Ensemble 4th day

## 앙상블 기법이란

참고 링크

<https://ybeaning.tistory.com/17>

<http://blog.naver.com/PostView.nhn?blogId=960125_hds&logNo=221046292116&parentCategoryNo=&categoryNo=7&viewDate=&isShowPopularPosts=true&from=search>

<https://3months.tistory.com/215>

<https://apple-rbox.tistory.com/6>

<https://stackoverrun.com/ko/q/11907138>

동일한 학습 알고리즘을 사용해서 여러 모델을 학습하는 개념

앙상블 – 여러 개의 값을 하나로 합치는 방법..!

모델링 기법 중 머신러닝에서 가장 많이 사용되고 있는 기법으로 여러 개의 분류 모형에 의한 결과를 종합하여 분류의 정확도를 높이는 방법이다.

이 모형은 ‘집단 지성’ 이 ’단일 지성’보다 낫다는 아이디어에 기초..!

일반적으로 학습에서 나타나는 오류는

1. 지나친 치우침(bias)으로 인한 underfitting
2. 높은 분산으로 인한 overfitting

앙상블 모형은 여러 모형의 평균을 취함으로써 어느 쪽에도 치우치지 않는 결과를 얻을 수 있으며 여러 개의 모형의 의견을 취합함으로서 분산을 감소시킬 수 있다.

앙상블 모형의 대표적인 알고리즘에는 ’배깅(bagging)’과 ’부스팅(boosting)’이 있다..!

# Bagging이란

배깅이란, bootstrap(부트스트랩) aggregating의 약어로 **데이터를 가방(bag)에 쓸어 담아 복원추출하여 여러 개의 표본을 만들어 이를 기반으로 각각의 모델을 개발한 후에 하나로 합쳐 하나의 모델을 만들어 내는 것**이다.

배깅을 통해서 얻을 수 있는 효과는 ‘알고리즘의 안전성’이다. 단일 seed하나의 값을 기준으로 데이터를 추출하여 모델을 생성해 내는 것 보다,

여러 개의 다양한 표본을 사용함으로서 모델을 만드는 것이 모집단을 잘 대표할 수 있게 된다.

명목형 변수의 경우 투표 방식, 혹은 가장 높은 확률값으로 예측 결과값을 합치며 연속형 변수의 경우에는 평균으로 값을 집계한다

또한 배깅은 병렬처리를 사용할 수 있는데, 독립적인 데이터 셋으로 독립된 모델을 만들기 때문에 모델 생성에 있어서 매우 효율적이다.

대표적인 예시로는 RandomForest기법을 들 수 있다.

# Boosting이란

배깅은 일반적인 모델을 만드는 것을 의미했다면, 부스팅은 틀린 문제를 노트에 적고 이것들에 집중을 하는 목적의 오답 노트와 비슷한 개념

즉, 틀린 케이스에 가중치를 줌으로써 이를 해결하는 것에 초점을 맞춘 모델

부스팅을 하는 목적은 ‘정확성을 향상’시키기 위함. 앞서 만든 모델이 오분류로 인한 케이스에 더 높은 가중치를 부여함으로써 이를 더 잘 해결할 수 있는 모델이 되도록 모델을 수정해나간다.

배깅이 병렬적으로 학습한다면 부스팅은 순차적으로 모델을 만든다 이로써 정확도가 더 높아질 수 있지만 그만큼 이상치에 취약해질 수 있게 된다.

대표적인 알고리즘에는 AdaBoost, xgBoost, GBM, C5.0 등이 있다. 이 중 가장 많이 사용하는 것은 xgBoost이다.

부스팅 기법 중 하나인 AdaBoost(회기 모형에 대한나 부스팅 기법)의 방법을 살펴보면 다음과 같다.

1. 분석 표본에 모형을 적합시켜 예측치와 오차를 산출
2. 오차가 큰 개체에 대해서는 큰 가중치를, 오차가 작은 개체에는 작은 가중치 부여
3. 변경된 가중치로 모형을 재적합하고 1)단계부터 실행(일정 횟수 반복)
4. 최종적으로 이제까지 도출된 모형을 종합 — 집계 방법 (Weighted Median – 가중 중간값)

대표적인 예시로는 xgboost를 살펴보자…!

우선 배깅의 논리를 decision tree로 직접 구현해보는 연습을 해본다.

## Classification

# 1. Random Forest

credit\_bg <- credit\_train  
str(credit\_bg)

## 'data.frame': 25000 obs. of 24 variables:  
## $ LIMIT\_BAL : int 500000 50000 230000 200000 240000 200000 310000 200000 230000 200000 ...  
## $ SEX : int 1 1 2 2 2 2 1 1 1 2 ...  
## $ EDUCATION : int 1 3 3 1 2 1 1 1 2 2 ...  
## $ MARRIAGE : int 2 2 1 1 2 1 1 2 1 1 ...  
## $ AGE : int 38 37 37 59 30 38 43 27 51 49 ...  
## $ PAY\_1 : int -2 2 0 0 0 2 -1 0 2 1 ...  
## $ PAY\_2 : int -2 2 0 0 0 -1 2 0 0 -2 ...  
## $ PAY\_3 : int -2 2 0 0 0 2 -1 -1 0 -2 ...  
## $ PAY\_4 : int -2 3 0 0 0 2 -1 -1 0 -2 ...  
## $ PAY\_5 : int -2 2 0 0 0 -2 -1 -1 0 -2 ...  
## $ PAY\_6 : int -2 2 0 0 0 -1 -1 -1 0 -2 ...  
## $ BILL\_AMT1 : int 0 46004 225800 92215 226315 166 2432 58725 204643 -5 ...  
## $ BILL\_AMT2 : int 0 45976 228342 94085 231582 1694 261 76287 197993 728 ...  
## $ BILL\_AMT3 : int 0 48953 189966 96174 226909 1519 2745 22869 84076 0 ...  
## $ BILL\_AMT4 : int 8086 48851 142377 98116 135626 0 -626 30268 77766 0 ...  
## $ BILL\_AMT5 : int 0 49318 143828 99490 138520 0 110 54466 70066 0 ...  
## $ BILL\_AMT6 : int 0 51143 143698 101412 141280 600 -154 37127 62865 0 ...  
## $ PAY\_AMT1 : int 0 1000 8440 4300 9100 1694 0 30229 5805 733 ...  
## $ PAY\_AMT2 : int 0 4035 6100 4500 9859 0 4000 22946 2965 0 ...  
## $ PAY\_AMT3 : int 8086 1000 7000 4438 4147 0 0 30684 3500 0 ...  
## $ PAY\_AMT4 : int 0 1400 5356 3900 3691 0 1000 54589 2600 0 ...  
## $ PAY\_AMT5 : int 0 2800 5400 3800 3700 600 0 37208 2500 0 ...  
## $ PAY\_AMT6 : int 0 0 5200 3500 3729 0 0 55031 1901 0 ...  
## $ default.payment.next.month: int 0 1 0 0 0 1 0 0 0 1 ...

보험사나 은행 같은 금융기관에서 해당 고객이 금액을 대출 받고 상환한 금액과 사용 금액들에 대한 정보가 나와있고

채무 불이행의 여부를 예측하는 모델이다.

library(caret)

## Warning: package 'caret' was built under R version 3.6.3

# caret 패키지의 createDataPartition 함수를 통해 train, test데이터를 나눈다.  
bg\_indx <- createDataPartition(credit\_bg$default.payment.next.month, p=.8, list=F)  
credit\_bg1 <- credit\_bg[bg\_indx, ]  
credit\_bg2 <- credit\_bg[-bg\_indx, ]

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.6.2

set.seed(2020)  
rf\_credit <- randomForest(default.payment.next.month~., ntree= 500, mtry=5, credit\_bg1, importance=T)  
rf\_credit

##   
## Call:  
## randomForest(formula = default.payment.next.month ~ ., data = credit\_bg1, ntree = 500, mtry = 5, importance = T)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 5  
##   
## OOB estimate of error rate: 18.29%  
## Confusion matrix:  
## 0 1 class.error  
## 0 14675 893 0.05736125  
## 1 2765 1667 0.62387184

parameter에 대한 설명

**mtry**는 **각각의 tree마다 몇 개의 feature를 사용할 것인지**를 정하는 것. 이는 각 트리의 랜덤성을 결정, 클수록 과적합을 줄인다.

**mtry == max\_features**

mtry값이 큰 경우 : 모든 특성을 고려하므로 피처 선택에 무작위성이 들어가지 않는다. 하지만 부트스트랩 샘플링으로 인한 무작위성은 그대로 유지된다. 따라서 각 트리들이 비슷해지고 가장 두드러진 특성을 이용해 데이터를 맞출 것이다.

mtry값이 작은 경우 : 트리의 분기에서 선택되는 피처가 적어진다. 1로 설정할 경우 테스트할 특성을 고를 필요가 없게 되며 그냥 무작위로 선택된 특성의 임계값만을 찾는다 따라서 각 트리들이 많이 달라지고 각 트리를 데이터에 맞추기 위해서 깊이가 깊어지게 된다.

regression의 경우 (변수갯수)/3 or (변수갯수)자체??, classification의 경우 sqrt(변수 갯수)

**ntree**는 \*\*tree의 총 갯수\*8를 의미 – 너무 많이하면 overfit, 너무 적게 하면 underfit

**importance**는 **변수의 중요도를 측정할 것인지**를 의미

**replace**는 복원 추출을 할 것인지를 의미

이외에도 많은 피처들이 있다. 자세한 건 ?randomForest()을 통해 확인해보자

이제 이를 predict함수를 이용하여 예측값을 만들고

혼동 행렬을 구해보면

pred\_credit <- predict(rf\_credit, credit\_bg1)  
table(pred = pred\_credit, actual = credit\_bg1$default.payment.next.month)

## actual  
## pred 0 1  
## 0 15559 87  
## 1 9 4345

pred\_credit2 <- predict(rf\_credit, credit\_bg2)  
table(pred= pred\_credit2, actual = credit\_bg2$default.payment.next.month)

## actual  
## pred 0 1  
## 0 3682 693  
## 1 210 415

AUC를 구해보면

library(ROCR)

## Warning: package 'ROCR' was built under R version 3.6.2

## Warning: package 'gplots' was built under R version 3.6.2

calc\_AUC <- function(model, new\_data, new\_label){  
 pred <- predict(model, newdata=new\_data, type='prob')[,2]  
 p <- prediction(pred, new\_label)  
 auc <- performance(p, 'auc')  
 return(auc@y.values[[1]])  
}

calc\_AUC(rf\_credit, credit\_bg1, credit\_bg1$default.payment.next.month)

## [1] 0.9998146

calc\_AUC(rf\_credit, credit\_bg2, credit\_bg2$default.payment.next.month)

## [1] 0.7695402

로지스틱과의 성능을 비교해보면

credit\_glm <- glm(default.payment.next.month~., credit\_bg1, family='binomial')  
pred <- predict(credit\_glm, credit\_bg2)  
p\_g <- prediction(pred, credit\_bg2$default.payment.next.month)  
aucss <- performance(p\_g, 'auc')  
aucss@y.values[[1]]

## [1] 0.7244696

**ntree의 수를 바꾸어서 실험해보자**

set.seed(2020)  
another\_rf1 <- randomForest(default.payment.next.month~., ntree=[ ], mtry=5, credit\_bg1, importance=T)  
calc\_AUC(another\_rf1, credit\_bg2, credit\_bg2$default.payment.next.month)

**mtry의 수를 바꾸어서 실험해보자** {r. eval=F}

set.seed(2020)  
another\_rf2 <- randomForest(default.payment.next.month~., ntree=500, mtry=[ ], credit\_bg1, importance=T)  
calc\_AUC(another\_rf2, credit\_bg2, credit\_bg2$default.payment.next.month)

**hyper parameter tuning**

그렇다면, mtry는 몇으로, ntree는 몇으로 하는 게 가장 성능이 좋을까?

이 것에 대해서는 데이터마다 다르고 seed값이 무엇이냐에 따라 다 다르기 때문에 매우 다양한데 기본적으로

grid 서치를 통해 많이 탐색을 한다

하지만 이것의 경우 시간이 오래 걸리므로 이번 캠프 시간에는 실행은 하지 않고 코드를 참고해서 학습만 진행한다.

fitContrl <- trainControl(method='repeatedcv', number=2, repeats=2, search='grid')  
grids <- expand.grid(.mtry=c(2:8))  
modellists1 <- list()  
for (ntrees in c(100,300,500,700)){  
 set.seed(2020)  
 rf\_fit <- train(default.payment.next.month~., data=credit\_bg1, method='rf', metric = 'Accuracy', tuneGrid = grids, trControl=fitContrl,ntree=ntrees,verbose=T)  
 key <- toString(ntrees)  
 modellists1[[key]] <- rf\_fit  
}

set.seed(2020)  
## 보다 나은 모델을 만들 수 있다...!  
## ntree 300 mtry 3으로  
library(randomForest)  
rf\_credit\_better <- randomForest(default.payment.next.month~., credit\_bg1, ntree=300, mtry=3, importance=T)  
pred\_credit\_better <- predict(rf\_credit\_better, credit\_bg2)  
table(pred= pred\_credit\_better, actual = credit\_bg2$default.payment.next.month)

## actual  
## pred 0 1  
## 0 3687 709  
## 1 205 399

calc\_AUC(rf\_credit\_better, credit\_bg1, credit\_bg1$default.payment.next.month)

## [1] 0.9986472

calc\_AUC(rf\_credit\_better, credit\_bg2, credit\_bg2$default.payment.next.month)

## [1] 0.7722746

# 2. C5.0

randomforest와 xgboost와 달리 **분류 모델에만** 사용 가능!

여기는 설명 변수에 factor형 변수가 없지만, 실제로 factor형 변수의 경우 levels 통일 해주어야하고

"" 이러한 levels 명이 되어 있는 거는 “missing”으로 치환해서 해주어야 하는 부분도 있음

**장점**

모든 문제에 적합한 분류기

결측치, 명목속성, 수치를 처리할수 있는 자동성이 높은 학습

가장 중요한 속성만사용

매우 많은 수 또는 상대적으로 적은 훈련 예제와 데이터 사용

수학적 배경 없이도 해석할 수 있는 모델 도출

다른 복잡한 모델보다 높은 효율

가지치기의 최적화…! – 자동으로 합리적인 기본 값을 사용해 많은 결정을 다룬다..

**단점**

결정 트리는 다수의 레벨을 가진 속성 쪽으로 구분하는 경향이 있음

모델이 쉽게 과적합화나 과소적합화됨

훈련 데이터에 대한 약간의 변경이 결정 논리에 큰 변화를 줌

큰 트리는 이해하기가 어렵고 직관적이지 않음

C5.0의 경우

먼저 훈련 데이터에 과적합해 트리는 크게 성장한다.

이후에 분류 오차를 가진 노드와 가지를 제거한다. 일부 경우에 전체 가지는 좀 더 단순한 결정으로 교체되거나 옮겨진다.

이러한 변경되는 과정을 부분 트리 생성, 부분 트리 대체라고 한다.

데이터 프레임을 새로 구축해본다

library(C50)

## Warning: package 'C50' was built under R version 3.6.3

새로운 데이터프레임 구축

credit\_c50 <- credit\_train  
  
credit\_c50$default.payment.next.month <- as.factor(credit\_c50$default.payment.next.month)  
  
c50\_idx <- createDataPartition(credit\_c50$default.payment.next.month, p=.8, list=F)  
  
credit\_c50\_train <- credit\_c50[c50\_idx, ]  
credit\_c50\_test <- credit\_c50[-c50\_idx,]

1번 수행한 경우

set.seed(2020)  
basic\_model <- C5.0(credit\_c50\_train[,-24], credit\_c50\_train$default.payment.next.month, trials = 1)  
# C5.0(train, class, trials=1, costs=NULL)  
# train : 훈련 데이터 프레임  
# class : 훈련 데이터의 각 행에 대해 범주를 가진 팩터 벡터  
# trial : 부스팅 반복의 수를 조절하기 위한 선택적 숫자  
# costs : 오차의 타입에 관련된 cost를 명시한 선택적 매트릭스

basic\_model

##   
## Call:  
## C5.0.default(x = credit\_c50\_train[, -24], y  
## = credit\_c50\_train$default.payment.next.month, trials = 1)  
##   
## Classification Tree  
## Number of samples: 20000   
## Number of predictors: 23   
##   
## Tree size: 7   
##   
## Non-standard options: attempt to group attributes

summary(basic\_model)

##   
## Call:  
## C5.0.default(x = credit\_c50\_train[, -24], y  
## = credit\_c50\_train$default.payment.next.month, trials = 1)  
##   
##   
## C5.0 [Release 2.07 GPL Edition] Sun Jul 26 18:28:11 2020  
## -------------------------------  
##   
## Class specified by attribute `outcome'  
##   
## Read 20000 cases (24 attributes) from undefined.data  
##   
## Decision tree:  
##   
## PAY\_1 > 1:  
## :...PAY\_3 > -1: 1 (2018/569)  
## : PAY\_3 <= -1:  
## : :...BILL\_AMT1 <= 2480: 0 (83/29)  
## : BILL\_AMT1 > 2480: 1 (49/15)  
## PAY\_1 <= 1:  
## :...PAY\_2 <= 0: 0 (16361/2304)  
## PAY\_2 > 0:  
## :...PAY\_5 <= 0: 0 (1051/387)  
## PAY\_5 > 0:  
## :...PAY\_6 <= 0: 0 (120/52)  
## PAY\_6 > 0: 1 (318/141)  
##   
##   
## Evaluation on training data (20000 cases):  
##   
## Decision Tree   
## ----------------   
## Size Errors   
##   
## 7 3497(17.5%) <<  
##   
##   
## (a) (b) <-classified as  
## ---- ----  
## 14843 725 (a): class 0  
## 2772 1660 (b): class 1  
##   
##   
## Attribute usage:  
##   
## 100.00% PAY\_1  
## 89.25% PAY\_2  
## 10.75% PAY\_3  
## 7.45% PAY\_5  
## 2.19% PAY\_6  
## 0.66% BILL\_AMT1  
##   
##   
## Time: 1.1 secs

c50\_pred1 <- predict(basic\_model, newdata=credit\_c50\_train)  
c50\_pred2 <- predict(basic\_model, newdata = credit\_c50\_test)  
table(pred=c50\_pred1, actual = credit\_c50\_train$default.payment.next.month)

## actual  
## pred 0 1  
## 0 14843 2772  
## 1 725 1660

table(pred=c50\_pred2, actual = credit\_c50\_test$default.payment.next.month)

## actual  
## pred 0 1  
## 0 3692 720  
## 1 200 388

c50\_prob1 <- predict(basic\_model, newdata=credit\_c50\_train, type='prob')  
c50\_prob2 <- predict(basic\_model, newdata=credit\_c50\_test, type='prob')  
  
calc\_AUC(basic\_model, credit\_c50\_train, credit\_c50\_train$default.payment.next.month)

## [1] 0.6994005

calc\_AUC(basic\_model, credit\_c50\_test, credit\_c50\_test$default.payment.next.month)

## [1] 0.693208

C5.0의 성능을 향상시킬수는 없을까..?

C5.0에 부스팅을 추가하는 방법은 trials 매개변수를 활용

알고리즘은 트리가 추가적인 trial이 정확도를 향상시키지 못할 것으로 보이면 트리 추가를 멈춘다.

또한 seed값을 재조정하는 방법도 있다.

seed값의 경우 123으로 다시 조정하여 수행해보았다.

set.seed(123)  
better\_model <- C5.0(credit\_c50\_train[,-24], credit\_c50\_train$default.payment.next.month, trials = 10)  
summary(better\_model)

##   
## Call:  
## C5.0.default(x = credit\_c50\_train[, -24], y  
## = credit\_c50\_train$default.payment.next.month, trials = 10)  
##   
##   
## C5.0 [Release 2.07 GPL Edition] Sun Jul 26 18:28:22 2020  
## -------------------------------  
##   
## Class specified by attribute `outcome'  
##   
## Read 20000 cases (24 attributes) from undefined.data  
##   
## ----- Trial 0: -----  
##   
## Decision tree:  
##   
## PAY\_1 > 1:  
## :...PAY\_3 > -1: 1 (2018/569)  
## : PAY\_3 <= -1:  
## : :...BILL\_AMT1 <= 2480: 0 (83/29)  
## : BILL\_AMT1 > 2480: 1 (49/15)  
## PAY\_1 <= 1:  
## :...PAY\_2 <= 0: 0 (16361/2304)  
## PAY\_2 > 0:  
## :...PAY\_5 <= 0: 0 (1051/387)  
## PAY\_5 > 0:  
## :...PAY\_6 <= 0: 0 (120/52)  
## PAY\_6 > 0: 1 (318/141)  
##   
## ----- Trial 1: -----  
##   
## Decision tree:  
##   
## PAY\_2 <= 0: 0 (16447.9/4801.9)  
## PAY\_2 > 0: 1 (3552.1/1671.7)  
##   
## ----- Trial 2: -----  
##   
## Decision tree:  
##   
## EDUCATION > 3: 0 (261.7/47.9)  
## EDUCATION <= 3:  
## :...PAY\_AMT3 > 676: 0 (13401.9/4521.5)  
## PAY\_AMT3 <= 676:  
## :...PAY\_6 > 6: 1 (30.1/7.1)  
## PAY\_6 <= 6:  
## :...PAY\_AMT6 > 5002: 0 (485.7/157.7)  
## PAY\_AMT6 <= 5002:  
## :...BILL\_AMT1 <= 511: 1 (1991.4/849.9)  
## BILL\_AMT1 > 511: 0 (3829.1/1738.3)  
##   
## ----- Trial 3: -----  
##   
## Decision tree:  
##   
## PAY\_1 <= 0:  
## :...PAY\_3 <= 1: 0 (12883/4678.2)  
## : PAY\_3 > 1: 1 (1137/451.5)  
## PAY\_1 > 0:  
## :...EDUCATION > 3: 0 (52.2/9.4)  
## EDUCATION <= 3:  
## :...BILL\_AMT5 > 320407: 1 (43.7/13.8)  
## BILL\_AMT5 <= 320407:  
## :...PAY\_AMT5 > 10073: 0 (210.4/85.5)  
## PAY\_AMT5 <= 10073:  
## :...PAY\_1 <= 1: 0 (2996.2/1440.2)  
## PAY\_1 > 1: 1 (2677.5/1075.9)  
##   
## ----- Trial 4: -----  
##   
## Decision tree:  
##   
## EDUCATION > 3: 0 (235.5/61)  
## EDUCATION <= 3:  
## :...PAY\_5 > 0: 1 (2644.8/1118.9)  
## PAY\_5 <= 0:  
## :...PAY\_6 > 2: 1 (37.3/6.9)  
## PAY\_6 <= 2:  
## :...PAY\_AMT1 > 2984:  
## :...PAY\_1 <= 1: 0 (5928.6/2020.7)  
## : PAY\_1 > 1: 1 (611.5/286.5)  
## PAY\_AMT1 <= 2984:  
## :...PAY\_AMT6 > 860: 0 (4759.5/2086.9)  
## PAY\_AMT6 <= 860:  
## :...PAY\_AMT5 > 19011: 0 (46/9.5)  
## PAY\_AMT5 <= 19011:  
## :...PAY\_4 > 1: 1 (368.2/142.9)  
## PAY\_4 <= 1:  
## :...PAY\_AMT4 > 7043: 0 (156.7/53.3)  
## PAY\_AMT4 <= 7043:  
## :...LIMIT\_BAL <= 120000: 1 (2834.5/1247)  
## LIMIT\_BAL > 120000: 0 (2377.3/1164.4)  
##   
## ----- Trial 5: -----  
##   
## Decision tree:  
##   
## EDUCATION > 3: 0 (226.4/65.3)  
## EDUCATION <= 3:  
## :...PAY\_AMT1 > 4550: 0 (4759/1788.2)  
## PAY\_AMT1 <= 4550:  
## :...PAY\_1 > 0: 1 (5266.2/2418.7)  
## PAY\_1 <= 0:  
## :...PAY\_4 > 1: 1 (816.8/328.6)  
## PAY\_4 <= 1:  
## :...BILL\_AMT5 > 43883: 0 (1263.6/420.6)  
## BILL\_AMT5 <= 43883:  
## :...BILL\_AMT1 <= 8341: 1 (3943/1941)  
## BILL\_AMT1 > 8341: 0 (3724.9/1582.5)  
##   
## ----- Trial 6: -----  
##   
## Decision tree:  
##   
## EDUCATION > 3: 0 (171/15.5)  
## EDUCATION <= 3:  
## :...PAY\_1 > 0: 0 (6701.5/3193)  
## PAY\_1 <= 0:  
## :...PAY\_5 <= 0: 0 (11397.7/3237)  
## PAY\_5 > 0: 1 (968.8/446.4)  
##   
## ----- Trial 7: -----  
##   
## Decision tree:  
##   
## PAY\_1 <= 0: 0 (10463.2/2063.1)  
## PAY\_1 > 0: 1 (8035.8/3245.2)  
##   
## ----- Trial 8: -----  
##   
## Decision tree:  
##   
## PAY\_1 <= 0:  
## :...PAY\_2 <= 1: 0 (7770.4/593.7)  
## : PAY\_2 > 1: 1 (519.5/229.3)  
## PAY\_1 > 0:  
## :...EDUCATION > 3: 0 (79.5)  
## EDUCATION <= 3:  
## :...PAY\_1 <= 1: 0 (5693.4/1853.6)  
## PAY\_1 > 1: 1 (3757.1/1389.1)  
##   
## ----- Trial 9: -----  
##   
## Decision tree:  
##   
## PAY\_1 <= 0: 0 (7584.9/132.4)  
## PAY\_1 > 0:  
## :...PAY\_2 <= -1: 0 (2231.3/210.6)  
## PAY\_2 > -1:  
## :...EDUCATION > 3: 0 (44.2)  
## EDUCATION <= 3:  
## :...PAY\_1 > 2: 1 (297.6/9.3)  
## PAY\_1 <= 2:  
## :...PAY\_AMT5 <= 10078: 1 (6385.3/2319.5)  
## PAY\_AMT5 > 10078: 0 (413.6/111.5)  
##   
##   
## Evaluation on training data (20000 cases):  
##   
## Trial Decision Tree   
## ----- ----------------   
## Size Errors   
##   
## 0 7 3497(17.5%)  
## 1 2 4026(20.1%)  
## 2 6 5105(25.5%)  
## 3 7 3909(19.5%)  
## 4 11 5085(25.4%)  
## 5 7 6733(33.7%)  
## 6 4 4673(23.4%)  
## 7 2 4345(21.7%)  
## 8 5 3620(18.1%)  
## 9 6 3727(18.6%)  
## boost 3491(17.5%) <<  
##   
##   
## (a) (b) <-classified as  
## ---- ----  
## 14842 726 (a): class 0  
## 2765 1667 (b): class 1  
##   
##   
## Attribute usage:  
##   
## 100.00% EDUCATION  
## 100.00% PAY\_1  
## 100.00% PAY\_2  
## 98.54% PAY\_5  
## 98.51% PAY\_AMT1  
## 98.51% PAY\_AMT3  
## 93.13% PAY\_6  
## 88.07% PAY\_3  
## 70.11% BILL\_AMT5  
## 57.81% PAY\_AMT6  
## 57.58% PAY\_4  
## 52.62% BILL\_AMT1  
## 40.76% PAY\_AMT5  
## 23.73% PAY\_AMT4  
## 22.86% LIMIT\_BAL  
##   
##   
## Time: 4.8 secs

c50\_better1\_prob <- predict(better\_model, newdata=credit\_c50\_train, type='prob')  
c50\_better2\_prob <- predict(better\_model, newdata=credit\_c50\_test, type='prob')  
  
calc\_AUC(better\_model, credit\_c50\_train, credit\_c50\_train$default.payment.next.month)

## [1] 0.7727951

calc\_AUC(better\_model, credit\_c50\_test, credit\_c50\_test$default.payment.next.month)

## [1] 0.7627686

성능이 조금 더 개선됨을 확인할 수 있다.

**다른 trial시도로도 한번 테스트해보자**

set.seed(123)

another\_model <- C5.0(credit\_c50\_train[,-24], credit\_c50\_train$default.payment.next.month, trials=[ ])

그렇다면 trials를 일정량 늘려 AUC가 trials의 변화에 따라 어떻게 달라지는지 확인하기 위한 그래프를 그리는 함수를 정의해보자

시각화를 통한 최적의 trials 갯수 확인해보기

여기도 다소 시간이 오래걸려 굳이 수행하지는 않는다.

get\_c50\_auc <- function(train,target\_tr,test,target\_te){  
 auc\_train <- c()  
 auc\_test <- c()  
 for(i in 1:30){  
 set.seed(123)  
 model <- C5.0(train,target\_tr,trials=i)  
 prob1 <- predict(model, newdata=train, type='prob')[,2]  
 prob2 <- predict(model, newdata=test, type='prob')[,2]  
 auc\_tr <- performance(prediction(prob1,target\_tr),'auc')  
 auc\_te <- performance(prediction(prob2,target\_te),'auc')  
 auc\_train <- c(auc\_train, auc\_tr@y.values[[1]])  
 auc\_test <- c(auc\_test, auc\_te@y.values[[1]])  
 }  
 df = data.frame(trials=1:30, auc\_train = auc\_train, auc\_test = auc\_test)  
 library(tidyr)  
 library(dplyr)  
 df %>% gather(key=type,value=score,-trials) %>% ggplot(aes(x=trials,y=score,color=type)) + geom\_line()  
}

get\_c50\_auc(credit\_c50\_train[,-24],credit\_bg1$default.payment.next.month, credit\_c50\_test[,-24], credit\_bg2$default.payment.next.month)

## Warning: package 'tidyr' was built under R version 3.6.3

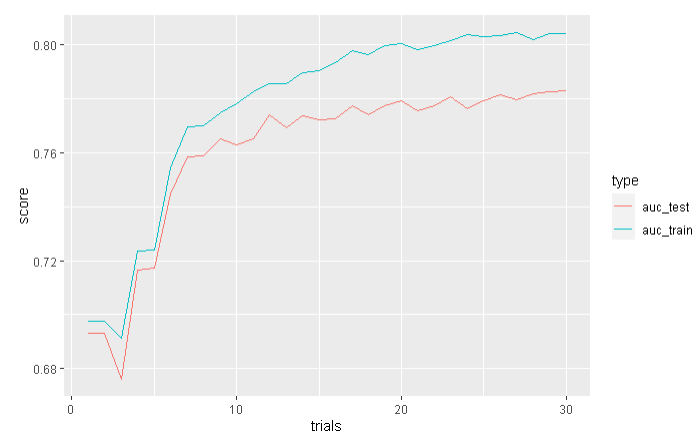
## Warning: package 'dplyr' was built under R version 3.6.3

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':  
##   
## combine

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

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



set.seed(123)  
best\_model <- C5.0(credit\_c50\_train[,-24], credit\_c50\_train$default.payment.next.month, trials = 28)  
  
calc\_AUC(best\_model, credit\_c50\_train, credit\_c50\_train$default.payment.next.month)

## [1] 0.7997104

calc\_AUC(best\_model, credit\_c50\_test, credit\_c50\_test$default.payment.next.month)

## [1] 0.7816783

# 3. Xgboost

library(xgboost)

## Warning: package 'xgboost' was built under R version 3.6.3

중요 파라미터들

**nrounds** = 최대 반복수

**eta** : learning rate default=0.3 [0,1]

Step size shrinkage로 학습 단계별로 가중치를 얼마나 적용할 지 결정하는 숫자, 가중치이므로 0~1 사이의 값을 지니며, 낮을 수록 more conservative

**gamma** : min split loss, default=0, [0,] gamma가 커질수록 보수적인 모델 information gain은 가지치기를 할 때 타겟변수에 대해 얼마나 설명할 수 있는 지표인데 이에 패널티를 부여하는 숫자가 바로 감마

gamma가 크다는 것은 가지치기를 잘 만들려 하지 않음을 의미하게 됨.

**max\_depth**: default=6, [0,] 말 그대로 의사결정나무의 깊이의 한도 깊이가 깊을수록 모델은 더 복잡해지고 overfitting의 문제를 야기할 수 있음

**subsample** : default=1, (0,1] training 데이터셋에서 subset를 만들지 전부를 사용할 지를 정하는 파라미터, 매번 나무를 만들때 적용하며 overfitting문제를 방지하려고 사용 각각의 트리를 만들때 데이터에서 사용할 행(row)의 비율(0~1)로도 볼 수 있다.

**colsample\_bytree** : default=1, (0,1] 나무를 만들때 칼럼, 즉 변수를 샘플링해서 쓸지에 대한 파라미터

나무를 만들기 전에 한 번 샘플링을 하게 됨

colsample\_bytree : 각각의 트리를 만들때 데이터에서 사용할 열의 비울(0~1)로도 볼 수 있다.

str(credit\_bg1) 현재 데이터에는 특별히 factor형 변수가 없지만 있는 경우는

**반드시 one - hot - encoding 과정을 거쳐야 함**을 명심!!! —- 이를 꼭 언급하면 좋을듯

**data.matrix()** 함수를 사용할 경우 이를 자동으로 처리해줌!

xgboost의 경우 **매트릭스 데이터를 이용**하므로 그에 맞게 형을 변형해준다!

library(dplyr)

set.seed(2020)  
# 데이터를 매트릭스 형태로 바꾸어줌  
x\_credit <- credit\_bg1 %>% select(-default.payment.next.month) %>% data.matrix  
y\_credit <- credit\_bg1$default.payment.next.month  
  
x\_credit2 <- credit\_bg2 %>% select(-default.payment.next.month) %>% data.matrix  
y\_credit2 <- credit\_bg2$default.payment.next.month

#y데이터는 xgboost는 0부터 시작하는 숫자 형태로 받아들이기 때문에 다음과 같은 전처리를 요한다.  
head(as.numeric(y\_credit),10)

## [1] 2 1 1 1 2 1 1 1 2 2

xg\_credit <- xgboost(data=x\_credit, label= as.numeric(y\_credit)-1, num\_class=2, nrounds=200,objective = "multi:softprob",eval\_metric = 'mlogloss', eta=0.001, max\_depth=5, gamma=3, prediction=T, verbose=F)

## [18:30:58] WARNING: amalgamation/../src/learner.cc:480:   
## Parameters: { prediction } might not be used.  
##   
## This may not be accurate due to some parameters are only used in language bindings but  
## passed down to XGBoost core. Or some parameters are not used but slip through this  
## verification. Please open an issue if you find above cases.

예측한 결과 값을 한번 확인해보면 다음과 같은 특징을 갖는다.

num\_class가 2이므로 0일떄의 예측확률, 1일때의 예측확률을 다 가지고 있음

pred\_credit4 <- predict(xg\_credit, newdata=x\_credit2)  
length(pred\_credit4)

## [1] 10000

nrow(credit\_bg2)

## [1] 5000

head(pred\_credit4,10)

## [1] 0.5669202 0.4330798 0.5427285 0.4572714 0.5394623 0.4605377 0.5573795  
## [8] 0.4426205 0.5573795 0.4426205

1로 예측한 경우의 확률만 가져오는 함수를 만듦

get\_pred <- function(vec){  
 pred <- c()  
 for(i in 1:length(vec)){  
 if((i%%2)==0){  
 pred <- c(pred, vec[i])  
 }  
 }  
 return(pred)  
}

pred\_credit4 <- get\_pred(pred\_credit4)  
# 0 과 1인 경우의 확률값의 평균은 다음과 같다.  
tapply(pred\_credit4, y\_credit2, mean)

## 0 1   
## 0.4408809 0.4790193

# 0.46을 대강의 threshold로 잡는다.  
pred\_logic <- ifelse(pred\_credit4>0.46,1,0)  
table(pred=pred\_logic, actual=y\_credit2)

## actual  
## pred 0 1  
## 0 3446 556  
## 1 446 552

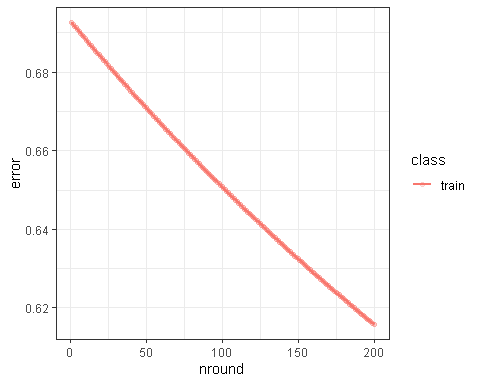
# AUC를 구해본다.  
pred\_xg <- prediction(pred\_credit4, y\_credit2)  
auc\_xg <- performance(pred\_xg, 'auc')  
auc\_xg@y.values[[1]]

## [1] 0.76991

nrounds가 변할 때마다 훈련데이터에서는 에러값이 준다.

하지만 그렇다고 nrounds를 계속해서 늘리는 것은 과적합의 우려를 야기할 수 있다.

cvplot = function(model){  
 eval.log = model$evaluation\_log  
 data.frame(error=c(unlist(eval.log[,2])),  
 class = c(rep('train',nrow(eval.log))),  
 nround=c(rep(1:nrow(eval.log),2))  
 ) %>% ggplot(aes(nround,error,col=class))+geom\_point(alpha=0.2)+geom\_smooth(alpha=0.4, se=F)+theme\_bw()  
}  
  
cvplot(xg\_credit)



**nrounds, eta, max\_depth, gamma**값을 바꾸어가며 AUC 수치를 구해보자

set.seed(2020)

another\_xg <- xgboost(data=x\_credit, label= as.numeric(y\_credit)-1, num\_class=2, nrounds=[ ],objective = "multi:softprob",eval\_metric = 'mlogloss', eta=[ ], max\_depth=[ ], gamma=[ ], prediction=T, verbose=F)

pred\_credit5 <- predict(another\_xg, newdata=x\_credit2)  
pred\_credit5 <- get\_pred(pred\_credit5)  
pred\_xg2 <- prediction(pred\_credit5, y\_credit2)  
auc\_xg2 <- performance(pred\_xg2, 'auc')  
auc\_xg2@y.values[[1]]

마찬가지로 그리드탐색을 수행해본다. 역시 시간이 오래걸리니 따로 시행은 하지 않는다.

set.seed(2020)  
grid <- expand.grid(nrounds=400,eta=c(0.05,0.07,0.1), max\_depth=c(3,5,7,10), gamma=c(0,1,3), colsample\_bytree=1, subsample=1, min\_child\_weight=1)  
trcontrol <- trainControl(method='cv', number=5)  
xgb\_hypter\_tuning <- train(x=x\_credit, y=y\_credit, trControl=trcontrol, tuneGrid=grid, method='xgbTree')

set.seed(2020)  
xg\_credit\_better <- xgboost(data=x\_credit, label= as.numeric(y\_credit)-1, num\_class = 2, nrounds=400,objective = "multi:softprob",  
 eval\_metric = 'mlogloss', eta=0.07, max\_depth=10, gamma=3, prediction=T, verbose=F)

pred\_credit4 <- predict(xg\_credit\_better, newdata=x\_credit2)  
  
pred\_credit4 <- get\_pred(pred\_credit4)  
  
pred\_xg\_better <- prediction(pred\_credit4, y\_credit2)  
auc\_xg\_better <- performance(pred\_xg\_better, 'auc')  
auc\_xg\_better@y.values[[1]]

## [1] 0.7776452

**xgb.DMatrix** 자료구조를 이용해서도 모델링을 만들 수 있음

**보통 이것을 더 애용한다…!**

xgb\_train <- xgb.DMatrix(data=x\_credit, label=y\_credit)  
xgb\_test <- xgb.DMatrix(data=x\_credit2, label=y\_credit2)  
  
set.seed(2020)  
xgbc <- xgboost(data=xgb\_train, nrounds=400,max\_depth=10,eta=0.07,gamma=3, prediction=T)

pred\_m <- predict(xgbc, newdata=xgb\_test)  
  
pred\_y <- as.factor(levels(y\_credit2))[round(pred\_m)]  
  
cm = confusionMatrix(y\_credit2, pred\_y)  
cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 3710 182  
## 1 722 386  
##   
## Accuracy : 0.8192   
## 95% CI : (0.8082, 0.8298)  
## No Information Rate : 0.8864   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3653   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.8371   
## Specificity : 0.6796   
## Pos Pred Value : 0.9532   
## Neg Pred Value : 0.3484   
## Prevalence : 0.8864   
## Detection Rate : 0.7420   
## Detection Prevalence : 0.7784   
## Balanced Accuracy : 0.7583   
##   
## 'Positive' Class : 0   
##

p\_m <- prediction(pred\_m, y\_credit2)  
auc\_m <- performance(p\_m, 'auc')  
auc\_m@y.values[[1]]

## [1] 0.7797225

## Regression

Regression 분석을 위한 데이터로는 센서 온도를 예측하는 데이터셋을 이용한다.

실제 이 데이터는 Dacon 대회에 사용된 데이터이며 데이터에 대한 설명은 다음과 같다.

**(대회 설명)**

str(temperature\_train)

## 'data.frame': 4752 obs. of 42 variables:  
## $ id : int 0 1 2 3 4 5 6 7 8 9 ...  
## $ temp1 : num 9.7 9.3 9.4 9.4 9.2 9.1 9.1 8.9 9 8.8 ...  
## $ pres1 : num 989 989 989 989 989 ...  
## $ wins1 : num 1.2 1.7 1.1 1.5 0.8 1.3 2.4 1 1.5 1.4 ...  
## $ wins2 : num 0.6 1.9 2.3 0.7 1.7 0.7 1.9 1.2 0.1 0.4 ...  
## $ acc\_rain1 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ sea\_pres1 : num 1009 1009 1009 1009 1009 ...  
## $ pres2 : num 990 990 990 990 990 ...  
## $ temp2 : num 12.2 12.1 12.1 12 12 12 12 11.9 11.9 11.8 ...  
## $ sea\_pres2 : num 1010 1010 1010 1010 1010 ...  
## $ sea\_pres3 : num 1010 1010 1010 1010 1010 ...  
## $ acc\_rain2 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ acc\_solar1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ hum1 : num 82.4 81.2 86.1 87.7 88.9 90.7 90.2 91.9 92.2 92.7 ...  
## $ wind1 : num 71.3 121.8 232 0 0 ...  
## $ acc\_solar2: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind2 : num 101 165 144 116 125 ...  
## $ acc\_solar3: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind3 : num 56.1 0 0 294.1 0 ...  
## $ wins3 : num 0.3 0.4 0.6 0.1 0 0.2 0.3 0.4 0.6 0.6 ...  
## $ acc\_solar4: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ hum2 : num 91 91.9 93.3 93.4 94.1 94.6 94.9 95 95.5 95.6 ...  
## $ acc\_rain3 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ pres3 : num 1002 1002 1002 1002 1002 ...  
## $ sea\_pres4 : num 1010 1010 1010 1010 1010 ...  
## $ wins4 : num 1.1 0 0 0 0 0 0 0 0 0 ...  
## $ wind4 : num 2.3 0 0 0 0 0 0 0 0 0 ...  
## $ wins5 : num 0.5 0 0 0.2 0 0 0 0 0 0 ...  
## $ pres4 : num 1009 1009 1009 1009 1009 ...  
## $ temp3 : num 12.3 12.1 12.1 12.1 12 12 11.8 11.8 11.6 11.5 ...  
## $ pres5 : num 1002 1002 1002 1002 1002 ...  
## $ hum3 : num 69.1 70.3 71.5 73.2 74.3 75.7 79 78.2 77.5 79.3 ...  
## $ temp4 : num 8.2 8.3 8 7.7 7.4 7.3 7.1 7.1 7 7.1 ...  
## $ temp5 : num 10.7 10.3 9.7 9.4 9.4 9.1 8.6 8.8 8.8 8.5 ...  
## $ sea\_pres5 : num 1010 1010 1010 1010 1010 ...  
## $ acc\_solar5: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind5 : num 256 215 235 214 175 ...  
## $ acc\_rain4 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ hum4 : num 77.2 77.3 77.3 77.5 78 78.7 78.6 78.6 78.8 78.8 ...  
## $ hum5 : num 62.6 63.5 63.9 64.5 65 65.1 65.5 65.7 66.5 67.6 ...  
## $ acc\_rain5 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Y18 : num 10 9.5 9.25 9 8.75 8.75 8.5 8.5 8.25 8.25 ...

str(temperature\_test)

## 'data.frame': 11520 obs. of 41 variables:  
## $ id : int 4752 4753 4754 4755 4756 4757 4758 4759 4760 4761 ...  
## $ temp1 : num 18.7 18.8 19 18.7 18.4 18.6 18.6 18.7 18.5 18.5 ...  
## $ pres1 : num 987 987 987 987 987 ...  
## $ wins1 : num 1.5 1.1 1.3 2.3 1.1 1.3 1 1.5 1.2 1.5 ...  
## $ wins2 : num 1.3 1.2 1.2 0.8 0.7 0 0.8 0.8 1.6 0 ...  
## $ acc\_rain1 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ sea\_pres1 : num 1007 1007 1007 1007 1007 ...  
## $ pres2 : num 988 988 988 988 988 ...  
## $ temp2 : num 21.1 21.2 21.2 21.1 20.9 20.7 20.6 20.5 20.4 20.4 ...  
## $ sea\_pres2 : num 1008 1008 1007 1007 1008 ...  
## $ sea\_pres3 : num 1007 1007 1007 1007 1007 ...  
## $ acc\_rain2 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ acc\_solar1: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ hum1 : num 96.1 96.2 96.4 96.5 96.6 96.6 96.9 96.8 96.8 96.9 ...  
## $ wind1 : num 334 336 0 253 238 ...  
## $ acc\_solar2: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind2 : num 200 194 173 154 174 ...  
## $ acc\_solar3: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind3 : num 36 127.7 59 54.9 48.8 ...  
## $ wins3 : num 1.1 0.6 0 0.4 0.8 0.1 0.8 0.8 0.1 0.2 ...  
## $ acc\_solar4: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ hum2 : num 94.7 95.3 93.9 94.4 95.7 96.7 96.1 96.6 95.9 96.1 ...  
## $ acc\_rain3 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ pres3 : num 1000 1000 1000 1000 1000 ...  
## $ sea\_pres4 : num 1008 1008 1007 1007 1007 ...  
## $ wins4 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind4 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wins5 : num 0.1 0.3 0.7 0.6 0.7 0.8 0.8 0.8 0.4 0.2 ...  
## $ pres4 : num 1006 1006 1006 1006 1006 ...  
## $ temp3 : num 23.2 23.1 23.1 22.8 22.8 22.6 22.6 22.5 22.2 22.1 ...  
## $ pres5 : num 1000 999 999 999 999 ...  
## $ hum3 : num 91.4 91.6 92 93 92.3 90.6 90.4 92 93.1 94.3 ...  
## $ temp4 : num 17.5 17.4 17.2 17.2 17.4 17.3 17.3 17.1 16.7 16.6 ...  
## $ temp5 : num 19.2 19.1 19 18.8 18.9 18.9 18.8 18.4 18.2 18.3 ...  
## $ sea\_pres5 : num 1008 1008 1008 1007 1008 ...  
## $ acc\_solar5: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ wind5 : num 195 173 209 185 163 ...  
## $ acc\_rain4 : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ hum4 : num 84 84 85 85.8 87.3 87.1 87.3 88.5 88.2 88.9 ...  
## $ hum5 : num 59.9 60.5 60.8 61.2 61.7 61.9 61.6 61.9 62.5 63.4 ...  
## $ acc\_rain5 : num 0 0 0 0 0 0 0 0 0 0 ...

우선 서버에 제출하기 전에

자체적으로 훈련 데이터셋을 나누어 간단하게 성능을 테스트해보는 연습을 해보자

훈련 데이터 내에서 작은 훈련 데이터, 테스트 데이터 셋을 나눈다

temp\_idx <- createDataPartition(temperature\_train$Y18, p=.8, list=F)  
temp\_tr <- temperature\_train[temp\_idx, ]  
temp\_te <- temperature\_train[-temp\_idx, ]

# id항목 아예 제거  
temp\_tr <- temp\_tr[-1]  
temp\_te <- temp\_te[-1]

## 1. Random Forest

간단한 randomForest 모형을 하나 돌려보면

set.seed(2020)  
rf\_temp <- randomForest(Y18~., data=temp\_tr)  
rf\_temp$ntree

## [1] 500

rf\_temp$mtry

## [1] 13

해당 모형에 대한 예측값을 만든 뒤  
pred\_temp2 <- predict(rf\_temp, temp\_te)

RMSE를 자체적으로 검증하는 함수를 정의한 뒤

calcRMSE <- function(label, estimation){  
 return(sqrt(mean((label-estimation)\*\*2)))  
}

calcRMSE(temp\_te$Y18, pred\_temp2)

## [1] 0.7828474

검사해보면 다음과 같이 나옴을 알 수 있다.

set.seed(2020)  
rf\_temp2 <- randomForest(Y18~., data=temp\_tr, ntree=300)  
  
pred\_temp4 <- predict(rf\_temp2, temp\_te)  
  
calcRMSE(temp\_te$Y18, pred\_temp4)

## [1] 0.780767

마찬가지로 하이퍼 파리미터 검증을 해보면

fitContrl2 <- trainControl(method='cv', number=3, search='grid')  
grids2 <- expand.grid(.mtry=c(10:15))  
modellists2 <- list()  
for (ntrees in c(100,300,500,700)){  
 set.seed(2020)  
 rf\_fit2 <- train(Y18~., data=temp\_tr, method='rf', metric = 'RMSE', tuneGrid = grids2, trControl=fitContrl2,ntree=ntrees,verbose=T)  
 key2 <- toString(ntrees)  
 modellists2[[key2]] <- rf\_fit2  
}

set.seed(2020)  
rf\_temp3 <- randomForest(Y18~., data=temp\_tr, mtry=13, ntree=700)  
  
pred\_temp6 <- predict(rf\_temp3, temp\_te)

calcRMSE(temp\_te$Y18, pred\_temp6)

## [1] 0.7778432

서버에 제출할 데이터 프레임 하나를 만들어보자  
rf\_temp\_sub <- randomForest(Y18~., data=temperature\_train, mtry=11, ntree=500)  
  
pred\_temp\_sub\_rf <- predict(rf\_temp\_sub, temperature\_test)  
  
sub\_temp$Y18 <- pred\_temp\_sub\_rf

write.csv(sub\_temp, file="submission\_temp\_rf.csv", row.names = F)

## 2. Xgboost

Xgboost 도 한번 수행해보자

temp\_tr\_mtx <- data.matrix(temp\_tr[-41])  
temp\_tr\_mty <- data.matrix(temp\_tr$Y18)  
temp\_te\_mtx <- data.matrix(temp\_te[-41])  
temp\_te\_mty <- data.matrix(temp\_te$Y18)

train\_data <- xgb.DMatrix(data=temp\_tr\_mtx, label=temp\_tr\_mty)  
set.seed(2020)  
xg\_temp <- xgboost(data=train\_data, nrounds=500, eta=0.05, max\_depth=20, gamma=3, prediction=T, verbose=F)

## [11:12:45] WARNING: amalgamation/../src/learner.cc:480:   
## Parameters: { prediction } might not be used.  
##   
## This may not be accurate due to some parameters are only used in language bindings but  
## passed down to XGBoost core. Or some parameters are not used but slip through this  
## verification. Please open an issue if you find above cases.

pred\_temps <- predict(xg\_temp, newdata=temp\_te\_mtx)  
  
  
calcRMSE(temp\_te\_mty, pred\_temps)

## [1] 0.7618633

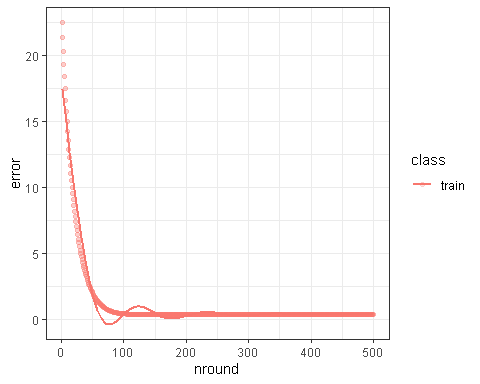
cvplot = function(model){  
 eval.log = model$evaluation\_log  
 data.frame(error=c(unlist(eval.log[,2])),  
 class = c(rep('train',nrow(eval.log))),  
 nround=c(rep(1:nrow(eval.log),2))  
 ) %>% ggplot(aes(nround,error,col=class))+geom\_point(alpha=0.2)+geom\_smooth(alpha=0.4, se=F)+theme\_bw()  
}

nround 변화에 따른 error값을 확인해보자

cvplot(xg\_temp)

## Warning in data.frame(error = c(unlist(eval.log[, 2])), class = c(rep("train", :  
## row names were found from a short variable and have been discarded

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



**xgb.cv**의 방법으로도 교차 검증을 수행할 수 있다

**caret의 train함수보다 더 빠른 장점이 있다!!**

best\_param = list()  
best\_logloss = Inf  
best\_logloss\_index = 0  
for (iter in 1:10){  
 params <- list(objective = "reg:linear",  
 eval\_metric = "rmse",  
 max\_depth=20,  
 eta=runif(1,0.01,0.3),  
 gamma=sample(1:4,1))  
 set.seed(2020)  
 xgbcv <- xgb.cv(data=train\_data,params=params,nfold=5,nrounds=500,early\_stopping\_rounds = 500,verbose=F)  
 min\_rmse <- min(xgbcv[[4]]$test\_rmse\_mean)  
 min\_rmse\_index <- which.min(xgbcv[[4]]$test\_rmse\_mean)  
   
 if(min\_rmse < best\_logloss){  
 best\_rmse <- min\_rmse  
 best\_rmse\_index <- min\_rmse\_index  
 best\_param = params  
 }  
   
}

best\_param$max\_depth <- 20  
best\_param

## $objective  
## [1] "reg:linear"  
##   
## $eval\_metric  
## [1] "rmse"  
##   
## $max\_depth  
## [1] 20  
##   
## $eta  
## [1] 0.2077155  
##   
## $gamma  
## [1] 2

set.seed(2020)  
xg\_temp2 <- xgboost(data=train\_data, params=best\_param, nrounds=500, verbose=F)

pred\_temps2 <- predict(xg\_temp2, temp\_te\_mtx)  
  
calcRMSE(temp\_te\_mty, pred\_temps2)

## [1] 0.73962

## xgboost로 temperature\_train 훈련 데이터 모델링을 돌린 뒤 temperature\_test에 맞게 제출을 해보자