```
library(rusboost)
library(rpart)
library(ISLR)
library(pROC)
library(tidyverse)
```

Since I already saved the final result, I won't rerun all code chunks. However, I still put them here for reviewing purpose and I will select a few code chunks to run so they can display some important findings.

Loading my data.

```
load("C:/Users/cjy2001/OneDrive - Middlebury College/Middlebury Academic/Fall 2021/Project/Final
_Data.RData")
data <- read_csv("uscecchini28.csv")
data$misstate <- factor(data$misstate)
```

Here is my original code to create train and test data. (Could use function assign)

```
split_train_test <- function (year, data) {
  train <<- data %>%
    filter(fyear <= year - 2)
  test <<- data %>%
    filter(fyear == year)
}

split_train_test(2003, data)
train_3 <- train
test_3 <- test
split_train_test(2004, data)
train_4 <- train
test_4 <- test
split_train_test(2005, data)
train_5 <- train
test_5 <- test
split_train_test(2006, data)
train_6 <- train
test_6 <- test
split_train_test(2007, data)
train_7 <- train
test_7 <- test
split_train_test(2008, data)
train_8 <- train
test_8 <- test
```

At first, I want to make sure the "iters" tuning parameter represents the number of trees in a model. I select year 2003 to be my train data, and then I compute some models with the number of trees ranging from 200 to 5000. I find that the number of true positives increases from 200 to 3000, and gradually stabilizes from 3000 to 5000. That validates the authors' statement that the model's performance becomes stable as the number of trees reaches 3000. Therefore, I set the number of trees for all models after 2003 to be equal to 3000.

```
# These are all for year 2003.
set.seed(5)
time_5000 <- system.time(model_5000 <- rusb(formula = misstate ~ act + ap + at + ceq + che + cog
s + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train_3,
              boot = FALSE,
              iters = 5000,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train_3$misstate == 0)
)

time_3000 <- system.time(model_3000 <- rusb(formula = misstate ~ act + ap + at + ceq + che + cog
s + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train,
              boot = FALSE,
              iters = 3000,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train$misstate == 0)
)

time_1500 <- system.time(model_1500 <- rusb(formula = misstate ~ act + ap + at + ceq + che + cog
s + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train,
              boot = FALSE,
              iters = 1500,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train$misstate == 0)
)

time_500 <- system.time(model_500 <- rusb(formula = misstate ~ act + ap + at + ceq + che + cogs
 + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train,
              boot = FALSE,
              iters = 500,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train$misstate == 0)
)
```

```
time_200 <- system.time(model_200 <- rusb(formula = misstate ~ act + ap + at + ceq + che + cogs
 + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train,
              boot = FALSE,
              iters = 200,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train$misstate == 0)
)
```

```
# Make predictions so that I can construct the confusion matrix.
preds_100 <- predict.rusb(year3_model_100, data.frame(test_3))
preds_200 <- predict.rusb(model_200, data.frame(test_3))
preds_500 <- predict.rusb(model_500, data.frame(test_3))
preds_1500 <- predict.rusb(model_1500, data.frame(test_3))
preds_3000 <- predict.rusb(model_3000, data.frame(test_3))
```

Next, I keep these tuning parameters (minsplit, sampleFraction, number of trees) the same and obtain models from year 2003 to 2008.

```
set.seed(5)
#.Random.seed  Check seed is the same

year3_model_3000 <- model_3000

year4_time_3000 <- system.time(year4_model_3000 <- rusb(formula = misstate ~ act + ap + at + ceq
+ che + cogs + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train_4,
              boot = FALSE,
              iters = 3000,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train_4$misstate == 0)
)

year5_time_3000 <- system.time(year5_model_3000 <- rusb(formula = misstate ~ act + ap + at + ceq
+ che + cogs + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train_5,
              boot = FALSE,
              iters = 3000,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train_5$misstate == 0)
)

year6_time_3000 <- system.time(year6_model_3000 <- rusb(formula = misstate ~ act + ap + at + ceq
+ che + cogs + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train_6,
              boot = FALSE,
              iters = 3000,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train_6$misstate == 0)
)

year7_time_3000 <- system.time(year7_model_3000 <- rusb(formula = misstate ~ act + ap + at + ceq
+ che + cogs + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train_7,
              boot = FALSE,
              iters = 3000,
              control = rpart.control(minsplit = 5),
```

```
              sampleFraction = 0.5,
              idx = train_7$misstate == 0)
)

year8_time_3000 <- system.time(year8_model_3000 <- rusb(formula = misstate ~ act + ap + at + ceq
+ che + cogs + csho + dlc + dltis + dltt + dp
              + ib + invt + ivao + ivst + lct + lt + ni + ppegt + pstk + + re + rect + sale + ss
tk + txp + txt + xint +
                prcc_f,
              data = train_8,
              boot = FALSE,
              iters = 3000,
              control = rpart.control(minsplit = 5),
              sampleFraction = 0.5,
              idx = train_8$misstate == 0)
)
```

Since I use the system time to time each model, I can also put the time used by each model here.

model with 100 trees: 13 mins model with 200 trees: 30 mins model with 500 trees: 1 hour model with 1500 trees: 2 hours model with 3000 trees: 5 hours model with 5000 trees: 8 hours

year3: 5 hours year4: 4 hours year5: 4 hours year6: 45 mins???
year7: 8 hours??? year8: 8 hours???

It is worth mentioning that we still don't know what causes that large variation in time used by models trained by different years. For example, the time used by year 2006 only takes 45 minutes; year 2007 and 2008 both take a much longer time, but their performances are the worst. Though the number of observations increases over years, it still cannot explain why we first see a stable trend from year 2003 to 2005 and then it changes so abruptly (because the increase of observations is similar between years.)

Finally, I record the performance of each model.

```
yearly_test_set <- list(test_3,test_4,test_5,test_6, test_7, test_8)
yearly_preds <- list(year3_preds_3000,year4_preds_3000,year5_preds_3000,year6_preds_3000,year7_p
reds_3000,year8_preds_3000)

yearly_sensitivity <- NULL
yearly_precision <- NULL
yearly_auc <- NULL
yearly_NDCG <- NULL

for (i in 1:6) {
  print(yearly_preds[[i]]$confusion)

  roc_obj <- roc(predictor = yearly_preds[[i]]$prob[,2],
                 response = yearly_test_set[[i]]$misstate)
  yearly_auc[i] = auc(roc_obj)
  #plot(roc_obj, legacy.axes = TRUE)

  yearly_sensitivity[i] <-
    yearly_preds[[i]]$confusion[2,2]/sum(yearly_preds[[i]]$confusion[,2])
  yearly_precision[i] <- yearly_preds[[i]]$confusion[2,2]/sum(yearly_preds[[i]]$confusion[2,])

  k <- floor(0.01*nrow(yearly_test_set[[i]]))
  RelevanceLevel <- as.numeric(as.character(sort(yearly_test_set[[i]]$misstate, decreasing = TRU
E)))
  testpred <- cbind(yearly_test_set[[i]], data.frame(yearly_preds[[i]]$prob[,1]))
  names(testpred)[52] <- "preds"
  testpred <- testpred %>% arrange(preds)
  PredictedRank <- as.numeric(as.character(testpred$misstate))
  yearly_NDCG[i] <- NDCG2(k, Relevance = RelevanceLevel, Predicted = PredictedRank)
}
```

```
yearly_auc
```

```
## [1] 0.7755731 0.7749642 0.7575131 0.7467930 0.7000114 0.5551344
```

```
yearly_NDCG
```

```
## [1] 0.12985427 0.03978712 0.04705198 0.00000000 0.01965068 0.00000000
```

```
yearly_sensitivity
```

```
## [1] 0.21739130 0.18965517 0.06666667 0.66666667 0.16666667 0.00000000
```

```
yearly_precision
```

```
## [1] 0.05617978 0.05069124 0.02521008 0.01237345 0.01228501 0.00000000
```

```
mean(yearly_auc)
```

```
## [1] 0.7183315
```

```
mean(yearly_NDCG)
```

```
## [1] 0.03939067
```

```
mean(yearly_sensitivity)
```

```
## [1] 0.2178411
```

```
mean(yearly_precision)
```

```
## [1] 0.02612326
```

Referring to the paper：“using the same 28 raw financial data items, the performance metrics averaged over the test period 2003–2008 is”
AUC = 0.725
NDCG = 0.049
Sensitivity = 4.88%
Precision = 4.48%

Here are some intuitive findings:

1. Our models’ averaged value of AUC is similar to paper’s, except the poor performance of year 2008.
2. Except for year 2003, 2004, and 2005, our models generally don’t perform well on the NDCG@k (mailto:NDCG@k) metric.
3. Our models have a much higher averaged sensitivity. Though it is very different from the paper, a higher sensitivity indicates that our models are doing a better job.
4. Our models have a slightly lower score of precision.

The general trend is that the overall accuracy rate declines over the years, but it still cannot explain why the 2008 model is so unusual, which requires further investigation.

Some additional notes:
1. Our model have not dealt with serial fraud mentioned by the paper yet.
2. As the global environment contains more data, it is very likely the RStudio will incur an error such as “cannot allocate vector of size 1.3 GB”. We can resolve this by cleaning the global environment and only load the data we want.