Challenge 1

Manny Duran, Sally Rong, Jenny Wu. Xixi Zheng

2/10/2021

Introduction

In this assignment, we use supervised learning methods to train and evaluate three different predictive models (logistic regression, Ridge regression, and boosted trees) to predict whether or not a criminal defendant will recidivate, utilizing data in the recidivism_data_sample.csv dataset. From the three models we train and evaluate, we proceed to select a final model to best deploy in the real world.

Key Decision Made in Assesing our Models

In performing model evaluation to select our best model (boosted trees in our case), we consider two error metrics mainly: False Negative Rate (FNR) and Mean Squared Error (MSE). We care about a low FNR, because of the costs on society for having a high FNR. A low FNR is especially relevant in the context of predicting recidivism, because we do not wish to have wrongly predicted that an individual won't recidivate when they do because that places a large cost to society as a whole. We optimize for a low FNR to lessen the probability of accidentally failing to account for an individual who will recidivate. We chose to set our threshold to 0.45 rather than the standard 0.5 after evaluating and comparing different FPR's and FNR's at different thresholds values to find the one with the smallest difference between FPRs and FNRs. We did this because we wanted to see if we could lower the FNR, which affects the cost of an individual who is wrongly classified as will recidivate in the future, while also making sure that in doing so, we do not add drastically to the FPR.

We also made the decision to evalute predicted probabilities with classification metrics because using predicted probability metrics allows us to assess how well the predicted values compare to true outcomes without having to choose thresholds. In this case, we looked at the MSE score. We chose MSE over alternatives because our outcome variable is binary, meaning that there are no outliers and that our data is not skewed. Mistakes made in prediction are all equally costly due to the binary nature of our outcome recidivate. In cases in which the outcome variable is binary (will recidivate or not), optimizing for a lower MSE is desirable. Additionally, we chose to look at predicted probability because our goal is not complete automation, but to provide an input into the richer decision making processes of court practices, procedures, and the judge's expertise. Predicted probabilities allow for a more comprehensive understanding of information being presented compared to binary values.

Data Set-up

```
dat = read.csv("recidivism_data_sample.csv")
pseudo.dat = read.csv("recidivism_data_pseudo_new.csv")
dat = dat[,-1] #Taking out the ID variable because it isn't pertinent in our data.
dat$race = as.factor(dat$race)
set.seed(54321)
n.total <- nrow(dat)
prop.train <- 0.8
k <- sample(1:n.total, size = round(n.total*prop.train), replace = FALSE)
train.dat = dat[k,]
test.dat = dat[-k,]
```

```
y <- dat$recidivate
x <- model.matrix(recidivate ~ ., data = dat)[,-1]

thresholds <- seq(from = 0.3, to = 0.7, by = 0.05)
set <- c("accuracy", "precision", "recall", "MSE", "FPR", "FNR", "Diff")
```

Logistic Regression

```
#modeling with a logistic regression, using family = "binomial" because our outcomes are binary
log1 = glm(recidivate ~., family = binomial(link = "logit"), data = train.dat)
# use logistic regression to predict on test data
mod2.pred = predict(log1, newdata = test.dat, type = "response")
```

Metrics

```
log.table <- data.frame(matrix(, nrow=9, ncol=7))
colnames(log.table) <- set
rownames(log.table) = thresholds
for(i in 1:length(thresholds)){
  mod2.class = as.numeric(mod2.pred >= thresholds[i]) # classifying predictions as recidivating or not
  TP<- sum(test.dat$recidivate == 1 & mod2.class == 1) # calculate true positives
  FP<- sum(test.dat$recidivate == 0 & mod2.class == 1) # calculate false positives
  TN<- sum(test.dat$recidivate == 0 & mod2.class == 0) # calculate true negatives
  FN<- sum(test.dat$recidivate == 1 & mod2.class == 0) # calculate false negatives
  accuracy <- (TP + TN)/ (TN + FN +TP +FP)
  precision <- (TP)/ (TP + FP)
  recall <- (TP)/ (TP + FN)
  MSE = mean((test.dat$recidivate - mod2.pred)^2)
  FPR <- FP / (FP +TN) # false positive rate
  FNR <- FN / (FN+TP) # false negative rate -- want a model with a low false negative rate
  diff = abs(FPR - FNR)
  log.table[i,] = cbind(accuracy, precision, recall, MSE, FPR, FNR, diff)
}

min.threshold = rownames(log.table)[which.min(log.table[,7])] #saving the threshold that gives us threshold that
 has the lowest absolute difference between FPR and FNR for comparison in the end
min.threshold
```

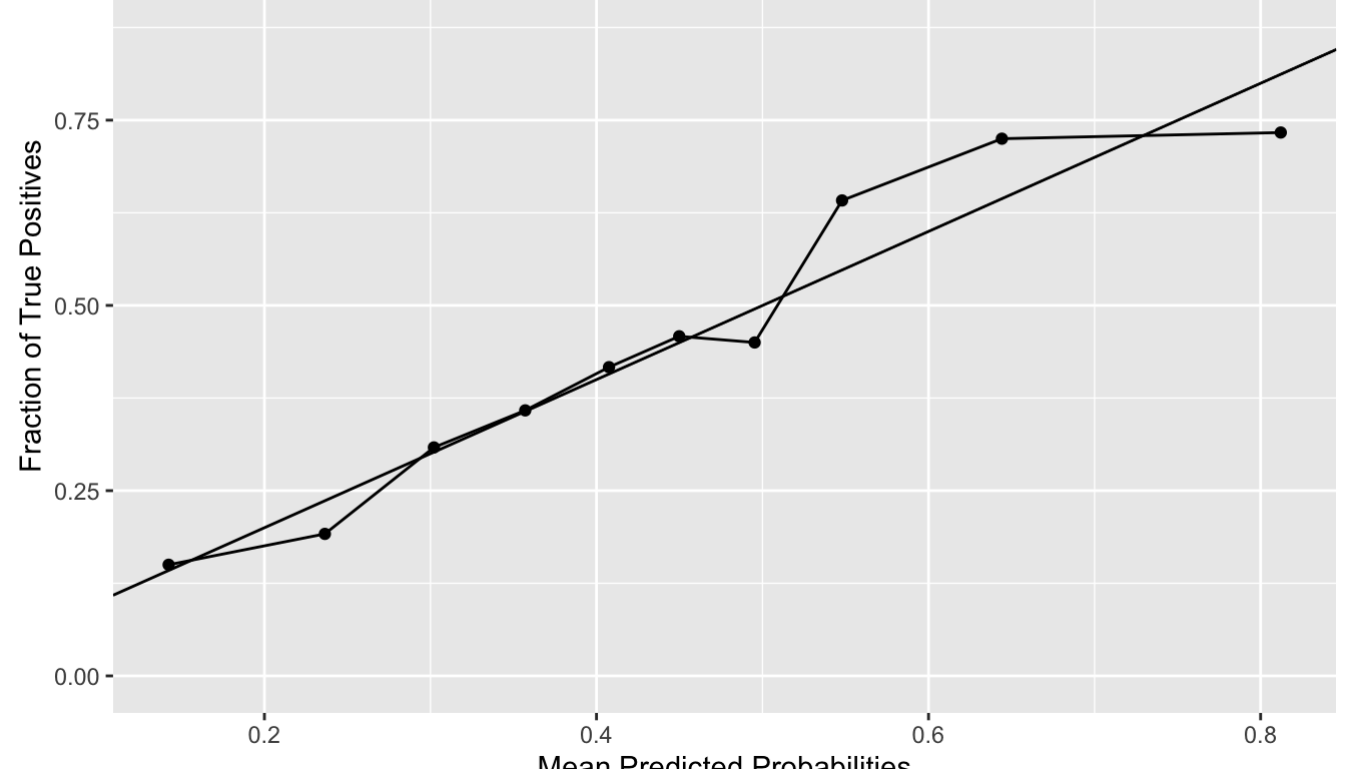
```
## [1] "0.45"
```

```
log.table
```

```
##       accuracy precision    recall       MSE        FPR       FNR       Diff
## 0.3  0.5950000 0.5253863 0.8947368 0.2100388 0.64371257 0.1052632 0.53844942
## 0.35 0.6200000 0.5475000 0.8233083 0.2100388 0.54191617 0.1766917 0.36522444
## 0.4  0.6391667 0.5729013 0.7312030 0.2100388 0.43413174 0.2687970 0.16533474
## 0.45 0.6650000 0.6208178 0.6278195 0.2100388 0.30538922 0.3721805 0.06679123
## 0.5  0.6758333 0.6783042 0.5112782 0.2100388 0.19311377 0.4887218 0.29560803
## 0.55 0.6625000 0.7212544 0.3890977 0.2100388 0.11976048 0.6109023 0.49114178
## 0.6  0.6416667 0.7339450 0.3007519 0.2100388 0.08682635 0.6992481 0.61242177
## 0.65 0.6358333 0.7653631 0.2575188 0.2100388 0.06287425 0.7424812 0.67960695
## 0.7  0.6075000 0.7328244 0.1804511 0.2100388 0.05239521 0.8195489 0.76715366
```

Logistic Calibration Plot

```
# Creating the logistic calibration plot

data = cbind(test.dat, mod2.pred, mod2.class)
data$TP = as.numeric(test.dat$recidivate == 1 & mod2.class == 1) # generate column for true positives
risk_deciles = quantile(data$mod2.pred, probs = seq(from = 0, to = 1, by = 0.1)) # generate risk deciles, produci
ng sample quantiles corresponding to probabilities from 0 to 1 in increments of 0.1

data$risk_decile_bin = as.numeric(cut(data$mod2.pred, breaks = risk_deciles, include.lowest = TRUE)) # create bin
s for risk deciles

some_results = data %>% group_by(risk_decile_bin) %>% summarise(pred.prob = mean(mod2.pred), pos = mean(recidivat
e)) #creating a new data table with the mean predicted probabilities and mean fraction of true positives separate
d into the 10 decile bins created in the previous line


ggplot(some_results, aes(x = pred.prob, y = pos)) + geom_point() + geom_line() +
  ggtitle("Logistic Calibration Plot") + ylim(0,1) + ylab("Fraction of True Positives") +
  xlab("Mean Predicted Probabilities") + geom_abline()
```



Logistic Regression Evaluations

A logistic regression model is useful since our outcome variable is categorical. A logistic regression allows flexibility by producing a predicted probability for recidivism for each individual, which can then be used to produce classification while employing a threshold. The predicted probability estimates are kept within the bounds [0,1]. A logistic regression is also not computationally intensive, and is a flexible model, allowing that we add or drop coefficients with ease. However, this might also pose inefficiency, for the user has to manually decide which coefficients to keep or drop, and some predictors are factors with multiple levels. Another con to note with logistic regression is that it's relatively low model complexity and easier explainability compromises model performance. It has a high FNR of 37.2%, which is not ideal, and one of the reasons we do not utilize logistic regression as our deployment model. The false positive rate (FPR) fares better at 30.5%, but we care more about FNR as an error metric in this context.

Ridge Regression

```
lambdas <- 10^seq(from = 6, to = -2, length = 100) # creating various lambda values
ridge.train <- glmnet(x = x[k,], y = y[k], alpha = 0, family = "binomial", lambda = lambdas) # creating ridge reg
ression with family = "binomial" because our outcomes are binary
test.mse.ridge <- rep(NA,length(lambdas)) # empty object to use with function

for (i in 1:length(lambdas)){
  ridge.pred.test <- predict(ridge.train, newx = x[-k,], s = lambdas[i], type= "response") # use ridge regression
model to predict onto test data iteratively with 1:i(length((lamdas)))
  test.mse.ridge[i] <- mean((y[-k] - ridge.pred.test)^2) # calculating mse for each iteration and storing it
}

min = lambdas[which.min(test.mse.ridge)] # locating the minimum test mse's and its optimal lamda value

ridge = glmnet(x = x[k,], y = y[k], alpha = 0, family = "binomial", lambda = min)# ridge regression model with op
timal lambda value
ridge.pred = predict(ridge, newx = x[-k,], type = "response", s = min) # use ridge regression to predict onto tes
t data
ridge.pred = as.vector(ridge.pred)
```

Metrics

```
ridge.table <- data.frame(matrix(, nrow=9, ncol=7))
colnames(ridge.table) <- set
rownames(ridge.table) = thresholds
for(i in 1:length(thresholds)){
  ridge.class = as.numeric(ridge.pred >= thresholds[i]) # classifying recidivate predictions iteratively for each
threshold value
  TP = sum((y[-k]) == 1 & ridge.class == 1) # calculate true positives
  TN = sum((y[-k]) == 0 & ridge.class == 0) # calculate false positives
  FP = sum((y[-k]) == 0 & ridge.class == 1) # calculate true negatives
  FN = sum((y[-k]) == 1 & ridge.class == 0) # calculate false negatives
  r.accuracy <- (TP + TN)/ (TN + FN +TP +FP)
  r.precision <- (TP)/ (TP + FP)
  r.recall <- (TP)/(TP + FN)
  r.FPR <- FP/(FP+ TN) # false positive rate
  r.FNR <- FN/(FN + TP) # false negative rate -- want a model with a low FPR
  r.mse = mean((y[-k] - ridge.pred)^2)
  r.diff = abs(r.FPR - r.FNR) # absolute value of difference between FPR and FNR
  ridge.table[i,] = cbind(r.accuracy, r.precision, r.recall, r.mse, r.FPR, r.FNR, r.diff)
}
r.min.threshold = rownames(ridge.table)[which.min(ridge.table[,7])] # choosing the lowest diff value's threshold
 as our classification threshold
r.min.threshold
```

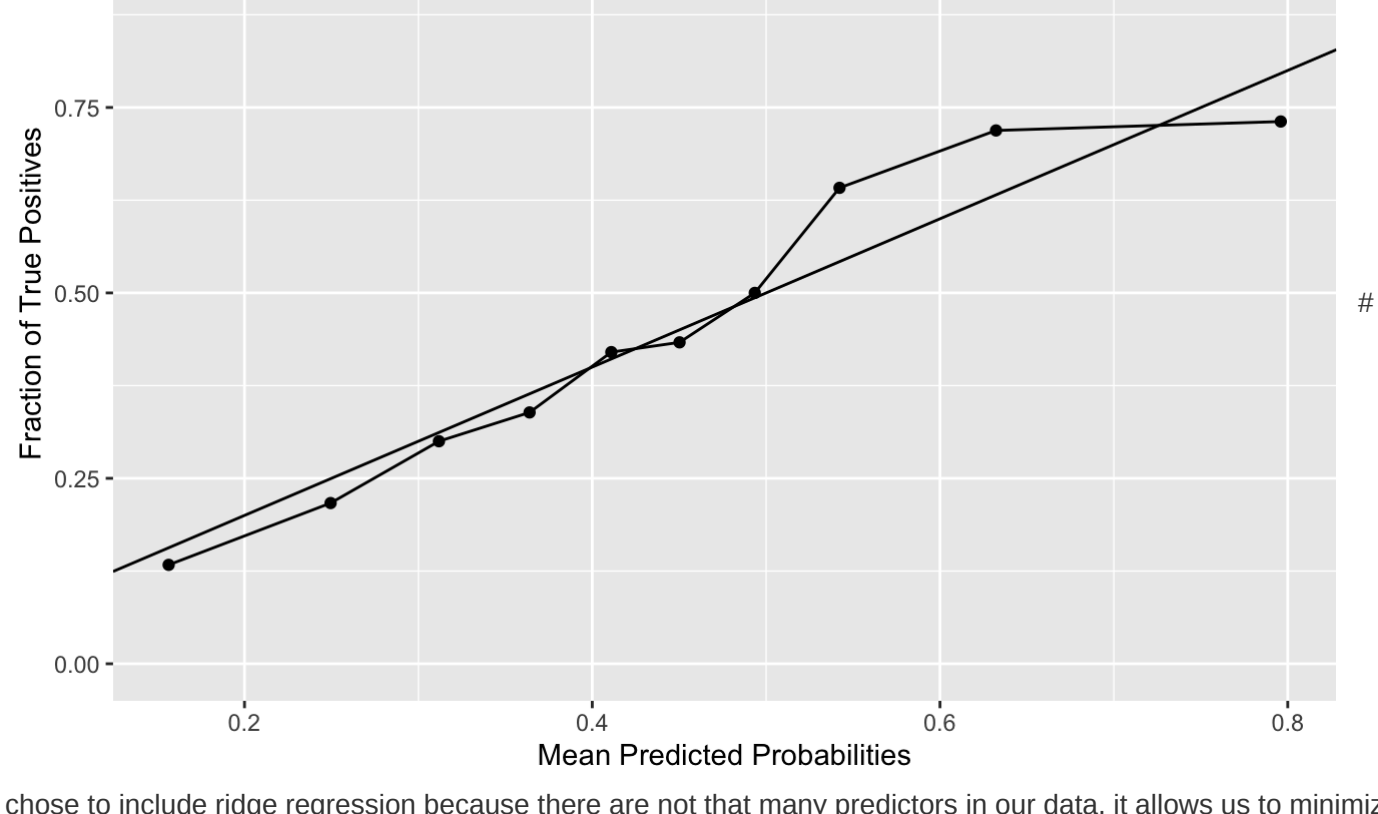
```
## [1] "0.45"
```

```
ridge.table
```

```
##       accuracy precision    recall       MSE        FPR        FNR       Diff
## 0.3  0.5925000 0.5232432 0.9097744 0.2098589 0.66017964 0.09022556 0.56995408
## 0.35 0.6150000 0.5427873 0.8345865 0.2098589 0.55988024 0.16541353 0.39446671
## 0.4  0.6391667 0.5718433 0.7406015 0.2098589 0.44161677 0.25939850 0.18221827
## 0.45 0.6691667 0.6256983 0.6315789 0.2098589 0.30089820 0.36842105 0.06752285
## 0.5  0.6791667 0.6814815 0.5187970 0.2098589 0.19311377 0.48120301 0.28808924
## 0.55 0.6600000 0.7230216 0.3778195 0.2098589 0.11526946 0.62218045 0.50691099
## 0.6  0.6408333 0.7463415 0.2875940 0.2098589 0.07784431 0.71240602 0.63456170
## 0.65 0.6216667 0.7407407 0.2255639 0.2098589 0.06287425 0.77443609 0.71156184
## 0.7  0.5983333 0.7232143 0.1522556 0.2098589 0.04640719 0.84774436 0.80133718
```

Ridge Regression Calibration Plot

```
# Creating the ridge regression calibration plot

# storing predicted values along with test values
ridge.pred = predict(ridge, newx = x[-k,], type = "response", s = min)
ridge.pred = as.vector(ridge.pred)
ridge.graph = cbind(test.dat, ridge.pred)


risk_deciles_ridge = quantile(ridge.graph$ridge.pred, probs = seq(from = 0, to = 1, by = 0.1)) # generate risk de
ciles, producing sample quantiles corresponding to probabilities from 0 to 1 in increments of 0.1


ridge.graph$risk_decile_bin = as.numeric(cut(ridge.graph$ridge.pred, breaks = risk_deciles_ridge, include.lowest
 = TRUE)) # create bins for risk deciles

some_results_ridge = ridge.graph %>% group_by(risk_decile_bin) %>% summarise(pred.prob = mean(ridge.pred), pos =
 mean(recidivate)) #creating a new data table with the mean predicted probabilities and mean fraction of true pos
itives separated into the 10 decile bins created in the previous line

ggplot(some_results_ridge, aes(x = pred.prob, y = pos)) + geom_point() + geom_line() +
  ggtitle("Ridge Regression Calibration Plot") + ylab("Fraction of True Positives") +
  xlab("Mean Predicted Probabilities")+ ylim(0,1) + geom_abline()
```



Ridge Regression Evalutions We chose to include ridge regression because there are not that many predictors in our data, it allows us to minimize irrelevant coefficients, and by doing so it becomes more optimized for prediction on new data sets. This model automatically decides what coefficients to shrink based on tuning parameters which is efficient and allows for optimal bias variance tradeoff. During tests, it produced a FNR of 0.368 and an MSE of 0.210, placing it among the better performing models that we tested (compared to logistic and LASSO regression).

Boosted Trees

```
#create vector of different interaction depth to train model
depths <- c(1,2,3,4,5,6,7)
#empty list that will hold the models trained with different depths
bt.models <- list()
set.seed(2019207)
#Loop through each with cross validation for each interaction depth for a training classification model of booste
d trees with shrinking parameter of 0.01, 1000 trees to fit and with 10 cross validation folds.
for (i in 1:length(depths)){

  bt.models[[i]] <- gbm(recidivate ~  .,
                               data = train.dat, shrinkage = 0.01,
                               distribution="bernoulli", n.trees = 1000,
                               interaction.depth = depths[i],
                               cv.folds = 10)
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

```
#Implementing CV over two parameters: depths and number of trees
n.depths <- length(bt.models)
depths <- rep(NA, n.depths)
min.cv.error <- rep(NA, n.depths)
best.n.trees <- rep(NA, n.depths)

#Considering all of the models, all values of interaction depth and all number of trees,
#we are looking to find the best model.

for (i in 1:n.depths){

  bt.curr <- bt.models[[i]]
  depths[i] <- bt.curr$interaction.depth
  min.cv.error[i] <- min(bt.curr$cv.error)
  best.n.trees[i] <- which.min(bt.curr$cv.error)
  rm(bt.curr)
}

#This gives us the best number of variables to predict upon.
m <- which.min(min.cv.error)
m
```

```
## [1] 3
```

```
#This gives us the best number of trees to create.
final.ntrees <- best.n.trees[m]
final.ntrees
```

```
## [1] 708
```

```
#This gives us the best number of nodes for a tree to have.
final.depth <- depths[m]
final.depth
```

```
## [1] 3
```

```
#finding the predicted probabilities using the best number of m and trees for the test data
test.dat.preds <- predict(bt.models[[m]], newdata = test.dat,
                          n.trees = final.ntrees, type = "response")
```

Boosted Trees Plot

```
bt.one <- bt.models[[3]]
bt.one$n.trees
```

```
## [1] 1000
```

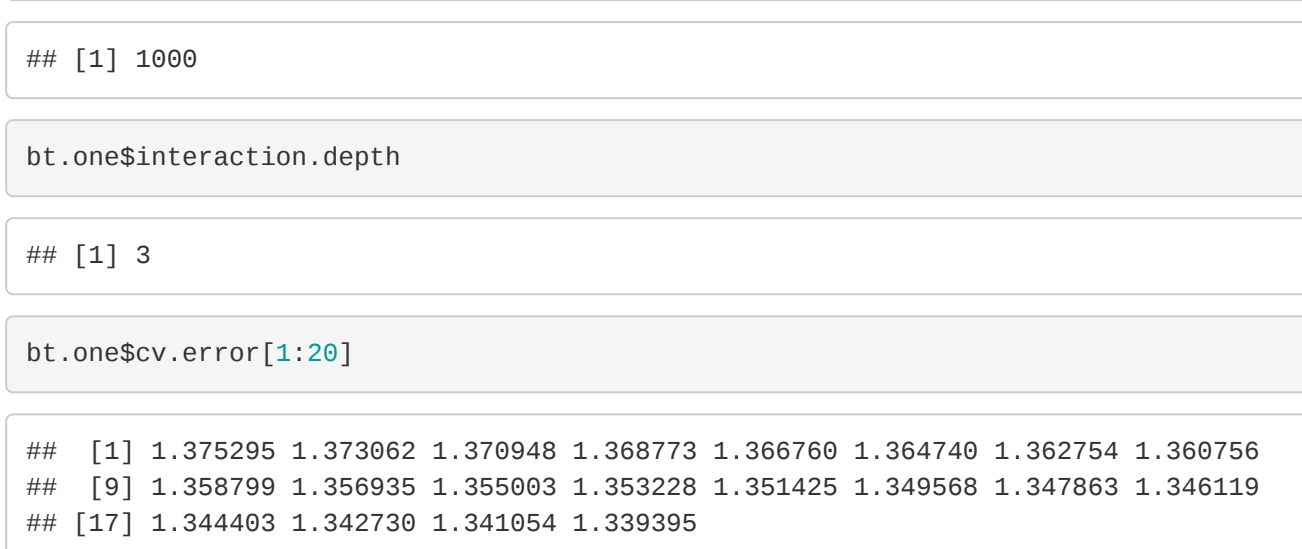
```
bt.one$interaction.depth
```

```
## [1] 3
```

```
bt.one$cv.error[1:20]
```

```
##  [1] 1.375295 1.373062 1.370948 1.368773 1.366760 1.364740 1.362754 1.360756
##  [9] 1.358799 1.356935 1.355003 1.353228 1.351425 1.349568 1.347863 1.346119
## [17] 1.344403 1.342730 1.341054 1.339395
```

```
plot(x = 1:bt.one$n.trees, y = bt.one$cv.error) #Plotting to get the gist of the number of trees that minimizes C
V error
```



Metrics

```
boost.table <- data.frame(matrix(, nrow=9, ncol=7))
colnames(boost.table) <- set
rownames(boost.table) = thresholds
boosteddat = cbind(test.dat, test.dat.preds)
for(i in 1:length(thresholds)){
  boosteddat$yhat = as.numeric(boosteddat$test.dat.preds >= thresholds[i])
  TP = sum(boosteddat$recidivate == 1 & boosteddat$yhat == 1)
  TN = sum(boosteddat$recidivate == 0 & boosteddat$yhat == 0)
  FP = sum(boosteddat$recidivate == 0 & boosteddat$yhat == 1)
  FN = sum(boosteddat$recidivate == 1 & boosteddat$yhat == 0)
  b.accuracy = (TP+TN)/nrow(boosteddat)
  b.precision = TP/(TP + FP)
  b.recall = TP/(TP+FN)
  b.FPR = FP/(FP+TN)
  b.FNR = FN/(FN+TP)
  b.mse = mean((test.dat.preds - test.dat$recidivate)^2)
  b.diff = abs(b.FPR -b.FNR)
  boost.table[i,]<-cbind(b.accuracy, b.precision, b.recall, b.mse, b.FPR, b.FNR, b.diff)
}
b.min.threshold = rownames(boost.table)[which.min(boost.table[,7])]
b.min.threshold
```

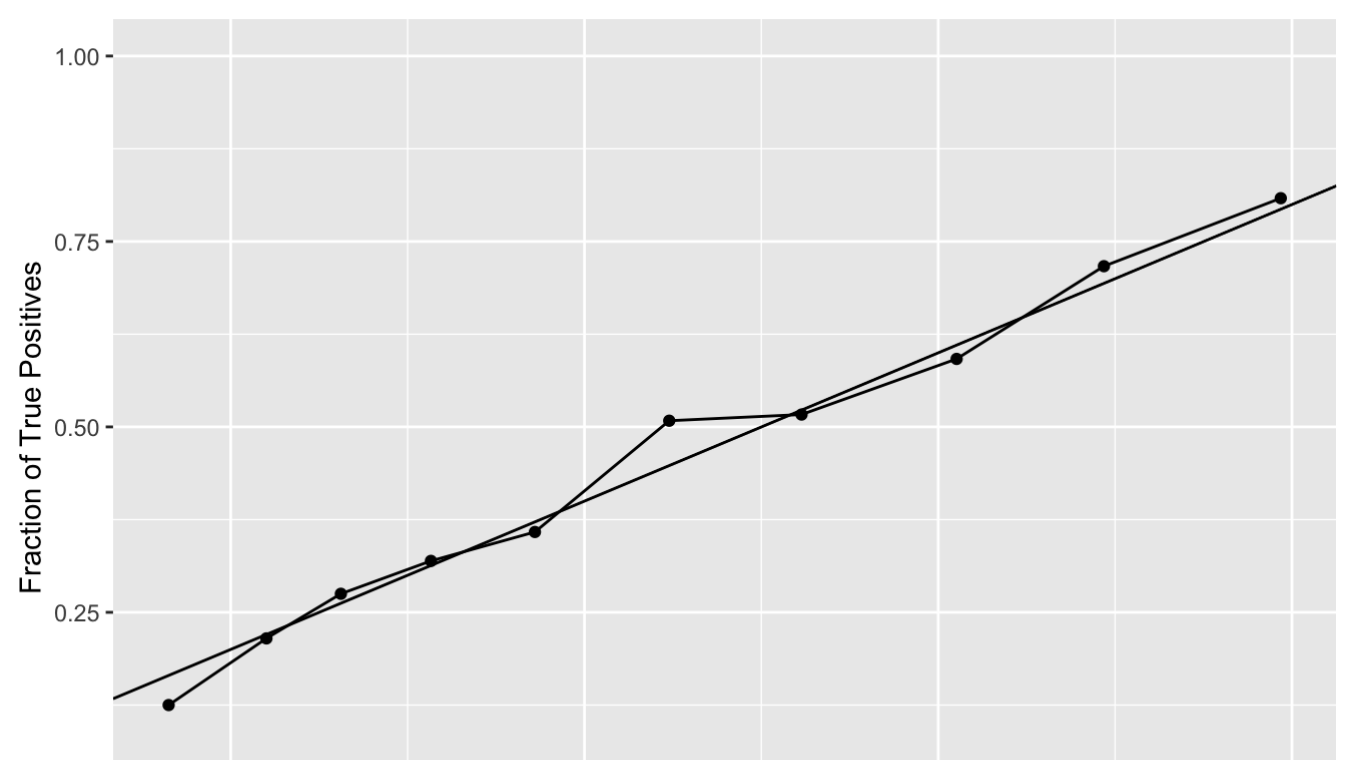
```
## [1] "0.45"
```

```
boost.table
```

```
##       accuracy precision    recall       MSE        FPR       FNR       Diff
## 0.3  0.6350000 0.5583127 0.8458647 0.2033125 0.53293413 0.1541353 0.37879879
## 0.35 0.6616667 0.5892351 0.7819549 0.2033125 0.43413174 0.2180451 0.21608662
## 0.4  0.6833333 0.6245902 0.7161654 0.2033125 0.34281437 0.2838346 0.05897978
## 0.45 0.6825000 0.6395564 0.6503759 0.2033125 0.29191617 0.3496241 0.05770789
## 0.5  0.6783333 0.6600877 0.5657895 0.2033125 0.23203593 0.4342105 0.20217460
## 0.55 0.6775000 0.6902887 0.4943609 0.2033125 0.17664671 0.5056391 0.32899239
## 0.6  0.6791667 0.7333333 0.4342105 0.2033125 0.12574850 0.5657895 0.44004097
## 0.65 0.6600000 0.7605042 0.3402256 0.2033125 0.08532934 0.6597744 0.57444509
## 0.7  0.6316667 0.7678571 0.2424812 0.2033125 0.05838323 0.7575188 0.69913556
```

Boosted Trees Calibration Plot

```
boosted.graph = boosteddat

risk_deciles_boosted = quantile(boosted.graph$test.dat.preds, probs = seq(from = 0, to = 1, by = 0.1))# generate
 risk deciles, producing sample quantiles corresponding to probabilities from 0 to 1 in increments of 0.1

boosted.graph$risk_decile_bin = as.numeric(cut(boosted.graph$test.dat.preds, breaks = risk_deciles_boosted, inclu
de.lowest = TRUE)) # create bins for risk deciles

some_results_boosted = boosted.graph %>% group_by(risk_decile_bin) %>% summarise(pred.prob = mean(test.dat.pred
s), pos = mean(recidivate)) #creating a new data table with the mean predicted probabilities and mean fraction of
true positives separated into the 10 decile bins created in the previous line


ggplot(some_results_boosted, aes(x = pred.prob, y = pos)) + geom_point() + geom_line() +
  ggtitle("Boosted Trees Calibration Plot") + ylab("Fraction of True Positives") +
  xlab("Mean Predicted Probabilities") + ylim(0,1) + geom_abline()
```



Boosted Trees Evaluation — Our Chosen Final Model

We selected the boosted trees model as our "best" model to deploy. In comparison to the two other predictive models we evaluated, the boosted trees model best optimizes both the MSE and the FNR. Since we are first and foremost optimizing MSE, boosted trees are our best model no matter the threshold. Within the boosted tree model, the threshold that optimized the FNR is the 0.45 (similar to both logistic and ridge regession). To reiterate, we aim to optimize for a low FNR, because there are high costs to society for accidentally missing and failing to predict an individual that will recidivate. One key thing to note about boosting is that it reweighs or modifies the data at each iteration based on how well its current models predict each data point. This is important to our dataset because the weights are not equal for predicted probabilities. Boosted trees trains each new model instance to emphasize data points that previous models predicted poorly or misclassified. It also incrementally adds individual trees (or models), and as such, is less likely to overfit the model. Our boosted model's calibration plot is also better "calibrated" to the other models' plots. On the calibration plot, closeness of the predicted probabilities to the identity line indicate how well the predicted probabilities are "calibrated."

Tables to Compare Models

```
min.threshold
```

```
## [1] "0.45"
```

```
log.table
```

```
##       accuracy precision    recall       MSE        FPR       FNR       Diff
## 0.3  0.5950000 0.5253863 0.8947368 0.2100388 0.64371257 0.1052632 0.53844942
## 0.35 0.6200000 0.5475000 0.8233083 0.2100388 0.54191617 0.1766917 0.36522444
## 0.4  0.6391667 0.5729013 0.7312030 0.2100388 0.43413174 0.2687970 0.16533474
## 0.45 0.6650000 0.6208178 0.6278195 0.2100388 0.30538922 0.3721805 0.06679123
## 0.5  0.6758333 0.6783042 0.5112782 0.2100388 0.19311377 0.4887218 0.29560803
## 0.55 0.6625000 0.7212544 0.3890977 0.2100388 0.11976048 0.6109023 0.49114178
## 0.6  0.6416667 0.7339450 0.3007519 0.2100388 0.08682635 0.6992481 0.61242177
## 0.65 0.6358333 0.7653631 0.2575188 0.2100388 0.06287425 0.7424812 0.67960695
## 0.7  0.6075000 0.7328244 0.1804511 0.2100388 0.05239521 0.8195489 0.76715366
```

```
r.min.threshold
```

```
## [1] "0.45"
```

```
ridge.table
```

```
##       accuracy precision    recall       MSE        FPR        FNR       Diff
## 0.3  0.5925000 0.5232432 0.9097744 0.2098589 0.66017964 0.09022556 0.56995408
## 0.35 0.6150000 0.5427873 0.8345865 0.2098589 0.55988024 0.16541353 0.39446671
## 0.4  0.6391667 0.5718433 0.7406015 0.2098589 0.44161677 0.25939850 0.18221827
## 0.45 0.6691667 0.6256983 0.6315789 0.2098589 0.30089820 0.36842105 0.06752285
## 0.5  0.6791667 0.6814815 0.5187970 0.2098589 0.19311377 0.48120301 0.28808924
## 0.55 0.6600000 0.7230216 0.3778195 0.2098589 0.11526946 0.62218045 0.50691099
## 0.6  0.6408333 0.7463415 0.2875940 0.2098589 0.07784431 0.71240602 0.63456170
## 0.65 0.6216667 0.7407407 0.2255639 0.2098589 0.06287425 0.77443609 0.71156184
## 0.7  0.5983333 0.7232143 0.1522556 0.2098589 0.04640719 0.84774436 0.80133718
```

```
b.min.threshold
```

```
## [1] "0.45"
```

```
boost.table
```

```
##       accuracy precision    recall       MSE        FPR       FNR       Diff
## 0.3  0.6350000 0.5583127 0.8458647 0.2033125 0.53293413 0.1541353 0.37879879
## 0.35 0.6616667 0.5892351 0.7819549 0.2033125 0.43413174 0.2180451 0.21608662
## 0.4  0.6833333 0.6245902 0.7161654 0.2033125 0.34281437 0.2838346 0.05897978
## 0.45 0.6825000 0.6395564 0.6503759 0.2033125 0.29191617 0.3496241 0.05770789
## 0.5  0.6783333 0.6600877 0.5657895 0.2033125 0.23203593 0.4342105 0.20217460
## 0.55 0.6775000 0.6902887 0.4943609 0.2033125 0.17664671 0.5056391 0.32899239
## 0.6  0.6791667 0.7333333 0.4342105 0.2033125 0.12574850 0.5657895 0.44004097
## 0.65 0.6600000 0.7605042 0.3402256 0.2033125 0.08532934 0.6597744 0.57444509
## 0.7  0.6316667 0.7678571 0.2424812 0.2033125 0.05838323 0.7575188 0.69913556
```

```
mods = rbind(log.table[min.threshold,], ridge.table[r.min.threshold,], boost.table[b.min.threshold,])
```

Build function for our model to use on pseudo code

```
test.dat.preds.fin <- predict(bt.models[[m]], newdata = pseudo.dat,
                              n.trees = final.ntrees, type = "response")

test.dat.preds.fin[1:100]
```

```
##   [1] 0.2842336 0.2959890 0.2173548 0.4968808 0.4155238 0.3977030 0.5753767
##   [8] 0.5372026 0.1780253 0.3972744 0.2071101 0.3949397 0.8469642 0.2732199
##  [15] 0.4024591 0.2936949 0.7521123 0.3648081 0.2640509 0.2757268 0.2223371
##  [22] 0.6457923 0.1915867 0.5265571 0.2847755 0.6709193 0.7064122 0.3712051
##  [29] 0.1791801 0.3206909 0.4047753 0.5048778 0.4925563 0.2160289 0.2701792
##  [36] 0.4945512 0.2954299 0.2300611 0.4030328 0.3083714 0.7285984 0.3284978
##  [43] 0.7040853 0.8179469 0.2074932 0.8414149 0.5185502 0.7588609 0.7770906
##  [50] 0.3696547 0.7440774 0.3636137 0.3379752 0.5805592 0.4756225 0.2658177
##  [57] 0.3452703 0.3877051 0.1949652 0.1785733 0.4903559 0.3162838 0.8364479
##  [64] 0.1912113 0.2088181 0.2567342 0.5140690 0.2088304 0.3265101 0.4314315
##  [71] 0.7624547 0.7294649 0.2515139 0.6310845 0.2289018 0.2515139 0.2993030
##  [78] 0.5952835 0.6039365 0.6066370 0.6379355 0.1346550 0.6623313 0.4439981
##  [85] 0.3736495 0.5815257 0.4934433 0.3424789 0.7183693 0.3511715 0.4518192
##  [92] 0.4293970 0.6303942 0.2327028 0.3002144 0.5286196 0.6406965 0.2999780
##  [99] 0.8130278 0.8661733
```