IS624 - Assignment4

James Quacinella

06/30/2015

Contents

8.1. Recreate the simulated data from Exercise 7.2:

```
simulated <- mlbench.friedman1(200, sd = 1)
simulated <- cbind(simulated$x, simulated$y)
simulated <- as.data.frame(simulated)
colnames(simulated)[ncol(simulated)] <- "y"
```

(a) Fit a random forest model to all of the predictors, then estimate the variable importance scores. Did the random forest model significantly use the uninformative predictors (V6 – V10)?

Answer:

```
model1 <- randomForest(y ~ ., data = simulated, importance = TRUE,
    ntree = 1000)
rfImp1 <- varImp(model1, scale = TRUE)
print(kable(rfImp1[order(-rfImp1), , drop = FALSE]))
```

	Overall
V1	56.5339235
V4	54.4219257
V2	47.0573003
V5	22.2624307
V3	11.2954742
V6	3.7921306
V7	0.7651840
V10	-0.7977709
V9	-1.2492892
V8	-1.6608095

Yes, the model uses V1 - V5 mostly, and the V6 - V10 predictors were not as important due to their negative importance values. I decided to set scale to TRUE, unlike the book, as I think the results are clearer.

(b) Now add an additional predictor that is highly correlated with one of the informative predictors. Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1 ?

Answer:

```
# Create model2 with only one dupe column
simulated2 <- simulated
```

```r
simulated2$duplicate1 <- simulated2$V1 + rnorm(200) *
    0.1
model2 <- randomForest(y ~ ., data = simulated2, importance = TRUE,
    ntree = 1000)
rfImp2 <- varImp(model2, scale = TRUE)

# Create model3 with two dupe columns
simulated3 <- simulated2
simulated3$duplicate2 <- simulated3$V1 + rnorm(200) *
    0.1
model3 <- randomForest(y ~ ., data = simulated3, importance = TRUE,
    ntree = 1000)
rfImp3 <- varImp(model3, scale = TRUE)

# Print results across models
rfImp1 <- rbind(rfImp1, data.frame(Overall = c(0, 0),
    row.names = c("duplicate1", "duplicate2")))
rfImp2 <- rbind(rfImp2, data.frame(Overall = c(0),
    row.names = c("duplicate2")))
results <- cbind(rfImp1, rfImp2, rfImp3)
colnames(results) <- c("Model1", "Model2", "Model3")
print(kable(results))
```

	Model1	Model2	Model3
V1	56.5339235	30.9149923	25.5658734
V2	47.0573003	47.4464000	49.3460618
V3	11.2954742	9.0746736	9.8212920
V4	54.4219257	49.5984101	52.5773767
V5	22.2624307	21.2223497	24.9173535
V6	3.7921306	2.8806927	1.6400997
V7	0.7651840	0.9219038	0.7310063
V8	-1.6608095	-0.2735145	-2.1458119
V9	-1.2492892	1.3157743	0.7552134
V10	-0.7977709	1.1440765	-0.0911344
duplicate1	0.0000000	28.1554053	26.3079847
duplicate2	0.0000000	0.0000000	17.4077553

Yes, the overall importance score for V1 decreases (from 39.08 to 28.8 to 23.62 during the sample run above). Other scores change but not by much, while the new duplicate predictors gain in their relative importance (for example, duplicate1 has similar scores as V3, which shows how dangerous correlated predictors can be).

(c) Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et al. (2007). Do these importances show the same pattern as the traditional random forest model?

Answer:

```r
# Build conditional forest models on the variant
# data sets
model1.cforest <- cforest(y ~ ., data = simulated)
model2.cforest <- cforest(y ~ ., data = simulated2)
model3.cforest <- cforest(y ~ ., data = simulated3)

# Find the variable importance from the conditional
# forest models
imp1.cforest <- party::varimp(model1.cforest)
imp2.cforest <- party::varimp(model2.cforest)
imp3.cforest <- party::varimp(model3.cforest)

# To combine into one table, need to add some rows
imp1.cforest <- rbind(as.data.frame(imp1.cforest),
    data.frame(imp1.cforest = c(0, 0), row.names = c("duplicate1",
        "duplicate2")))
imp2.cforest <- rbind(as.data.frame(imp2.cforest),
    data.frame(imp2.cforest = c(0), row.names = c("duplicate1")))

# Print results
results.cforest <- data.frame(model1 = imp1.cforest,
    model2 = imp2.cforest, model3 = imp3.cforest)
colnames(results.cforest) <- c("Orig Model", "Model w/ Dupe",
    "Model w/ 2 Dupes")
print(kable(results.cforest, caption = "Variable Importance Across Models"))
```

Table 3: Variable Importance Across Models

	Orig Model	Model w/ Dupe	Model w/ 2 Dupes
V1	8.5468369	3.8812715	3.4090842
V2	6.8326442	6.1510202	5.9973252
V3	0.0730809	0.0067587	-0.0254594
V4	8.4233886	7.0283161	6.5538858
V5	1.8948773	1.5198802	1.4704666
V6	0.0035832	-0.0431127	0.0084757
V7	0.0395047	0.0635187	0.0379936
V8	-0.0112969	-0.0008807	-0.0151039
V9	-0.0104745	0.0251427	0.0173890
V10	-0.0331175	-0.0135344	-0.0463026
duplicate1	0.0000000	5.9682738	5.1321124
duplicate2	0.0000000	0.0000000	1.4324943

```r
# Find the variable importance from the conditional
# forest models
imp1.cforest_true <- party::varimp(model1.cforest,
    conditional = TRUE)
imp2.cforest_true <- party::varimp(model2.cforest,
    conditional = TRUE)
imp3.cforest_true <- party::varimp(model3.cforest,
    conditional = TRUE)
```

```r
# To combine into one table, need to add some rows
imp1.cforest_true <- rbind(as.data.frame(imp1.cforest_true),
    data.frame(imp1.cforest_true = c(0, 0), row.names = c("duplicate1",
        "duplicate2")))
imp2.cforest_true <- rbind(as.data.frame(imp2.cforest_true),
    data.frame(imp2.cforest_true = c(0), row.names = c("duplicate1")))

# Print results
results.cforest_true <- data.frame(model1 = imp1.cforest_true,
    model2 = imp2.cforest_true, model3 = imp3.cforest_true)
colnames(results.cforest_true) <- c("Orig Model", "Model w/ Dupe",
    "Model w/ 2 Dupes")
print(kable(results.cforest_true))
```

	Orig Model	Model w/ Dupe	Model w/ 2 Dupes
V1	5.3802458	1.6025194	1.2530366
V2	5.4926456	4.8881107	4.6901711
V3	0.0132711	-0.0141322	-0.0227616
V4	6.7023292	5.3234814	5.3868344
V5	1.1894489	1.0374333	0.9950283
V6	-0.0051026	-0.0130307	0.0059756
V7	0.0188659	0.0079896	0.0233327
V8	-0.0069813	0.0013708	-0.0217417
V9	0.0022450	0.0066764	0.0114505
V10	0.0174840	0.0221737	-0.0205907
duplicate1	0.0000000	2.3155143	1.7130853
duplicate2	0.0000000	0.0000000	0.4552489

The pattern does seem to repeat itself for conditional models, with both ways of calculating variable importance.

(d) Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?

Answer:

Boosted Trees

```r
# Build conditional forest models on the variant
# data sets
model1.gbm <- gbm(y ~ ., data = simulated, distribution = "gaussian")
model2.gbm <- gbm(y ~ ., data = simulated2, distribution = "gaussian")
model3.gbm <- gbm(y ~ ., data = simulated3, distribution = "gaussian")

# Print results
results.gbm <- data.frame(model1 = c(summary.gbm(model1.gbm,
    plotit = FALSE)$rel.inf, 0, 0), model2 = c(summary.gbm(model2.gbm,
    plotit = FALSE)$rel.inf, 0), model3 = summary.gbm(model3.gbm,
    plotit = FALSE)$rel.inf, row.names = rownames(results.cforest_true))
```

```r
colnames(results.gbm) <- c("Orig Model", "Model w/ Dupe",
    "Model w/ 2 Dupes")
print(kable(results.gbm, caption = "Variable Importance Across gbm Models"))
```

Table 5: Variable Importance Across gbm Models

	Orig Model	Model w/ Dupe	Model w/ 2 Dupes
V1	56.00400	35.527170	45.677192
V2	26.91284	35.207799	29.474551
V3	17.08316	19.889965	14.667900
V4	0.00000	9.375067	8.370285
V5	0.00000	0.000000	1.810073
V6	0.00000	0.000000	0.000000
V7	0.00000	0.000000	0.000000
V8	0.00000	0.000000	0.000000
V9	0.00000	0.000000	0.000000
V10	0.00000	0.000000	0.000000
duplicate1	0.00000	0.000000	0.000000
duplicate2	0.00000	0.000000	0.000000

Cubist

```r
# Create cibist models
model1.cubist <- cubist(x = subset(simulated, select = c(-y)),
    y = simulated$y)
model2.cubist <- cubist(x = subset(simulated2, select = c(-y)),
    y = simulated2$y)
model3.cubist <- cubist(x = subset(simulated3, select = c(-y)),
    y = simulated3$y)

imp1.cubist <- varImp(model1.cubist)
imp2.cubist <- varImp(model2.cubist)

imp1.cubist <- rbind(as.data.frame(imp1.cubist), data.frame(Overall = c(0,
    0), row.names = c("duplicate1", "duplicate2")))
imp2.cubist <- rbind(as.data.frame(imp2.cubist), data.frame(Overall = c(0),
    row.names = c("duplicate1")))

# Print results
results.cubist <- data.frame(model1 = imp1.cubist,
    model2 = imp2.cubist, model3 = varImp(model3.cubist),
    row.names = rownames(results.cforest_true))
colnames(results.cubist) <- c("Orig Model", "Model w/ Dupe",
    "Model w/ 2 Dupes")
print(kable(results.cubist, caption = "Variable Importance Across cubist Models"))
```

Table 6: Variable Importance Across cubist Models

	Orig Model	Model w/ Dupe	Model w/ 2 Dupes
V1	50	50	50
V2	50	50	50
V3	50	50	50
V4	50	50	50
V5	0	50	0
V6	0	50	0
V7	0	0	0
V8	0	0	0
V9	0	0	0
V10	0	0	0
duplicate1	0	0	0
duplicate2	0	0	0

Answer: A similar patterns emerges with V1's importance oging down as we add duplicate columns. Unlike the previous models, however, we see that the other non0important variables are just not used at all in the inital model, and adding dupes has the effect of adding some importance to other variables. The cubist models are esp. odd, since V5 and V6 are added with one dupe, but two dupes brings them back to 0.

8.6. Return to the permeability problem described in Exercises 6.2 and 7.4. Train several tree-based models and evaluate the resampling and test set performance:

(a) Which tree-based model gives the optimal resampling and test set performance?

(b) Do any of these models outperform the covariance or non-covariance based regression models you have previously developed for these data? What criteria did you use to compare models' performance?

(c) Of all the models you have developed thus far, which, if any, would you recommend to replace the permeability laboratory experiment?

Answer:

Cleanup

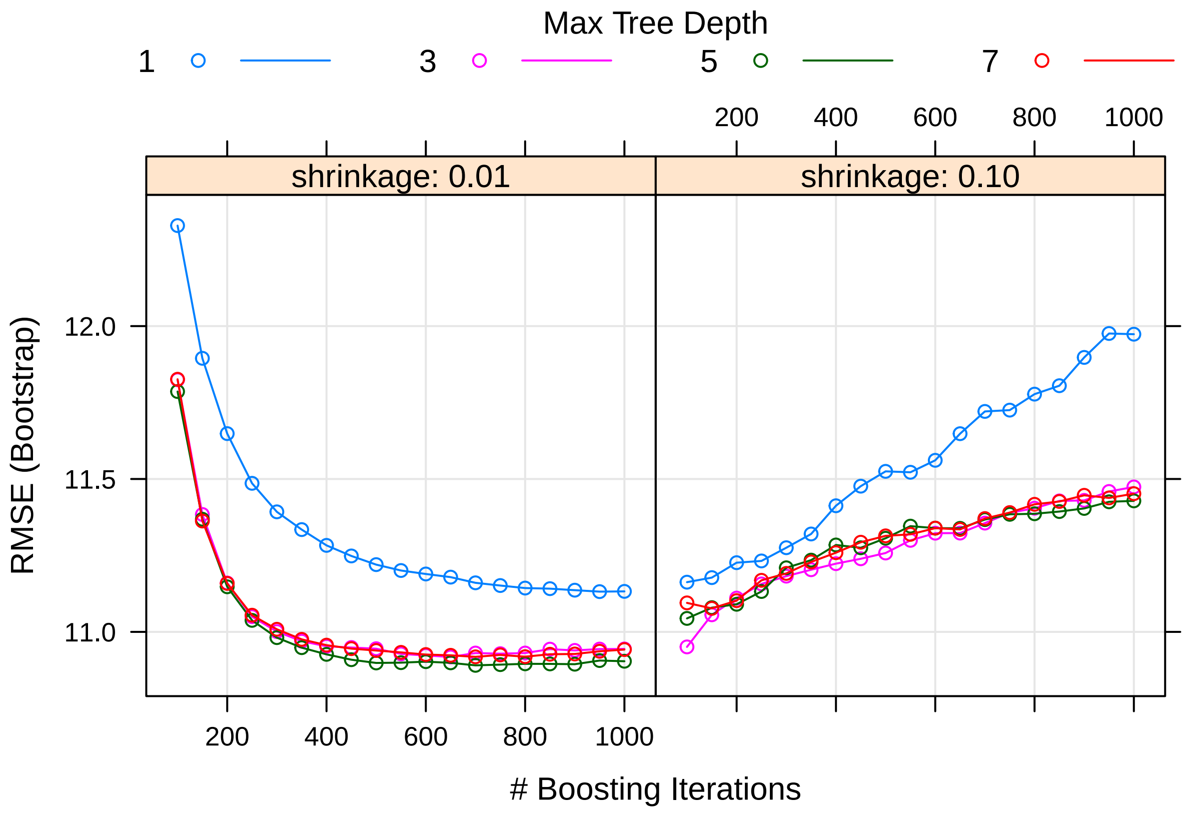
```r
# Get rid of any predictors that are nero-zero
# variance
nearZero <- nearZeroVar(fingerprints)
fingerprints.filtered <- fingerprints[, -nearZero]

# Filter out highly correlated predictors
# correlations <- cor(fingerprints.filtered)
# highCorr <- findCorrelation(correlations, cutoff
# = .9) fingerprints.filtered <-
# fingerprints.filtered[, -highCorr]

# Split the data into a training and test set
indx <- createDataPartition(permeability, p = 0.8,
    list = FALSE)
fingerprints.train <- fingerprints.filtered[indx, ]
fingerprints.test <- fingerprints.filtered[-indx, ]
permeability.train <- permeability[indx, ]
permeability.test <- permeability[-indx, ]

# fingerprints.train <-
# fingerprints.filtered[1:124, ] fingerprints.test
# <- fingerprints.filtered[125:165, ]
# permeability.train <- permeability[1:124, ]
# permeability.test <- permeability[125:165, ]
```

Boosted Trees

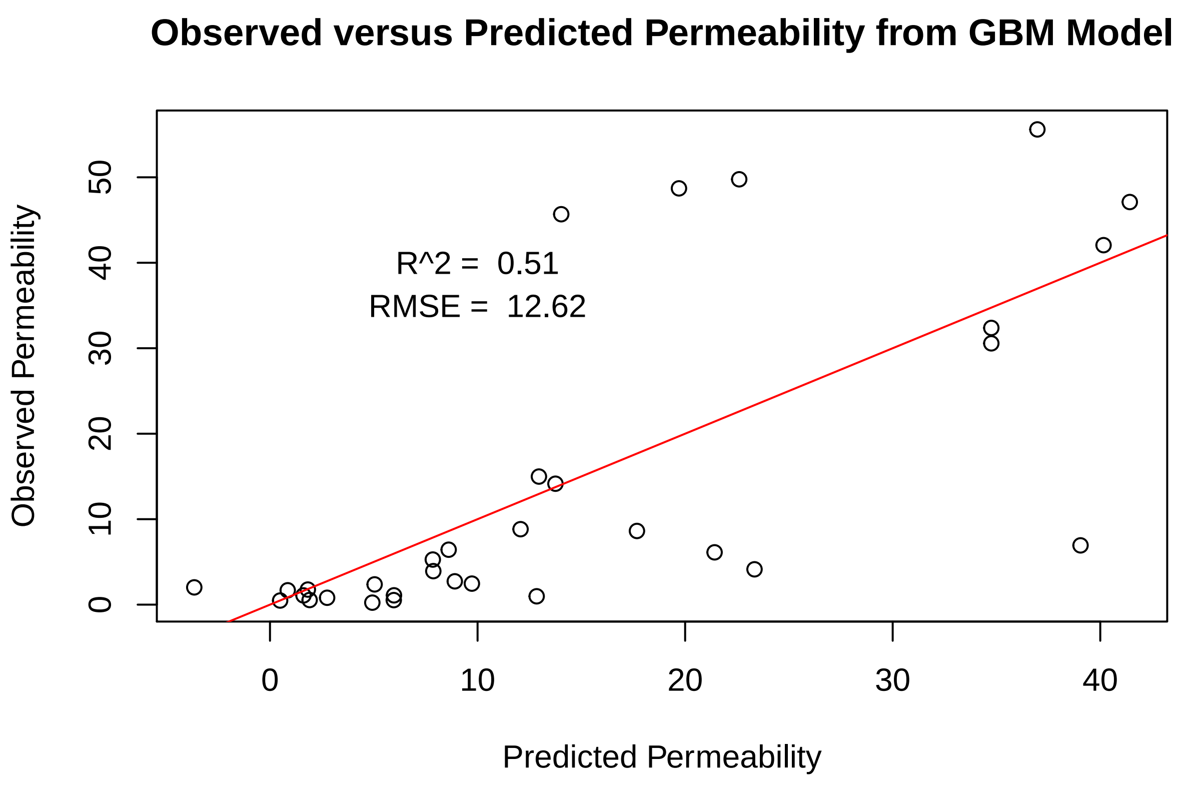
```r
# Use train to get a gbm model, using train params
# from book NOTE: .n.minobsinnode = c(10) was not
# in the book
gbmGrid <- expand.grid(.interaction.depth = seq(1,
    7, by = 2), .n.trees = seq(100, 1000, by = 50),
    .shrinkage = c(0.01, 0.1), .n.minobsinnode = c(10))
permeability.models.gbm <- train(fingerprints.train,
    permeability.train, method = "gbm", tuneGrid = gbmGrid,
    verbose = FALSE)
```

```
## Loading required package: plyr
##
## Attaching package: 'plyr'
##
## The following object is masked from 'package:modeltools':
##
##     empty
```

```
# Plot model
plot(permeability.models.gbm)
```

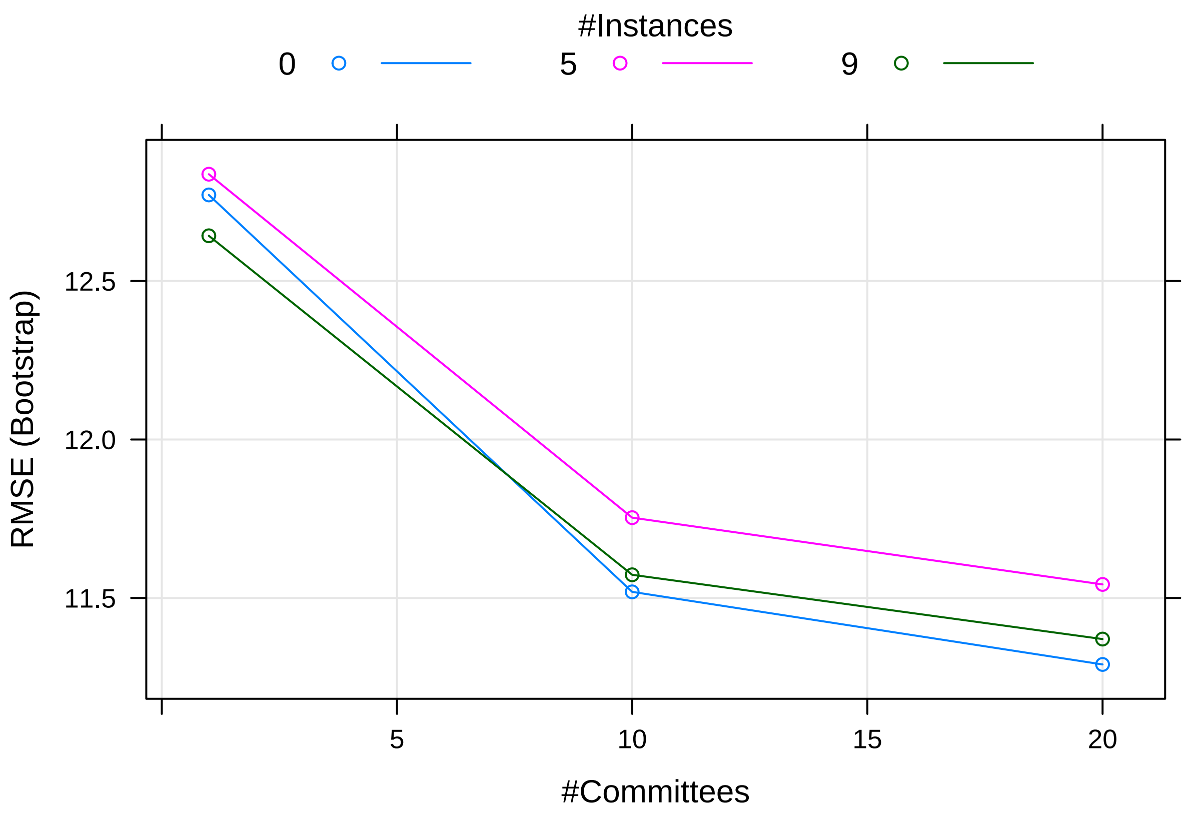


```
# Residuals
permeability.predict.gbm <- predict(permeability.models.gbm,
    fingerprints.test)
plot(permeability.predict.gbm, permeability.test, main = "Observed versus Predicted Permeability from GB
    xlab = "Predicted Permeability", ylab = "Observed Permeability")
abline(0, 1, col = "red")
text(10, 40, paste("R^2 = ", round(cor(permeability.test,
    permeability.predict.gbm)^2, 2)))
text(10, 35, paste("RMSE = ", round(sqrt(sum((permeability.test -
    permeability.predict.gbm)^2)/length(permeability.test)),
    2)))
```

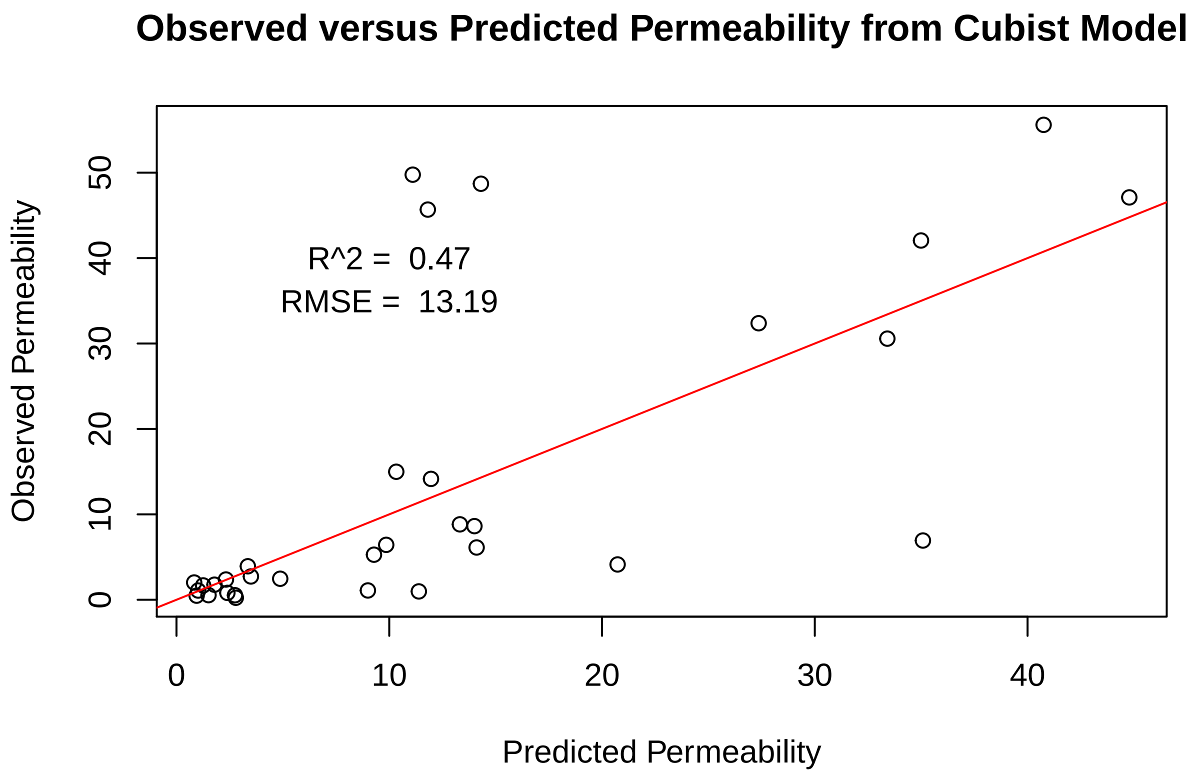


Cubist

```
# permeability.models.cubist <-
# cubist(x=fingerprints.train,
# y=permeability.train)

# Use train to get a cubist model
permeability.models.cubist <- train(x = fingerprints.train,
    y = permeability.train, method = "cubist")

# Show top 10 important variables
# head(varImp(permeability.models.cubist)$importance,
# n=10)

# Plot model
plot(permeability.models.cubist)
```

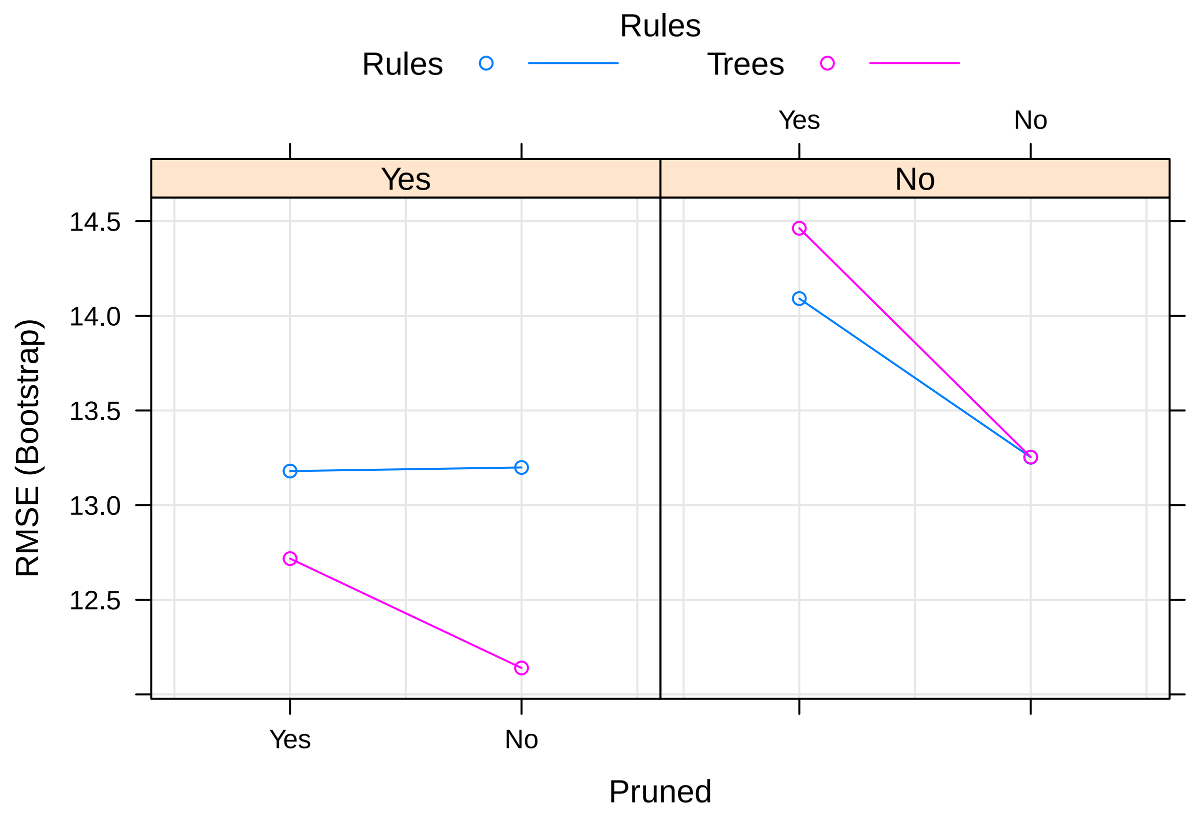


```
# Residuals
permeability.predict.cubist <- predict(permeability.models.cubist,
    fingerprints.test)
plot(permeability.predict.cubist, permeability.test,
    main = "Observed versus Predicted Permeability from Cubist Model",
    xlab = "Predicted Permeability", ylab = "Observed Permeability")
abline(0, 1, col = "red")
text(10, 40, paste("R^2 = ", round(cor(permeability.test,
    permeability.predict.cubist)^2, 2)))
text(10, 35, paste("RMSE = ", round(sqrt(sum((permeability.test -
    permeability.predict.cubist)^2)/length(permeability.test)),
    2)))
```

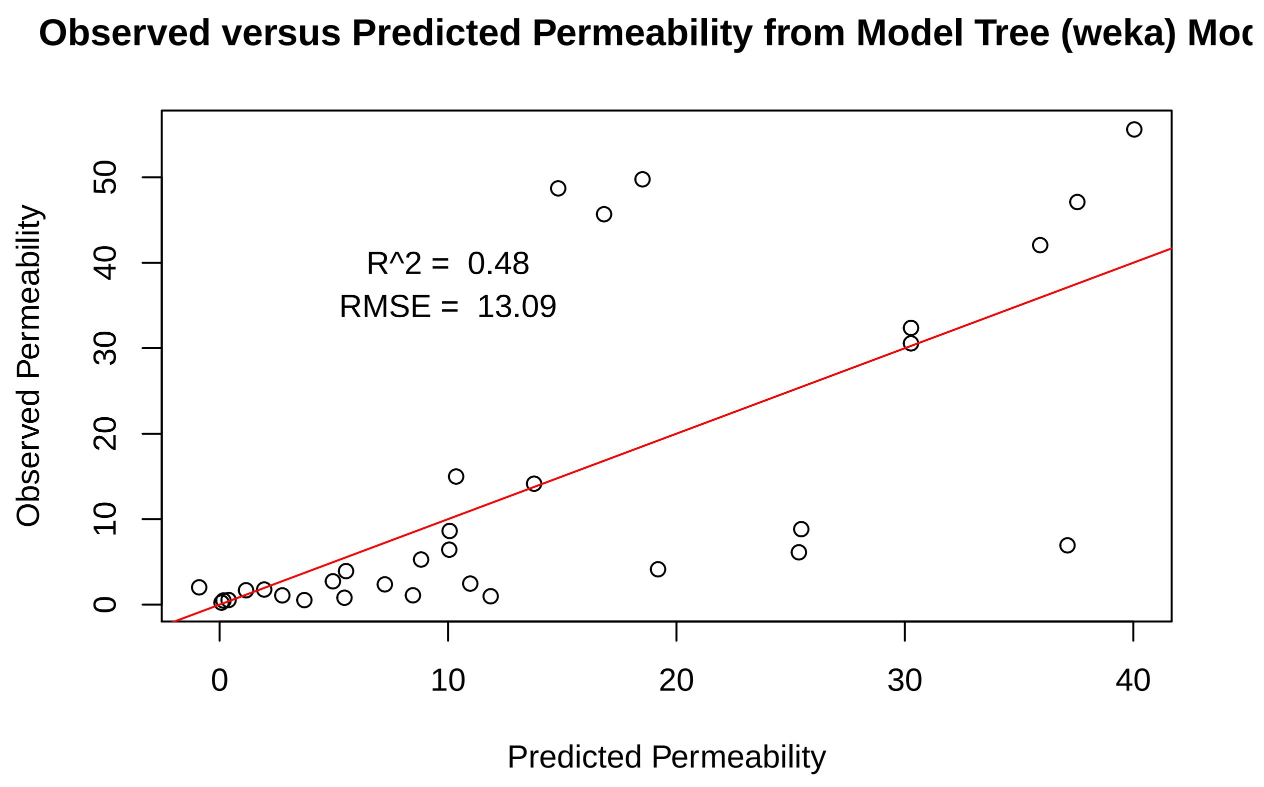


Model Trees

```
permeability.models.m5 <- train(x = fingerprints.train,
    y = permeability.train, method = "M5")
plot(permeability.models.m5)
```



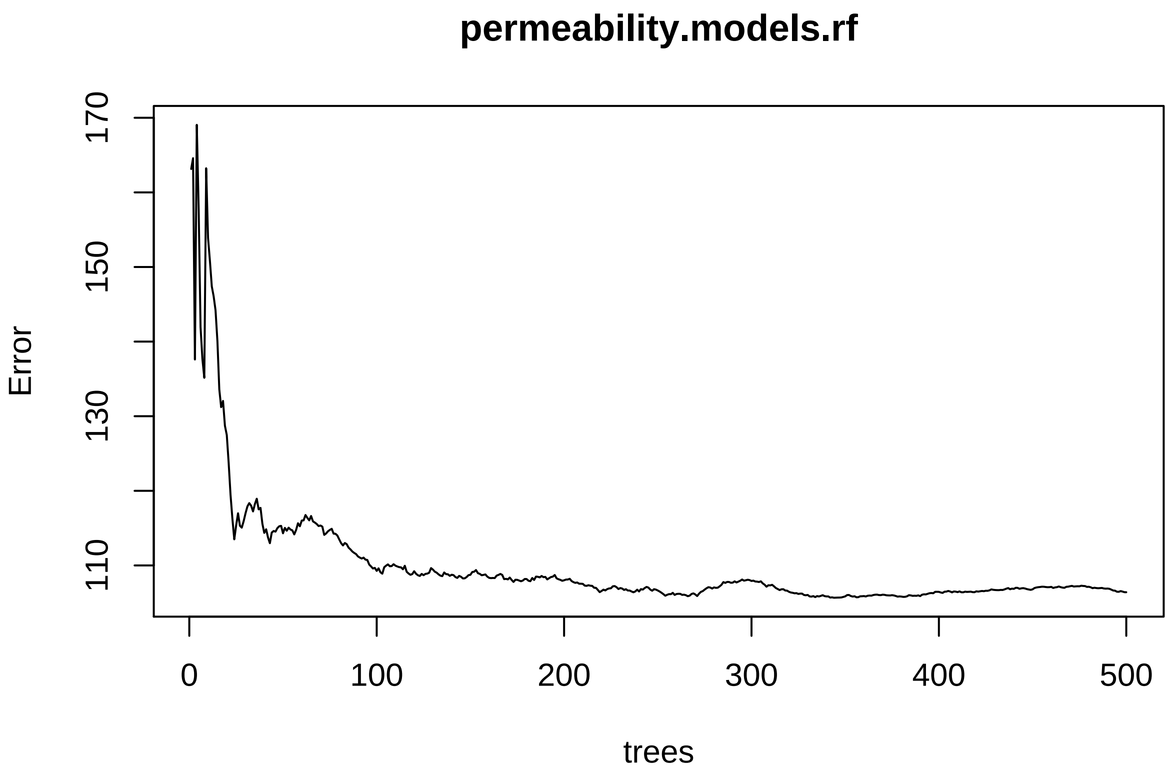
```
permeability.predict.m5 <- predict(permeability.models.m5,
    fingerprints.test)
plot(permeability.predict.m5, permeability.test, main = "Observed versus Predicted Permeability from Moc
    xlab = "Predicted Permeability", ylab = "Observed Permeability")
abline(0, 1, col = "red")
text(10, 40, paste("R^2 = ", round(cor(permeability.test,
    permeability.predict.m5)^2, 2)))
text(10, 35, paste("RMSE = ", round(sqrt(sum((permeability.test -
    permeability.predict.m5)^2)/length(permeability.test)),
    2)))
```



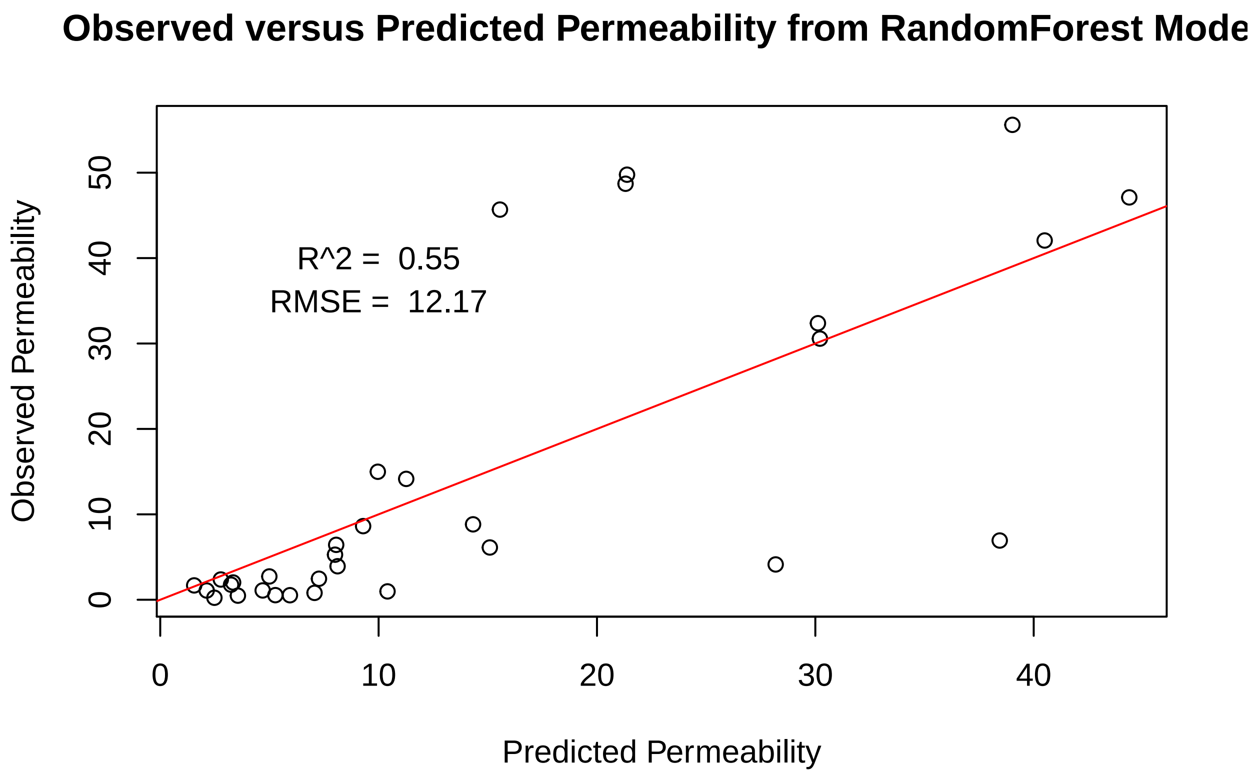
“‘

Random Forest

```
# Build a random forest
permeability.models.rf <- randomForest(fingerprints.train,
    permeability.train, ntrees = 1000)

# Plot model
plot(permeability.models.rf)
```



```
# Residuals
permeability.predict.rf <- predict(permeability.models.rf,
    fingerprints.test)
plot(permeability.predict.rf, permeability.test, main = "Observed versus Predicted Permeability from Ra
    xlab = "Predicted Permeability", ylab = "Observed Permeability")
abline(0, 1, col = "red")
text(10, 40, paste("R^2 = ", round(cor(permeability.test,
    permeability.predict.rf)^2, 2)))
text(10, 35, paste("RMSE = ", round(sqrt(sum((permeability.test -
    permeability.predict.rf)^2)/length(permeability.test)),
    2)))
```



Disucssion

Sadly, looking back at my old models, I realized I messed up when doing the data split, and ended up using the same train and test sets. This makes it impossible to go back and compare my old models until I go back to them and re-eavluate them. This makes me quite sad.

For now, none of these models really did that well and I would not be comfortable suggesting their use. This week I cannot spend too much time on this, but I would play with the preprocessing code (should I increase threshold for correlated variables) as well as the model params.