НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Системы искусственного интеллекта

Лабораторная работа № 3

Выполнил студент

Кузнецов Максим

Группа № P33131

Преподаватель: Авдюшина Анна Евгеньевна

г. Санкт-Петербург

2022

**Задание:**

1. Датасет с классификацией грибов -- https://archive.ics.uci.edu/ml/datasets/Mushroom
2. Отобрать случайным образом sqrt(n) признаков
3. Реализовать без использования сторонних библиотек построение дерева решений (numpy и pandas использовать можно)
4. Провести оценку реализованного алгоритма с использованием Accuracy, precision и recall
5. Построить AUC-ROC и AUC-PR

**Код программы, реализующий данную задачу:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_recall\_curve, roc\_curve, auc, average\_precision\_score, precision\_score, recall\_score, RocCurveDisplay, PrecisionRecallDisplay

import matplotlib.pyplot as plt

column\_names = {

'1': 'cap-shape',

'2': 'cap-surface',

'3': 'cap-color',

'4': 'bruises?',

'5': 'odor',

'6': 'gill-attachment',

'7': 'gill-spacing',

'8': 'gill-size',

'9': 'gill-color',

'10': 'stalk-shape',

'11': 'stalk-root',

'12': 'stalk-surface-above-ring',

'13': 'stalk-surface-below-ring',

'14': 'stalk-color-above-ring',

'15': 'stalk-color-below-ring',

'16': 'veil-type',

'17': 'veil-color',

'18': 'ring-number',

'19': 'ring-type',

'20': 'spore-print-color',

'21': 'population',

'22': 'habitat'

}

def draw\_plt(predict\_arr, expect\_arr):

y\_true = np.array([0 if x == 'p' else 1 for x in predict\_arr])

y\_score = np.array([0 if x == 'p' else 1 if x == 'e' else '-1' for x in expect\_arr])

fpr, tpr, \_ = roc\_curve(y\_true, y\_score)

roc\_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()

precision, recall, \_ = precision\_recall\_curve(y\_true, y\_score)

pr\_display = PrecisionRecallDisplay(precision=precision, recall=recall).plot()

auc\_roc = auc(fpr, tpr)

auc\_pr = average\_precision\_score(y\_true, y\_score)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 8))

roc\_display.plot(ax=ax1)

pr\_display.plot(ax=ax2)

plt.show()

def analyze(predict\_arr, expect\_arr):

print("Recall:", recall\_score(expect\_arr, predict\_arr, average="micro"))

print("Accuracy:", accuracy\_score(expect\_arr, predict\_arr))

print("Precision:", precision\_score(expect\_arr, predict\_arr, average="micro"))

def get\_entropy(names):

entropy = 0

for name in set(names):

p = names[names == name].shape[0] / names.shape[0]

entropy -= p \* np.log2(p)

return entropy

def get\_condition\_entropy(names, cl):

entropy = 0

for name in set(names):

c = cl[names == name]

tmp\_ent = get\_entropy(c)

p = c.shape[0] / cl.shape[0]

entropy += p \* tmp\_ent

return entropy

def get\_split\_info(names, cl):

info = 0

for name in set(names):

c = cl[names == name]

p = c.shape[0] / cl.shape[0]

info -= p \* np.log2(p)

return info

def gain\_info(names, cl):

return (get\_entropy(names) - get\_condition\_entropy(names, cl)) / (get\_split\_info(names, cl)+0.0000001)

def built\_tree(names, cl, depth=1):

if len(set(cl)) == 1:

return cl[0]

class\_1, class\_2 = set(cl)

max = class\_1 if np.sum(cl == class\_1) > np.sum(cl == class\_2) else class\_2

if depth > 10:

return max

if len(cl) < 1:

return max

if len(names[0]) == 0:

return max

index\_best = 0

max\_gain = 0

for name\_index in range(len(names[0])):

gain = gain\_info(names[:, name\_index], cl)

if max\_gain < gain:

max\_gain = gain

index\_best = name\_index

T = {}

sub\_T = {}

for best\_name in set(names[:, index\_best]):

sub\_y = cl[names[:, index\_best] == best\_name]

sub\_X = names[names[:, index\_best] == best\_name]

sub\_X = np.delete(sub\_X, index\_best, 1)

sub\_T[best\_name + "--" + str(len(sub\_X))] = built\_tree(sub\_X, sub\_y, depth + 1)

T[cols\_name[index\_best] + "--" + str(len(names))] = sub\_T

return T

def make\_pred(x, tree=None):

if x.ndim == 2:

res = []

for x\_element in x:

res.append(make\_pred(x\_element))

return res

global main\_tree

if not tree:

tree = main\_tree

key = list(tree.keys())[0]

name = key.split("--")[0]

try:

index = cols\_name.index(name)

except ValueError:

return '-'

curr\_tree = tree[key]

for k in curr\_tree.keys():

if k.split("--")[0] == x[index]:

key = k

curr\_tree = curr\_tree[key]

if type(curr\_tree) == dict:

return make\_pred(x, curr\_tree)

else:

return curr\_tree

data = pd.read\_csv("agaricus-lepiota.data", header=None)

Class = data.iloc[:, 0]

Names = data.iloc[:, 1:]

Names = Names.sample(n=5, axis=1)

cols = Names.columns.tolist()

cols\_new = [column\_names.get(str(x)) for x in cols]

global cols\_name

global main\_tree

main\_tree = dict()

cols\_name = cols\_new

Names = Names.values

Class = Class.values

Names\_train, Names\_test, Class\_train, Class\_test = train\_test\_split(Names, Class, random\_state=0)

main\_tree = built\_tree(Names\_train, Class\_train)

predict = make\_pred(Names\_test)

print(main\_tree)

print("Выбранные признаки:")

print(cols\_new)

print("Предсказанный:")

print(predict)

print("Ожидаемый:")

print(Class\_test.tolist())

analyze(predict, Class\_test)

draw\_plt(predict, Class\_test)

Пример вывода:Изображение выглядит как текст

Автоматически созданное описание



**Вывод** -- в результате данной работы я:

* Познакомился с методом классификации «деревья решений»
* Применил знания математики для написания алгоритма C4.5, улучшенную надстройку