Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Методы машинного обучения

Лабораторная работа №2

Решающие деревья

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: доцент кафедры ПМиК

Ракитский Антон Андреевич

Новосибирск, 2020

**Содержание**

1. Задание.
2. Текст программы.
3. Результаты работы
4. Вывод

**Задание**

Данная работа носит творческий характер и призвана показать,

насколько студент подготовлен к реальному применению полученных знаний

на практике. Как известно, в реальной работе никаких вводных данных не

предоставляется, тем не менее, мы слегка пренебрегли данным правилом и

предоставили теорию и предпочтительный метод для применения.

В приложенном файле (heart\_data.csv) располагаются реальные данные

по сердечной заболеваемости, собранные различными медицинскими

учреждениями. Каждый человек представлен 13-ю характеристиками и

полем goal, которое показывает наличие болезни сердца, поле принимает

значение 0 или 1 (0 – нет болезни, 1 - есть). Символ ‘?’ в каком-либо поле

означает, что для конкретного человека отсутствуют данные в этом поле

(либо не производились замеры, либо не записывались в базу).

Требуется имеющиеся данные разбить на обучающую и тестовую

выборки в процентном соотношении 70 к 30. После чего по обучающей

выборке необходимо построить решающее дерево. Для построения дерева

можно пользоваться любыми существующими средствами. Кроме того, для

построения дерева необходимо будет решить задачу выделения

информативных решающих правил относительно имеющихся числовых

признаков.

Разрешается использовать уже реализованные решающие деревья из

известных библиотек (например, scikit-learn для Python), либо реализовывать

алгоритм построения дерева самостоятельно (все необходимые алгоритмы

представлены в теории по ссылке).

В качестве результата работы необходимо сделать не менее 10

случайных разбиений исходных данных на обучающую и тестовую выборки,

для каждой построить дерево и протестировать, после чего построить

таблицу, в которой указать процент правильно классифицированных данных.

Полученную таблицу необходимо включить в отчёт по лабораторной работе.

В отчёте следует отразить следующие изменяемые параметры: глубина

дерева и количество деревьев для каждого тестируемого случая.

**Текст программы**

import csv

import math

import random

from collections import defaultdict

from collections import Counter

from typing import \*

import numpy as np

from sklearn import tree

from sklearn.impute import SimpleImputer

class DataItem:

def \_\_init\_\_(self, row: list):

self.attributes = [None] \* 13

for i in range(len(self.attributes)):

self.attributes[i] = None if row[i] == "?" else row[i]

self.final\_class = row[len(row) - 1]

def \_\_str\_\_(self):

return str(self.attributes) + ", " + str(self.final\_class)

class DecisionTree:

def \_\_init\_\_(self, train\_mas: List[DataItem], trees\_depth: int):

n\_attributes = len(train\_mas[0].attributes)

self.attr\_subset\_indexes = random.sample(range(n\_attributes), trees\_depth)

attribute\_subset = [subset\_by\_indexes(item.attributes, self.attr\_subset\_indexes) for item in train\_mas]

classes = [item.final\_class for item in train\_mas]

self.sk\_dec\_tree = tree.DecisionTreeClassifier()

self.sk\_dec\_tree.fit(attribute\_subset, classes)

def predict(self, X) -> list:

X = [subset\_by\_indexes(attrs, self.attr\_subset\_indexes) for attrs in X]

return self.sk\_dec\_tree.predict(X)

class decision\_forest:

def \_\_init\_\_(self, decision\_trees: List[DecisionTree]):

self.decision\_trees = decision\_trees

def predict(self, X) -> list:

vote\_table = zip(\*[dec\_tree.predict(X) for dec\_tree in self.decision\_trees])

return [Counter(votes).most\_common(1)[0][0] for votes in vote\_table]

def read\_csv\_file(file: str) -> List[DataItem]:

with (open(file)) as csv\_file:

reader = csv.reader(csv\_file)

next(reader)

return [DataItem(row) for row in reader]

def fill\_missing\_attributes(data: List[DataItem]) -> List[DataItem]:

imp = SimpleImputer(missing\_values=np.nan, strategy='mean')

attributes = [item.attributes for item in data]

imp = imp.fit\_transform(attributes)

return [DataItem([\*imp[i], data[i].final\_class]) for i in range(len(data))]

def initialize():

full\_data\_mas = read\_csv\_file('heart\_data.csv')

full\_data\_mas = fill\_missing\_attributes(full\_data\_mas)

return full\_data\_mas

def subset\_by\_indexes(l: list, indexes: List[int]) -> list:

return [l[i] for i in indexes]

def split\_data(data: List[DataItem], train\_mas\_ratio: float) -> Tuple[List[DataItem], List[DataItem]]:

n = len(data)

n\_training = int(n \* train\_mas\_ratio)

grouped\_by\_class = defaultdict(list)

for entry in data:

grouped\_by\_class[entry.final\_class].append(entry)

classes = list(grouped\_by\_class.keys())

train\_mas = []

test\_mas = []

class\_i = 0

for i in range(n):

class\_i = (class\_i + 1) % len(classes)

final\_class = classes[class\_i]

group = grouped\_by\_class[final\_class]

if len(group) == 0:

classes.remove(final\_class)

continue

group.pop

item = group.pop(random.randint(0, len(group) - 1))

target\_data = train\_mas if i < n\_training else test\_mas

target\_data.append(item)

return train\_mas, test\_mas

def main():

optimal\_iteration = 0

optimal\_trees = 0

optimal\_depth = 0

optimal\_average = 0

iterations = 1

while iterations <= 10:

average = 0

full\_data\_mas = initialize()

(train\_mas, test\_mas) = split\_data(full\_data\_mas, 0.7)

trees\_count = 10

while trees\_count <= 50:

trees\_depth = 2

while trees\_depth <= 8:

correct\_precisions = 0

decision\_trees = [DecisionTree(train\_mas, trees\_depth) for i in range(trees\_count)]

decision\_trees\_forest = decision\_forest(decision\_trees)

predicted: list = decision\_trees\_forest.predict([item.attributes for item in test\_mas])

for i in range(len(predicted)):

if predicted[i] == test\_mas[i].final\_class:

correct\_precisions = correct\_precisions + 1

print("Iteration:" + str(iterations) + " Trees:" + str(trees\_count) + " Depth:" + str(trees\_depth) +" Result:" + str(correct\_precisions / len(test\_mas)))

if correct\_precisions / len(test\_mas) > optimal\_average:

optimal\_iteration = iterations

optimal\_average = (correct\_precisions / len(test\_mas))

optimal\_trees = trees\_count

optimal\_depth = trees\_depth

average += (correct\_precisions / len(test\_mas))

trees\_depth += 1

trees\_count += 10

print("Average for Iteration" + str(iterations) + " " + str(average / 350))

iterations += 1

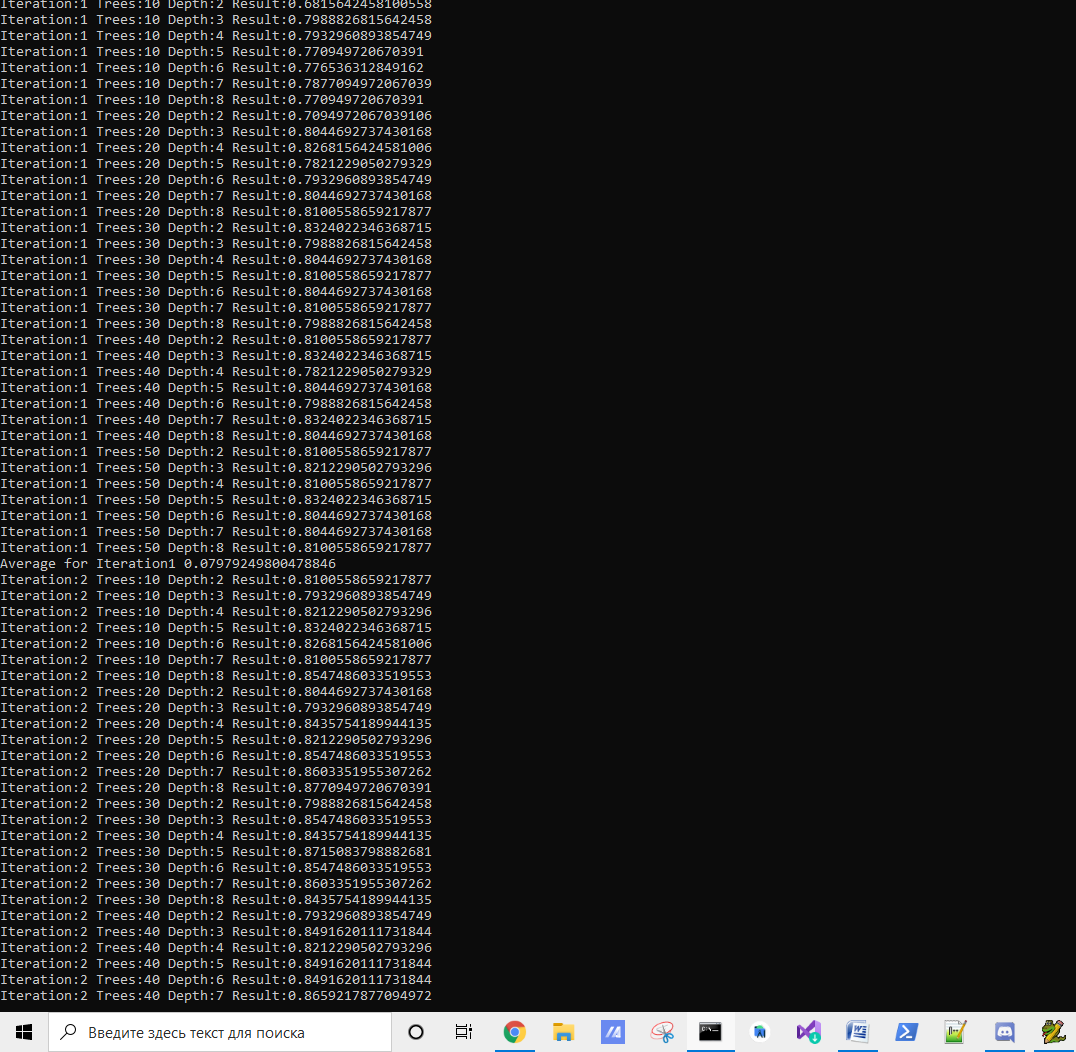
print("Optmal parametrs:" + "Iteration: " + str(optimal\_iteration) + " Trees: " + str(optimal\_trees) + " Depth: " + str(optimal\_depth) + " Precisions: " + str(optimal\_average))

if \_\_name\_\_ == '\_\_main\_\_':

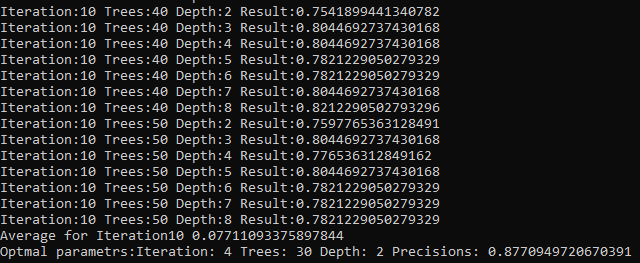
main()

exit()

**Результаты работы**



Оптимальные параметры:



**Вывод**

В результате практической работы были построены решающие деревья для решения задач классификации