САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. ПЕТРА ВЕЛИКОГО

Институт прикладной математики и механики

Высшая школа прикладной математики и вычислительной физики

Отчёт

по лабораторной работе №7 «Алгоритмы композиции»
по дисциплине «Системы искусственного интеллекта»

Студентка гр. 3630201/70101 _____ О. В. Саксина

Преподаватель _____ Л. В. Уткин

Санкт-Петербург,
2021г.

Содержание

1 Задание 1

1.1 Постановка задачи

Исследуйте зависимость тестовой ошибки от количества деревьев в ансамбле для алгоритма adaboost.M1 на наборе данных Vehicle из пакета mlbench (обучающая выборка должна состоять из 7/10 всех прецедентов, содержащихся в данном наборе данных). Постройте график зависимости тестовой ошибки при числе деревьев, равном 1, 11, 21, . . . , 301, объясните полученные результаты.

1.2 Реализация

Результат представлен на рис. 1. При увеличении количества деревьев в ансамбле до 80, тестовая ошибка уменьшается. Затем колеблется в пределах от 0.225 до 0.3, принимая наименьшее значение на 100 деревьях.

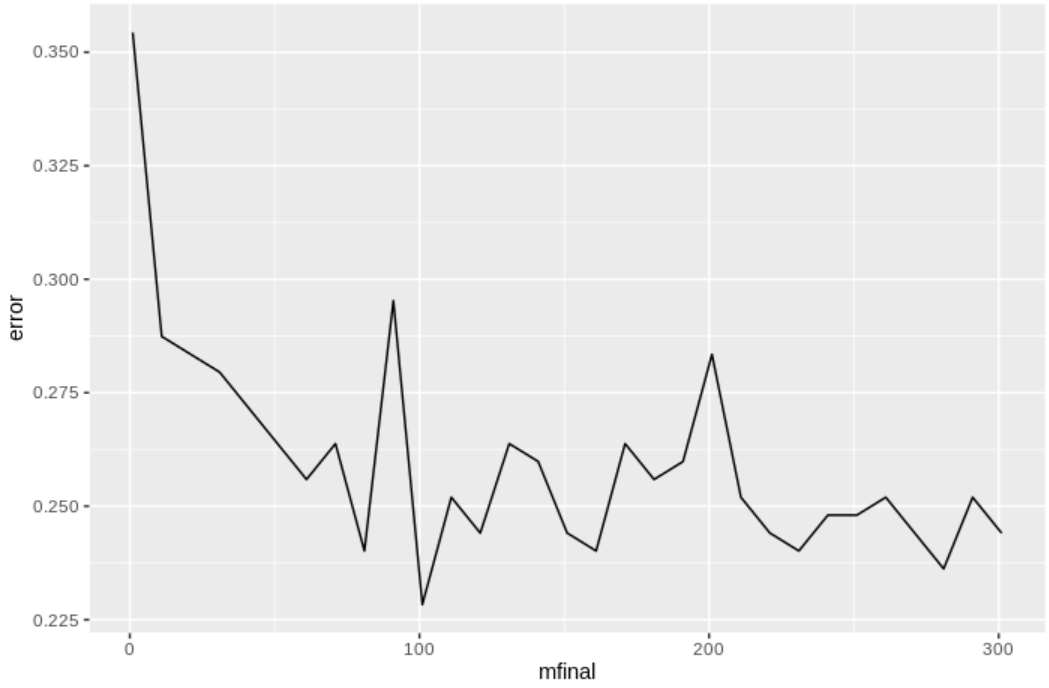

Рис. 1: Зависимость ошибки от количества деревьев в ансамбле для алгоритма adaboost

2 Задание 2

2.1 Постановка задачи

Исследуйте 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). Постройте график зависимости тестовой ошибки при числе деревьев, равном 1, 11, 21, . . . , 201, объясните полученные результаты.

2.2 Реализация

Результат представлен на рис. 2. При увеличении количества деревьев в ансамбле до 30 тестовая ошибка уменьшается. Затем колеблется в пределах от 0.24 до 0.29, принимая наименьшее значение на 70 деревьях.

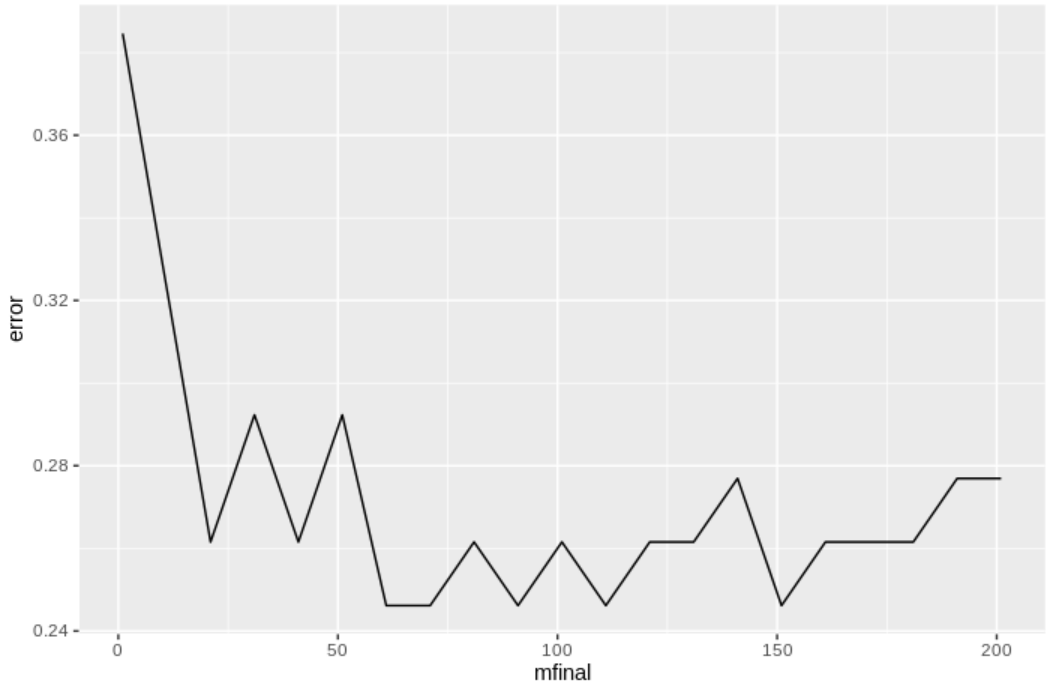

Рис. 2: Зависимость ошибки от количества деревьев в ансамбле для алгоритма bagging

3 Задание 3

3.1 Постановка задачи

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

3.2 Реализация

Описание алгоритма на естественном языке:

1. Присвоить каждому элементу из набора данных вес, равный 1.
2. Разделить данные на тестовые и обучающие.
3. Обучающие данные методом leave one out настроить следующим образом:
   (a) Найти k ближайших соседей для i-го элемента.
   (b) Определить количество голосов за каждый класс, сложив по отдельности веса соседей каждого класса. Присвоить i-му элементу класс, за который отдано больше всех голосов.
   (c) Если этот класс не совпадает с реальным классом i-го элемента, то для каждого соседа из k ближайших соседей увеличить на 0.5 вес тех, кто имеет реальный класс i-го элемента, и уменьшить на 0.5 вес тех, кто имеет назначенный класс i-го элемента.
   (d) Повторить шаги a-c 2 раза
4. Предсказать классы элементов из тестового набора путём поиска k ближайших соседей в обучающих данных с изменёнными весами.

Средняя тестовая ошибка данного классификатора на наборе данных Glass при k = 8 — 0.35. Средняя тестовая ошибка единичного дерева классификации глубины 4 — 0.31.

Средняя тестовая ошибка данного классификатора на наборе данных Vehicle при k = 8 — 0.41. Средняя тестовая ошибка единичного дерева классификации глубины 4 — 0.32.

Приложение

Задание 1

```
library(ggplot2)
library(rpart)
library(mlbench)
library(adabag)

data(Vehicle)
l = length(Vehicle[,1])
sub = sample(1:l,7*l/10)
mfinal = seq(1, 301, 10)
maxdepth = 5
res = rep(0, length(mfinal))
for(i in 1:length(mfinal)){
  Vehicle.adaboost = boosting(Class ~.,data=Vehicle[sub,], mfinal=mfinal[i], maxdepth=maxdepth)
  Vehicle.adaboost.pred = predict.boosting(Vehicle.adaboost, newdata=Vehicle[-sub, ])
  res[i] = Vehicle.adaboost.pred$error
}
df = data.frame(mfinal = mfinal, error = res)
ggplot(df, aes(x = mfinal, y = error)) +
  geom_line()
```

Задание 2

```
library(ggplot2)
library(rpart)
library(mlbench)
library(adabag)

data(Glass)
l = length(Glass[,1])
sub = sample(1:l,7*l/10)
mfinal = seq(1, 201, 10)
maxdepth = 5
res = rep(0, length(mfinal))
for(i in 1:length(mfinal)){
  Glass.adaboost = bagging(Type ~.,data=Glass[sub,], mfinal=mfinal[i], maxdepth=maxdepth)
  Glass.adaboost.pred = predict.bagging(Glass.adaboost, newdata=Glass[-sub, ])
  res[i] = Glass.adaboost.pred$error
}
df = data.frame(mfinal = mfinal, error = res)
ggplot(df, aes(x = mfinal, y = error)) +
  geom_line()
```

Задание 3

```
import numpy as np
from collections import Counter
import csv
import random
import copy
from sklearn.tree import DecisionTreeClassifier

def distance(instance1, instance2):
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)
    return np.linalg.norm(instance1 - instance2)

def get_distances(tr, te):
    train = copy.deepcopy(tr)
    test = copy.deepcopy(te)
    for i in range(len(te)):
```

```
        test[i].pop(-1)
    for i in range(len(tr)):
        train[i].pop(-1)
    distances = {}
    test_ind = 0
    for i in test:
        distances[test_ind] = []
        train_ind = 0
        for j in train:
            distances[test_ind].append([train_ind, distance(i, j)])
            train_ind += 1
        test_ind += 1
    return distances

def get_neighbors(train, train_labels, test_num, k, distances):
    #neighbour = (index, dist, class, weight)
    neighbors = []
    distances[test_num].sort(key=lambda x: x[1])
    neighbors = copy.deepcopy(distances[test_num][:k])
    for i in range(k):
        neighbors[i].append(train_labels[neighbors[i][0]])
        neighbors[i].append(train[neighbors[i][0]][-1])
    return(neighbors)

def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += neighbor[3]
    return class_counter.most_common(1)[0][0]

def add_weight(test_num, right_label, wrong_label, train, train_labels, neighbors):
    for n in neighbors:
        ind = n[0]
        if train_labels[ind] == right_label:
            train[ind][-1] += 0.5
        elif train_labels[ind] == wrong_label:
            train[ind][-1] -= 0.5

def read_data(file):
    data = []
    with open(file) as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            data.append(row)
    for row in range(1,len(data)):
        for el in range(len(data[row]) - 1):
            data[row][el] = float(data[row][el])
        try:
            data[row][-1] = int(float(data[row][-1]))
        except:
            pass
    return data

def knn(train0, main_train_labels):
    for s in range(len(train0)):
        print('\n____________________')
        print(str(s) + '___________________\n')
        k = 6
        train_labels = copy.deepcopy(main_train_labels)
        test = [train0[s]]
        test_labels = [train_labels[s]]
        if s == 0:
            train = train0[1:]
        elif s == len(train0):
            train = train0[:s]
        else:
```

```
            train = train0[:s] + train0[s+1:]
        d = get_distances(train, test)
        votes = []
        for test_ind in range(len(test)):
            neighbors = get_neighbors(train, train_labels, test_ind, k, d)
            votes.append(vote(neighbors))
        wrongs = {}
        for i in range(len(votes)):
            if votes[i] != test_labels[i]:
                wrongs[i] = (test_labels[i], votes[i])
                print(str(i) + ') predicted: ' + str(votes[i]) + ' real: ' + str(test_labels[i]))
        for i in wrongs:
            add_weight(i, wrongs[i][0], wrongs[i][1], train, train_labels, get_neighbors(train, train_labels, i, k+1, d))

        print('\n')
        if wrongs:
            k += 1
            votes = []
            for test_ind in range(len(test)):
                neighbors = get_neighbors(train, train_labels, test_ind, k, d)
                votes.append(vote(neighbors))
            wrongs = {}
            for i in range(len(votes)):
                if votes[i] != test_labels[i]:
                    wrongs[i] = (test_labels[i], votes[i])
                    print(str(i) + ') predicted: ' + str(votes[i]) + ' real: ' + str(test_labels[i]) )
            for i in wrongs:
                add_weight(i, wrongs[i][0], wrongs[i][1], train, train_labels, get_neighbors(train, train_labels, i, k+1, d))
            print('\n')

            if wrongs:
                k += 1
                votes = []
                for test_ind in range(len(test)):
                    neighbors = get_neighbors(train, train_labels, test_ind, k, d)
                    votes.append(vote(neighbors))
                wrongs = {}
                for i in range(len(votes)):
                    if votes[i] != test_labels[i]:
                        wrongs[i] = (test_labels[i], votes[i])
                        print(str(i) + ') predicted: ' + str(votes[i]) + ' real: ' + str(test_labels[i]) )
                for i in wrongs:
                    add_weight(i, wrongs[i][0], wrongs[i][1], train, train_labels, get_neighbors(train, train_labels, i, k+1, d
    return train0

def main():
    #data = read_data('glass.csv')
    data = read_data('vehicle_csv.csv')
    rows = len(data)
    data[0].insert(-1, "weight")
    for i in range(rows):
        data[i].insert(-1, 1)
    cols = len(data[0])
    names = data[0]
    data.pop(0)
    random.shuffle(data)
    train_size = round(0.7 * rows)
    train0 = data[:train_size]
    test0 = data[train_size:]
    main_train_labels = [row[-1] for row in train0]
    main_test_labels = [row[-1] for row in test0]
    for row in test0:
        row.pop(-1)
    for row in train0:
        row.pop(-1)
    #tree
```

```
    clf = DecisionTreeClassifier(max_depth = 4, random_state = 0)
    clf.fit(train0, main_train_labels)
    clf.predict(test0)
    score = clf.score(test0, main_test_labels)
    #knn
    train = knn(train0, main_train_labels)
    k = 8
    test = test0
    train_labels = copy.deepcopy(main_train_labels)
    test_labels = copy.deepcopy(main_test_labels)
    d = get_distances(train, test)
    votes = []
    for test_ind in range(len(test)):
        neighbors = get_neighbors(train, train_labels, test_ind, k, d)
        votes.append(vote(neighbors))
    wrongs = {}
    print('\n test \n')
    for i in range(len(votes)):
        if votes[i] != test_labels[i]:
            wrongs[i] = test_labels[i]
            print(str(i) + ') predicted: ' + str(votes[i]) + ' real: ' + str(test_labels[i]))
    print('\n knn missclasification = ' + str(len(wrongs)/len(test)))
    print('\n tree missclasification = ' + str(1 - score))


main()
```