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, часть1»

1. Выполнил: студент группы
2. 3540201/20302 С.А. Ляхова

*<подпись>*

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

Санкт-Петербург

2022

**Содержание**

[1. Цель работы 3](#_Toc121670354)

[2. Формулировка задания 3](#_Toc121670355)

[3. Ход работы 3](#_Toc121670356)

[4. Вывод 6](#_Toc121670357)

[Приложение 1 7](#_Toc121670358)

[Приложение 2 8](#_Toc121670359)

[Приложение 3 9](#_Toc121670360)

# **Цель работы**

В лабораторной работе необходимо ознакомится с алгоритмами композиций, используя библиотеки *adabag* и *rpart*.

# **Формулировка задания**

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**

**Задание №1**

На рисунке 1 изображена зависимость тестовой ошибки от числа деревьев в ансамбле для метода adaboost.M1. Обучающая выборка состоит из 0.7 всех прецедентов, содержащихся в наборе данных Vehicle. Число деревьев равно 1, 11, 21, ... , 301.

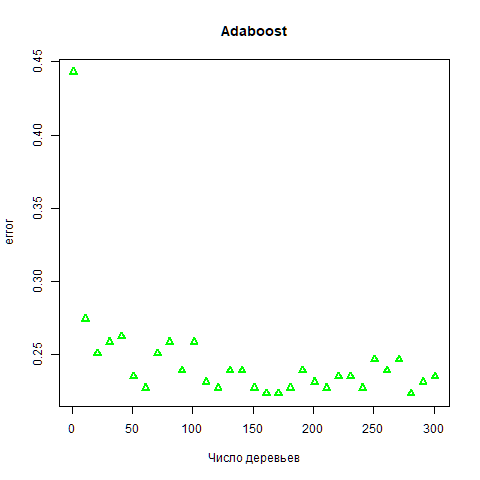


Рисунок 1. Зависимость тестовой ошибки от числа деревьев в ансамбле для метода adaboost

Самая большая ошибка 0.4431373 при одном дереве в ансамбле. Минимальная ошибка 0.2235294 достигнута при 161, 171, 281 дереве в ансамбле. Среднее значение ошибки составило 24.6%.

Судя по рисунку 1, эффект недообучения заметен лишь при небольшом числе деревьев (0-100).

**Задание №2**

На рисунке 2 изображена зависимость тестовой ошибки от числа деревьев в ансамбле для метода bagging. Обучающая выборка состоит из 0.7 всех прецедентов, содержащихся в наборе данных Glass. Число деревьев равно 1, 11, 21, ... , 201.

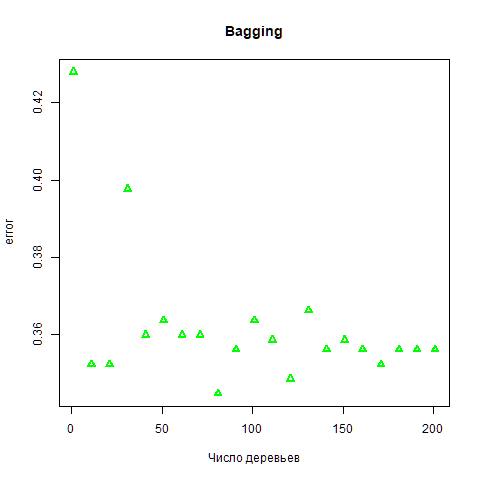


Рисунок 2. Зависимость тестовой ошибки от числа деревьев в ансамбле в методе bagging

Для поиска среднего значения ошибки для каждого количества деревьев алгоритм был запущен 4 раза, а затем результаты усреднялись.

Самая большая ошибка 0.4280303 при одном дереве в ансамбле. Минимальная ошибка 0.344697 достигнута при 81 дереве в ансамбле. Среднее значение ошибки составило 36.3%.

Судя по рисунку 2, эффект недообучения, аналогично adaboost, заметен при небольшом числе деревьев, .

**Задание №3**

Реализация алгоритма adaboosting со слабым классификатором, основанном на алгоритме k ближайших соседей, представляет собой набор функций, реализующих алгоритм и тестирующих полученные данные, а также циклический эксперимент с итерациями по параметру mfinal – количеству использований слабого классификатора. Данный цикл также содержит внутренний цикл тестирования, количество итераций которого даже являются параметром реализации.

На каждой итерации главного цикла совершается классификация методом k ближайших соседей на наборах данных Glass и Vehicle, затем полученные матрицы весов и меток классов подаются на вход функциям алгоритма бустинга.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество слабых классификаторов |  |  | 1 | 2 | 3 | 4 |
| Glass | Разработанный классификатор |  | 34.8% | 36.4% | 37.8% | 34.8% |
| rpart | 33.3% |  |  |  |  |
| Vehicle | Разработанный классификатор |  | 27.8% | 26.1% | 24.9% |  |
| rpart | 23.9% |  |  |  |  |

Таблица 1. Тестовая ошибка для встроенного и реализованного алгоритмов

Была реализована модификация алгоритма k ближайших соседей с использованием весов. Особенностью такого подхода является то, что результатом классификации является не тот класс, к которому принадлежит наибольшее число соседей, а класс с наибольшим весом, который складывается из весов соседей.

Из таблицы 1 можем сделать вывод, что точность классификатора довольно сильно зависит от выборки – на каких-то данных можно получить классификаторы, которые работают лучше одиночного дерева, а на каких-то – хуже. Также стоит отметить, что классификатор, написанный вручную, работает значительно медленнее функции rpart.

# **Вывод**

В ходе выполнения лабораторной работы были изучены алгоритмы композиции: бэггинг и бустинг.

Алгоритмы эффективны с вычислительной точки зрения, позволяют решать сложные задачи, которые плохо решаются отдельными алгоритмами, просты в программировании, имеют только один тъюнинг параметр - число итераций, во многих случаях обеспечивают высокую точность прогнозирования и просты в модификации.

Недостатки: слишком эффективные или сложные слабые классификаторы могут привести к переобучению, чувствительны к выбросам, требуют достаточно большие выборки.

# **Приложение 1**

# Задание 1 #########################################

data(Vehicle)

ratio <- 0.7

n <- nrow(Vehicle)

n\_train <- as.integer(n\*ratio)

Vehicle\_rand <- Vehicle[order(runif(n)), ]

Vehicle\_train <- Vehicle\_rand[1:n\_train, ]

Vehicle\_test <- Vehicle\_rand[n\_train:n, ]

tree\_num <- seq(1, 301, 10)

error <- vector()

for (k in tree\_num)

{

clf <- boosting(Class ~ ., data = Vehicle\_train, mfinal = k)

err <- predict(clf, Vehicle\_test)$error

error <- append(error, err)

}

max(error)

min(error)

mean(error)

png(paste(path, 'adaboost.png'))

plot(x = tree\_num, y = error, xlab = "Число деревьев", pch = 2, lwd = 2, col = "green", main = "Adaboost")

dev.off()

# **Приложение 2**

# Задание 2 ##########################################

data("Glass")

n <- nrow(Glass)

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

Glass\_rand <- Glass[order(runif(n)), ]

Glass\_train <- Glass\_rand[1:n\_train, ]

Glass\_test <- Glass\_rand[n\_train:n, ]

error <- vector()

tree\_num <- seq(1, 201, 10)

for (k in tree\_num)

{

err <- vector()

#for(i in 1:4)

#{

clf <- bagging(Type ~ ., data = Glass\_train, mfinal = k)

err <- append(err, predict(clf, Glass\_test)$error)

#}

#error <- append(error, mean(err))

error <- append(error, err)

}

max(error)

min(error)

mean(error)

png(paste(path, 'bagging1.png'))

plot(x = tree\_num, y = error, xlab = "Число деревьев", pch = 2, lwd = 2, col = "green", main = "Bagging")

dev.off()

# **Приложение 3**

# Задание 3 ##########################################

library(dplyr)

knn\_w <- function(target, train, k, w) return(list(target = target, train = train, levels = levels(train[, target]), k = k, w = w))

knn\_w\_predicted <- function(clfier, testdata) {

n <- nrow(testdata)

pred <- rep(NA\_character\_, n)

trainlabels <- clfier$train[, clfier$target]

train <- clfier$train[, !(names(clfier$train) %in% clfier$target)]

test <- testdata[, !(names(testdata) %in% clfier$target)]

for (i in 1:n) {

n\_number <- order(apply(train, 1, function(x)

sum((test[i,] - x)^2)))[1:clfier$k]

myfreq <- data.frame(names = clfier$levels,

freq = rep(0, length(clfier$levels)))

for (t in n\_number) {

myfreq[myfreq$names == trainlabels[t], ][2] <- myfreq[myfreq$names == trainlabels[t], ][2] + clfier$w[t]

}

most\_frequent <- clfier$levels[myfreq$freq == max(myfreq$freq)]

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

}

factor(pred, levels = levels(trainlabels))

}

knn\_boosting <- function(target,

data,

k = 11,

mfinal = 1,

...) {

n <- nrow(data)

w <- rep(1/n, each = n)

classifiers <- list()

alphas <- vector()

for (t in 1:mfinal) {

clfier <- knn\_w(target, train = data, k = k, w)

knn\_predicted <- knn\_w\_predicted(clfier, data)

error <- vector()

for (i in 1:n) {

if (data[[target]][i] != knn\_predicted[i]) error <- append(error, w[i])

}

if (sum(error) >= 0.5) {

break()

}

classifiers[[t]] <- clfier

alphas[[t]] <- log((1 - sum(error)) / sum(error)) / 2

for (i in 1:n) {

if (knn\_predicted[i] != data[[target]][i])

{

w[i] <- w[i]\*exp(alphas[[t]])

} else{

w[i] <- w[i]\*exp(-alphas[[t]])

}

}

}

result <- list()

result$classifiers <- classifiers

result$alphas <- alphas

result$levels <- levels(data[, target])

return(result)

}

boosting\_pred <- function(clfier, testdata) {

n <- nrow(testdata)

pred = rep(NA\_character\_, n)

for (i in 1:n) {

myfreq <- data.frame(names = clfier$levels,

freq = rep(0, length(clfier$levels)))

for (j in 1:length(clfier$classifiers)) {

prediction <- knn\_w\_predicted(clfier$classifiers[[j]], testdata[i, ])

myfreq[myfreq$names == prediction, ][2] <- myfreq[myfreq$names == prediction, ][2] + clfier$alphas[j]

}

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

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

}

factor(pred, levels = clfier$levels)

}

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

library(mlbench)

library(adabag)

data("Glass")

n <- nrow(Glass)

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

Glass\_rand <- Glass[order(runif(n)), ]

Glass\_train <- Glass\_rand[1:n\_train, ]

Glass\_test <- Glass\_rand[n\_train:n, ]

Glass\_rpart <- rpart(Type ~ ., data = Glass\_train, maxdepth = 5)

Glass\_rpart\_pred <- predict(Glass\_rpart, Glass\_test, type = 'class')

tbl\_rpart <- table(Glass\_rpart\_pred, Glass\_test$Type)

1 - (sum(diag(tbl\_rpart)) / sum(tbl\_rpart))

clfier <- knn\_boosting('Type', Glass\_train, mfinal = 4)

pred <- boosting\_pred(clfier, Glass\_test)

tbl\_knn <- table(Glass\_test$Type, pred)

1 - sum(diag(tbl\_knn)) / sum(tbl\_knn)

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

data(Vehicle)

n <- nrow(Vehicle)

n\_train <- as.integer(n\*ratio)

Vehicle\_rand <- Vehicle[order(runif(n)), ]

Vehicle\_train <- Vehicle\_rand[1:n\_train, ]

Vehicle\_test <- Vehicle\_rand[n\_train:n, ]

Vehicle\_rpart <- rpart(Class ~ ., data = Vehicle\_train, maxdepth = 5)

Vehicle\_rpart\_pred <-

predict(Vehicle\_rpart, newdata = Vehicle\_test, type = 'class')

tbl\_rpart <- table(Vehicle\_rpart\_pred, Vehicle\_test$Class)

1 - (sum(diag(tbl\_rpart)) / sum(tbl\_rpart))

clfier <- knn\_boosting('Class', Vehicle\_train, mfinal = 1)

pred <- boosting\_pred(clfier, Vehicle\_test)

tbl\_knn <- table(Vehicle\_test$Class, pred)

1 - sum(diag(tbl\_knn)) / sum(tbl\_knn)