Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

ЛАБОРАТОРНАЯ РАБОТА №7

«Алгоритмы композиции»

по дисциплине «Машинное обучение, часть1»

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

<подпись>

Проверил: Л.В. Уткин
д.т.н., профессор

<подпись>

Санкт-Петербург
2022

Содержание

1. Цель работы

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

2. 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, с тестовой ошибкой, полученной с использованием единичного дерева классификации.

3. 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.

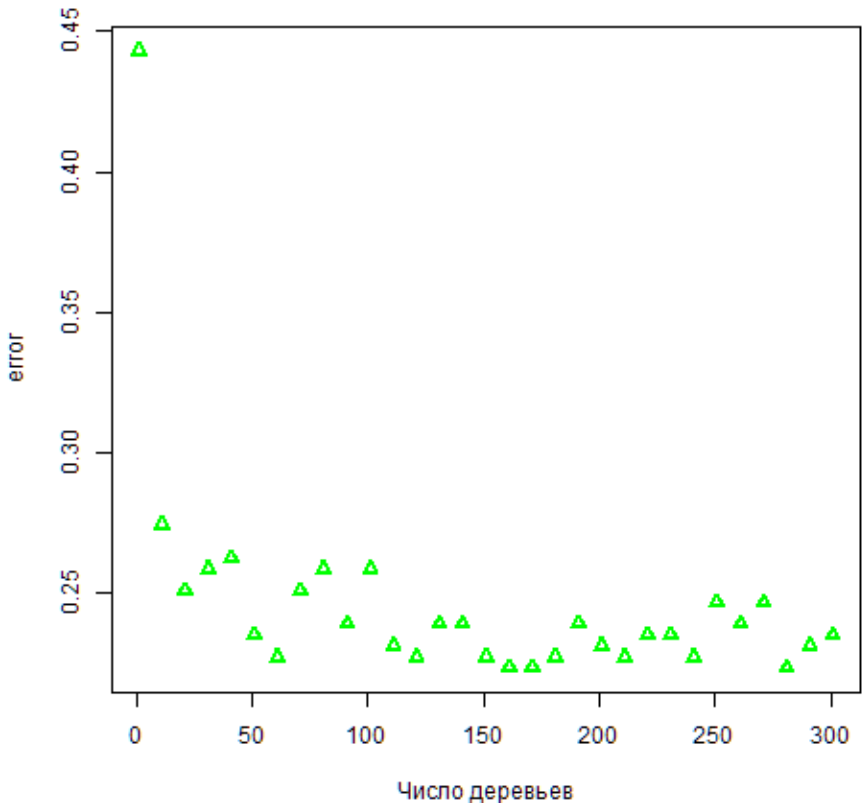

Рисунок 1. Зависимость тестовой ошибки от числа деревьев в ансамбле для метода adaboost

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.

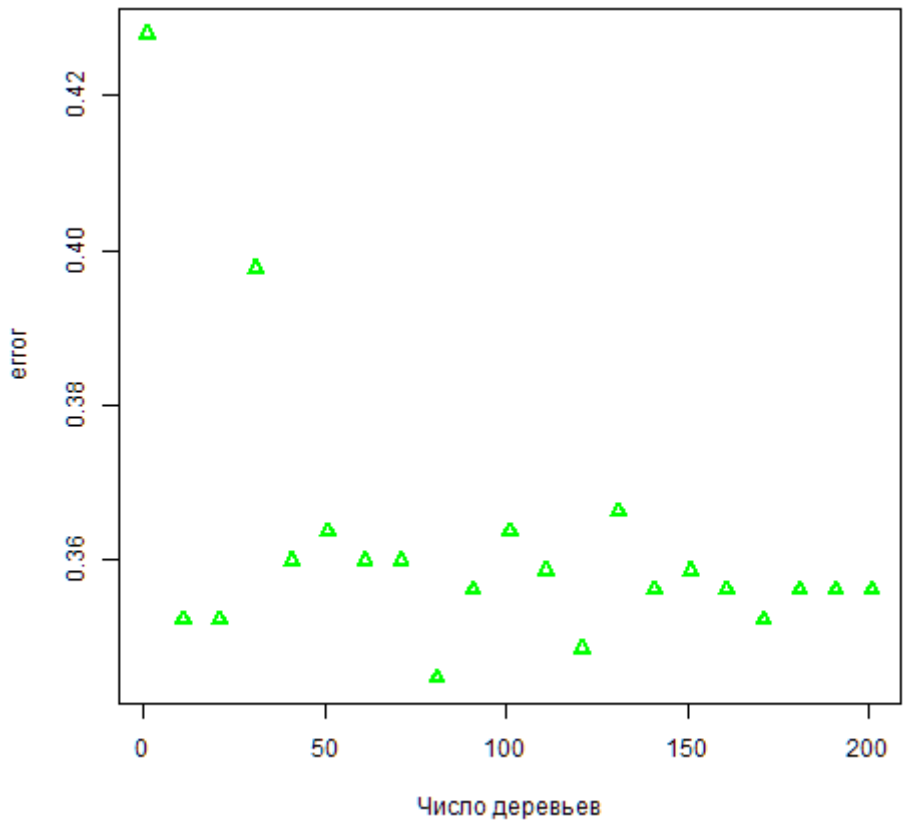

Рисунок 2. Зависимость тестовой ошибки от числа деревьев в ансамбле в методе bagging

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.

4. 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)
```