SL HW 9

Joshua Ingram

11/28/2019

Problem 1

a. In your own words, why do we perform pruning of a large decision tree? What might be the issues with a large tree, and how does pruning help?

With a large tree, it may do very well with predicting the training data, but when used with test data it may have a large error rate. When we prune the tree, we attempt to lower the test error rate and avoid over-fitting.

b. Describe what "bagging" means for decision trees? How do we perform it? Why do we perform it,instead of just using a single pruned decision tree?

Bagging is a form of bootstrap for decision trees. For a given dataset, we create a number of training samples by using sampling with replacement, then we can create a number of trees from these training data. Then we can use this "bag" of trees to obtain a prediction by taking oberservation, putting it through each tree, and obtaining the prediction from each tree, then taking the average (or majority vote for classification) from these trees as the prediciton. This significantly lowers the variance that we see with a single tree method.

c. Explain what "out-of-bag" test error means, and how it is calculated (for regression and classification scenarios, respectively).

The out-of-bag test error is a test error for the observation i of the group of given trees that do not include the observation i in their training data. We find the overall test error for regression by using the predicitons of all the tress that do not include the observation i, for every observation i. Then we take the average of the prediction (like in bagging) and find the MSE. We do this for every observation i, then we average all of the out-of-bag MSE's to find the test error for regression. For classification it is very similar, but the out-of-bag test error we calculuate is the classification error and we average that out.

d. Describe the main difference between bagging and random forests. What tweak is applied in random forests, and why is it applied?

Random Forests work similarly to bagging except at each split for each tree created, the variables used to determine this split is out of m variables from the total amount. This can elimate a problem found in bagging that does not actually lower the variance that much. This problem occurs when there is a variable that dominates across each tree, so there is some correlation between all the bagged trees. By randomizing the variables selected for each split, the change of this dominant variable staying dominant is decreased and the variance is significantly lowered by "decorrelating" these trees.

Problem 2

a.

```
set.seed(1)

carseat.tree.obj <- tree(Sales ~ ., data = Carseats)

carseat.tree.cv <- cv.tree(carseat.tree.obj, K=nrow(Carseats))
best.size <- carseat.tree.cv$size[which.min(carseat.tree.cv$dev)]
carseat.pruned.obj <- prune.tree(carseat.tree.obj, best=best.size)
summary(carseat.pruned.obj)
```

```
##
## Regression tree:
## snip.tree(tree = carseat.tree.obj, nodes = c(10L, 11L, 7L, 4L
## ))
## Variables actually used in tree construction:
## [1] "ShelveLoc" "Price"
## Number of terminal nodes:  5
## Residual mean deviance:  4.412 = 1743 / 395
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -6.56000 -1.45300 -0.08439  0.00000  1.36300  5.41800
```

```
tree.test.MSE <- min(carseat.tree.cv$dev)/nrow(Carseats)
tree.test.MSE
```

```
## [1] 4.974786
```

The optimal pruned decision has a 5 terminal nodes and a MSE test error of 4.97

b.

```
set.seed(1)

carseat.bag.obj <- randomForest(Sales~., data = Carseats, ntree = 200, mtry = ncol(Carseats)-1)
carseat.bag.obj
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, ntree = 200,      mtry = ncol(Carseats) - 1)
##                Type of random forest: regression
##                      Number of trees: 200
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 2.311163
##                     % Var explained: 70.95
```

OOB test error: 2.31

c

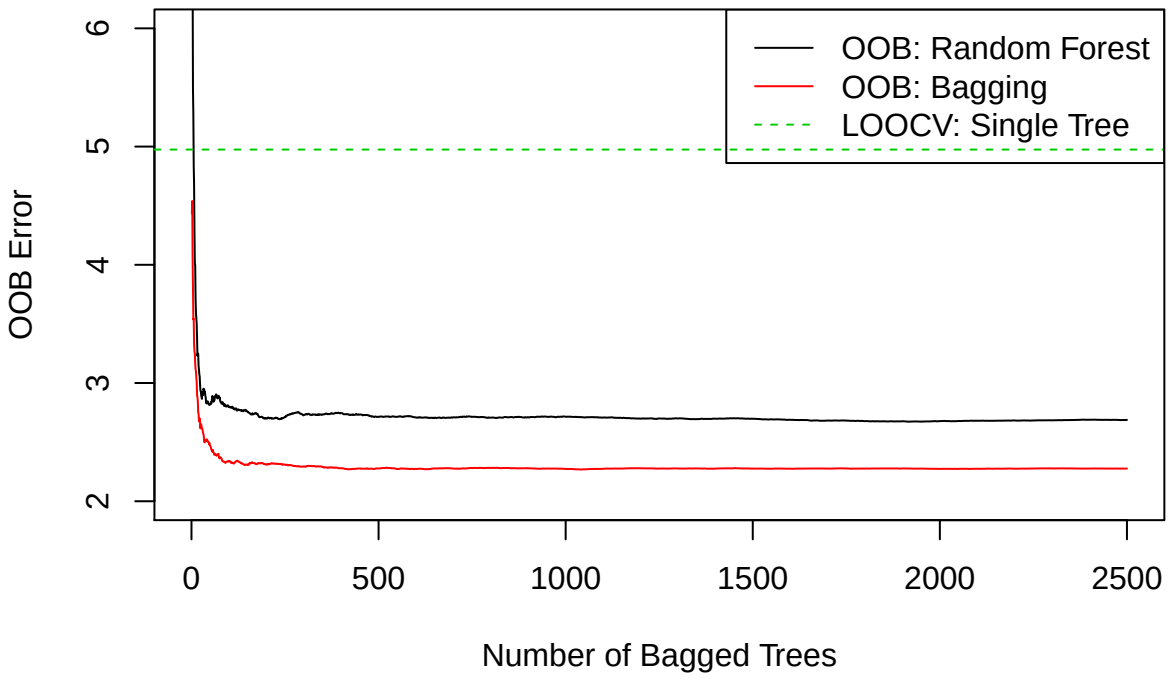
```
set.seed(1)

carseat.forest.obj <- randomForest(Sales~., data = Carseats, ntree = 200, mtry = (ncol(Carseats)-1)/3, i
carseat.forest.obj
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, ntree = 200,      mtry = (ncol(Carseats) - 1)/3,
##                Type of random forest: regression
##                      Number of trees: 200
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 2.666861
##                     % Var explained: 66.48
```

OOB test error: 2.67

d.

```
set.seed(1)
# creating the graph that is similar to that in the slides (slide 22) but with 2500 ntrees as max
bag.obj <- randomForest(Sales~., data = Carseats, ntree = 2500, mtry = ncol(Carseats)-1)
random.forest.obj <-randomForest(Sales~., data = Carseats, ntree = 2500, mtry = (ncol(Carseats)-1)/3, in

plot(random.forest.obj$mse, type='l', xlab = "Number of Bagged Trees", ylab = "OOB Error", col = 1, ylim
lines(bag.obj$mse, type = 'l', col = 2)
abline(h=tree.test.MSE, lty = 2, col = 3)
legend("topright", lty=c(1, 1, 2), col = c(1,2,3), legend=c("OOB: Random Forest", "OOB: Bagging", "LOOC
```

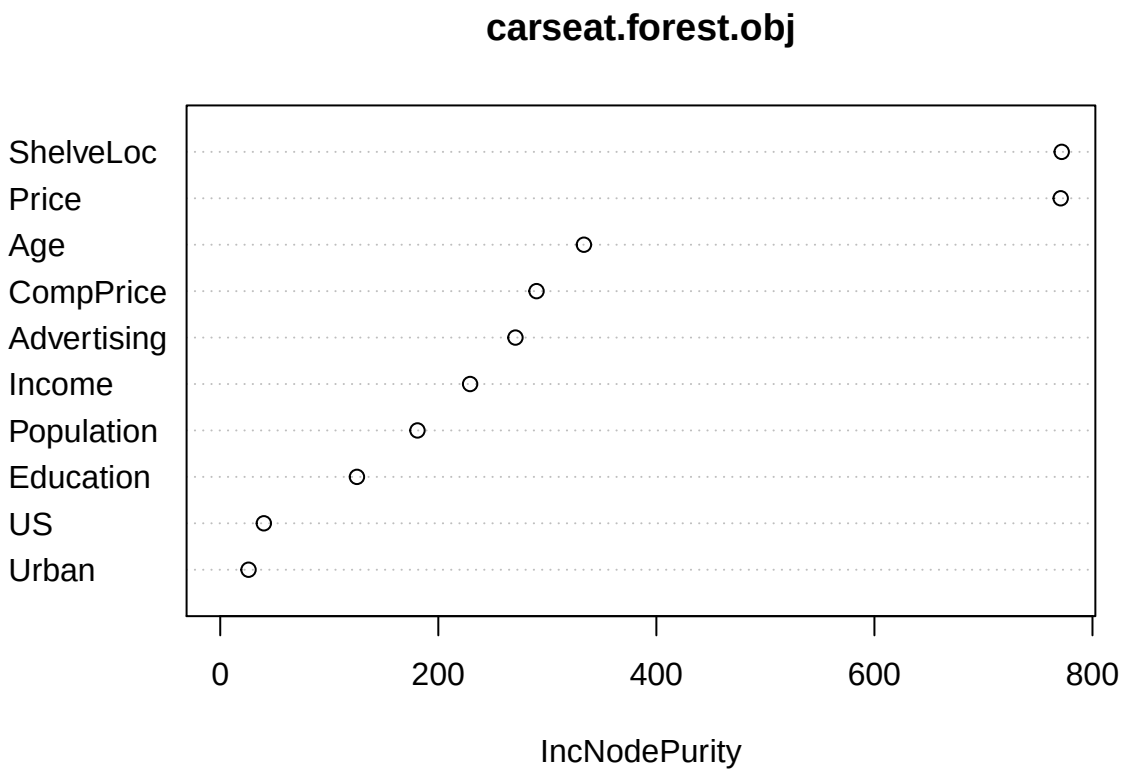


The test error for LOOCV with a single tree is nearly 5, being much larger than the OOB error for both bagging and random forests. However, I am genuinely confused by the fact that the bagging OOB error is not only lower, but seems to be more stable than random forest OOB error.

*** Could this be the from pure random chance for set.seed(1)? I would really appreciate some feedback, as I checked my code and nothing seems wrong.

e.

```
set.seed(1)
varImpPlot(carseat.forest.obj, type = 2)
```



ShelveLoc and Price seem to be the most important in predicting carseat sales

Problem 3

a.

```
set.seed(1)

oj.bag.obj <- randomForest(Purchase~., data = OJ, ntree = 200, mtry = ncol(OJ)-1)
oj.bag.obj
```

```
##
## Call:
##  randomForest(formula = Purchase ~ ., data = OJ, ntree = 200,      mtry = ncol(OJ) - 1)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 17
##
##         OOB estimate of  error rate: 19.63%
## Confusion matrix:
##     CH  MM class.error
## CH 544 109   0.1669219
## MM 101 316   0.2422062
```

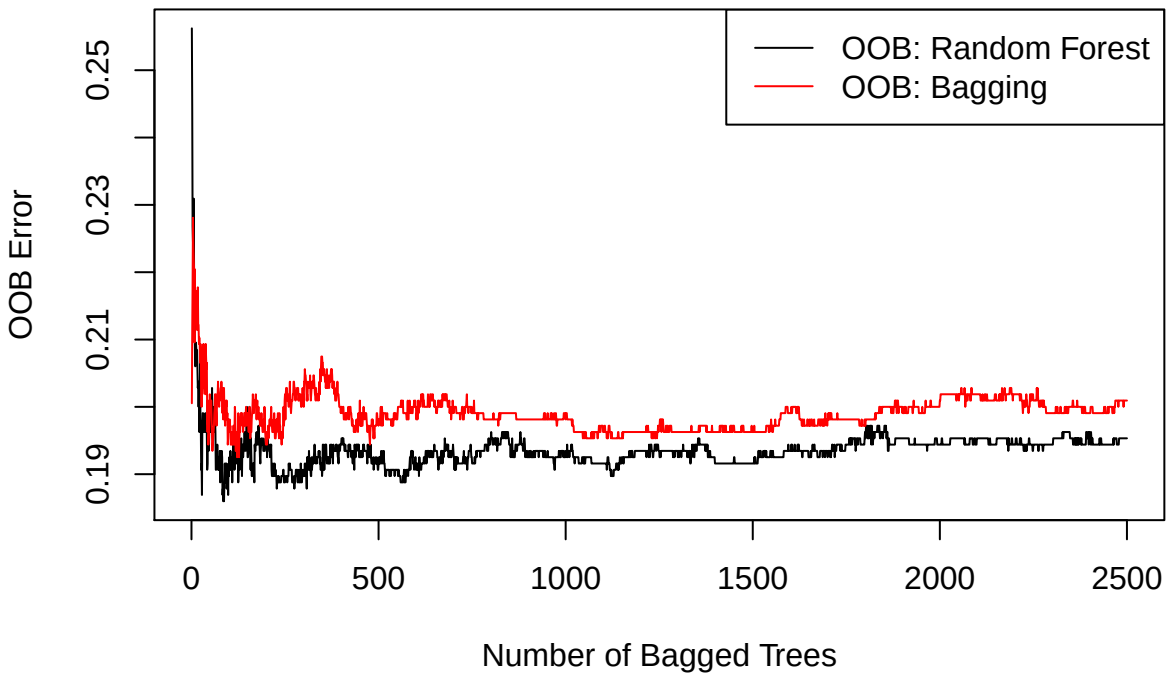
b.

```
set.seed(1)

oj.forest.obj <- randomForest(Purchase~., data = OJ, ntree = 200, mtry = sqrt(ncol(OJ)-1), importance =
oj.forest.obj
```

```
##
## Call:
##  randomForest(formula = Purchase ~ ., data = OJ, ntree = 200,      mtry = sqrt(ncol(OJ) - 1), importa
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 19.25%
## Confusion matrix:
##     CH  MM class.error
## CH 565  88   0.1347626
## MM 118 299   0.2829736
```

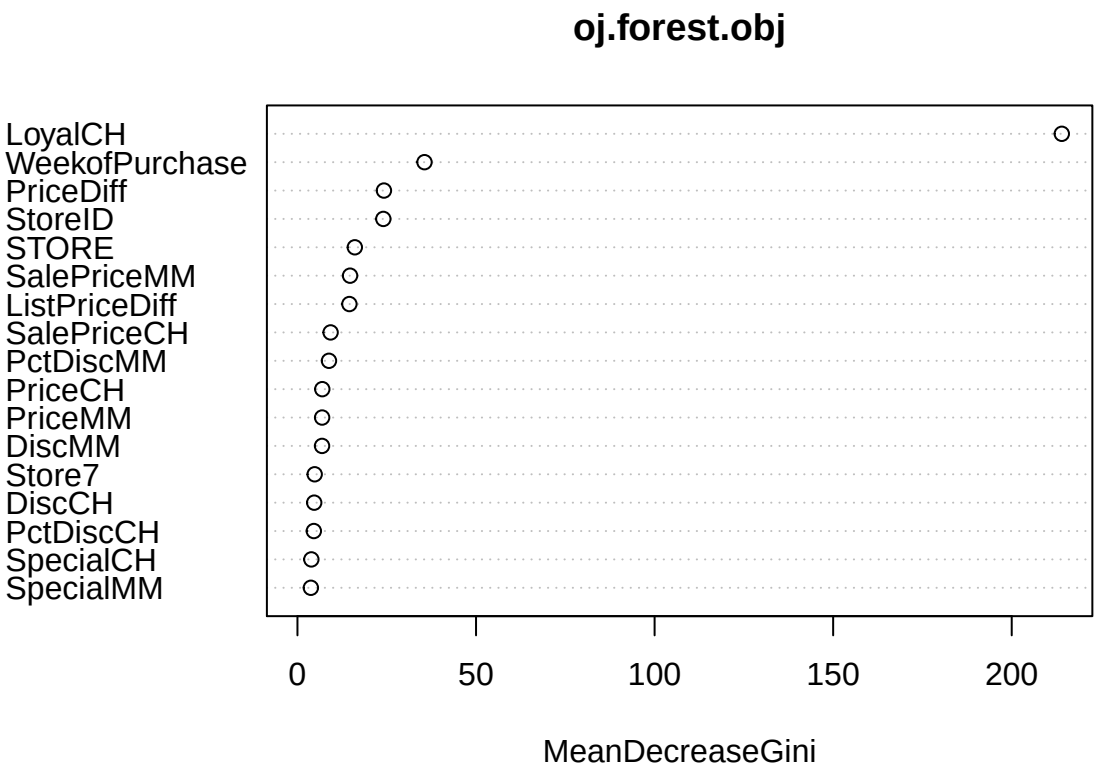
c.

```
set.seed(1)
# creating the graph that is similar to that in the slides (slide 22) but with 2500 ntrees as max
bag.obj <- randomForest(Purchase~., data = OJ, ntree = 2500, mtry = ncol(OJ)-1)
random.forest.obj <-randomForest(Purchase~., data = OJ, ntree = 2500, mtry = sqrt(ncol(OJ)-1), importan

plot(random.forest.obj$err.rate[,1], type='l', xlab = "Number of Bagged Trees", ylab = "OOB Error", col
lines(bag.obj$err.rate[,1], type = 'l', col = 2)
legend("topright", lty=c(1, 1), col = c(1,2), legend=c("OOB: Random Forest", "OOB: Bagging"))
```



d.

```
set.seed(1)
varImpPlot(oj.forest.obj, type = 2)
```



LoyalCH is the most important predictor.

Problem 4

a.

```
set.seed(1)
tree.obj <-tree(factor(Khan$ytrain)~. ,data = data.frame(Khan$xtrain))

tree.cv <- cv.tree(tree.obj, K=nrow(data.frame(Khan$xtrain)))
best.size <- tree.cv$size[which.min(tree.cv$dev)]
pruned.obj <- prune.tree(tree.obj, best=best.size)
summary(pruned.obj)
```

```
##
## Classification tree:
## snip.tree(tree = tree.obj, nodes = 5L)
## Variables actually used in tree construction:
## [1] "X1389" "X108"  "X153"
## Number of terminal nodes:  4
## Residual mean deviance:  0.2385 = 14.07 / 59
## Misclassification error rate: 0.03175 = 2 / 63
```

```
test.error <- min(tree.cv$dev)/nrow(data.frame(Khan$xtrain))
test.error
```

```
## [1] 3.117095
```

Test misclassifcaiton error rate = 3.12%

b.

```
set.seed(1)

bag.obj <- randomForest(factor(Khan$ytrain)~., data = data.frame(Khan$xtrain), ntree = 500, mtry = 2308)
bag.obj
```

```
##
## Call:
##  randomForest(formula = factor(Khan$ytrain) ~ ., data = data.frame(Khan$xtrain),      ntree = 500, mt
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2308
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##    1  2  3  4 class.error
## 1 8  0  0  0           0
## 2 0 23  0  0           0
## 3 0  0 12  0           0
## 4 0  0  0 20           0
```

OOB test misclassifcation error = 0% ## c.

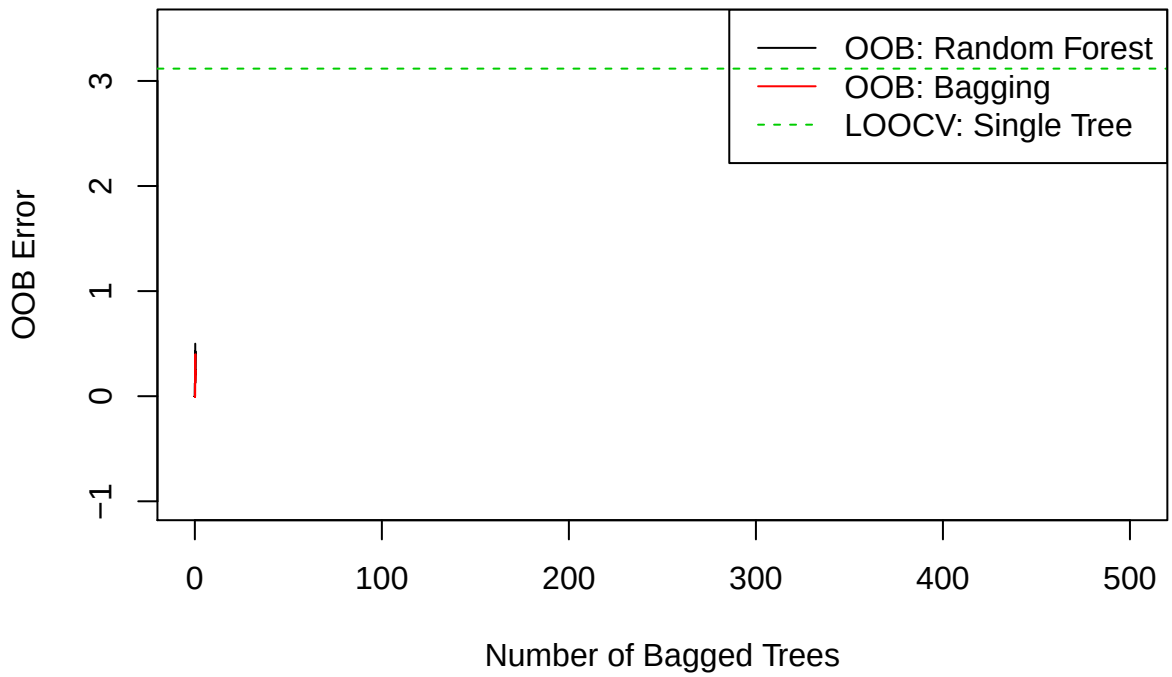
```
set.seed(1)

forest.obj <- randomForest(factor(Khan$ytrain)~., data = data.frame(Khan$xtrain), ntree = 500, mtry = sq
forest.obj
```

```
##
## Call:
##  randomForest(formula = factor(Khan$ytrain) ~ ., data = data.frame(Khan$xtrain),      ntree = 500, mt
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 48
##
##         OOB estimate of  error rate: 1.59%
## Confusion matrix:
##    1  2  3  4 class.error
## 1 8  0  0  0  0.00000000
## 2 0 22  0  1  0.04347826
## 3 0  0 12  0  0.00000000
## 4 0  0  0 20  0.00000000
```

OOB misclassification error rate = 1.59%

d.

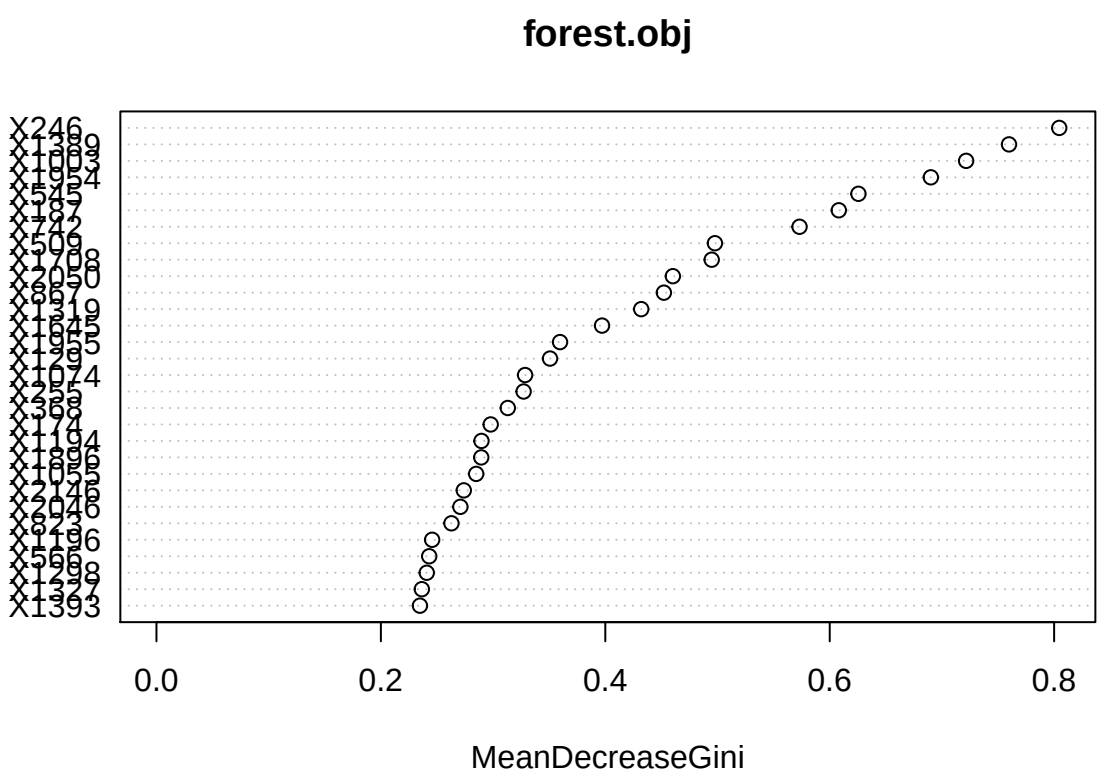
```
set.seed(1)

bag.obj.1 <- randomForest(factor(Khan$ytrain)~., data = data.frame(Khan$xtrain), ntree = 500, mtry = 23
random.forest.obj.1 <- randomForest(factor(Khan$ytrain)~., data = data.frame(Khan$xtrain), ntree = 500,

plot(random.forest.obj.1$err.rate, type='l', xlab = "Number of Bagged Trees", ylab = "OOB Error", col =
lines(bag.obj.1$err.rate, type = 'l', col = 2)
abline(h=test.error, lty = 2, col = 3)
legend("topright", lty=c(1, 1, 2), col = c(1,2,3), legend=c("OOB: Random Forest", "OOB: Bagging", "LOOC
```



A bit odd, but the single tree test error rate expecctedly remains constant at 3.12%, whereas the bagging and random forest rates start somewhere below 1 and quickly drop to 0 (or very close to it). It seems using random forests or bagging with gene predictors is extremely accurate with minimal error rates.

e.

```
set.seed(1)
varImpPlot(forest.obj, type = 2)
```



X246, X1389, X1003, and X1954 seem to be the most singicant in prediction, closely followed by a number of other geenes, but these are the two most notable

f.

```
set.seed(1)

tree.predictions <- predict(tree.obj, newdata = data.frame(Khan$xtest))

predic <- c()
col.index <- 0
for (i in 1:20){
  col.index <- match(max(tree.predictions[i,]), tree.predictions[i,])
  prediction <- as.numeric(colnames(tree.predictions)[col.index])
  predic[i] <- prediction
}
tree.predictions <- factor(predic)

bag.predictions <- predict(bag.obj, newdata  = data.frame(Khan$xtest))

forest.predictions <- predict(forest.obj, newdata = data.frame(Khan$xtest))

counts <- 0
for (i in 1:20){
  if (tree.predictions[i] == Khan$ytest[i]){
```

```
    counts <- counts + 1
}
}

counts <- 20 - counts

error.rate.tree <- counts / 20

counts <- 0
for (i in 1:20){
  if (bag.predictions[i] == Khan$ytest[i]){
    counts <- counts + 1
}
}

counts <- 20 - counts

error.rate.bag <- counts / 20

counts <- 0
for (i in 1:20){
  if (forest.predictions[i] == Khan$ytest[i]){
    counts <- counts + 1
}
}

counts <- 20 - counts

error.rate.forest <- counts / 20

errors <- c(error.rate.tree, error.rate.bag, error.rate.forest)

errors
```

```
## [1] 0.25 0.15 0.10
```

misclassification error rates of Khan$ytest: single tree model: 25% bagging: 15% random forest: 10%