Data Mining lab 2.1

Градиентный бустинг

1. Данные

Вначале подгрузим все необходимые библиотеки. В данной лабораторной для бустинга используется библиотека XGBoost от Kaggle.

#библиотеки

library(xgboost)

library(caTools)

library(dplyr)

library(caret)

#Загрузим данные

myData←read.csv("C:/R/tourism.csv")

head(myData)

#в этих данных все переменные представляют собой категории, а расходы переведены в группировочные интервалы. Как следствие, необходимо присвоить им числовые метки. По умолчанию строковые данные имеют тип «string», и в целочисленный формат преобразованы с помощью функции as.integer быть не могут, для этого нам нужно преобразовать их в тип «factor», чтобы такие метки были присвоены как категории.

myData[]<-lapply(myData, factor)

#случайное зерно (seed) задают для повторяемости результатов псевдогенератора случайных чисел, например, чтобы была возможность в точности воспроизвести эксперимент или построение модели.

set.seed(1234)

#разделим данные с помощью случайной выборки на обучающие и тестовые

sample\_split <- sample.split(Y = myData$TYPE, SplitRatio = 0.7)

train\_set <- subset(x = myData, sample\_split == TRUE)

test\_set <- subset(x = myData, sample\_split == FALSE)

y\_train <- as.integer(train\_set$TYPE) - 1

y\_test <- as.integer(test\_set$TYPE) - 1

X\_train <- train\_set %>% select(-TYPE)

X\_test <- test\_set %>% select(-TYPE)

2. Моделирование

#DMatrix это не более чем способ хранения табличных данных в машинном обучении, для оптимизации памяти и скорости обучения

xgb\_train <- xgb.DMatrix(data = as.matrix(sapply(X\_train, as.numeric)), label = y\_train)

xgb\_test <- xgb.DMatrix(data = as.matrix(sapply(X\_test, as.numeric)), label = y\_test)

#небольшой кусок кода для хранения различных параметров модели, чтобы не вводить их каждый раз в саму функцию ее обучения

xgb\_params <- list(

booster = "gbtree",

eta = 0.01,

max\_depth = 8,

gamma = 4,

subsample = 0.75,

colsample\_bytree = 1,

objective = "multi:softprob",

eval\_metric = "mlogloss",

num\_class = length(levels(myData$TYPE))

)

#Подгонка модели

xgb\_model <- xgb.train(

params = xgb\_params,

data = xgb\_train,

nrounds = 5000,

verbose = 1

)

xgb\_model

#Предсказание; в R большинство моделей вычисляют прогнозируемые значения с помощью функции predict()

xgb\_preds <- predict(xgb\_model, as.matrix(sapply(X\_test, as.numeric)), reshape = TRUE)

xgb\_preds <- as.data.frame(xgb\_preds)

colnames(xgb\_preds) <- levels(myData$TYPE)

xgb\_preds

#Для удобства просмотра результатов прогноза, создадим в тестовых данных две колонки, с предсказанной группой (PredictedClass) и фактической (ActualClass)

xgb\_preds$PredictedClass <- apply(xgb\_preds, 1, function(y) colnames(xgb\_preds)[which.max(y)])

xgb\_preds$ActualClass <- levels(myData$TYPE)[y\_test + 1]

xgb\_preds

#Точность модели составит 33%, так как размер данных очень мал

accuracy <- sum(xgb\_preds$PredictedClass == xgb\_preds$ActualClass) / nrow(xgb\_preds)

accuracy

#Матрица классификации:

confusionMatrix(factor(xgb\_preds$ActualClass), factor(xgb\_preds$PredictedClass))