
```

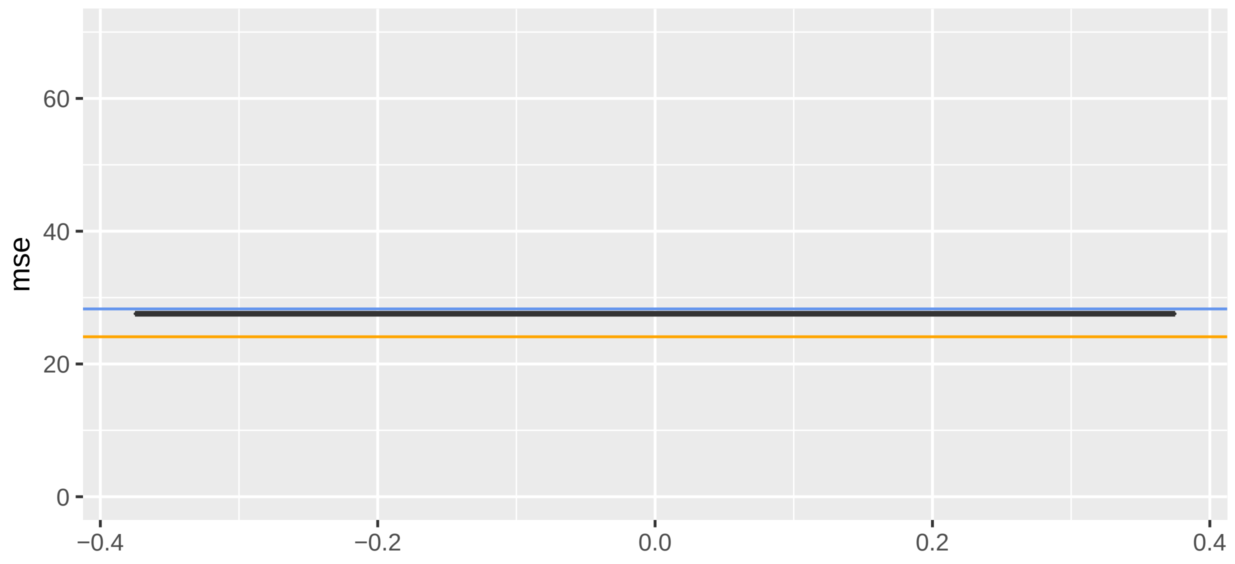
```r
single_tree_test_mse
```

```
## [1] 28.29364
```

```r
ensemble_test_mse
```

```
## [1] 24.09867
```

```r
ggplot() +
  geom_boxplot(
    data = data.frame(mse = component_test_mses),
    mapping = aes(y = mse)) +
  geom_hline(
    yintercept = ensemble_test_mse,
    color = "orange") +
  geom_hline(
    yintercept = single_tree_test_mse,
    color = "cornflowerblue") +
  ylim(c(0, 70))
```



Random Forests

```r
library(randomForest)
rf_fit <- train(
  form = medv ~ .,
  data = train_set,
  method = "rf",
  trControl = trainControl(method = "oob",
    returnResamp = "all",
    savePredictions = TRUE),
  tuneLength = 10
)

rf_fit$results
```

```
##        RMSE  Rsquared mtry
## 1  3.646170 0.8433786    2
## 2  3.395492 0.8641741    3
## 3  3.406912 0.8632589    4
## 4  3.239559 0.8763629    5
## 5  3.314726 0.8705588    6
## 6  3.318329 0.8702773    8
## 7  3.270926 0.8739571    9
## 8  3.341645 0.8684479   10
## 9  3.353521 0.8675112   11
## 10 3.400481 0.8637747   13
```

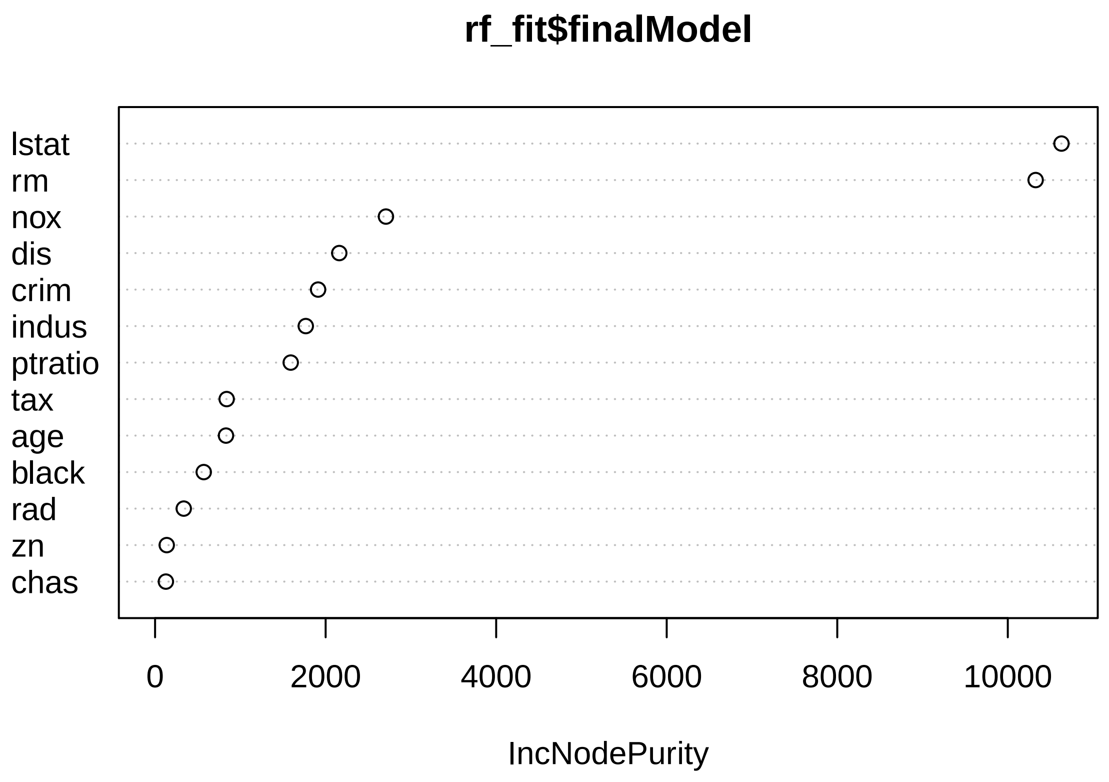
```r
rf_mse <- mean((test_set$medv - predict(rf_fit, newdata = test_set))^2)
rf_mse
```

```
## [1] 7.94216
```

```r
importance(rf_fit$finalModel, type = 2)
```

```
##         IncNodePurity
## crim        1911.7451
## zn           136.6524
## indus       1767.1486
## chas         126.5285
## nox         2707.2736
## rm         10326.6731
## age          831.3984
## dis         2159.9545
## rad          336.1964
## tax          839.4039
## ptratio     1590.6246
## black        570.6858
## lstat      10629.7072
```

```r
varImpPlot(rf_fit$finalModel, type = 2)
```


rf_fit$finalModel
lstat
rm
nox
dis
crim
indus
ptratio
tax
age
black
rad
zn
chas
0
2000
4000
6000
8000
10000
IncNodePurity