```
> setwd('C:/Rtest')

>

> #install.packages('ipred')

> #install.packages('caret')

> #install.packages('randomForest')

> #install.packages('MASS')

> #install.packages('gbm')

>

> library(ipred)

Warning message:

패키지 'ipred'는 R 버전 4.1.3에서 작성되었습니다

> library(rpart)

> library(reshape)

Warning message:

패키지 'reshape'는 R 버전 4.1.3에서 작성되었습니다

> library(caret)

> library(MASS)

> require(randomForest)

> require(gbm)

>

>

> pro_train <- read.csv('project_data_train.csv',header=T)

> pro_test <- read.csv('project_data_test.csv',header=T)

> #colSums(is.na(pro_train));colSums(is.na(pro_test))

>

> #train data

>

> pro_train <- as.data.frame(pro_train)
```

```
> id <- as.factor(pro_train[,1])
> gender <- as.factor(pro_train[,2])
> age <- as.factor(pro_train[,3])
> device <- as.factor(pro_train[,4])
> channel <- as.factor(pro_train[,5])
> period <- as.numeric(pro_train[,6])
> ani_regist <- as.factor(pro_train[,7])
> breeds <- as.factor(pro_train[,8])
> ani_gender <- as.factor(pro_train[,9])
> ani_age <- as.factor(pro_train[,10])
> ani_weight <- as.numeric(pro_train[,11])
> mkt_agree <- as.factor(pro_train[,12])
> push_agree <- as.factor(pro_train[,13])
> interest <- as.factor(pro_train[,14])
> coupon <- as.factor(pro_train[,15])
> payment <- as.factor(pro_train[,16])
>
> pro_train<-cbind(id,gender,age,device,channel,period,ani_regist,breeds,ani_gender,ani_age,ani_weight,mkt_agree,
+                  push_agree,interest,coupon,payment,pro_train[,-1:-16])
>
> pro_train_sr <- pro_train[,-1]
> pro_train_sr <- pro_train_sr[,-2]
> pro_train_sr <- pro_train_sr[,-9]
> pro_train_sr <- pro_train_sr[,-4]
> pro_train_sr <- pro_train_sr[,-5]
> pro_train_sr <- pro_train_sr[,-5]
>
> str(pro_train_sr)
```

```
'data.frame':	51901 obs. of  10 variables:
 $ gender    : Factor w/ 3 levels "1","2","3": 3 3 3 3 3 3 3 3 3 3 ...
 $ device    : Factor w/ 4 levels "1","2","3","4": 1 2 3 2 1 3 1 2 1 1 ...
 $ channel   : Factor w/ 5 levels "1","2","3","4",..: 4 5 1 5 5 4 5 2 1 4 ...
 $ ani_regist: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 2 1 ...
 $ ani_age   : Factor w/ 18 levels "0","1","2","3",..: 1 12 4 1 9 11 3 1 18 1 ...
 $ mkt_agree : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
 $ push_agree: Factor w/ 2 levels "0","1": 2 2 2 2 2 2 1 2 2 2 ...
 $ interest  : Factor w/ 2 levels "0","1": 2 2 2 2 2 1 2 1 1 1 ...
 $ coupon    : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 2 1 2 ...
 $ payment   : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 1 2 ...
> pro_test <- as.data.frame(pro_test)
> id <- as.factor(pro_test[,1])
> gender <- as.factor(pro_test[,2])
> age <- as.factor(pro_test[,3])
> device <- as.factor(pro_test[,4])
> channel <- as.factor(pro_test[,5])
> period <- as.numeric(pro_test[,6])
> ani_regist <- as.factor(pro_test[,7])
> breeds <- as.factor(pro_test[,8])
> ani_gender <- as.factor(pro_test[,9])
> ani_age <- as.factor(pro_test[,10])
> ani_weight <- as.numeric(pro_test[,11])
> mkt_agree <- as.factor(pro_test[,12])
> push_agree <- as.factor(pro_test[,13])
> interest <- as.factor(pro_test[,14])
> coupon <- as.factor(pro_test[,15])
> payment <- as.factor(pro_test[,16])
```

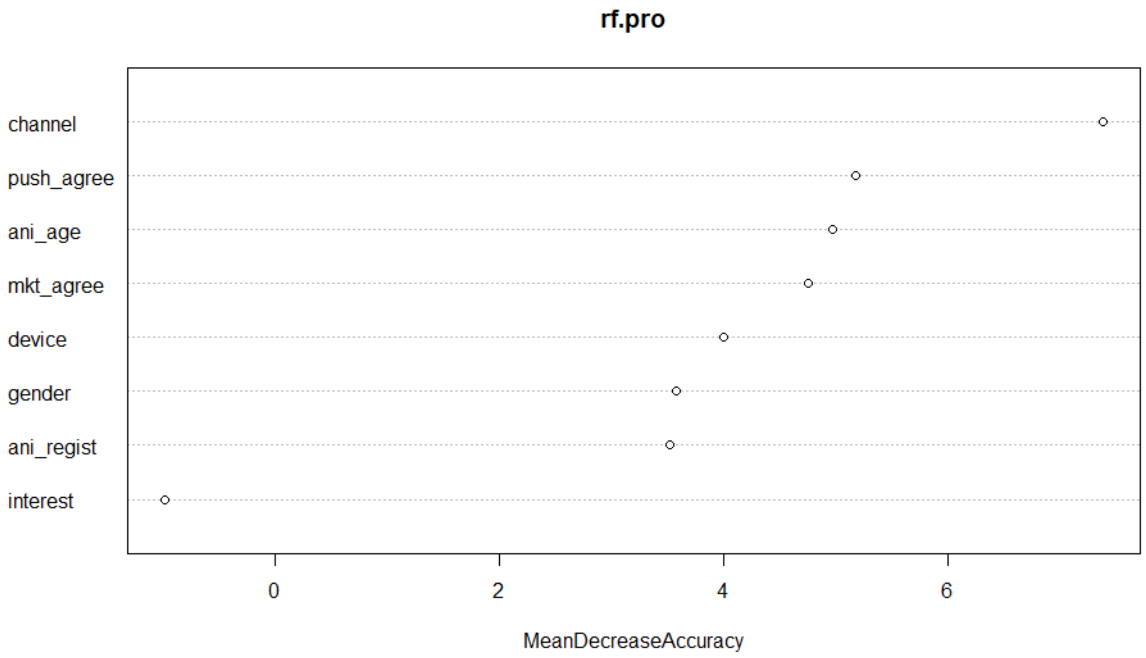
```
>
> pro_test<-cbind(id,gender,age,device,channel,period,ani_regist,breeds,ani_gender,ani_age,ani_weight,mkt_agree,
+                    push_agree,interest,coupon,payment,pro_test[,-1:-16])
>
> pro_test_sr <- pro_test[,-1]
> pro_test_sr <- pro_test_sr[,-2]
> pro_test_sr <- pro_test_sr[,-9]
> pro_test_sr <- pro_test_sr[,-4]
> pro_test_sr <- pro_test_sr[,-5]
> pro_test_sr <- pro_test_sr[,-5]
> str(pro_test_sr)
'data.frame':	34654 obs. of  10 variables:
 $ gender    : Factor w/ 3 levels "1","2","3": 3 3 3 3 3 3 3 3 3 3 ...
 $ device    : Factor w/ 4 levels "1","2","3","4": 2 2 2 2 2 1 2 2 2 2 ...
 $ channel   : Factor w/ 5 levels "1","2","3","4",..: 5 2 5 4 2 5 5 4 5 4 ...
 $ ani_regist: Factor w/ 2 levels "0","1": 1 2 2 2 2 2 2 2 2 2 ...
 $ ani_age   : Factor w/ 18 levels "0","1","2","3",..: 3 18 18 18 18 18 18 18 18 18 ...
 $ mkt_agree : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
 $ push_agree: Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
 $ interest  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ coupon    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ payment   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
> bagg.pro<-bagging(payment~.,
                   data=pro_train_sr,
                    nbag=100,
                    control=rpart.control(minsplit=10),
                    coob=T)
> bagg.pro
```

```
Bagging classification trees with 100 bootstrap replications

Call: bagging.data.frame(formula = payment ~ ., data = pro_train_sr,
    nbag = 100, control = rpart.control(minsplit = 10), coob = T)

Out-of-bag estimate of misclassification error:  0.0028

> bagg.predict <- predict(bagg.pro, pro_test_sr, type='class')
> confusionMatrix(bagg.predict,pro_test_sr$payment)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 32175    47
         1     0  2432

               Accuracy : 0.9986
                 95% CI : (0.9982, 0.999)
    No Information Rate : 0.9285
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9897

 Mcnemar's Test P-Value : 1.949e-11

            Sensitivity : 1.0000
            Specificity : 0.9810
```

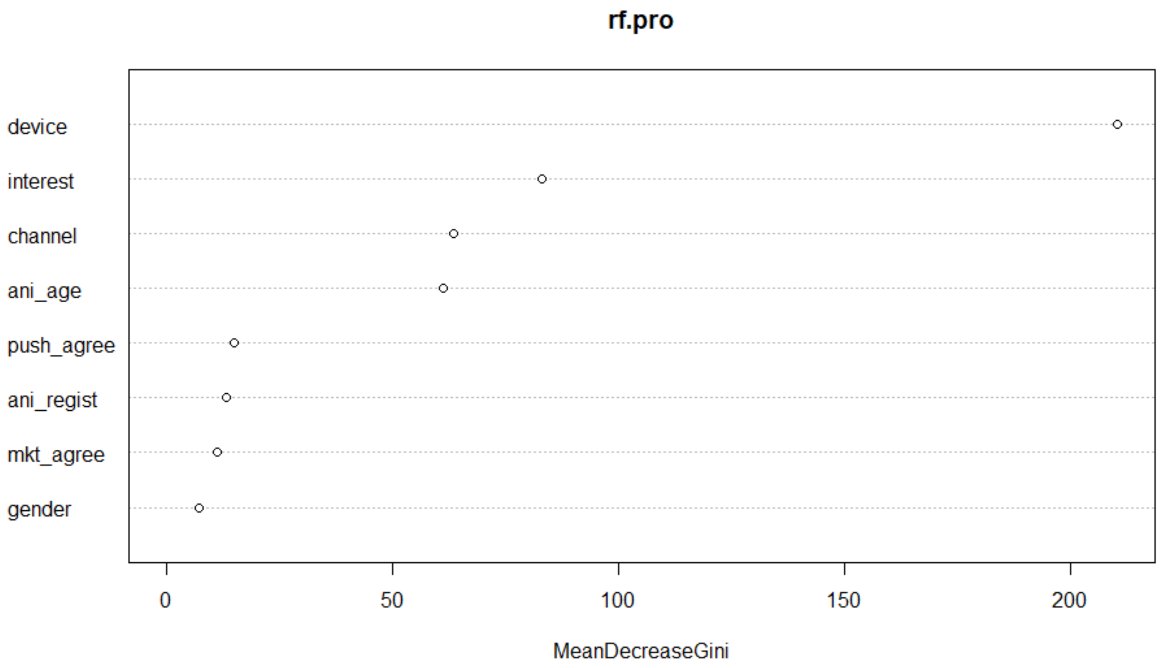
```
         Pos Pred Value : 0.9985
         Neg Pred Value : 1.0000
             Prevalence : 0.9285
         Detection Rate : 0.9285
   Detection Prevalence : 0.9298
      Balanced Accuracy : 0.9905

       'Positive' Class : 0

> pro_train_srs <- pro_train_sr[,-9] #coupon data 제거(편중됨)
>rf.pro <- randomForest(payment~.,
                      data=pro_train_srs,
                      importance=TRUE,
                      ntree=100,
                      mtry=2)
> rf.pro

Call:
 randomForest(formula = payment ~ ., data = pro_train_srs, importance = TRUE,      ntree = 100, mtry = 2)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 8.48%
Confusion matrix:
      0 1  class.error
0 47500 1 2.105219e-05
1  4400 0 1.000000e+00
```

```
> rf.predict <- predict(rf.pro, pro_test_sr, type='class')
> summary(rf.predict)
    0     1
34551   103
> confusionMatrix(rf.predict, pro_test_sr$payment)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 32089  2462
         1    86    17

               Accuracy : 0.9265
                 95% CI : (0.9237, 0.9292)
    No Information Rate : 0.9285
    P-Value [Acc > NIR] : 0.9258

                  Kappa : 0.0075

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.997327
            Specificity : 0.006858
         Pos Pred Value : 0.928743
         Neg Pred Value : 0.165049
             Prevalence : 0.928464
         Detection Rate : 0.925983
   Detection Prevalence : 0.997028
```

```
         Balanced Accuracy : 0.502092

          'Positive' Class : 0

> importance(rf.pro)
                    0          1 MeanDecreaseAccuracy MeanDecreaseGini
gender      -2.302447   6.935742            3.5851299         7.239799
device       3.387417   8.870006            4.0036601       210.194545
channel      7.529542  -5.321079            7.3858222        63.432384
ani_regist   3.896974  -5.003157            3.5276637        13.116524
ani_age      5.317141  -6.049824            4.9723044        61.269025
mkt_agree    4.723286  -4.428302            4.7570080        11.080905
push_agree   5.212757  -5.101233            5.1764056        15.033586
interest    -5.490202   8.401072           -0.9789458        82.882161
> importance(rf.pro, type=1)
           MeanDecreaseAccuracy
gender                3.5851299
device                4.0036601
channel               7.3858222
ani_regist            3.5276637
ani_age               4.9723044
mkt_agree             4.7570080
push_agree            5.1764056
interest             -0.9789458

> varImpPlot(rf.pro, type=1)
```



> varImpPlot(rf.pro, type=2)




```
>
> ##########gbm###########
>boost.pro <- gbm(payment~.,
                  data=pro_train_sr,
                  distribution="multinomial",
                   n.trees=1000,
                  shrinkage=0.01,
                  interaction.depth=4)
```

```
>boost.predict <- predict.gbm(object = boost.pro,

                                newdata = pro_test_sr,

                                n.trees = 1000,

                                type = "response")

> print(boost.predict)
 , , 1000

                  0            1
 [1,] 9.974173e-01 0.0025827496
 [2,] 9.989307e-01 0.0010693253
 [3,] 9.989427e-01 0.0010573455
 [4,] 9.990089e-01 0.0009911196
 [5,] 9.989307e-01 0.0010693253
 [6,] 9.973960e-01 0.0026040244
 [7,] 9.989427e-01 0.0010573455
 [8,] 9.990089e-01 0.0009911196
 [9,] 9.989427e-01 0.0010573455
[10,] 9.990089e-01 0.0009911196
[11,] 9.989307e-01 0.0010693253
[12,] 9.989307e-01 0.0010693253
[13,] 9.989307e-01 0.0010693253
[14,] 9.989307e-01 0.0010693253
[15,] 9.989912e-01 0.0010088418
[16,] 9.973531e-01 0.0026469214
[17,] 9.989307e-01 0.0010693253
[18,] 9.973960e-01 0.0026040244
[19,] 9.989427e-01 0.0010573455

> value <- apply(boost.predict, 1,which.max)

> value

  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [37] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1
 [73] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[109] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1
[145] 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 2 1 1 2 1 1 1 1 1
[181] 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2
[217] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
[253] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
[289] 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2
[325] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2
[361] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[397] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[433] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
[469] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
[505] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
> result = data.frame(pro_test_sr$payment, value)

> result
    pro_test_sr.payment value
1                     0     1
2                     0     1
3                     0     1
4                     0     1
5                     0     1
6                     0     1
7                     0     1
8                     0     1
9                     0     1
10                    0     1
11                    0     1
12                    0     1
13                    0     1
14                    0     1
15                    0     1
16                    0     1
17                    0     1

> with(result, table(pro_test_sr.payment, value))

                   value
pro_test_sr.payment     1     2
                  0 32057   118
                  1    47  2432

> print((32057+2432)/(32057+2432+118+47))

[1] 0.9952386

>
```