Ушаков Владимир Александрович, группа МПиКИ

Лабораторная работа № 6

Деревья решений

Цель работы:

Целью лабораторной работы является изучение деревьев решений.

Код:

```
from dataclasses import dataclass
from enum import Enum


class Value(Enum):
    LOW = 1
    MEDIUM = 2
    HIGH = 3


@dataclass
class Client:
    id: int
    saving: Value
    assets: Value
    income: int
    risks: Value

    def __hash__(self):
        return self.id

    def __str__(self):
        return f'Client({self.id}: {self.risks})'


clients = [
    Client(1, Value.MEDIUM, Value.HIGH, 75, Value.LOW),
    # Client(2, Value.LOW, Value.LOW, 50, Value.HIGH),
    Client(3, Value.HIGH, Value.MEDIUM, 25, Value.HIGH),
    Client(4, Value.MEDIUM, Value.MEDIUM, 50, Value.LOW),
    Client(5, Value.LOW, Value.MEDIUM, 100, Value.LOW),
    Client(6, Value.HIGH, Value.HIGH, 25, Value.LOW),
    # Client(7, Value.LOW, Value.LOW, 25, Value.HIGH),
    Client(8, Value.MEDIUM, Value.MEDIUM, 75, Value.LOW),
]

conditions = [
    lambda x: x.saving == Value.LOW,
    lambda x: x.saving == Value.MEDIUM,
    lambda x: x.saving == Value.HIGH,
    lambda x: x.assets == Value.LOW,
    lambda x: x.assets == Value.MEDIUM,
    lambda x: x.assets == Value.HIGH,
    lambda x: x.income <= 25,
```

```
    lambda x: x.income <= 50,
    lambda x: x.income <= 75,
]

split = [
    [
        tuple([client, condition(client)])
        for client in clients
    ]
    for condition in conditions
]

Pl = [
    float(len([i for i in s if i[1] == False])) / len(s)
    for s in split
]

Pr = [
    float(len([i for i in s if i[1] == True])) / len(s)
    for s in split
]

Pjtl = [
    tuple([
        0 if len([i for i in s if i[1] == False]) == 0 else
        float(len(
            [i for i in s if i[0].risks == Value.LOW and i[1] == False]
        )) / len([i for i in s if i[1] == False]),
        0 if len([i for i in s if i[1] == False]) == 0 else
        float(len(
            [i for i in s if i[0].risks == Value.HIGH and i[1] == False]
        )) / len([i for i in s if i[1] == False])])
    for s in split
]

Pjtr = [
    tuple([
        0 if len([i for i in s if i[1] == True]) == 0 else
        float(len(
            [i for i in s if i[0].risks == Value.LOW and i[1] == True]
        )) / len([i for i in s if i[1] == True]),
        0 if len([i for i in s if i[1] == True]) == 0 else
        float(len(
            [i for i in s if i[0].risks == Value.HIGH and i[1] == True]
        )) / len([i for i in s if i[1] == True])
    ])
    for s in split
]

print(f'Pjtl: {Pjtl}')
print(f'Pjtr: {Pjtr}')
_2PlPr = [
    Pl[i] * Pr[i] * 2 for i in range(len(Pl))
]
print(f'2PlPr: {_2PlPr}')
W = [
    sum([abs(Pjtl[i][j] - Pjtr[i][j]) for j in range(2)])
    for i in range(len(Pjtl))
]
print(f'W: {W}')
Q = [_2PlPr[i] * W[i] for i in range(len(_2PlPr))]
print(f'Q: {Q}')
for tree in split:
    print('-' * 99)
```

```
    print([str(i[0]) for i in tree if i[1] == False])
    print([str(i[0]) for i in tree if i[1] == True])
```

Вывод программы:

Pjtl: [(0.8, 0.2), (0.6666666666666666, 0.3333333333333333), (1.0, 0.0), (0.8333333333333334, 0.16666666666666666), (1.0, 0.0), (0.75, 0.25), (1.0, 0.0), (1.0, 0.0), (1.0, 0.0)]

Pjtr: [(1.0, 0.0), (1.0, 0.0), (0.5, 0.5), (0, 0), (0.75, 0.25), (1.0, 0.0), (0.5, 0.5), (0.6666666666666666, 0.3333333333333333), (0.8, 0.2)]

2PlPr: [0.2777777777777778, 0.5, 0.4444444444444444, 0.0, 0.4444444444444444, 0.4444444444444444, 0.4444444444444444, 0.5, 0.2777777777777778]

W: [0.39999999999999997, 0.6666666666666667, 1.0, 1.0, 0.5, 0.5, 1.0, 0.6666666666666667, 0.39999999999999997]

Q: [0.1111111111111111, 0.33333333333333337, 0.4444444444444444, 0.0, 0.2222222222222222, 0.2222222222222222, 0.4444444444444444, 0.33333333333333337, 0.1111111111111111]

['Client(1: Value.LOW)', 'Client(3: Value.HIGH)', 'Client(4: Value.LOW)', 'Client(6: Value.LOW)', 'Client(8: Value.LOW)']

['Client(5: Value.LOW)']

['Client(3: Value.HIGH)', 'Client(5: Value.LOW)', 'Client(6: Value.LOW)']

['Client(1: Value.LOW)', 'Client(4: Value.LOW)', 'Client(8: Value.LOW)']

['Client(1: Value.LOW)', 'Client(4: Value.LOW)', 'Client(5: Value.LOW)', 'Client(8: Value.LOW)']

['Client(3: Value.HIGH)', 'Client(6: Value.LOW)']

['Client(1: Value.LOW)', 'Client(3: Value.HIGH)', 'Client(4: Value.LOW)', 'Client(5: Value.LOW)', 'Client(6: Value.LOW)', 'Client(8: Value.LOW)']

[]

['Client(1: Value.LOW)', 'Client(6: Value.LOW)']

['Client(3: Value.HIGH)', 'Client(4: Value.LOW)', 'Client(5: Value.LOW)', 'Client(8: Value.LOW)']

['Client(3: Value.HIGH)', 'Client(4: Value.LOW)', 'Client(5: Value.LOW)', 'Client(8: Value.LOW)']

['Client(1: Value.LOW)', 'Client(6: Value.LOW)']

['Client(1: Value.LOW)', 'Client(4: Value.LOW)', 'Client(5: Value.LOW)', 'Client(8: Value.LOW)']

['Client(3: Value.HIGH)', 'Client(6: Value.LOW)']

['Client(1: Value.LOW)', 'Client(5: Value.LOW)', 'Client(8: Value.LOW)']

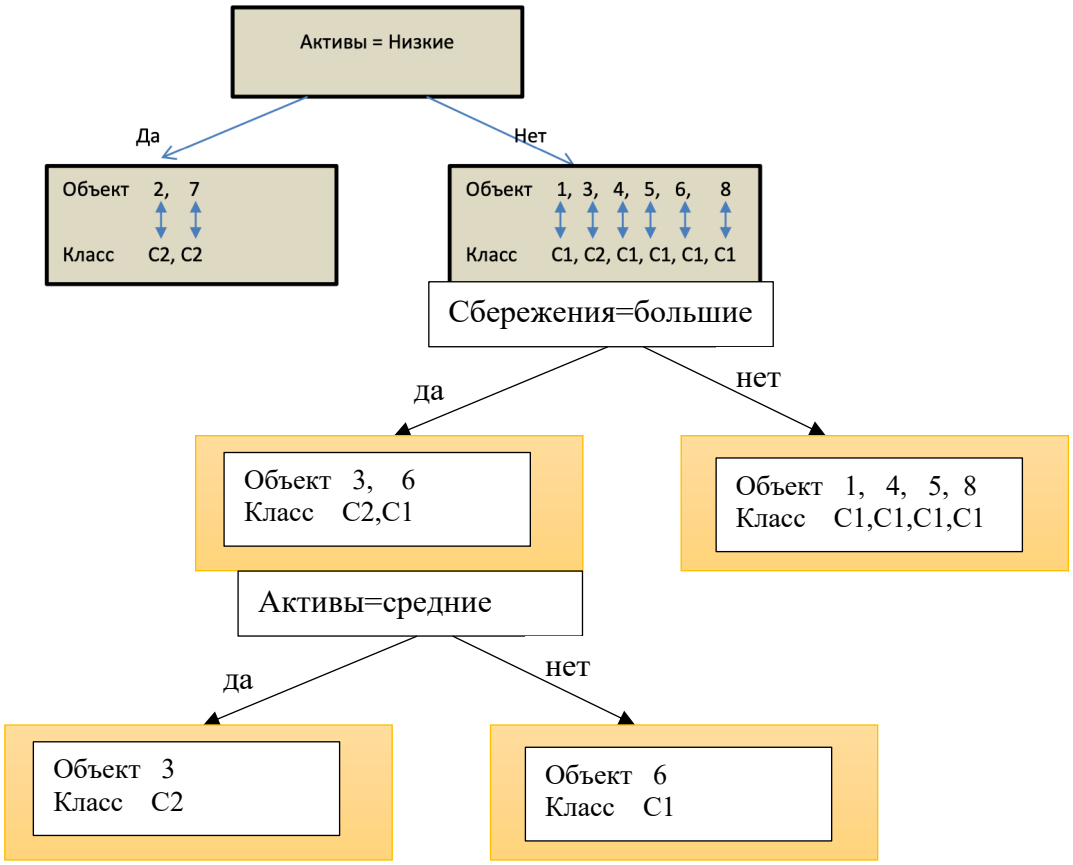
['Client(3: Value.HIGH)', 'Client(4: Value.LOW)', 'Client(6: Value.LOW)']

['Client(5: Value.LOW)']

['Client(1: Value.LOW)', 'Client(3: Value.HIGH)', 'Client(4: Value.LOW)', 'Client(6: Value.LOW)', 'Client(8: Value.LOW)']

Согласно полученным значениям оптимальным разбиением будут 3 и 7. Они равносильны, так как в обоих случаях приводят к клиентам 3 и 6. Для завершения

разбиения потребуется дополнительно еще одно разбиение, которое просто логически выбирается по оставшимся клиентам – это будет разбиение 5 по Активы = средние
В результате дерево выглядит следующим образом:



Вывод:

В результате лабораторной работы было достроено бинарное дерево решений при помощи алгоритма CART