Fraud Detection prediction using XGBoosting

Haijie CAO

10/05/2017

Reading Data

```
card<-read.csv("creditcard.csv")
```

```
#card$Class<-as.factor(card$Class)
table(card$Class)
```

```
##
##      0      1
## 284315    492
```

```
taux<-nrow(card[card$Class==1,])/nrow(card)
taux
```

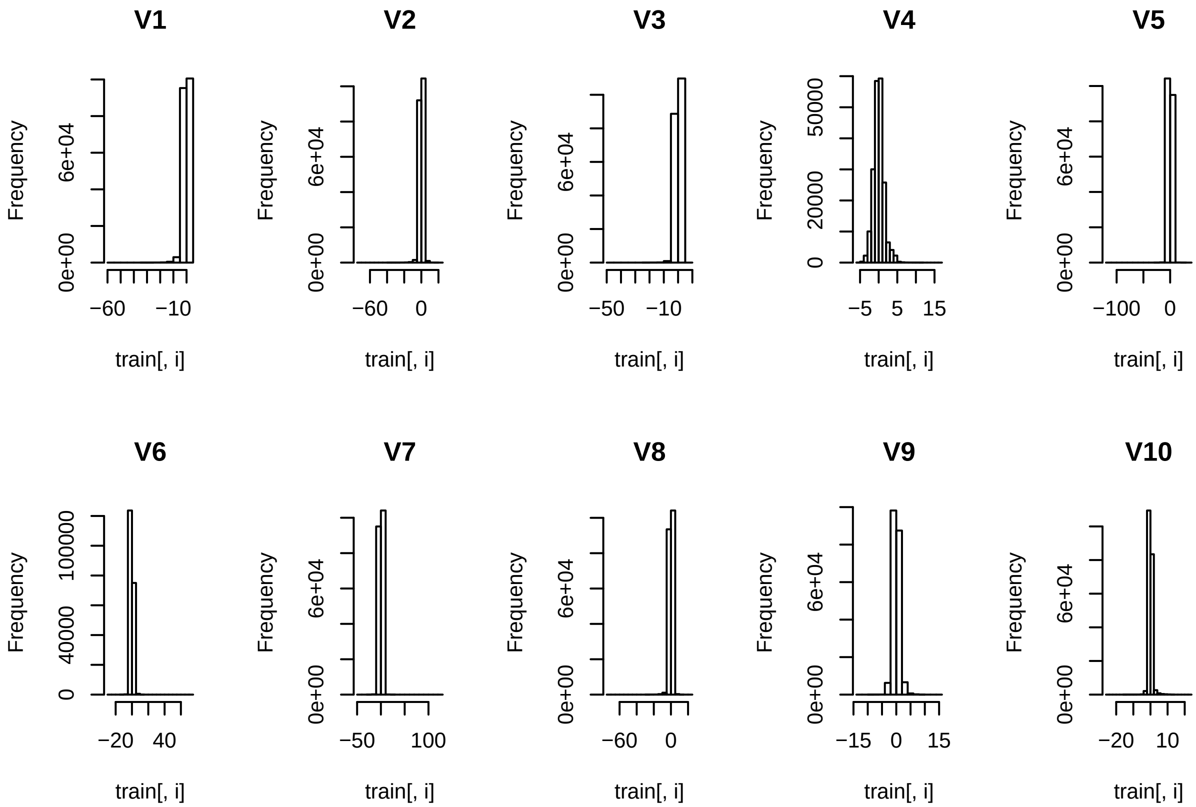
```
## [1] 0.001727486
```

```
## seperate the data into training set and testing set
cardsub<-card[,-1]
set.seed(5)
ind<-sample(2,nrow(cardsub),replace = T,prob=c(0.7,0.3))
train<-cardsub[ind==1,]
test<-cardsub[ind==2,]

######=================================================================================
feature.names=names(train)

for (f in feature.names) {
  if (class(train[[f]])=="factor") {
    levels <- unique(c(train[[f]]))
    train[[f]] <- factor(train[[f]],
                   labels=make.names(levels))
  }
}

for (f in feature.names) {
  if (class(test[[f]])=="factor") {
    levels <- unique(c(test[[f]]))
    test[[f]] <- factor(test[[f]],
                   labels=make.names(levels))
  }
}
#####===================================================================================
```

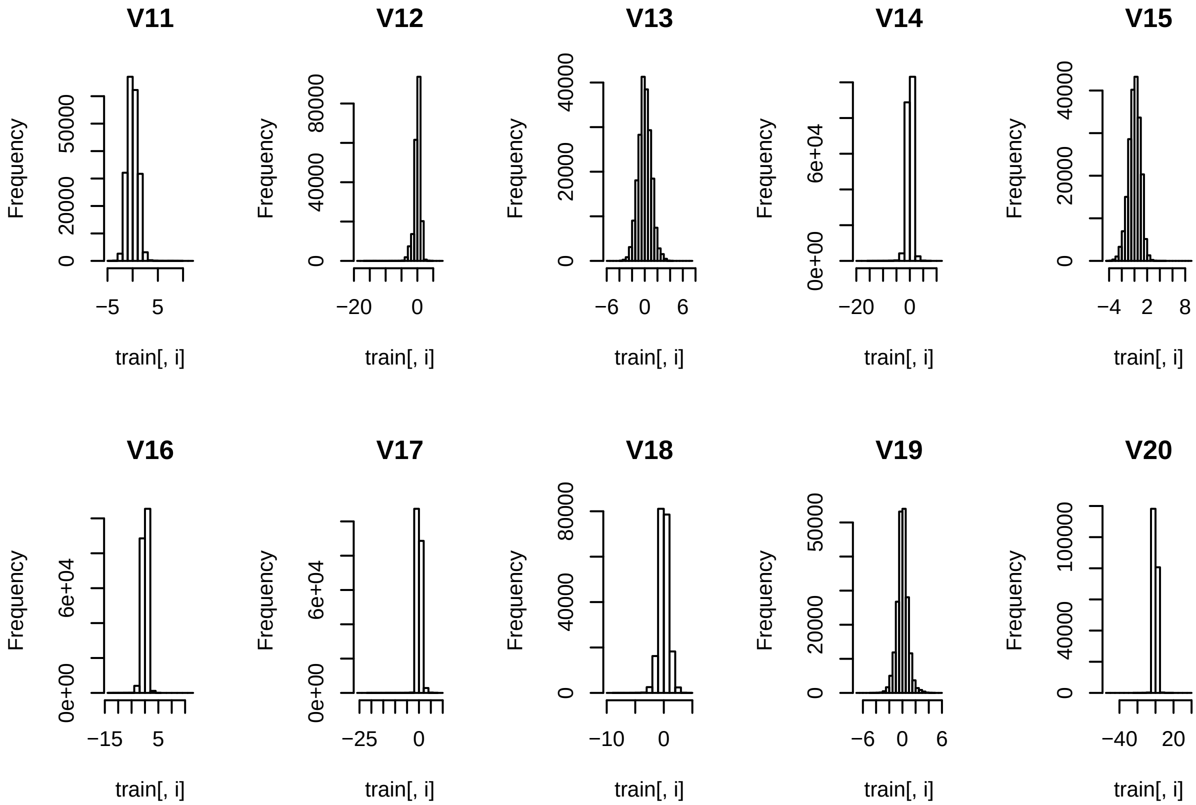
Dada Exploratory

```
par(mfrow=c(2,5))
for(i in 1:10){
```

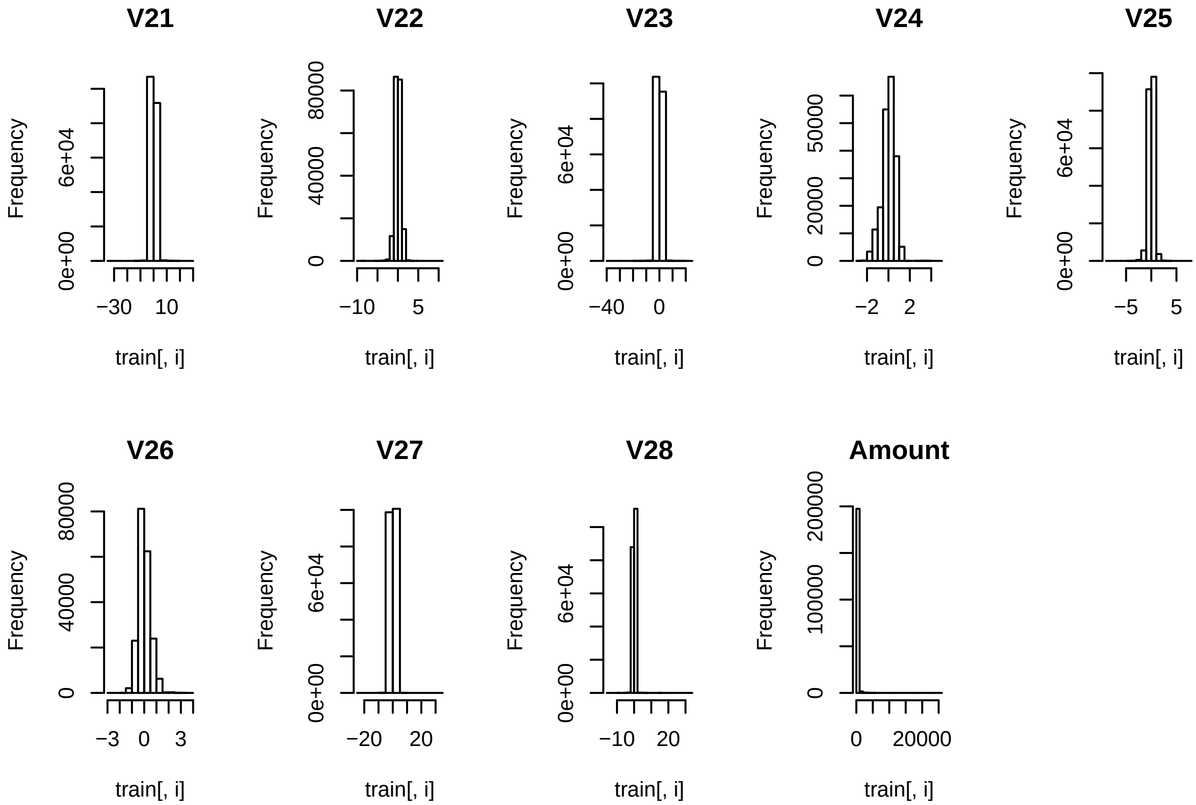
```r
        hist(train[,i], main=names(train)[i])
}
```




```r
par(mfrow=c(2,5))
for(i in 11:20){
        hist(train[,i], main=names(train)[i])
}
```




```r
par(mfrow=c(2,5))
for(i in 21:29){
        hist(train[,i], main=names(train)[i])
}
```



We can see that the variables are very skewed.

XGBoost

```
library(ggplot2,quietly = T)
library(plyr)
library(dplyr,quietly = T)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr,quietly = T)
library(readr,quietly = T)
library(xgboost,quietly = T)
```

```
##
```

```
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(caret,quietly = T)
library(pROC,quietly = T)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
# xgboost fitting with arbitrary parameters
xgb_params = list(
        objective = "binary:logistic",          # binary classification
        eta = 0.01,                             # learning rate
        max.depth = 3,                          # max tree depth
        eval_metric = "auc"                     # evaluation/loss metric
)

# fit the model with the arbitrary parameters specified above
xgb = xgboost(data = as.matrix(train[,1:29]),
                label = train$Class,
                params = xgb_params,
                nrounds = 100,                  # max number of trees to build
                verbose = TRUE,
                print_every_n = 20,
                early_stop_round = 10          # stop if no improvement within 10 trees
)
```

```
## [1]  train-auc:0.909587
## [21] train-auc:0.912663
## [41] train-auc:0.912664
## [61] train-auc:0.912673
## [81] train-auc:0.912677
## [100]    train-auc:0.912698
```

```r
# cross-validate xgboost to get the accurate measure of error
xgb_cv = xgb.cv(params = xgb_params,
                  data = as.matrix(train[,1:29]),
                  label = train$Class,
                  nrounds = 100,
                  nfold = 5,                    # number of folds in K-fold
                  prediction = TRUE,            # return the prediction using the final model
                  showsd = TRUE,                # standard deviation of loss across folds
                  stratified = TRUE,            # sample is unbalanced; use stratified sampling
                  verbose = TRUE,
                  print_every_n = 20,
                  early_stop_round = 10
)
```

```
## [1]  train-auc:0.909600+0.003094 test-auc:0.904845+0.007888
```

```
## [21] train-auc:0.911566+0.002337 test-auc:0.904872+0.007886
## [41] train-auc:0.912704+0.003098 test-auc:0.909549+0.006195
## [61] train-auc:0.913846+0.003201 test-auc:0.911158+0.008934
## [81] train-auc:0.913859+0.003198 test-auc:0.911169+0.008936
## [100]    train-auc:0.914230+0.002974 test-auc:0.911171+0.008934
```

```r
# set up the cross-validated hyper-parameter search
xgb_grid = expand.grid(
        nrounds = 100,
        eta = c(0.1),
        max_depth = c(2,6),
        gamma = 1,
        colsample_bytree=1,
        min_child_weight=10,
        subsample=1)


# pack the training control parameters
xgb_trcontrol = trainControl(
        method = "cv",
        number = 5,
        verboseIter = TRUE,
        returnData = FALSE,
        returnResamp = "all",                              # save losses across all models
        classProbs = TRUE,                                 # set to TRUE for AUC to be computed
        summaryFunction = twoClassSummary,
        allowParallel = TRUE
)

# train the model for each parameter combination in the grid,
#  using CV to evaluate
train$Class[train$Class==0]<-"No"
train$Class[train$Class==1]<-"Yes"

xgb_train = train(
        x = as.matrix(train[,1:29]),
        y = as.factor(train$Class),
        trControl = xgb_trcontrol,
        metric="ROC",
        tuneGrid = xgb_grid,
        method = "xgbTree"
)
```

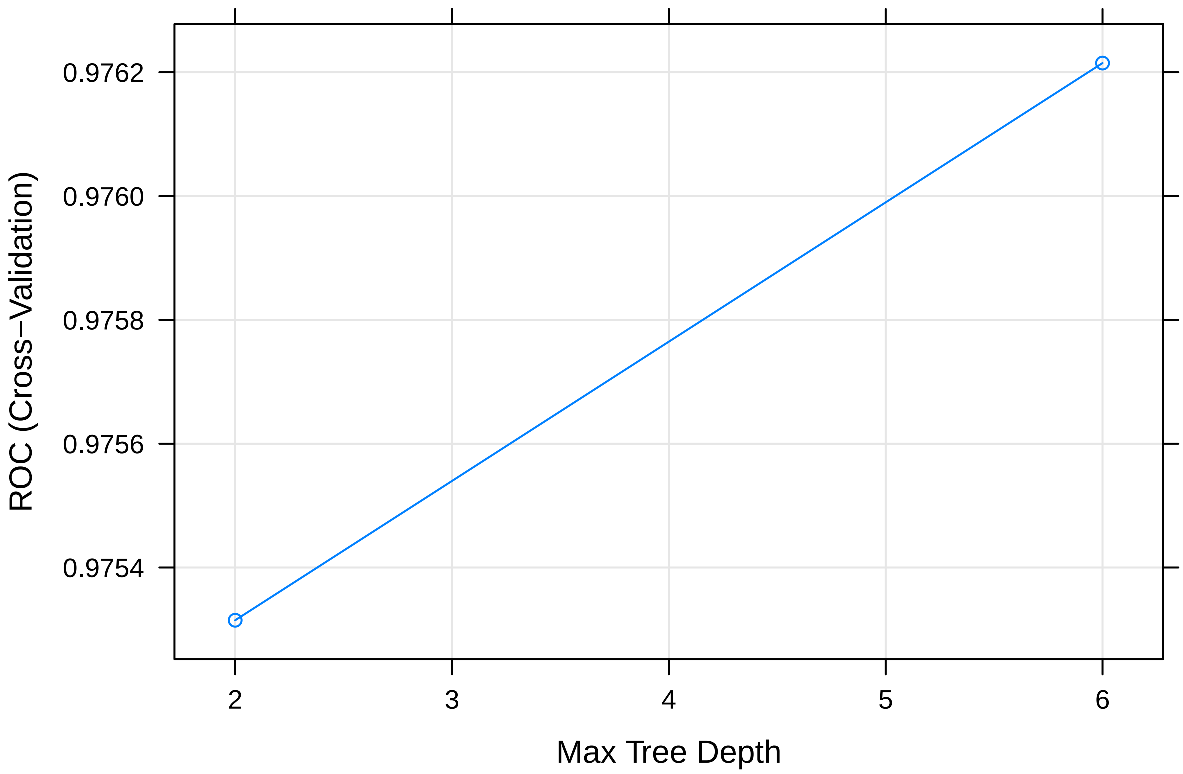
```
## + Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## - Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## + Fold1: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## - Fold1: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## + Fold2: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## - Fold2: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## + Fold2: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## - Fold2: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## + Fold3: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## - Fold3: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## + Fold3: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
## - Fold3: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nround
```

```
## + Fold4: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## - Fold4: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## + Fold4: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## - Fold4: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## + Fold5: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## - Fold5: eta=0.1, max_depth=2, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## + Fold5: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## - Fold5: eta=0.1, max_depth=6, gamma=1, colsample_bytree=1, min_child_weight=10, subsample=1, nrounds
## Aggregating results
## Selecting tuning parameters
## Fitting nrounds = 100, max_depth = 6, eta = 0.1, gamma = 1, colsample_bytree = 1, min_child_weight =
```

```
xgb_train$bestTune
```

```
##   nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 2     100         6 0.1     1                1               10         1
```

```
plot(xgb_train)
```




```
res <- xgb_train$results
res
```

```
##   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.1         2     1                1               10         1     100
## 2 0.1         6     1                1               10         1     100
##         ROC      Sens      Spec      ROCSD       SensSD     SpecSD
## 1 0.9753148 0.9998042 0.7430303 0.01280935 7.615431e-05 0.09711998
## 2 0.9762149 0.9998644 0.7676457 0.01154988 2.862677e-05 0.08889640
```

```
### xgboostModel Predictions and Performance
# Make predictions using the test data set
xgb.pred <- predict(xgb_train,test)
```

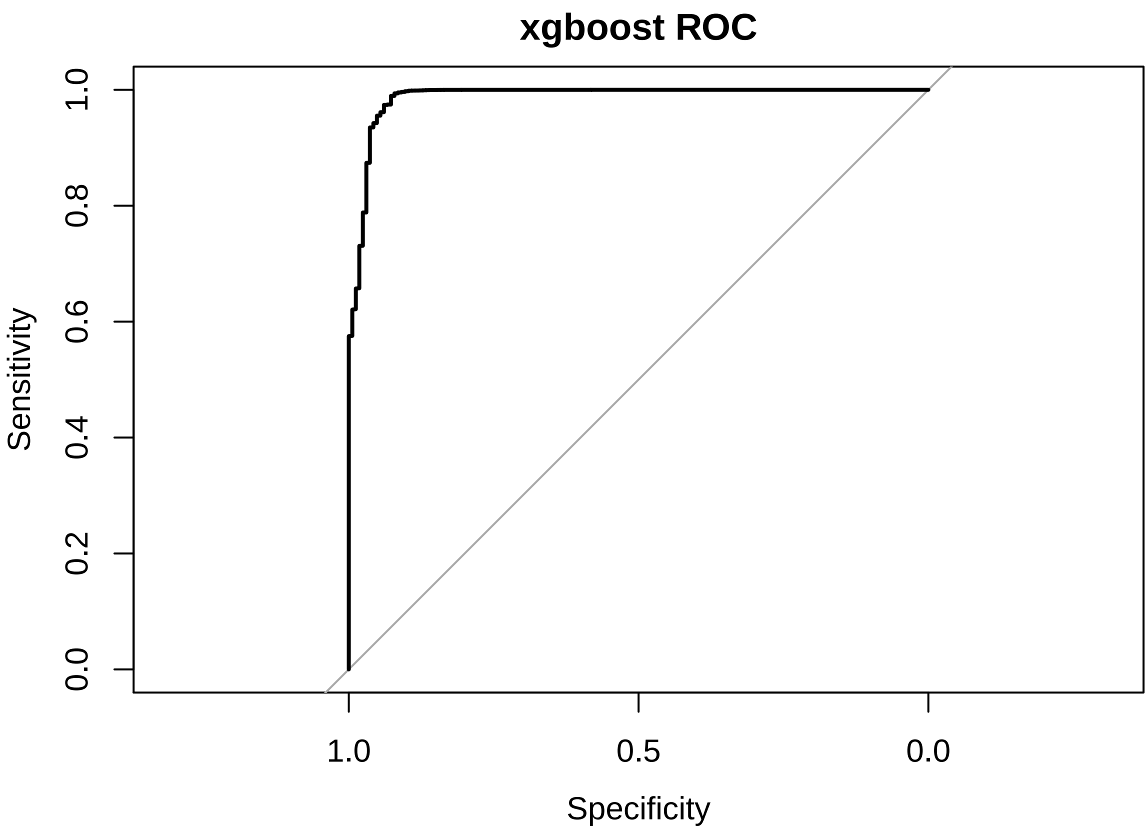
```r
test$Class[test$Class==0]<-"No"
test$Class[test$Class==1]<-"Yes"
test$Class<-as.factor(test$Class)

#Look at the confusion matrix
confusionMatrix(xgb.pred,test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  85164    29
##        Yes     6   136
##
##                Accuracy : 0.9996
##                  95% CI : (0.9994, 0.9997)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8858
##  Mcnemar's Test P-Value : 0.0002003
##
##             Sensitivity : 0.9999
##             Specificity : 0.8242
##          Pos Pred Value : 0.9997
##          Neg Pred Value : 0.9577
##              Prevalence : 0.9981
##          Detection Rate : 0.9980
##    Detection Prevalence : 0.9983
##       Balanced Accuracy : 0.9121
##
##        'Positive' Class : No
##
```

```r
#Draw the ROC curve
xgb.probs <- predict(xgb_train,test,type="prob")

#head(xgb.probs)
xgb.ROC <- roc(predictor=xgb.probs$No,
               response=test$Class,
               levels=rev(levels(test$Class)))
xgb.ROC$auc
```

```
## Area under the curve: 0.9876
```

```r
plot(xgb.ROC,main="xgboost ROC")
```




```r
# Plot the propability of poor segmentation
histogram(~xgb.probs$No|test$Class,xlab="Probability of Poor Segmentation")
```