A Data Science Lab Project

Zhiyu Quan[*]

26 April 2018

Abstract

The fourth actuarial pricing game is a Kaggle-style competition in insurance pricing. The dataset from historical home insurance in France and the player need to provide the predictive modeling on the claim on the policy. In this competition, we compared the traditional Tweedie GLM with Tweedie LASSO, Tweedie Group LASSO, Tweedie Elastic Net, Tweedie Grouped Elastic Net and Tweedie Tree-Boosting. We examine the prediction accuracy and variable selection through different methods. Finally, provide the best Tweedie-based Model. This work served for the Data Science in Action course offered by the University of Connecticut.

Contents

1 Introduction

The fourth actuarial pricing game is a Kaggle-style competition in insurance pricing. In this iteration of the game, the game organizers will provide the players with real historical home insurance data and ask the players to give the game organizers their most profitable model for offering insurance premium prices according to the data.

The game organizers will then use previously unseen data and submitted pricing models to simulate a competitive insurance market where players will compete for customers that choose the cheapest premium offered to them. In each market, the player that makes the most money will win.

[*] zhiyu.quan@uconn.edu; Ph.D. Candidate at Department of Mathematics, University of Connecticut.

2 Actuarial Pricing

The insurance premium is the amount of money the insured is charged by the insurer for coverage set forth in the insurance policy. If the insured experiences a loss covered by the policy, they can submit a claim to the insurer. The profit of an insurer is the difference between their earned premiums and their incurred losses. Consider a person facing a total possible loss of S during a period of one year. The actuarial pure premium, or fair premium, is the expected value of S, $E[S]$.

3 Tweedie-Compound Poisson Distribution

In insurance premium prediction problems, the total claim amount has a probability mass at zero, and a small portion of the total claim has positive continuous values. One traditional approach in actuarial science predictive modeling on such special data is using Tweedie compound Poisson models.

Let N be the number of claims and follow the Poisson distribution with parameter λ and $Z_k, k = 0 \dots N$ be the claim sizes and follow the $Gamma(\alpha, \gamma)$. Hence the total claim S is the Poisson sum of independent Gamma random variables.

$$P(S = 0) = P(N = 0) = exp(-\lambda)$$

$$P(S \leqslant y) = exp(-\lambda) + \sum_{n=1}^{\infty} P(N = n)P(S_n \leqslant y), \quad y \geqslant 0$$

For density function:

$$\begin{aligned} f_S(s|\lambda, \alpha, \gamma) &= P(N = 0)d_0(s) + \sum_{n=1}^{\infty} P(N = n) f_{S|N=n}(s) \\ &= exp(-\lambda)d_0(s) + \sum_{n=1}^{\infty} \frac{\lambda^n e^{-\lambda}}{n!} \frac{s^{n\alpha-1} e^{\frac{-z}{\gamma}}}{\gamma^{n\alpha}\Gamma(n\alpha)} \end{aligned}$$

here d_0 is the Dirac delta function at 0. The moments of total claim S is

$$ES_N = \lambda \frac{\alpha}{\gamma} \quad and \quad VarS_N = \frac{\lambda\alpha}{\gamma^2}(1 + \alpha)$$

Define three parameters μ, ϕ, p by

$$\lambda = \frac{\mu^{2-p}}{\phi(2-p)}, \quad \alpha = \frac{2-p}{p-1}, \quad and \quad \gamma = \phi(p-1)\mu^{p-1}$$

The tweedie model can be rewritten to, more general case, exponential dispersion models form:

$$f_S(s|\mu,\phi,p) = exp[\frac{1}{\phi}(s\frac{\mu^{1-p}}{1-p} - \frac{\mu^{2-p}}{2-p}) + \log a(s,\phi,p) \quad where$$

$$a(s,\phi,p) = \begin{cases} \frac{1}{s}\sum_{t=1}^{\infty}\frac{s^{t\alpha}}{(p-1)^{t\alpha}\phi^{t(1+\alpha)}(2-p)^t t!\Gamma(t\alpha)} & \text{for z>0} \\ 1 & \text{for z=0} \end{cases} \tag{1}$$

After the rewrite, the Tweedie distribution by the exponential dispersion models the moments simply rewritten by

$$ES_N = \mu \quad and \quad VarS_N = \phi\mu^p \quad where \quad 1 < p < 2$$

For the Tweedie GLM, we use a logarithmic link. We apply greedy search for the variable selection. Initially wrote binary add function to loop all the possibility of the combination of the variables. It obviously took too much time to find the result. So later improved the algorithm that using the combination function which can split the task into parallel computing. For our dataset, there are 25 explanatory variables. The binary add function gives $2^{25} - 1$ combinations. This is equal to $\binom{25}{1} + \binom{25}{2} + \cdots + \binom{25}{25}$. Latter can be split into parallel computing.

4 Tweedie Grouped Elastic Net

For n independent insurance contracts. Each contract i has $\mathbf{x}_i$ and claim y_i. The negative log-likelihood can be written as

$$l(\beta_0,\beta|y_i,\mathbf{x}_i) = \sum_{i=1}^{n}(\frac{y_i e^{(1-p)(\beta_0+\beta^T\mathbf{x}_i)}}{1-p} + \frac{e^{(2-p)(\beta_0+\beta^T\mathbf{x}_i)}}{2-p})$$

Qian, Yang, and Zou (2016) proposed a new blockwise coordinate descent algorithm which extends from the iteratively reweighted least square (IRLS) strategy. The blockwise majorization descent method is embedded into the IRLS strategy to solve the penalized weighted least square. Let the p-dimensional coefficient vector β is grouped by g blocks, then the objective function with grouped elastic net penalties is

$$(\hat{\beta_0},\hat{\beta}) = \underset{\beta_0,\beta}{\text{argmin}} l(\beta_0,\beta) + \lambda\sum_{j=1}^{g}(\tau w_j||\beta_j||_2 + \frac{1}{2}(1-\tau)||\beta_j||_2^2)$$

here τ and λ regulization tuning parameters and w_j are the positive weights for the grouped lasso penalties.

5 Gradient Tree-Boosted Tweedie

For n independent insurance contracts. Each contract i has $\mathbf{x}_i$ and claim y_i.

$$log(\mu_i) = log(E(Y_i|\mathbf{x}_i)) = F(\mathbf{x}_i)$$

Here the $F()$ is the boosted Tweedie model. Given the data, the log-likelyhood function can be specified as

$$\begin{aligned} l(F, \phi, p|y_i, \mathbf{x}_i) &= \sum_{i=1}^{n} log f_Y(y_i|\mu_i, \phi, p) \\ &= \sum_{i=1}^{n} \frac{1}{\phi}(y_i \frac{\mu_i^{1-p}}{1-p} - \frac{\mu_i^{2-p}}{2-p}) + \log a(y_i, \phi, p) \end{aligned}$$

Given ϕ and p, the minimizer function F* can be found by

$$\begin{aligned} F^*(\mathbf{x}) &= \underset{F \in \mathcal{F}}{\operatorname{argmin}} \{-l(F, \phi, p|y_i, \mathbf{x}_i)\} \\ &= \underset{F \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \{-y_i \frac{\exp[(1-p)F(\mathbf{x}_i)]}{1-p} + \frac{\exp[(2-p)F(\mathbf{x}_i)]}{2-p}\} \end{aligned}$$

Define the objective function for the tree-based boosting,

$$\Phi(y_i, F(\mathbf{x}_i)|p) = -y_i \frac{\exp[(1-p)F(\mathbf{x}_i)]}{1-p} + \frac{\exp[(2-p)F(\mathbf{x}_i)]}{2-p}$$

Following steps are common gradient boosting algorithm. For further detail refers to Yang, Qian, and Zou (2017).

6 Dataset and Explanatory Analysis

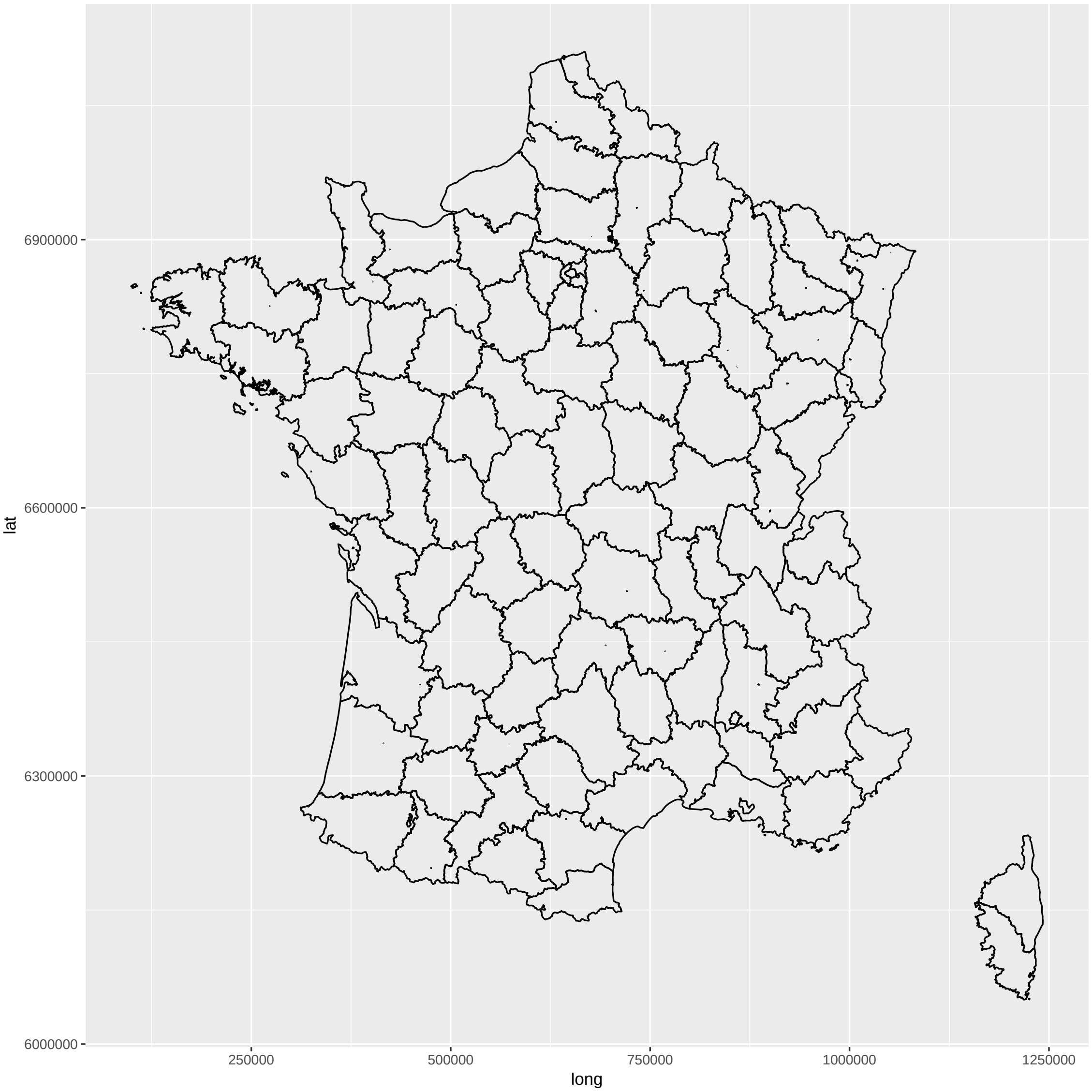
The dataset contains approximately 2 million real historical contracts across 3 consecutive years. These are contracts concerning home insurance policies from this decade in France.

For the purposes of the game, the player will each receive a separate and disjoint set of contracts from the first two years of the dataset. This will be training data during the game.

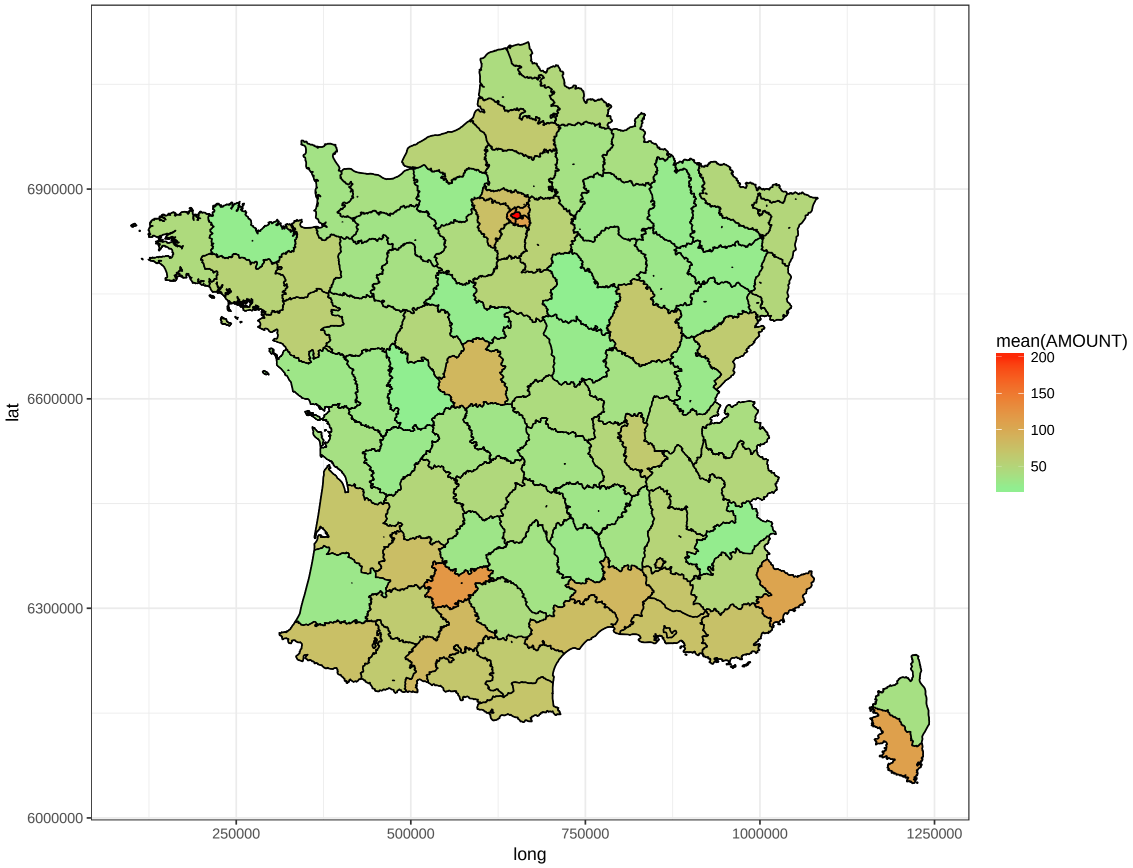
In this iteration of the game, there are more than 200 registered players. However, each market, the game organizers had simulated, will contain a random selection of 20 players. This is so that each player obtains a sizeable part of the dataset.

The remaining contracts from the third year (about 716, 000) will be used in the market simulation and will not be provided to players in advance (like a real market). However, players are provided with a fake, randomly generated, dataset (100 contracts) for which players should offer premiums. The purpose of this fake dataset is to ensure players' model is reproducible.

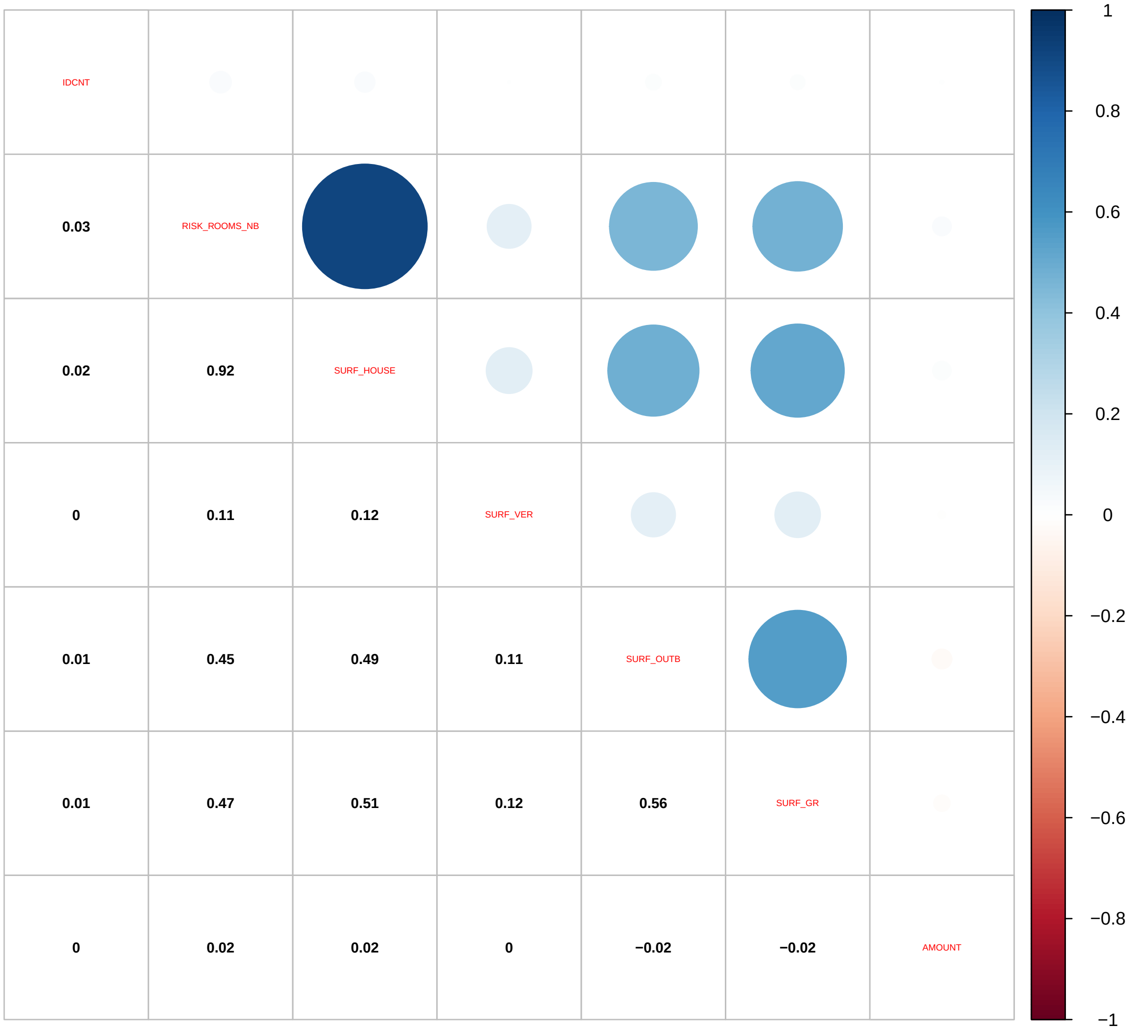
```r
# setwd('/Users/ZhiyuQuan/Downloads/DataSclass/Pricing
# Game/PricingGame/data') setwd('D:/GithubResearch/PricingGame/data')

giniTest <- function(y, py) {
    # Calculate a gini score Parameters: y: The real data from observation
    # py: The predicted value from model Return: gini score for this
    # prediction Algorithm: 1. Use a uniform random vector to break ties
    # from predicted values 2. Use predicted values to rank observed y 3.
    # Calculated gini score used ranked observed y.
    data <- as.data.frame(cbind(y, py))
    set.seed(1)
    n <- length(y)
    data$rand.unif <- runif(n)

    sorted.y <- data[order(data$py, data$rand.unif), ][, 1]
    i <- seq(n)
    giniIndex <- 1 - 2/(n - 1) * (n - sum(sorted.y * i)/sum(sorted.y))
    return(giniIndex)
}

set.seed(1)

load("train.RData")
load("test.RData")
load("spatial.RData")

CombinedData = rbind.data.frame(train, test)

library(dplyr)
MeanAmount = CombinedData %>% group_by(DEPT) %>% summarise(mean(AMOUNT))
DEPARTMENTS.spatial = dplyr::inner_join(MeanAmount, DEPARTMENTS.df, by = "DEPT")

library(ggplot2)
ggplot(data = DEPARTMENTS.df, aes(x = long, y = lat, group = group)) +
    geom_path()
```



```
ggplot(data = DEPARTMENTS.spatial, mapping = aes(x = long, y = lat, group = group)) +
    coord_equal() + geom_polygon(color = "black", fill = "white") + geom_polygon(data = DEPART
    aes(fill = `mean(AMOUNT)`), color = "black") + theme_bw() + scale_fill_gradient(low = "lig
    high = "red")
```



```r
NumericVariables <- sapply(CombinedData, is.numeric)
correlation = cor(CombinedData[, NumericVariables])

library(corrplot)
corrplot.mixed(correlation, lower.col = "black", number.cex = 0.7, tl.cex = 0.4)
```

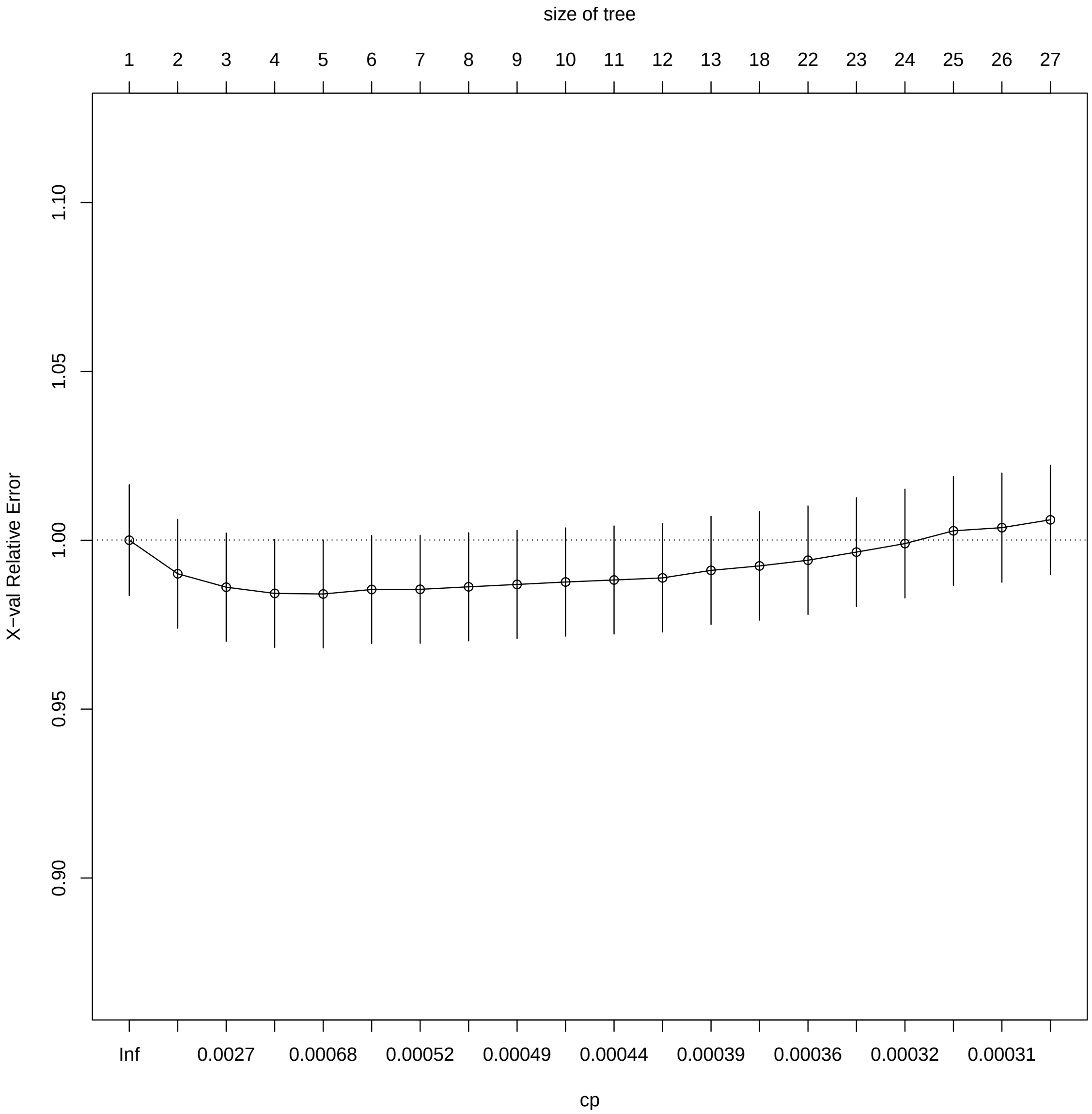


```
CombinedData$DEPT = as.factor(CombinedData$DEPT)
train$DEPT = as.factor(train$DEPT)
test$DEPT = as.factor(test$DEPT)
```

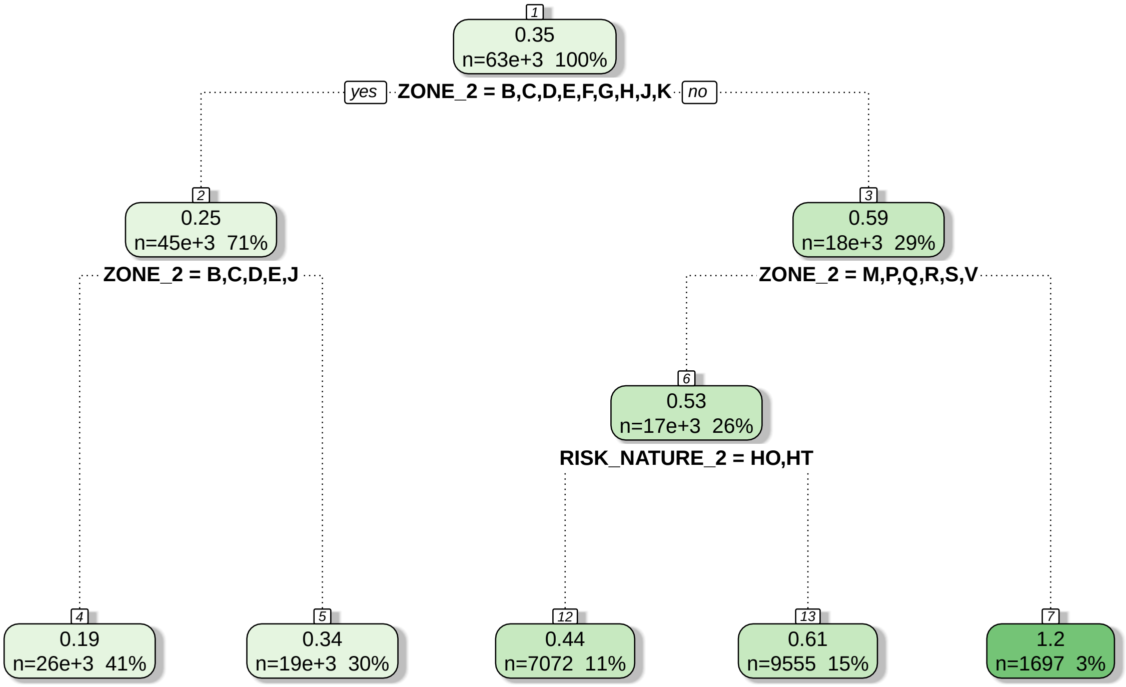
```
# paste(names(CombinedData), collapse=' + ')
library(rpart)
# grow tree
FitRpart = rpart::rpart(log(AMOUNT + 1) ~ RISK_NATURE_1 + RISK_NATURE_2 +
    RISK_NATURE_3 + RISK_FAMILY_STRUCT + RISK_ROOMS_NB + RISK_PROT_1 +
    RISK_PROT_2 + SURF_HOUSE + SURF_VER + SURF_OUTB + SURF_GR + OPTION_1 +
    OPTION_2 + OPTION_3 + OPTION_4 + OPTION_5 + OPTION_6 + OPTION_7 + OPTION_8 +
    ZONE_1 + ZONE_2 + ZONE_3, method = "anova", data = train, control = rpart.control(minsplit
    cp = 3e-04))

printcp(FitRpart)  # display the results
```

```
##
## Regression tree:
## rpart::rpart(formula = log(AMOUNT + 1) ~ RISK_NATURE_1 + RISK_NATURE_2 +
##     RISK_NATURE_3 + RISK_FAMILY_STRUCT + RISK_ROOMS_NB + RISK_PROT_1 +
##     RISK_PROT_2 + SURF_HOUSE + SURF_VER + SURF_OUTB + SURF_GR +
##     OPTION_1 + OPTION_2 + OPTION_3 + OPTION_4 + OPTION_5 + OPTION_6 +
##     OPTION_7 + OPTION_8 + ZONE_1 + ZONE_2 + ZONE_3, data = train,
##     method = "anova", control = rpart.control(minsplit = 3, cp = 3e-04))
##
## Variables actually used in tree construction:
##  [1] OPTION_8           RISK_FAMILY_STRUCT RISK_NATURE_1
##  [4] RISK_NATURE_2      RISK_NATURE_3      RISK_PROT_1
##  [7] RISK_ROOMS_NB      SURF_HOUSE         SURF_OUTB
## [10] SURF_VER           ZONE_1             ZONE_2
## [13] ZONE_3
##
## Root node error: 140427/62952 = 2.2307
##
## n= 62952
##
##            CP nsplit rel error  xerror     xstd
## 1  0.01072154      0   1.00000 1.00002 0.016426
## 2  0.00434420      1   0.98928 0.99009 0.016128
## 3  0.00166820      2   0.98493 0.98611 0.016034
## 4  0.00081651      3   0.98327 0.98427 0.015977
## 5  0.00056059      4   0.98245 0.98410 0.015974
## 6  0.00052124      5   0.98189 0.98543 0.015982
## 7  0.00051157      6   0.98137 0.98548 0.015981
## 8  0.00050696      7   0.98086 0.98622 0.015988
## 9  0.00046954      8   0.98035 0.98692 0.015993
## 10 0.00043887      9   0.97988 0.98764 0.015999
## 11 0.00043209     10   0.97944 0.98824 0.016004
## 12 0.00040175     11   0.97901 0.98886 0.016003
## 13 0.00038086     12   0.97861 0.99108 0.016020
## 14 0.00037150     17   0.97664 0.99241 0.016037
## 15 0.00034330     21   0.97515 0.99409 0.016055
## 16 0.00032969     22   0.97481 0.99650 0.016078
## 17 0.00031430     23   0.97448 0.99902 0.016105
## 18 0.00030763     24   0.97417 1.00281 0.016144
## 19 0.00030748     25   0.97386 1.00374 0.016153
## 20 0.00030000     26   0.97355 1.00606 0.016175
```

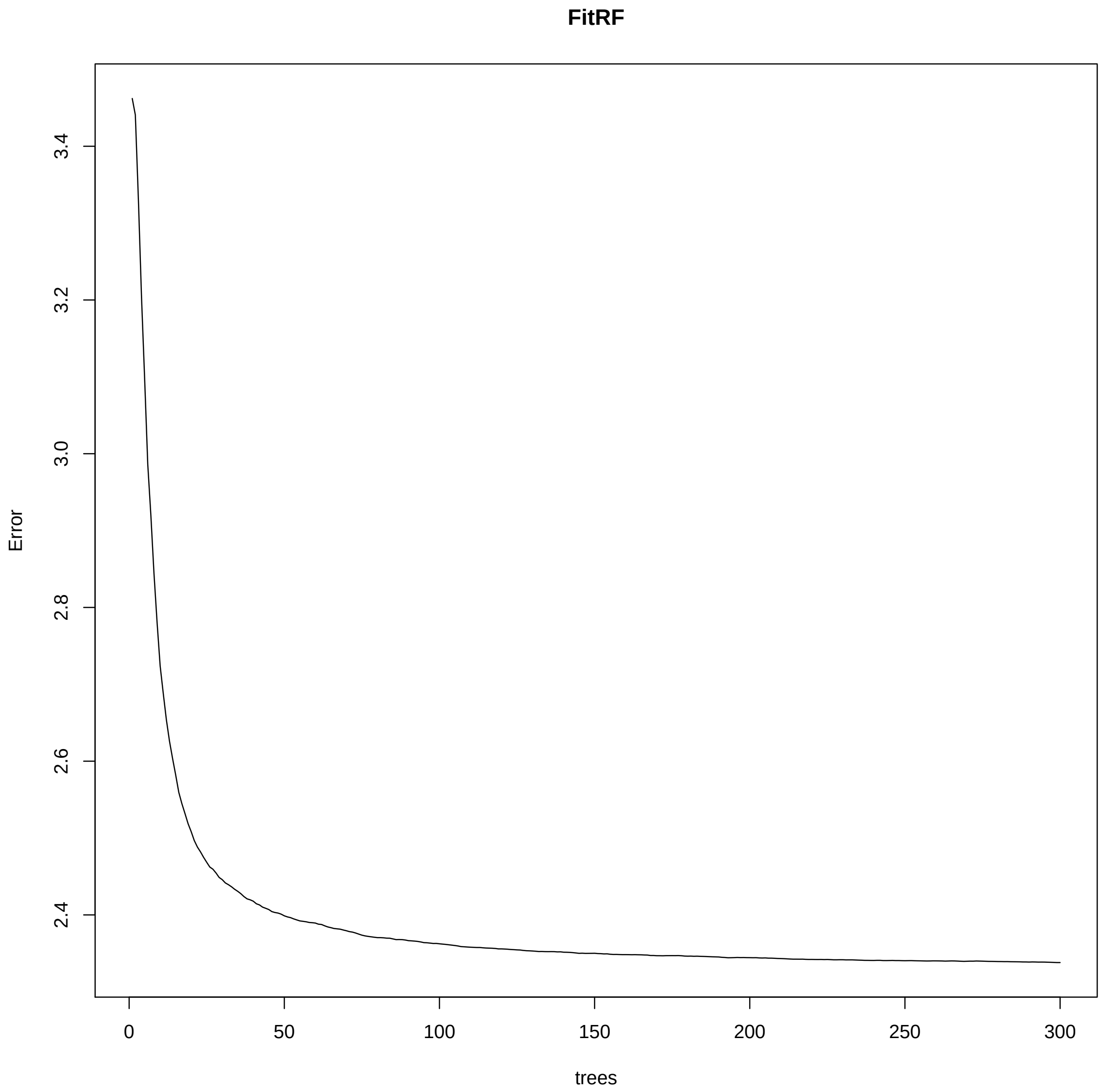
```r
plotcp(FitRpart)  # display the results
```



```r
# summary(FitRpart) # detailed summary of splits

# prune tree
PruneFit = rpart::prune(FitRpart, cp = FitRpart$cptable[which.min(FitRpart$cptable[,
    "xerror"]), "CP"])
library(rattle)
fancyRpartPlot(PruneFit, caption = NULL)
```



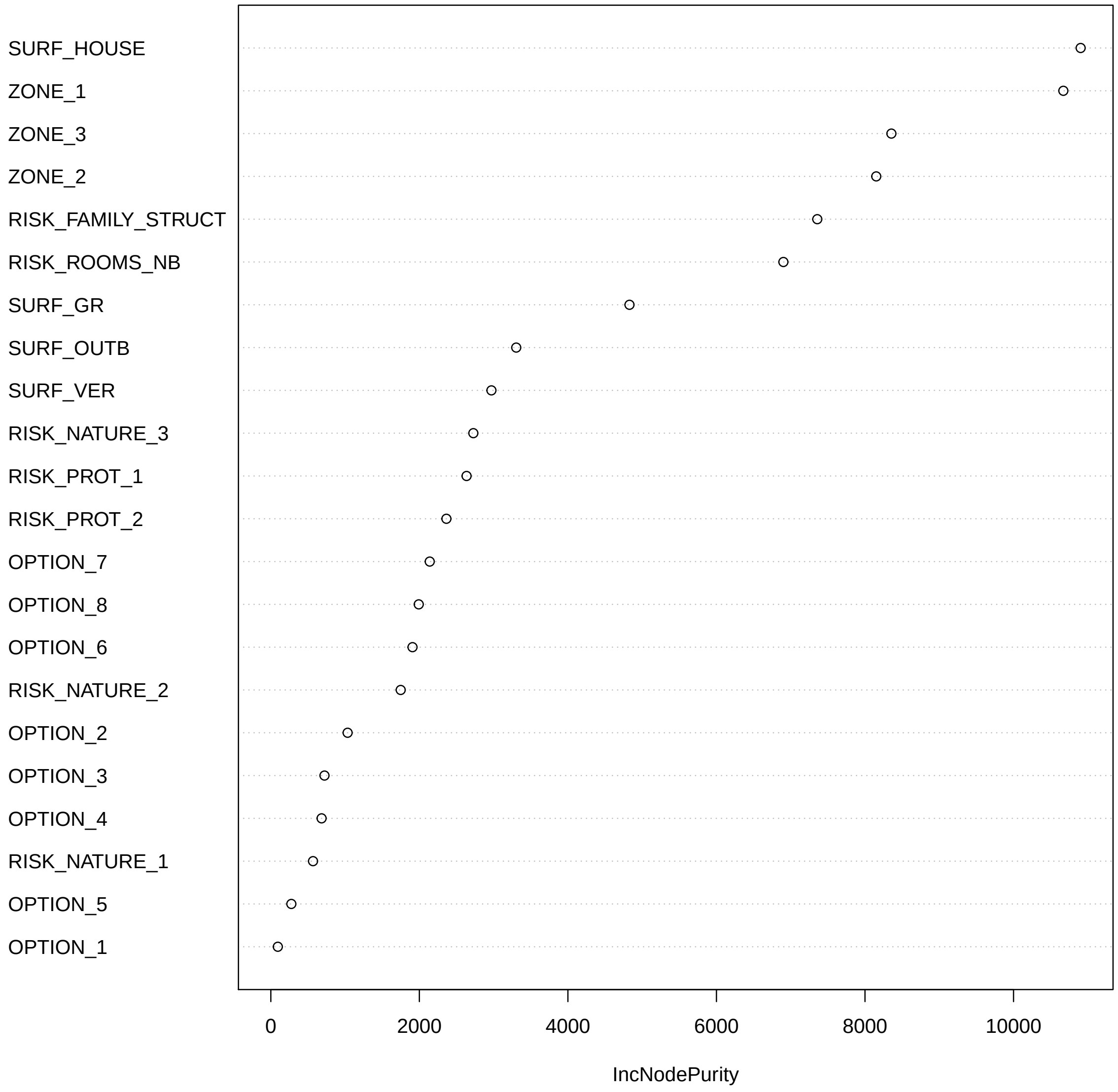
```
# Fit randomForest
library(randomForest)
# FitRF = randomForest(log(AMOUNT + 1) ~ RISK_NATURE_1 + RISK_NATURE_2
# + RISK_NATURE_3 + RISK_FAMILY_STRUCT + RISK_ROOMS_NB + RISK_PROT_1 +
# RISK_PROT_2 + SURF_HOUSE + SURF_VER + SURF_OUTB + SURF_GR + OPTION_1
# + OPTION_2 + OPTION_3 + OPTION_4 + OPTION_5 + OPTION_6 + OPTION_7 +
# OPTION_8 + ZONE_1 + ZONE_2 + ZONE_3, method = 'anova', data = train,
# importance = TRUE, ntree = 300, nodesize = 3, mtry = 6 )

# saveRDS(FitRF, 'FitRF.rds')
FitRF = readRDS("FitRF.rds")
plot(FitRF)
```



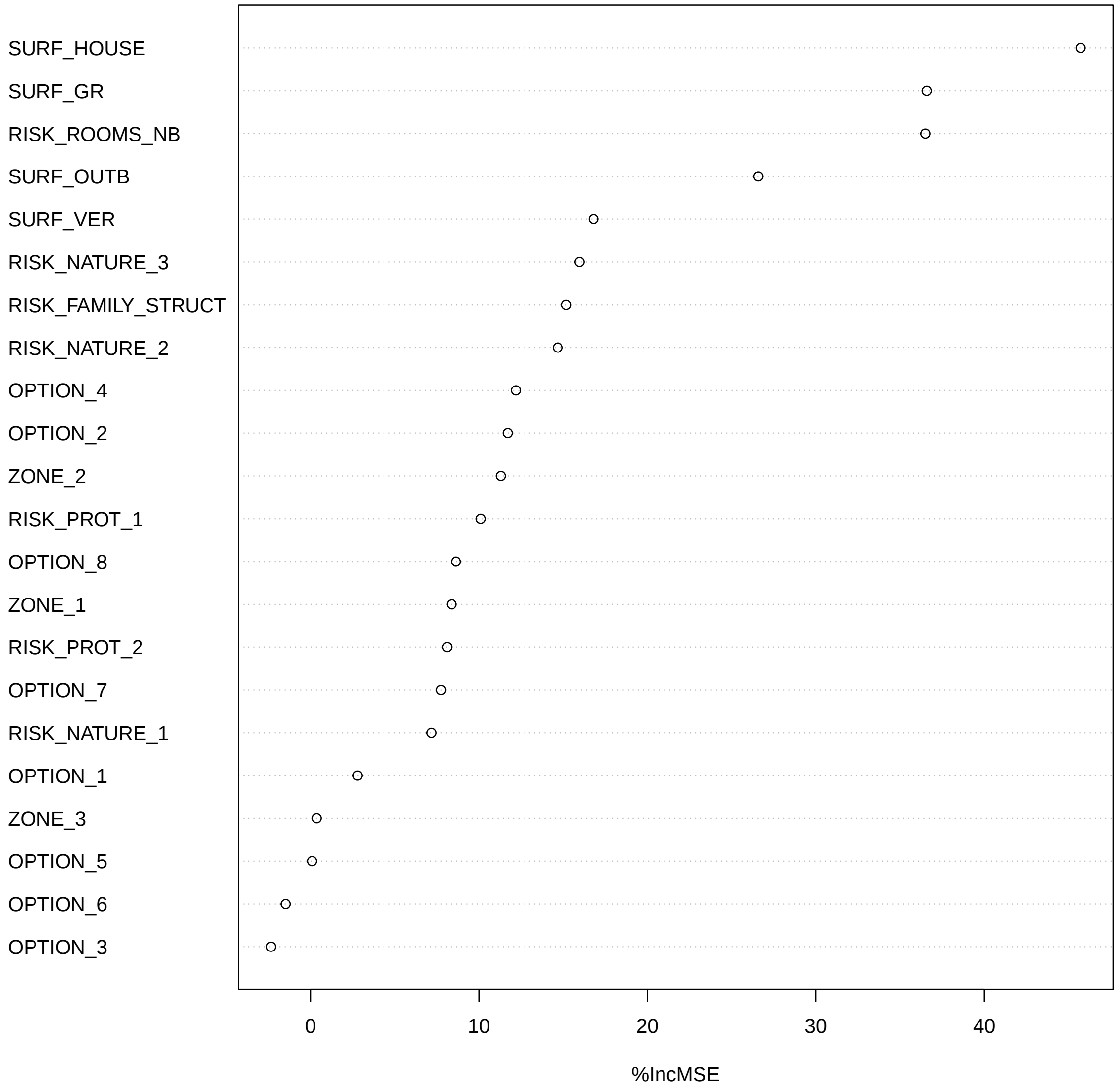
```
PredictionRandomForest <- predict(FitRF, test)
giniTest(test$AMOUNT, exp(PredictionRandomForest) - 1)
```

```
## [1] 0.2188753
```

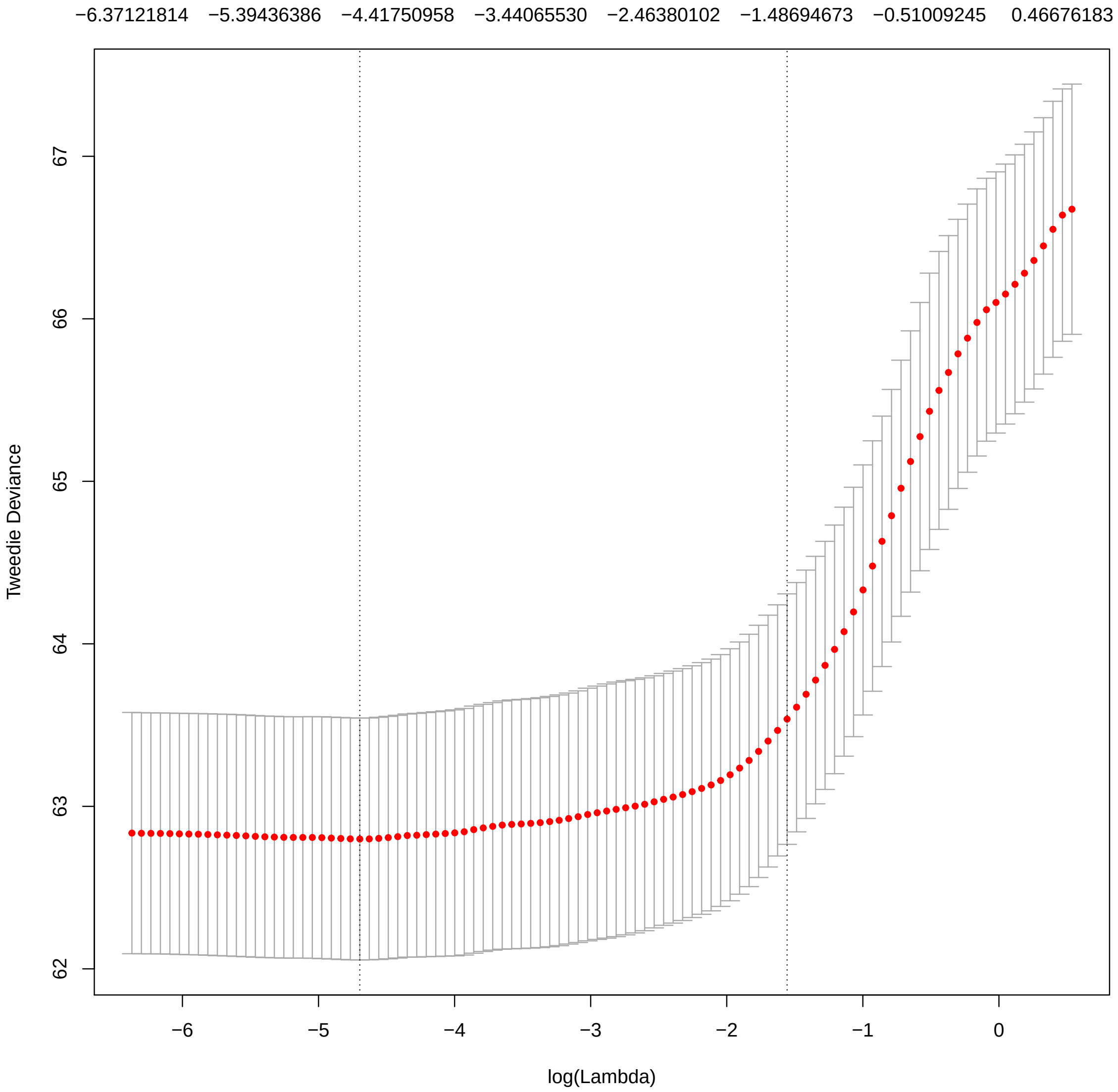
```
varImpPlot(FitRF, type = 2, main = NULL)  # IncNodePurity
```



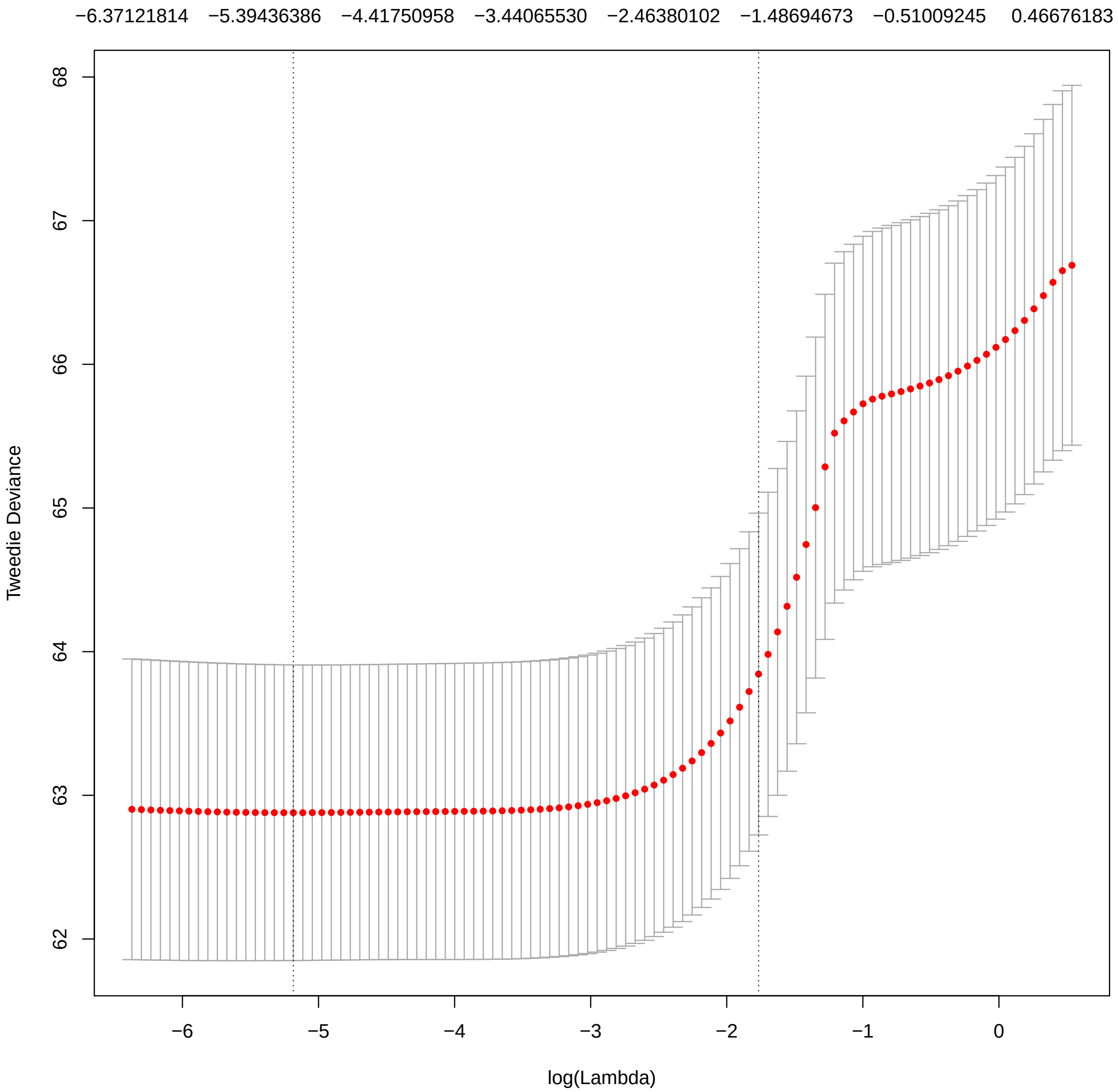
```r
varImpPlot(FitRF, type = 1, main = NULL)   # IncMSE
```



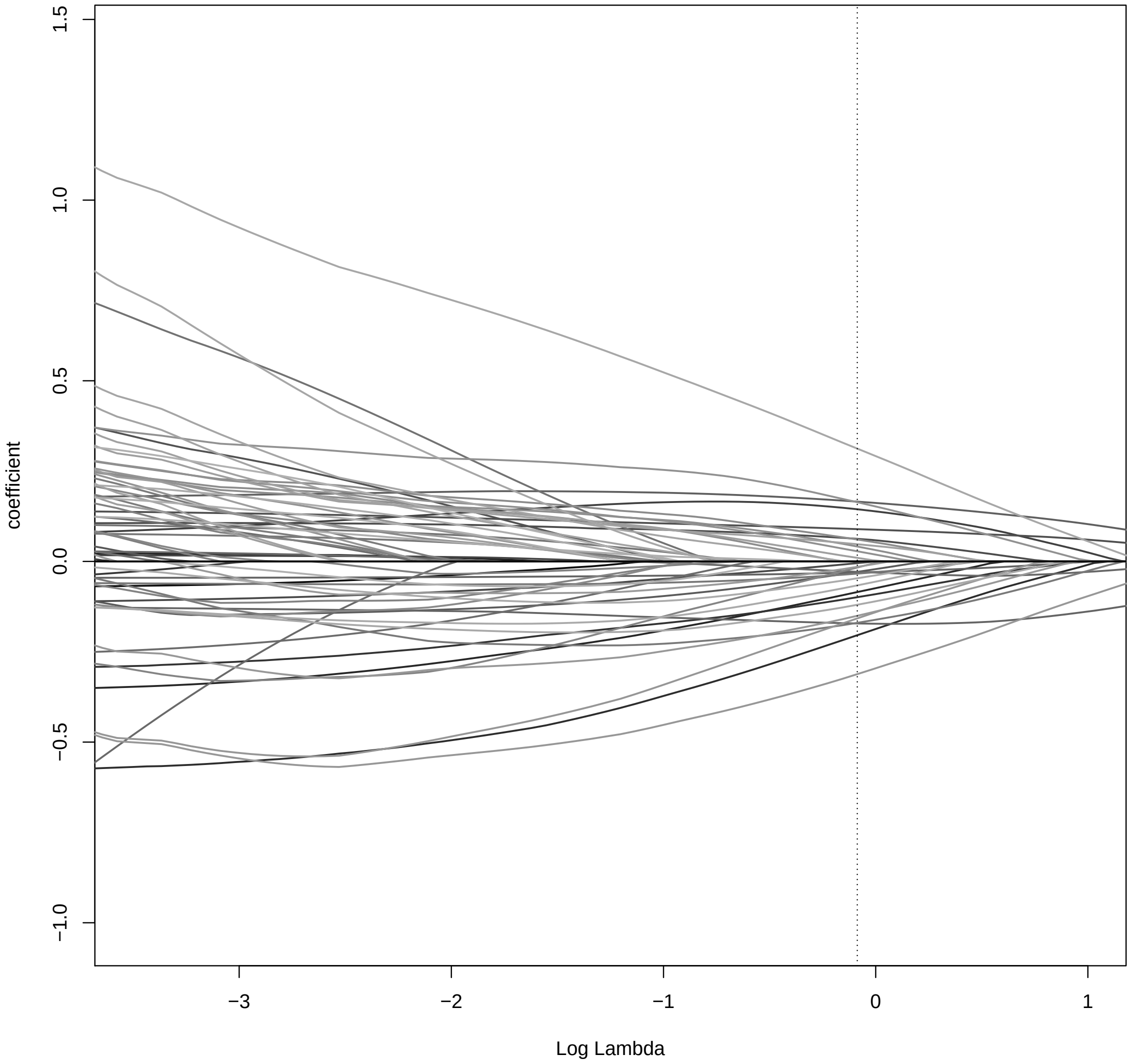
```r
library(cplm)
library(tweedie)
library(statmod)
library(HDtweedie)

xnames = names(train)[2:24][-20]
xContinuous = xnames[5:11][-2][-2]

y = train[, "AMOUNT"]

Dummy = caret::dummyVars(" ~ . ", data = train[, xnames], fullRank = TRUE)
X = data.frame(predict(Dummy, newdata = train[, xnames]))

X[, xContinuous] = scale(X[, xContinuous])

# CvTweedieLasso = HDtweedie::cv.HDtweedie(x = as.matrix(X), y = y,
# p=1.4, nfolds = 5) saveRDS(CvTweedieLasso, 'CvTweedieLasso.rds')
CvTweedieLasso = readRDS("CvTweedieLasso.rds")
plot(CvTweedieLasso)
```



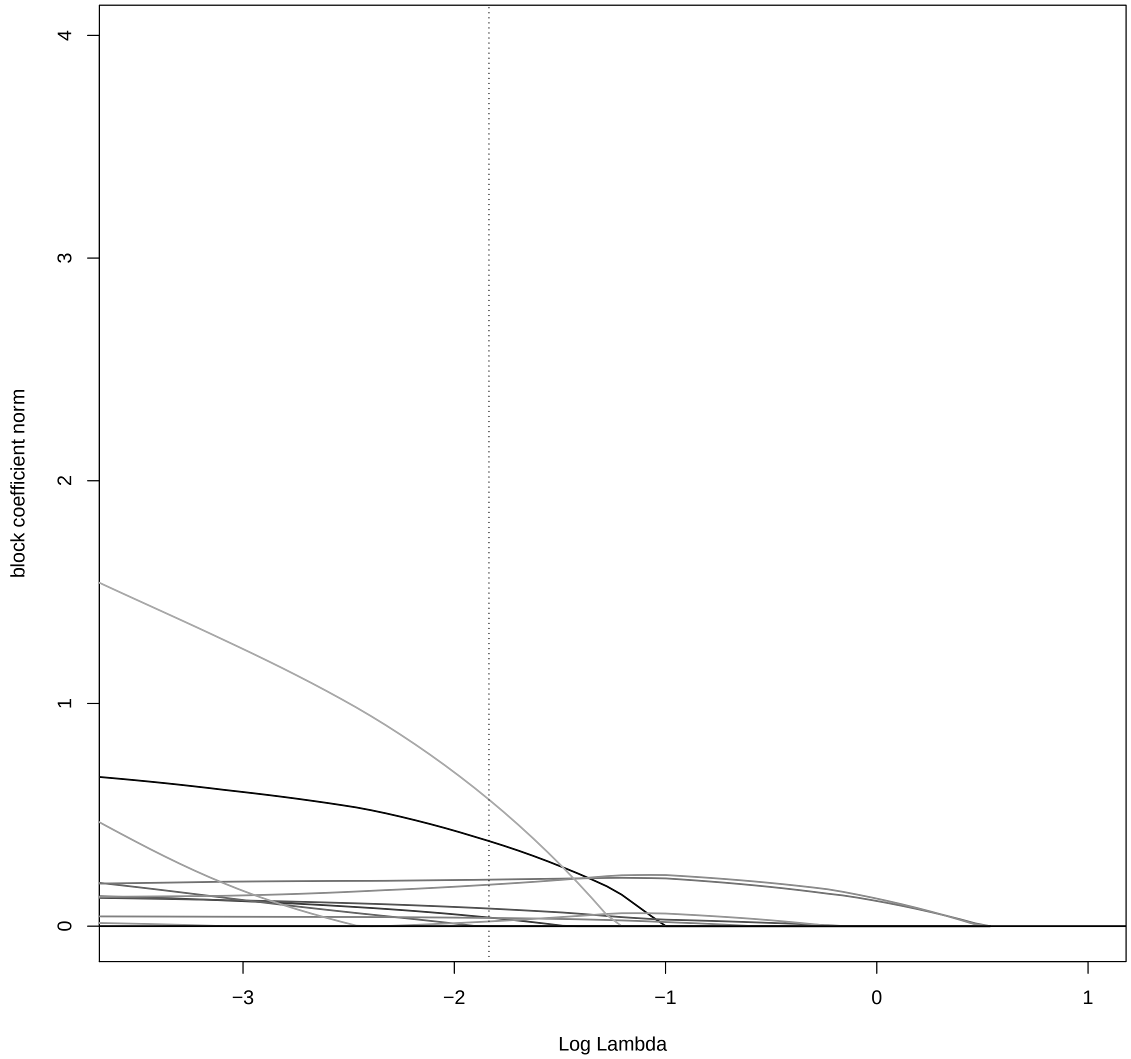
```
# group = c(rep(1,8), rep(2,4), rep(3,1), rep(4,5), 5:8, rep(9,8),
# rep(10,43)) CvTweedieGroupLasso = HDtweedie::cv.HDtweedie(x =
# as.matrix(X), y = y, p=1.4, group = group, nfolds = 5)
# saveRDS(CvTweedieGroupLasso, 'CvTweedieGroupLasso.rds')
CvTweedieGroupLasso = readRDS("CvTweedieGroupLasso.rds")
plot(CvTweedieGroupLasso)
```



```
CvTweedieNet = readRDS("CvTweedieNet.rds")
plot(CvTweedieNet$HDtweedie.fit, ylab = "coefficient", xlim = c(-3.5, 1))
abline(v = log(CvTweedieNet$lambda.1se), lty = 3)
```



```
CvTweedieGroupNet = readRDS("CvTweedieGroupNet.rds")
plot(CvTweedieGroupNet$HDtweedie.fit, group = T, ylab = "block coefficient norm",
    xlim = c(-3.5, 1))
abline(v = log(CvTweedieGroupNet$lambda.1se), lty = 3)
```



```r
load("test.RData")



Dummy = caret::dummyVars(" ~ . ", data = test[, xnames], fullRank = TRUE)
newX = data.frame(predict(Dummy, newdata = test[, xnames]))
newX[, xContinuous] = scale(newX[, xContinuous])



PredictionCvTweedieLasso = predict(CvTweedieLasso, newx = newX, type = "response")
giniTest(test$AMOUNT, PredictionCvTweedieLasso[, 1])
```

```
## [1] 0.293131
```

```r
coef(CvTweedieLasso, s = CvTweedieLasso$lambda.min)
```

```
##                               1
## (Intercept)          3.70212291
## RISK_NATURE_1.SH    -0.06867953
## RISK_NATURE_2.FT       0.00000000
## RISK_NATURE_2.HO      -0.34920365
## RISK_NATURE_2.HT      -0.59810008
## RISK_NATURE_3.2       -0.27166888
## RISK_NATURE_3.3       -0.01283313
## RISK_NATURE_3.4        0.00000000
## RISK_NATURE_3.5        0.07033417
## RISK_FAMILY_STRUCT.1  0.01247123
## RISK_FAMILY_STRUCT.2  0.00000000
## RISK_FAMILY_STRUCT.3 -0.11437078
## RISK_FAMILY_STRUCT.4  0.10341294
## RISK_ROOMS_NB         0.14178233
## RISK_PROT_1.2         0.42770046
## RISK_PROT_1.3         0.00000000
## RISK_PROT_1.4         0.08856674
## RISK_PROT_2.1        -0.06269516
## RISK_PROT_2.2         0.00000000
## SURF_HOUSE            0.17582161
## SURF_VER             -0.04419053
## SURF_OUTB            -0.12438925
## SURF_GR               0.02855478
## OPTION_1.True        -0.99354545
## OPTION_2.True        -0.24571076
## OPTION_3.True         0.11198731
## OPTION_4.True         0.04407437
## OPTION_5.True         0.84918798
## OPTION_6.True         0.09977914
## OPTION_7.True         0.07452389
## OPTION_8.True        -0.05264832
## ZONE_1.C             -0.02324024
## ZONE_1.D              0.22455806
## ZONE_1.E              0.16682130
## ZONE_1.F              0.05009945
## ZONE_1.G              0.12407199
## ZONE_1.H             -0.09597752
## ZONE_1.J             -0.33234015
## ZONE_1.K              0.01717591
## ZONE_1.L              0.12426932
## ZONE_1.M             -0.20187855
## ZONE_1.N              0.16156483
## ZONE_1.P              0.11686598
## ZONE_1.Q              0.13143970
## ZONE_1.R              0.13757140
## ZONE_1.S              0.22542339
## ZONE_1.T              0.11058516
## ZONE_1.U              0.13988385
## ZONE_1.Z              0.00000000
## ZONE_1.W              0.00000000
## ZONE_2.C             -0.27023885
## ZONE_2.D             -0.25910304
## ZONE_2.E              0.00000000
## ZONE_2.F              0.26543092
## ZONE_2.G              0.00000000
## ZONE_2.H              0.51617274
## ZONE_2.J              0.00000000
## ZONE_2.K              0.47453471
## ZONE_2.M              0.62115854
## ZONE_2.P              0.69667048
## ZONE_2.Q              0.57304771
## ZONE_2.R              0.81551204
## ZONE_2.S              0.85239634
## ZONE_2.V              1.25686878
## ZONE_2.W              1.50688663
## ZONE_3.E             -0.11879427
## ZONE_3.G             -0.10486273
## ZONE_3.H              0.00000000
## ZONE_3.J              0.22584120
## ZONE_3.K              0.13687793
## ZONE_3.L              0.15459880
## ZONE_3.M              0.01106211
## ZONE_3.P              0.37261401
## ZONE_3.Z              0.00000000
```

```
x1 = coef(CvTweedieLasso, s = CvTweedieLasso$lambda.min)
ImpCvTweedieLasso = rownames(x1)[abs(x1) > 0.4]
ImpCvTweedieLasso
```

```
##  [1] "(Intercept)"      "RISK_NATURE_2.HT" "RISK_PROT_1.2"
##  [4] "OPTION_1.True"    "OPTION_5.True"    "ZONE_2.H"
##  [7] "ZONE_2.K"         "ZONE_2.M"         "ZONE_2.P"
## [10] "ZONE_2.Q"         "ZONE_2.R"         "ZONE_2.S"
## [13] "ZONE_2.V"         "ZONE_2.W"
```

```
PredictionCvTweedieGroupLasso = predict(CvTweedieGroupLasso, newx = newX,
    type = "response")
giniTest(test$AMOUNT, PredictionCvTweedieGroupLasso[, 1])
```

```
## [1] 0.2940705
```

```
coef(CvTweedieGroupLasso, s = CvTweedieGroupLasso$lambda.min)
```

```
##                                  1
## (Intercept)           3.6598713118
## RISK_NATURE_1.SH     -0.0711015354
## RISK_NATURE_2.FT      0.1088697874
## RISK_NATURE_2.HO     -0.3495216864
## RISK_NATURE_2.HT     -0.4895595749
## RISK_NATURE_3.2      -0.3806897875
## RISK_NATURE_3.3      -0.0699957996
## RISK_NATURE_3.4       0.0993291448
## RISK_NATURE_3.5       0.0797918601
## RISK_FAMILY_STRUCT.1  0.0206656673
## RISK_FAMILY_STRUCT.2  0.0004809425
## RISK_FAMILY_STRUCT.3 -0.1110944090
## RISK_FAMILY_STRUCT.4  0.1047222367
## RISK_ROOMS_NB         0.1410068964
## RISK_PROT_1.2         0.4134381011
## RISK_PROT_1.3         0.1068676612
## RISK_PROT_1.4         0.1030658174
## RISK_PROT_2.1        -0.0526722338
## RISK_PROT_2.2        -0.0055556039
## SURF_HOUSE            0.1761413234
## SURF_VER             -0.0449734881
## SURF_OUTB            -0.1253530937
## SURF_GR               0.0293764351
## OPTION_1.True        -0.8691999237
## OPTION_2.True        -0.2669034402
## OPTION_3.True         0.1633503144
## OPTION_4.True         0.0888232429
## OPTION_5.True         0.8455668209
## OPTION_6.True         0.1106391052
## OPTION_7.True         0.0824773908
## OPTION_8.True        -0.0604818339
## ZONE_1.C              0.0488030028
## ZONE_1.D              0.3264932971
## ZONE_1.E              0.2731558416
## ZONE_1.F              0.1695106756
## ZONE_1.G              0.2473250987
## ZONE_1.H             -0.0079847705
## ZONE_1.J             -0.2205038503
## ZONE_1.K              0.1662278107
## ZONE_1.L              0.3011286187
## ZONE_1.M             -0.0593672583
## ZONE_1.N              0.3534238700
## ZONE_1.P              0.3189143956
## ZONE_1.Q              0.3279461931
## ZONE_1.R              0.3736840519
## ZONE_1.S              0.4597872301
## ZONE_1.T              0.3700691571
## ZONE_1.U              0.3739627568
## ZONE_1.Z              0.0240119665
## ZONE_1.W             -0.0088313321
## ZONE_2.C             -0.3297040532
## ZONE_2.D             -0.3365285387
## ZONE_2.E             -0.0941321952
## ZONE_2.F              0.1726548628
## ZONE_2.G              0.1994486554
## ZONE_2.H              0.4133191927
## ZONE_2.J             -0.0735959357
## ZONE_2.K              0.3431939469
## ZONE_2.M              0.4716374210
## ZONE_2.P              0.5137781178
## ZONE_2.Q              0.3995632154
## ZONE_2.R              0.6110816979
## ZONE_2.S              0.6793426324
## ZONE_2.V              1.0391967476
## ZONE_2.W              1.2744625481
## ZONE_3.E             -0.1026671715
## ZONE_3.G             -0.1025446285
## ZONE_3.H              0.0176973822
## ZONE_3.J              0.2378915953
## ZONE_3.K              0.1431943333
## ZONE_3.L              0.2111802962
## ZONE_3.M              0.0298785391
## ZONE_3.P              0.3567862575
## ZONE_3.Z             -0.0088313321
```

```
x2 = coef(CvTweedieGroupLasso, s = CvTweedieGroupLasso$lambda.min)
ImpCvTweedieGroupLasso = rownames(x2)[abs(x2) > 0.4]
ImpCvTweedieGroupLasso
```

```
##  [1] "(Intercept)"      "RISK_NATURE_2.HT" "RISK_PROT_1.2"
##  [4] "OPTION_1.True"    "OPTION_5.True"    "ZONE_1.S"
##  [7] "ZONE_2.H"         "ZONE_2.M"         "ZONE_2.P"
## [10] "ZONE_2.R"         "ZONE_2.S"         "ZONE_2.V"
## [13] "ZONE_2.W"
```

```
PredictionCvTweedieNet = predict(CvTweedieNet, newx = newX, type = "response")
giniTest(test$AMOUNT, PredictionCvTweedieNet[, 1])
```

```
## [1] 0.2889836
```

```
coef(CvTweedieNet, s = CvTweedieNet$lambda.min)
```

```
##                                 1
## (Intercept)           3.584360573
## RISK_NATURE_1.SH     -0.070868135
## RISK_NATURE_2.FT      0.015717705
## RISK_NATURE_2.HO     -0.357887158
## RISK_NATURE_2.HT     -0.587098833
## RISK_NATURE_3.2      -0.295151141
## RISK_NATURE_3.3      -0.063787757
## RISK_NATURE_3.4       0.073524744
## RISK_NATURE_3.5       0.069487693
## RISK_FAMILY_STRUCT.1  0.018827201
## RISK_FAMILY_STRUCT.2  0.000000000
## RISK_FAMILY_STRUCT.3 -0.115052179
## RISK_FAMILY_STRUCT.4  0.105450666
## RISK_ROOMS_NB         0.142398776
## RISK_PROT_1.2         0.496467869
## RISK_PROT_1.3         0.124227134
## RISK_PROT_1.4         0.120007227
## RISK_PROT_2.1        -0.036439126
## RISK_PROT_2.2        -0.049624284
## SURF_HOUSE            0.174324146
## SURF_VER             -0.045144010
## SURF_OUTB            -0.124639136
## SURF_GR               0.031234503
## OPTION_1.True        -0.934789424
## OPTION_2.True        -0.262124404
## OPTION_3.True         0.153769470
## OPTION_4.True         0.086259907
## OPTION_5.True         0.871199361
## OPTION_6.True         0.107218638
## OPTION_7.True         0.081278461
## OPTION_8.True        -0.057528230
## ZONE_1.C              0.048327809
## ZONE_1.D              0.322195705
## ZONE_1.E              0.267689319
## ZONE_1.F              0.161875117
## ZONE_1.G              0.238818421
## ZONE_1.H             -0.009905355
## ZONE_1.J             -0.227387014
## ZONE_1.K              0.152934022
## ZONE_1.L              0.284793544
## ZONE_1.M             -0.070804745
## ZONE_1.N              0.331076266
## ZONE_1.P              0.294382935
## ZONE_1.Q              0.305902415
## ZONE_1.R              0.344267924
## ZONE_1.S              0.429854554
## ZONE_1.T              0.339612630
## ZONE_1.U              0.355681856
## ZONE_1.Z              0.000000000
## ZONE_1.W              0.000000000
## ZONE_2.C             -0.264152548
## ZONE_2.D             -0.268895743
## ZONE_2.E             -0.023254694
## ZONE_2.F              0.242052005
## ZONE_2.G              0.184380808
## ZONE_2.H              0.485143718
## ZONE_2.J              0.000000000
## ZONE_2.K              0.421494013
## ZONE_2.M              0.551916364
## ZONE_2.P              0.600770069
## ZONE_2.Q              0.489965416
## ZONE_2.R              0.709505307
## ZONE_2.S              0.772497106
## ZONE_2.V              1.155874782
## ZONE_2.W              1.380228476
## ZONE_3.E             -0.099421912
## ZONE_3.G             -0.094782117
## ZONE_3.H              0.022472901
## ZONE_3.J              0.239962510
## ZONE_3.K              0.146386416
## ZONE_3.L              0.207006130
## ZONE_3.M              0.031908382
## ZONE_3.P              0.367295016
## ZONE_3.Z              0.000000000
```

```
x3 = coef(CvTweedieNet, s = CvTweedieGroupLasso$lambda.min)
ImpCvTweedieNet = rownames(x3)[abs(x3) > 0.4]
ImpCvTweedieNet
```

```
##  [1] "(Intercept)"      "RISK_NATURE_2.HT" "RISK_PROT_1.2"
##  [4] "OPTION_1.True"    "OPTION_5.True"    "ZONE_1.S"
##  [7] "ZONE_2.H"         "ZONE_2.K"         "ZONE_2.M"
## [10] "ZONE_2.P"         "ZONE_2.Q"         "ZONE_2.R"
## [13] "ZONE_2.S"         "ZONE_2.V"         "ZONE_2.W"
```

```
PredictionCvTweedieGroupNet = predict(CvTweedieGroupNet, newx = newX, type = "response")
giniTest(test$AMOUNT, PredictionCvTweedieGroupNet[, 1])
```

```
## [1] 0.2967058
```

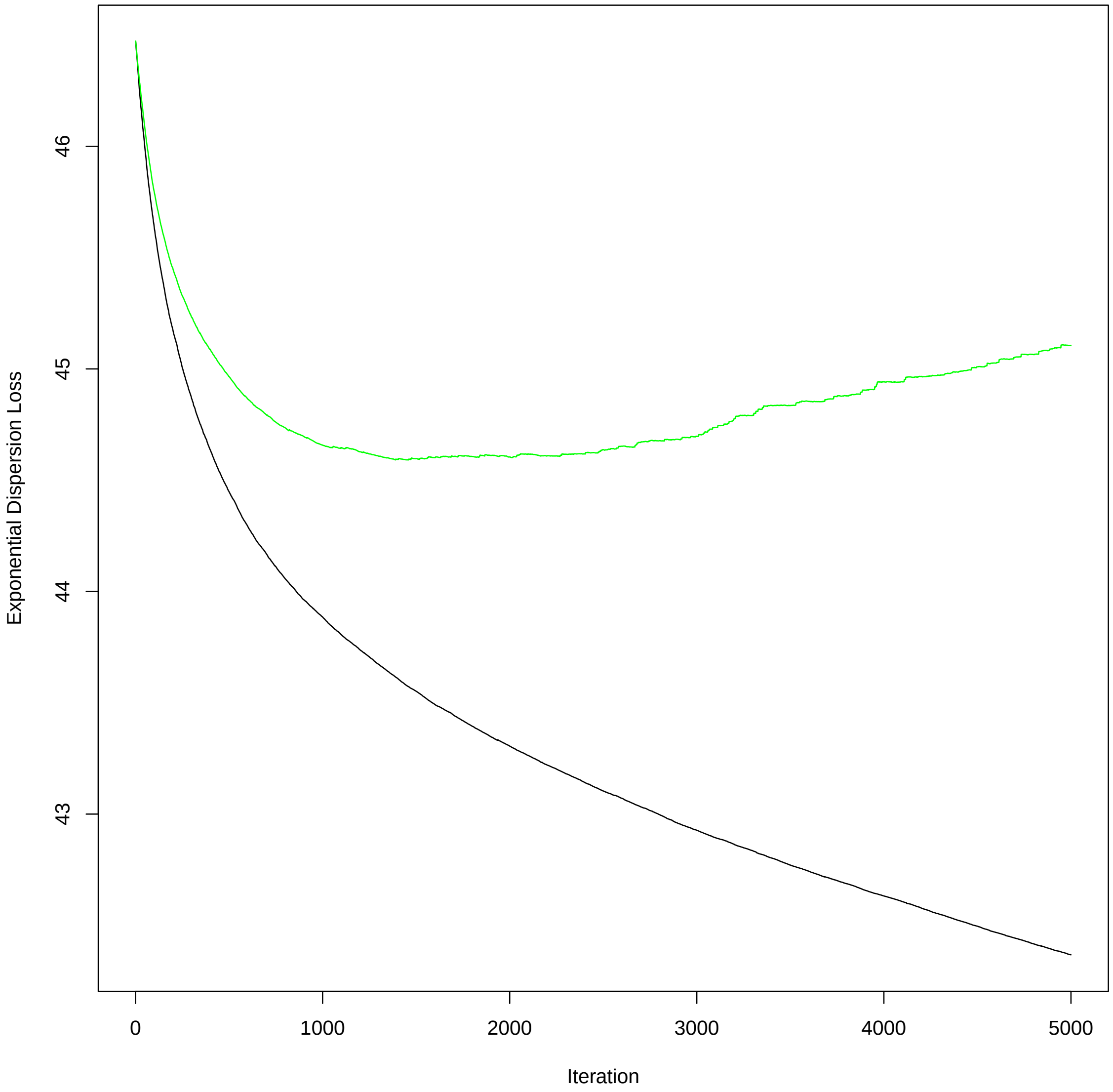
```
coef(CvTweedieGroupNet, s = CvTweedieGroupNet$lambda.min)
```

```
##                                  1
## (Intercept)           3.4653641653
## RISK_NATURE_1.SH     -0.0702512193
## RISK_NATURE_2.FT      0.1080702630
## RISK_NATURE_2.HO     -0.3518372155
## RISK_NATURE_2.HT     -0.4918745563
## RISK_NATURE_3.2      -0.3838042933
## RISK_NATURE_3.3      -0.0750094012
## RISK_NATURE_3.4       0.1091097543
## RISK_NATURE_3.5       0.0736852117
## RISK_FAMILY_STRUCT.1  0.0208559383
## RISK_FAMILY_STRUCT.2  0.0005343567
## RISK_FAMILY_STRUCT.3 -0.1133710074
## RISK_FAMILY_STRUCT.4  0.1048404012
## RISK_ROOMS_NB         0.1428495637
## RISK_PROT_1.2         0.4866977133
## RISK_PROT_1.3         0.1612945665
## RISK_PROT_1.4         0.1330425990
## RISK_PROT_2.1        -0.0243970395
## RISK_PROT_2.2        -0.0453711388
## SURF_HOUSE            0.1736093145
## SURF_VER             -0.0449706158
## SURF_OUTB            -0.1241657931
## SURF_GR               0.0309537187
## OPTION_1.True        -1.0487917777
## OPTION_2.True        -0.2682181946
## OPTION_3.True         0.1686467154
## OPTION_4.True         0.1054277659
## OPTION_5.True         0.9033481708
## OPTION_6.True         0.1119360146
## OPTION_7.True         0.0832950279
## OPTION_8.True        -0.0585920720
## ZONE_1.C              0.0950164154
## ZONE_1.D              0.3653150478
## ZONE_1.E              0.3084000169
## ZONE_1.F              0.2030180046
## ZONE_1.G              0.2802750729
## ZONE_1.H              0.0196994702
## ZONE_1.J             -0.1929170723
## ZONE_1.K              0.1938529204
## ZONE_1.L              0.3304095802
## ZONE_1.M             -0.0354197677
## ZONE_1.N              0.3743166931
## ZONE_1.P              0.3342641019
## ZONE_1.Q              0.3505528074
## ZONE_1.R              0.3976428526
## ZONE_1.S              0.4805179225
## ZONE_1.T              0.3985485743
## ZONE_1.U              0.4164219564
## ZONE_1.Z              0.0397524796
## ZONE_1.W             -0.0139196882
## ZONE_2.C             -0.2144897433
## ZONE_2.D             -0.2203199254
## ZONE_2.E              0.0226154585
## ZONE_2.F              0.2933699173
## ZONE_2.G              0.3194748731
## ZONE_2.H              0.5374767062
## ZONE_2.J             -0.1091247726
## ZONE_2.K              0.4732445367
## ZONE_2.M              0.5990876157
## ZONE_2.P              0.6482059614
## ZONE_2.Q              0.5480528866
## ZONE_2.R              0.7633250495
## ZONE_2.S              0.8328297764
## ZONE_2.V              1.2233148129
## ZONE_2.W              1.4318109408
## ZONE_3.E             -0.0886324499
## ZONE_3.G             -0.0846706057
## ZONE_3.H              0.0405926165
## ZONE_3.J              0.2510075368
## ZONE_3.K              0.1547699628
## ZONE_3.L              0.2238675456
## ZONE_3.M              0.0453873388
## ZONE_3.P              0.3794681359
## ZONE_3.Z             -0.0139196882
```

```
x4 = coef(CvTweedieGroupNet, s = CvTweedieGroupLasso$lambda.min)
ImpCvTweedieGroupNet = rownames(x4)[abs(x4) > 0.4]
ImpCvTweedieGroupNet
```

```
##  [1] "(Intercept)"      "RISK_NATURE_2.HT" "RISK_PROT_1.2"
##  [4] "OPTION_1.True"    "OPTION_5.True"    "ZONE_1.S"
##  [7] "ZONE_2.H"         "ZONE_2.M"         "ZONE_2.P"
## [10] "ZONE_2.R"         "ZONE_2.S"         "ZONE_2.V"
## [13] "ZONE_2.W"
```

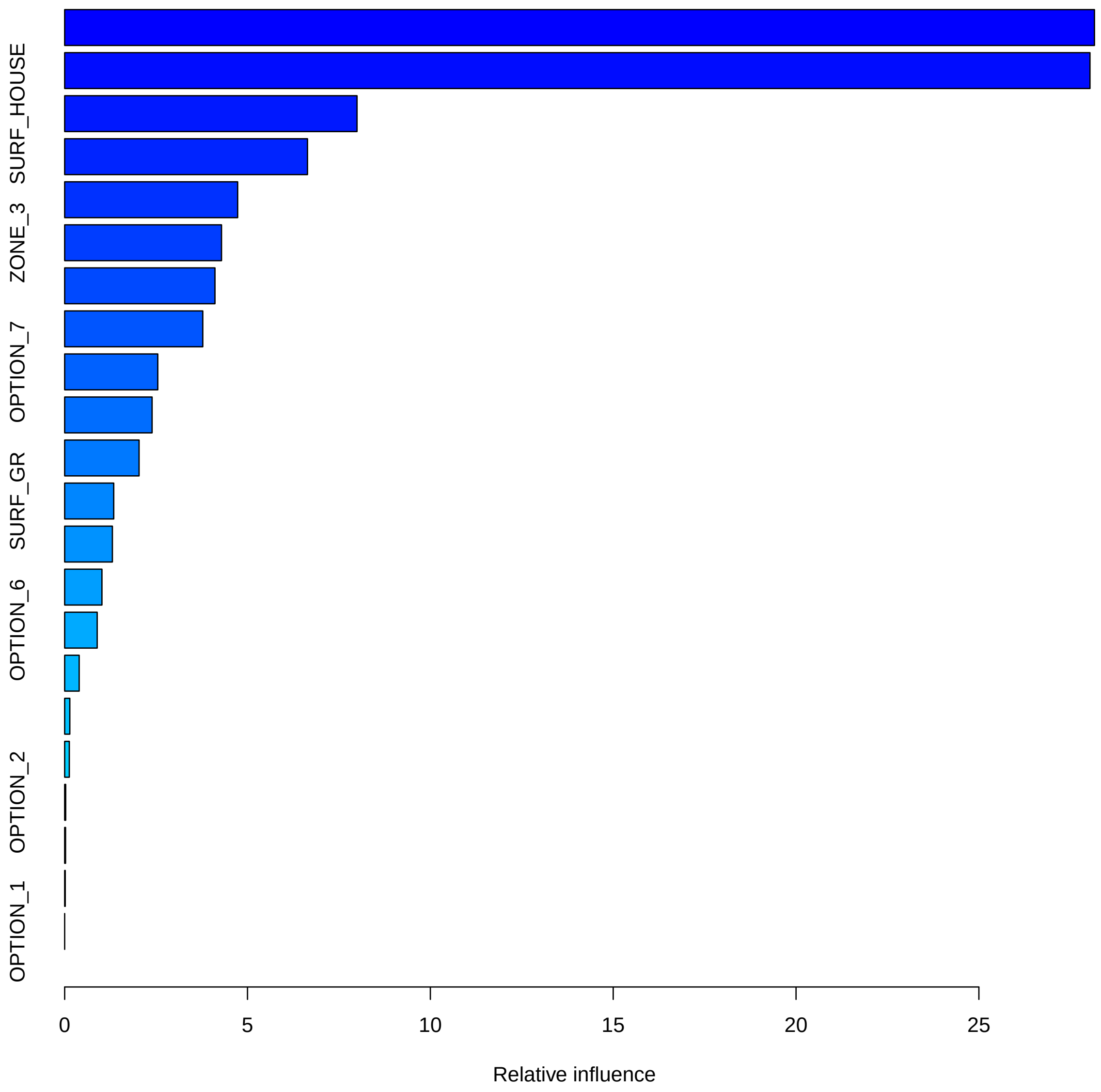
```
library(TDboost)

# FitTDboost = TDboost(AMOUNT ~ RISK_NATURE_1 + RISK_NATURE_2 +
# RISK_NATURE_3 + RISK_FAMILY_STRUCT + RISK_ROOMS_NB + RISK_PROT_1 +
# RISK_PROT_2 + SURF_HOUSE + SURF_VER + SURF_OUTB + SURF_GR + OPTION_1
# + OPTION_2 + OPTION_3 + OPTION_4 + OPTION_5 + OPTION_6 + OPTION_7 +
# OPTION_8 + ZONE_1 + ZONE_2 + ZONE_3,
# distribution=list(name='EDM',alpha=1.4), n.trees=5000,
# shrinkage=0.005, interaction.depth=3, n.minobsinnode = 10,
# cv.folds=5, data = train ) saveRDS(FitTDboost, 'FitTDboost.rds')

FitTDboost = readRDS("FitTDboost.rds")
best.iter <- TDboost.perf(FitTDboost, method = "cv")
```



```
PredictionCvTDboost <- predict.TDboost(FitTDboost, test, best.iter)
giniTest(test$AMOUNT, PredictionCvTDboost)
```

```
## [1] 0.2924537
```

```
summary(FitTDboost, n.trees = best.iter)
```



```
##                  var     rel.inf
## 1             ZONE_1 28.16045405
## 2             ZONE_2 28.03807027
## 3         SURF_HOUSE  7.99465662
## 4      RISK_ROOMS_NB  6.64053532
## 5           OPTION_5  4.73079253
## 6             ZONE_3  4.28921639
## 7        RISK_PROT_1  4.10977911
## 8  RISK_FAMILY_STRUCT  3.77709294
## 9           OPTION_7  2.54642862
## 10         SURF_OUTB  2.39030998
## 11     RISK_NATURE_2  2.03477426
## 12           SURF_GR  1.34185696
## 13     RISK_NATURE_3  1.30493943
## 14     RISK_NATURE_1  1.01918417
## 15          OPTION_6  0.88963715
## 16        RISK_PROT_2  0.39678155
## 17           OPTION_8  0.14104089
## 18           SURF_VER  0.12689617
## 19           OPTION_2  0.02773766
## 20           OPTION_4  0.02392972
## 21           OPTION_3  0.01588622
## 22           OPTION_1  0.00000000
```

7 Summary and Discussion

Appendix: Variable dictionary

The two training datasets (`data year1.csv` and `data year2.csv`) contain the following 25 variables:

1. `IDCNT` Unique contract ID
2. `RISK NATURE 1` An Indicator for whether this is a primary house PH or a secondary house SH.
3. `RISK NATURE 2` Nature of the ownership and house type. This indicates, both if this is a house H or a flat F, and, if the client is the owner O or a tenant T. Hence there are 4 possible values FO, FT, HO, HT.)
4. `RISK NATURE 3` The type of occupancy. This can be one of '001', '002', '003', '004', '005'. For example, '002' indicates a tenant while the others are different types of occupancy.
5. `RISK FAMILY STRUCT` The family structure. This can be one of '000', '001', '002', '003', '004' representing different combinations of adults and children in the client family.
6. `RISK ROOMS NB` Number of declared rooms. Note that '011' means 11 or more (can be up to 20 in real life).
7. `RISK PROT 1` Protection indicator 1. This can be one of '001', '002', '003', '004' and represents different house protection schemes.
8. `RISK PROT 2` Protection indicator 2. This can be one of '000', '001', '002'. This represents another class of house protection schemes.
9. `SURF HOUSE` Total house surface area. This ranges from '001' to '028', representing the real house surface area in ascending order.
10. `SURF VER` Veranda surface area. This ranges from '000' to '021', in ascending order of the declared veranda surface area. For example, '000' means no veranda present.
11. `SURF OUTB` Extra buildings surface area. This ranges from '000' to '011', in ascending order of area. This captures the total area of external buildings other than the main house. For example, '000' means no other building than the main house.
12. `SURF GR` Ground surface. Ranges from '000' to '014', in ascending order of the declared terrain surface. Again '000' means no free terrain surface.
13. `OPTION 1` Different options (true or false)

14. `OPTION 2` Different options (true or false)
15. `OPTION 3` Different options (true or false)
16. `OPTION 4` Different options (true or false)
17. `OPTION 5` Different options (true or false)
18. `OPTION 6` Different options (true or false)
19. `OPTION 7` Different options (true or false)
20. `OPTION 8` Different options (true or false)
21. `INSEE CODE` The INSEE code is a numerical index used by the French National Institute for Statistics and Economic Studies (INSEE) to identify various entities, including communes, departments.
22. `ZONE 1` Some geographical information. The zone scales, alphabetically ordered. Each zone is a combination of communes.
23. `ZONE 2` Other geographical zones.
24. `ZONE 3` Other geographical zones.
25. `AMOUNT` Total loss for that policy.

Please note: The `INSEE CODE` is a 5-digits alphanumeric code used by the French National Institute for Statistics and Economic Studies identify communes and departments in France. There are about 36,000 communes in France, but not every one of them is present in the dataset. The first 2 digits of INSEE CODE identifies the department. The `INSEE CODE` (or department code) can be used to merge external data to the datasets: population density, OpenStreetMap data, etc. If needed, two shapefiles are available online:

For French regional Departments https://pricing-game.dsi.ic.ac.uk/static/DEPARTEMENTS.zip

For communes https://pricing-game.dsi.ic.ac.uk/static/COMMUNES.zip

Be aware that, if you need to graph geographical information, the French reference system is RGF93 / Lambert-93 (EPSG: 2154) and not the common WGS84.

Reference

Qian, Wei, Yi Yang, and Hui Zou. 2016. "Tweedie's Compound Poisson Model with Grouped Elastic Net." *Journal of Computational and Graphical Statistics* 25 (2). Taylor & Francis: 606–25.

Yang, Yi, Wei Qian, and Hui Zou. 2017. "Insurance Premium Prediction via Gradient Tree-Boosted Tweedie Compound Poisson Models." *Journal of Business & Economic Statistics*. Taylor & Francis, 1–15.