1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. Высшая школа искусственного интеллекта

**ЛАБОРАТОРНАЯ РАБОТА №7**

**«Алгоритмы композиции»**

по дисциплине «Машинное обучение»

1. Выполнил: студент группы
2. 3540201/20302 А.И. Обидина

*<подпись>*

1. Проверил: Л.В. Уткин
2. д.т.н., профессор
3. *<подпись>*

Санкт-Петербург

2022

**Задание**

1. Исследовать зависимость тестовой ошибки от количества деревьев в ансамбле для алгоритма adaboost на наборе данных Vehicle из пакета mlbench (обучающая выборка должна состоять из 7/10 всех прецедентов, содержащихся в данном наборе данных). Построить график зависимости тестовой ошибки при числе деревьев, равном 1, 11, 21, ... , 301, объясните полученные результаты.

2. Исследовать зависимость тестовой ошибки от количества деревьев в ансамбле для алгоритма bagging на наборе данных Glass из пакета mlbench (обучающая выборка должна состоять из 7/10 всех прецедентов, содержащихся в данном наборе данных). Построить график зависимости тестовой ошибки при числе деревьев, равном 1, 11, 21, ... , 201, объясните полученные результаты.

3. Реализовать бустинг алгоритм с классификатором K ближайших соседей. Сравнить тестовую ошибку, полученную с использованием данного классификатора на наборах данных Vehicle и Glass, c тестовой ошибкой, полученной с использованием единичного дерева классификации.

# Ход работы

## **Пункт 1**

Была определена зависимость ошибки классификации от числа деревьев (метод adaboost.M1). График зависимости представлен на рисунке 1.

Обучающая выборка составляет 70 % от всего датасета, количество деревьев менялось от 1 до 301 с шагом 10.

Максимальная ошибка классификации (0,397) возникла при одном дереве в ансамбле, минимальная ошибка (0,129) – при 141 и 271 деревьях в ансамбле.

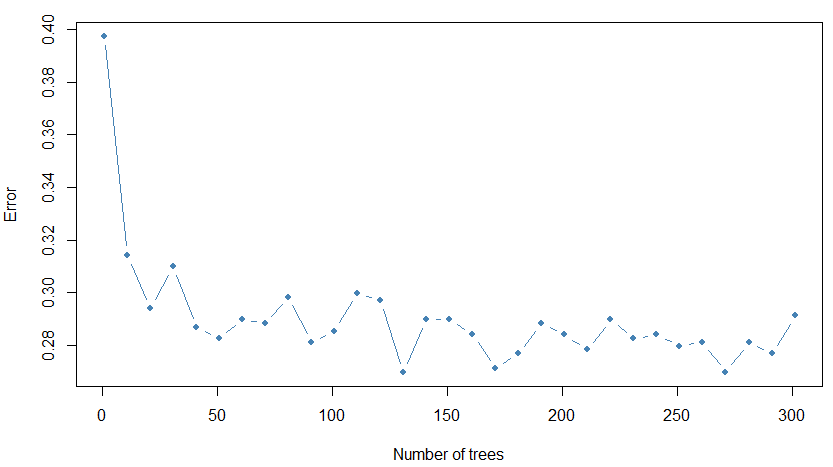


Рисунок 1 – График зависимости ошибки классификации от числа деревьев (метод adaboost.M1)

**Код:**

library(adabag)

library(rpart)

library(mlbench)

data(Vehicle)

strings\_number<-length(Vehicle[,1])

train\_num<-sample(1:strings\_number,0.7\*strings\_number)

misclass <- c()

k=1

for (i in seq(1, 301, 10))

{

print(i)

model <- boosting(Class ~ ., data = Vehicle[train\_num,], mfinal = i)

pr <- predict(model, Vehicle[-train\_num,])

misclass[k] <- pr$error

k=k+1

}

print(misclass)

plot(seq(1, 301, 10), misclass, col = "steelblue", xlab="Number of trees", ylab="Error", pch = 20, cex = 1.2, type="b")

## **Пункт 2**

Была определена зависимость ошибки классификации от числа деревьев (метод bagging). График зависимости представлен на рисунке 2.

Обучающая выборка составляет 70 % от всего датасета, количество деревьев менялось от 1 до 301 с шагом 10.

Максимальная ошибка классификации (0,343) возникла при одном дереве в ансамбле, минимальная ошибка (0,073) – при 121, 131, 151 и 171 деревьях в ансамбле.

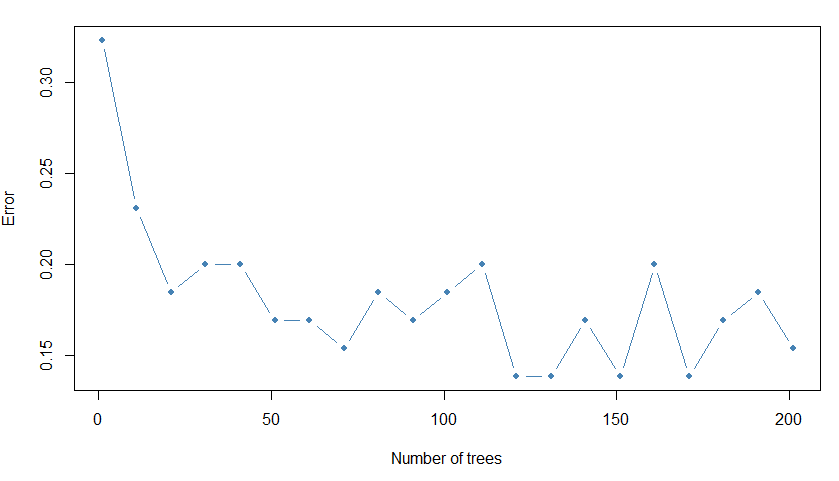


Рисунок 2 – График зависимости ошибки классификации от числа деревьев (метод bagging)

**Код:**

library(adabag)

library(rpart)

library(mlbench)

data(Glass)

strings\_number<-length(Glass[,1])

train\_num<-sample(1:strings\_number,0.7\*strings\_number)

misclass <- c()

k=1

for (i in seq(1, 201, 10))

{

print(i)

model <- bagging(Type ~ ., data = Glass[train\_num,], mfinal = i)

pr <- predict(model, Glass[-train\_num,])

misclass[k] <- pr$error

k=k+1

}

print(misclass)

plot(seq(1, 201, 10), misclass, col = "steelblue", xlab="Number of trees", ylab="Error", pch = 20, cex = 1.2, type="b")

## **Пункт 3**

Был реализован алгоритм adaboosting со слабым классификатором K ближайших соседей. Для этого были реализованы функции обучения классификатора - AdaBoost <- function(train, Y, t, k) (цикл, повторяющийся t раз – количество слабых классификаторов, k – количество ближайших соседей) и предсказания - AdaBoost\_pred <- function(model, test\_data). На каждой итерации цикла обучения происходит классификация методом k ближайших соседей и модификация весов, зависящая от полученных результатов классификации.

Также был реализован классификатор K ближайших соседей - kknn\_pr <- function(model, test\_data). Особенностью данного классификатора является то, что он выбирает результирующую метку не как наиболее часто встречающуюся, а как метку с наибольшим суммарным весов среди K соседей.

С помощью реализованного алгоритма были классифицированы объекты датасетов Glass и Vehicle (k = 7), результаты представлены в таблице 1.

Таблица 1. Тестовая ошибка для реализованного алгоритма AdaBoost

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество knn-классификаторов | 1 | 2 | 3 | 4 |
| Датасет Glass | 0,318 | 0,324 | 0,338 | 0,318 |
| Датасет Vehicle | 0,333 | 0,346 | 0,327 | 0,316 |

С помощью полученных результатов можно сделать вывод, что точность классификатора зависит от выборки, а также скорость его работы меньше скорости классификаторов, используемых в пунктах 1 и 2. Ошибки, допущенные реализованным классификатором, примерно соответствуют ошибкам, допущенным классификатором rpart с одним деревом – 0,333 и 0,276 для датасетов Glass и Vehicle соответственно.

**Код:**

AdaBoost <- function(train, Y, t, k) {

weak\_cls <- list() #list with weak classifiers

strings\_number <- nrow(train)

weights <- rep(1/strings\_number, strings\_number)

a <- c() # alphas - weights of weak classifiers

for (i in 1:t) {

labels <- train[, Y]

Y\_without\_repeating <- levels(train[, Y])

model <- list(Y = Y, train = train, levels = Y\_without\_repeating, weights = weights, k = k)

weak\_cls[[i]] <- model

# kknn---

predict <- kknn\_pr(model, train)

print(predict)

print(train[,Y])

#---

e = 0 # sum of errors

for (j in 1:strings\_number) {

if (labels[j] != predict[j]) {

e <- e + weights[j]

}

}

if (e != 0)

a[[i]] <- 0.5\*log((1-e)/e)

else

stop("Error = 0")

for (l in 1:strings\_number) {

if (predict[l] != train[[Y]][l]) misCl = 1

else misCl = -1

weights[l] <- weights[l]\*exp(a[[i]]\*misCl)

}

}

r <- list(weak\_cls=weak\_cls, a=a, levels = levels(train[, Y]))

return(r)

}

AdaBoost\_pred <- function(model, test\_data) {

strings\_number\_test <- nrow(test\_data)

pred = c()

for (i in 1:strings\_number\_test) {

myfreq <- data.frame(names = model$levels, freq = rep(0, length(model$levels)))

for (j in 1:length(model$weak\_cls)) {

prediction <- kknn\_pr(model$weak\_cls[[j]], test\_data[i, ])

myfreq[myfreq$names == prediction, ][2] <- myfreq[myfreq$names == prediction, ][2] + model$a[j]

}

most\_frequent = model$levels[myfreq$freq == max(myfreq$freq)]

pred[i] <- sample(most\_frequent, 1)

}

return(pred)

}

kknn\_pr <- function(model, test\_data) {

strings\_number\_test <- nrow(test\_data)

pred <- c()

features <- !(names(model$train) %in% model$Y)

train <- model$train[, features]

dim\_dist<- ncol(train) - 1 # how many features

strings\_number\_train <- nrow(train)

test <- test\_data[, features]

for (i in 1:strings\_number\_test) {

#взяли i-й пример из тестовой выборки

dist <- c()

for (j in 1:strings\_number\_train) {

#взяли j-й пример из обучающей выборки

dist[j] <- 0

for (l in 1:dim\_dist) {

dist[j] <- dist[j]+(test[i,l] - train[j,l])^2

}

}

order\_dist <- order(dist)

max\_frq <- 0

labels <- model$train[, model$Y]

y <- labels[order\_dist[1]]

for (j in 1:model$k) {

frq=model$weights[order\_dist[j]]

for (l in (j+1):model$k) {

if (labels[order\_dist[j]] == labels[order\_dist[l]]) {

frq=frq+model$weights[order\_dist[l]]

}

}

if (frq > max\_frq) {

max\_frq = frq

y <- labels[order\_dist[j]]

}

}

pred<-append(pred, y)

}

return(pred)

}

#----------------------------------------------------------------------------

library(mlbench)

library(adabag)

data("Glass")

strings\_number <- nrow(Glass)

n\_train <- as.integer(strings\_number\*0.7)

data\_rand <- Glass[order(runif(strings\_number)), ]

train\_data <- data\_rand[1:n\_train, ]

test\_data <- data\_rand[n\_train:strings\_number, ]

model <- AdaBoost(train\_data, 'Type', t = 1, k = 7)

predict <- AdaBoost\_pred(model, test\_data)

predict\_table <- table(test\_data$Type, predict)

predict\_table

miscl <- 1 - sum(diag(predict\_table))/sum(predict\_table)

miscl

data("Vehicle")

strings\_number <- nrow(Glass)

n\_train <- as.integer(strings\_number\*0.7)

data\_rand <- Vehicle[order(runif(strings\_number)), ]

train\_data <- data\_rand[1:n\_train, ]

test\_data <- data\_rand[n\_train:strings\_number, ]

model <- AdaBoost(train\_data, 'Class', t = 1, k = 7)

predict <- AdaBoost\_pred(model, test\_data)

predict\_table <- table(test\_data$Class, predict)

predict\_table

miscl <- 1 - sum(diag(predict\_table))/sum(predict\_table)

miscl

## **Вывод**

При выполнении данной лабораторной работы были изучены алгоритмы композиции, такие как бэггинг и бустинг. Также был реализован алгоритм AdaBoost. Данные алгоритмы хороши в решении сложных задач, которые плохо поддаются отдельным алгоритмам, также показывают хорошие результаты с вычислительной точки зрения. Алгоритмы имеют достаточно маленькую ошибку и один настраиваемый параметр – количество итераций.

Но данные алгоритмы достаточно чувствительны к выбросам и могут привести к обучению (для более эффективной работы требуется соответствующая обработка набора данных; набор данных должен быть большого размера).