# NBA 4920/6921 Lecture 20

## Ensemble Methods: Boosting Application

Murat Unal

11/09/2021

```r
rm(list=ls())
options(digits = 3, scipen = 999)
library(tidyverse)
library(ISLR)
library(jtools)
library(caret)
library(glmnet)
library(rpart)
library(rpart.plot)
library(ROCR)
library(ipred)
library(vip)
library(randomForest)
library(gbm)
library(ranger)
set.seed(2)
```

```r
Hitters <- ISLR::Hitters
Hitters <- na.omit(Hitters)
train = sample(1:nrow(Hitters), 0.7*nrow(Hitters))
```

# Boosting

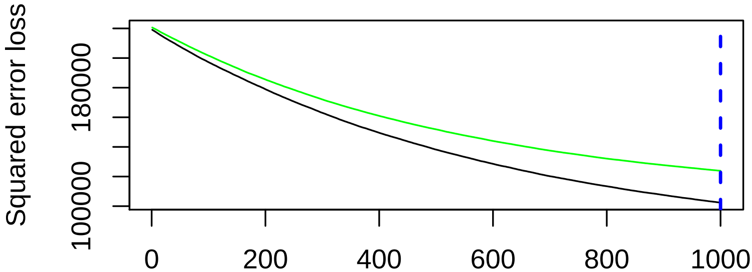
- Let's run a basic GBM model

```
hit_gbm <- gbm(
  formula = Salary ~ .,
  data = Hitters[train,],
  distribution = "gaussian",# SSE loss function
  n.trees = 1000,
  shrinkage = 0.001, #learning rate
  cv.folds = 10,
  interaction.depth = 3 #depth of each tree
)
# find index for number trees with minimum CV error
best <- which.min(hit_gbm$cv.error)
# get MSE and compute RMSE
sqrt(hit_gbm$cv.error[best])
```

```
[1] 352
```

- Results show cross-validated RMSE of rsqrt(hit_gbm$cv.error[best])‘ which we achieved with 1000 trees.
- Training and cross-validated MSE as trees are added to the GBM algorithm
- The small learning rate is resulting in very small incremental improvements which means many trees are required

```
gbm.perf(hit_gbm, method = "cv")
```

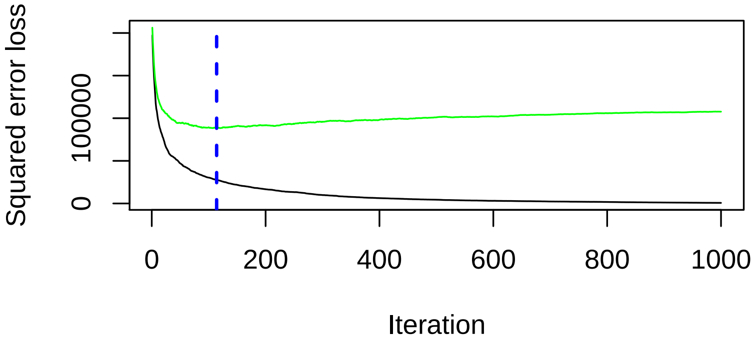


- Let's increase the learning rate to take larger steps down the gradient descent

```r
hit_gbm2 <- gbm(
  formula = Salary ~ .,
  data = Hitters[train,],
  distribution = "gaussian",# SSE loss function
  n.trees = 1000,
  shrinkage = 0.1, #learning rate
  cv.folds = 10,
  interaction.depth = 3 #depth of each tree
)
# find index for number trees with minimum CV error
best <- which.min(hit_gbm2$cv.error)
# get MSE and compute RMSE
sqrt(hit_gbm2$cv.error[best])
```

```
[1] 297
```

```r
gbm.perf(hit_gbm2, method = "cv")
```




```
[1] 114
```

- Make predictions on the test data
- Like most models, we simply use the `predict` function; however, we also need to supply the number of trees to use

```
pred.gbm <- predict.gbm(hit_gbm2,n.trees=1000,Hitters[-trai
rmse.gbm <- sqrt(mean((Hitters[-train,"Salary"] -
                          pred.gbm)^2))
rmse.gbm
```

```
[1] 319
```

- A better option than manually tweaking hyperparameters one at a time is to perform a grid search which iterates over every combination of hyperparameter values and allows us to assess which combination tends to perform well.
- Let's search across 16 models with varying learning rates and tree depth. Let's also vary the minimum number of observations allowed in the trees terminal nodes `n.minobsinnode` and introduce stochastic gradient descent by allowing `bag.fraction < 1`

```
# create hyperparameter grid
hyper_grid <- expand.grid(
  shrinkage = c(.01, .1),
  interaction.depth = c(1, 3),
  n.minobsinnode = c(5, 10),
  bag.fraction = c(.7, .8),
  optimal_trees = 0,
  min_RMSE = 0
)
```

```r
# grid search
for(i in 1:nrow(hyper_grid)) {
   print(i)
  # train model
  gbm.tune <- gbm(
    formula = Salary ~ .,
    distribution = "gaussian",
    data = Hitters[train,],
    n.trees = 1000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    cv.folds = 10)

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$cv.erro
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$cv.error))
}
```

```r
hyper_grid %>%
  arrange(min_RMSE) %>%
  head(10)
```

| shrinkage | interaction.depth | n.minobsinnode | bag.fraction | optimal_trees | min_RMSE |
|---|---|---|---|---|---|
| 0.01 | 3 | 5 | 0.8 | 738 | 286 |
| 0.01 | 3 | 5 | 0.7 | 807 | 294 |
| 0.01 | 3 | 10 | 0.7 | 775 | 296 |
| 0.10 | 3 | 10 | 0.8 | 61 | 298 |
| 0.01 | 3 | 10 | 0.8 | 579 | 298 |
| 0.10 | 3 | 5 | 0.8 | 89 | 302 |
| 0.10 | 3 | 10 | 0.7 | 51 | 305 |
| 0.10 | 3 | 5 | 0.7 | 98 | 306 |
| 0.10 | 1 | 10 | 0.7 | 135 | 308 |
| 0.01 | 1 | 10 | 0.7 | 980 | 315 |

- Once we have found our top model we train a model with those specific parameters.

```
best.model <- hyper_grid %>%
  arrange(min_RMSE) %>%
  head(1)
best.model
```

| shrinkage | interaction.depth | n.minobsinnode | bag.fraction | optimal_trees | min_RMSE |
|---|---|---|---|---|---|
| 0.01 | 3 | 5 | 0.8 | 738 | 286 |

- Let's re-run the GBM model with optimal hyper parameters

```
hit_gbm.final <- gbm(
  formula = Salary ~ .,
  data = Hitters[train,],
  distribution = "gaussian",
  n.trees = 1000,
  interaction.depth = best.model$interaction.depth,
  shrinkage = best.model$shrinkage,
  n.minobsinnode = best.model$n.minobsinnode,
  bag.fraction = best.model$bag.fraction,
  cv.folds = 10)
# find index for number trees with minimum CV error
best <- which.min(hit_gbm.final$cv.error)
# get MSE and compute RMSE
sqrt(hit_gbm.final$cv.error[best])
```

```
[1] 298
```

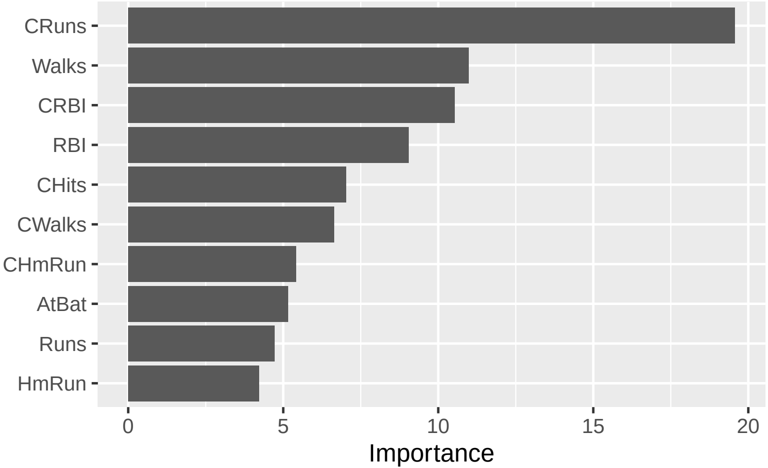
- Make predictions on the test data

```
pred.gbm.final <- predict.gbm(hit_gbm.final, n.trees=1000,
rmse.gbm.final <- sqrt(mean((Hitters[-train,"Salary"] -
                            pred.gbm.final)^2))
rmse.gbm.final
```

```
[1] 276
```

- Variable Importance Plot

```
vip(hit_gbm.final)
```



- Compare prediction performance

```
rmse.tree <- 323
rmse.bag <- 269
rmse.rf <- 255
rmse.gbm
```

```
[1] 319
```

```
rmse.gbm.final
```

```
[1] 276
```

# Exercise

## Boosting

```
cars_train <- read.csv("cayugacars_train.csv")
cars_test <- read.csv("cayugacars_test.csv")
cars_train$customer_bid <- ifelse(cars_train$customer_bid==
cars_test$customer_bid <- ifelse(cars_test$customer_bid=="Y
```

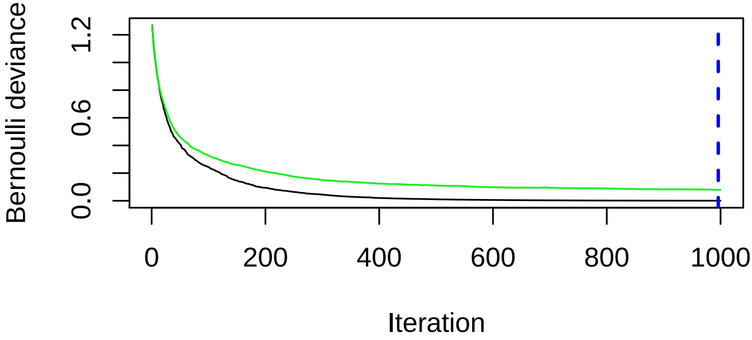
- Run a simple boosting model

```r
cars.gbm <- gbm(
  formula = customer_bid ~ .,
  data = cars_train,
  distribution = "bernoulli",
  n.trees = 1000,
  shrinkage = 0.1, #learning rate
  cv.folds = 10,
  interaction.depth = 3 #depth of each tree
)
# find index for number trees with minimum CV error
best <- which.min(cars.gbm$cv.error)
# get MSE and compute RMSE
cars.gbm$cv.error[best]
```

```
[1] 0.0788
```

- Plot the cv.error

```
gbm.perf(cars.gbm, method = "cv")
```




```
[1] 996
```

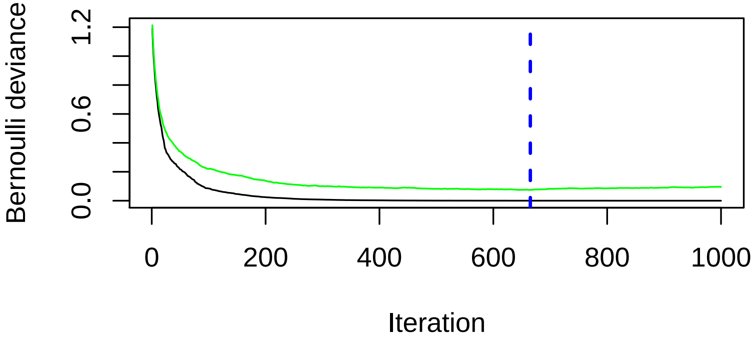
- Change the learning rate

```r
cars.gbm <- gbm(
  formula = customer_bid ~ .,
  data = cars_train,
  distribution = "bernoulli",
  n.trees = 1000,
  shrinkage = 0.2, #learning rate
  cv.folds = 10,
  interaction.depth = 3 #depth of each tree
)
# find index for number trees with minimum CV error
best <- which.min(cars.gbm$cv.error)
# get MSE and compute RMSE
cars.gbm$cv.error[best]
```

```
[1] 0.0742
```

- Plot the cv.error

```r
gbm.perf(cars.gbm, method = "cv")
```




```
[1] 665
```

- Create hyperparameter grid

```r
hyper_grid <- expand.grid(
  shrinkage = c(.01, .2),
  interaction.depth = c(1, 3),
  n.minobsinnode = c(5, 10),
  bag.fraction = c(.7, .8),
  optimal_trees = 0,
  min_RMSE = 0
)
# total number of combinations
nrow(hyper_grid)
```

```
[1] 16
```

- Run the model

```
# grid search
for(i in 1:nrow(hyper_grid)) {
   print(i)
  # train model
  gbm.tune <- gbm(
    formula = customer_bid ~ .,
    distribution = "bernoulli",
    data = cars_train,
    n.trees = 1000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    cv.folds = 10)
  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$cv.erro
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$cv.error))
}
```

- Sort the results

```
hyper_grid %>%
  arrange(min_RMSE) %>%
  head(10)
```

| shrinkage | interaction.depth | n.minobsinnode | bag.fraction | optimal_trees | min_RMSE |
|---|---|---|---|---|---|
| 0.20 | 3 | 5 | 0.8 | 839 | 0.244 |
| 0.20 | 3 | 10 | 0.8 | 652 | 0.267 |
| 0.20 | 3 | 10 | 0.7 | 664 | 0.271 |
| 0.20 | 3 | 5 | 0.7 | 571 | 0.298 |
| 0.01 | 3 | 10 | 0.8 | 1000 | 0.551 |
| 0.01 | 3 | 10 | 0.7 | 1000 | 0.560 |
| 0.01 | 3 | 5 | 0.8 | 1000 | 0.565 |
| 0.01 | 3 | 5 | 0.7 | 1000 | 0.565 |
| 0.20 | 1 | 10 | 0.7 | 66 | 0.918 |
| 0.20 | 1 | 10 | 0.8 | 85 | 0.921 |

- Train a model with the optimal parameters.

```
best.model <- hyper_grid %>%
  arrange(min_RMSE) %>%
  head(1)
best.model
```

| shrinkage | interaction.depth | n.minobsinnode | bag.fraction | optimal_trees | min_RMSE |
|---|---|---|---|---|---|
| 0.2 | 3 | 5 | 0.8 | 839 | 0.244 |

- Re-run the GBM model with optimal hyper parameters

```
cars.gbm.final <- gbm(
  formula = customer_bid ~ .,
  distribution = "bernoulli",
  data = cars_train,
  n.trees = 1000,
  interaction.depth = best.model$interaction.depth,
  shrinkage = best.model$shrinkage,
  n.minobsinnode = best.model$n.minobsinnode,
  bag.fraction = best.model$bag.fraction,
  cv.folds = 10)
# find index for number trees with minimum CV error
best <- which.min(cars.gbm.final$cv.error)
cars.gbm.final$cv.error[best]
```

```
[1] 0.0682
```

- Make predictions on the test data and classify into classes

```
pred.gbm.final <- predict.gbm(cars.gbm.final, n.trees=1000,
                              type="response",cars_test)
yhat.gbm.final <- as.factor(ifelse(pred.gbm.final>=0.5,1,0)
```

- Confusion Matrix

```r
cm <- confusionMatrix(data=yhat.gbm.final,
reference=as.factor(cars_test$customer_bid),
positive="1")
cm$table
```

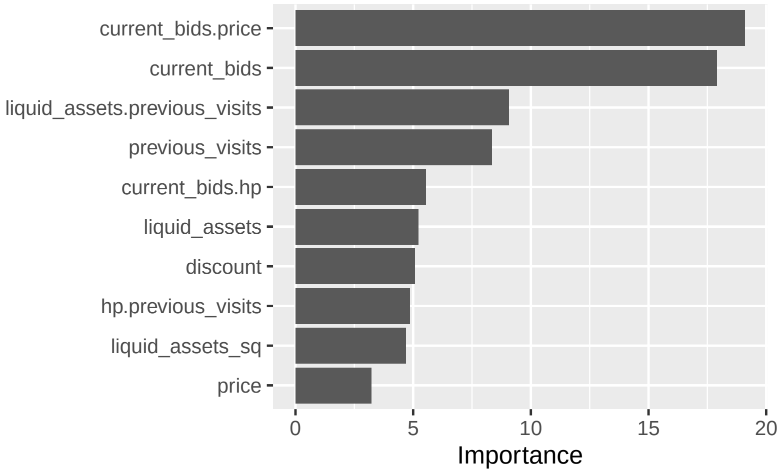
```
          Reference
Prediction   0   1
         0 187   0
         1   1 119
```

```r
c(cm$overall[1],cm$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.997       1.000       0.995       0.996
```

- Variable Importance Plot

```
vip(cars.gbm.final)
```



## Random Forest

- Define tuning grid

```
hyper_grid <- expand.grid(
  mtry        = seq(2, 12, by = 2),
  node_size   = seq(2, 8, by = 2),
  sample_size = c(.5, .70, .80),
  OOB_RMSE    = 0
)
```

## Apply tuning

```
for(i in 1:nrow(hyper_grid)) {
  # train model
  model <- ranger(
    formula         = customer_bid ~ .,
    data            = cars_train,
    num.trees       = 1000,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i]  )
  # add OOB error to grid
  hyper_grid$OOB_RMSE[i] <- model$prediction.error
}
```

- Show tuning results

```
hyper_grid %>%
  arrange(OOB_RMSE) %>% head(10)
```

| mtry | node_size | sample_size | OOB_RMSE |
|---|---|---|---|
| 12 | 4 | 0.8 | 0.066 |
| 12 | 2 | 0.8 | 0.066 |
| 12 | 6 | 0.8 | 0.067 |
| 12 | 8 | 0.8 | 0.068 |
| 12 | 2 | 0.7 | 0.068 |
| 12 | 4 | 0.7 | 0.068 |
| 10 | 2 | 0.8 | 0.068 |
| 12 | 6 | 0.7 | 0.068 |
| 10 | 4 | 0.8 | 0.069 |
| 10 | 6 | 0.8 | 0.069 |

```
best.rf <-hyper_grid %>%
  arrange(OOB_RMSE) %>% head(1)
```

```r
optimal_rf <- ranger(
    formula         = customer_bid ~ .,
    data            = cars_train,
    num.trees       = 1000,
    mtry            = best.rf$mtry,
    min.node.size   = best.rf$node_size,
    sample.fraction = best.rf$sample_size,
    importance      = 'impurity')
```

- Make predictions

```
predict_rf <- predict(optimal_rf, cars_test,type="response'
y.hat_rf <- ifelse(predict_rf>=0.5,1,0)
```

- Confusion Matrix

```
cm <- confusionMatrix(data=as.factor(y.hat_rf),
reference=as.factor(cars_test$customer_bid),
positive="1")
cm$table
```

```
          Reference
Prediction   0   1
         0 184  15
         1   4 104
```

```
c(cm$overall[1],cm$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.938       0.874       0.979       0.916
```

## Bagging

```
cars_bag <- bagging(as.factor(customer_bid) ~ .,data = cars
                    nbagg=500,coob=TRUE,
                    control=rpart.control(cp=0))
cars_bag
```

```
Bagging classification trees with 500 bootstrap replicatio

Call: bagging.data.frame(formula = as.factor(customer_bid)
    nbagg = 500, coob = TRUE, control = rpart.control(cp =

Out-of-bag estimate of misclassification error:  0.0833
```

- Make predictions on the test data

```
cars_pred_bag <- data.frame("p_hat"=predict(cars_bag,
                cars_test,type = "prob")[,"1"],
                        "predicted"=predict(cars_bag,
                cars_test, type = "class"),
                        "actual"=cars_test$customer_bid)
```

- Call the confusion matrix

```
cm_bag <- confusionMatrix(data=as.factor(cars_pred_bag$pred
                  reference=as.factor(cars_pred_bag$actual),
                  positive="1")
cm_bag$table
```

```
          Reference
Prediction   0   1
         0 182  15
         1   6 104
```

```
c(cm_bag$overall[1],cm_bag$byClass[c(1,2,7)])
```

```
   Accuracy Sensitivity Specificity          F1
      0.932       0.874       0.968       0.908
```