МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**Факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №3**

по дисциплине

“Системы искусственного интеллекта”

Деревья решений

Вариант №8

**Студент:**

Воробьев Кирилл Олегович

            Группа P33302

**Преподаватель:**

Королёва Ю.А.



Санкт-Петербург, 2022

**Цель работы:**

Исследование алгоритмов решения задач методом поиска

**Задание:**

1. Для студентов с четным порядковым номером в группе – датасет с классификацией грибов, а нечетным – датасет с данными про оценки студентов инженерного и педагогического факультетов (для данного датасета нужно ввести метрику: студент успешный /неуспешный на основании грейда)
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

**Выполнение работы:**

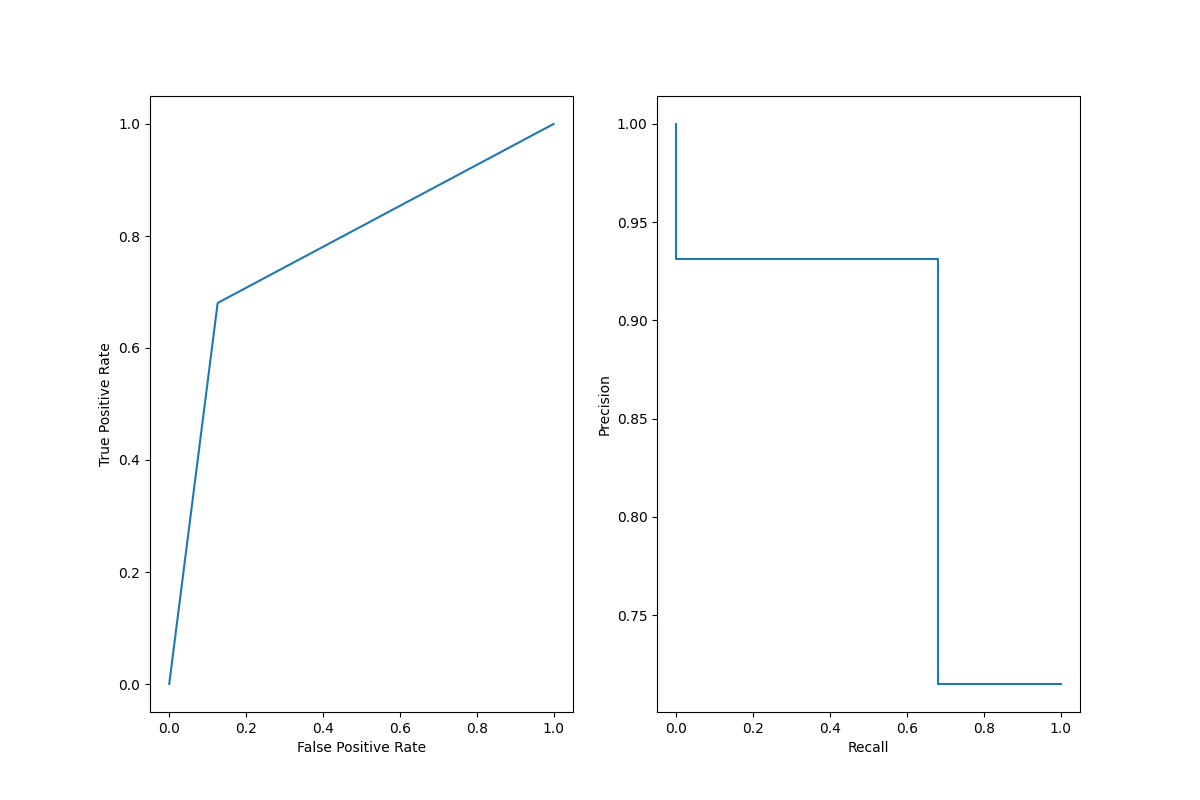
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, precision\_recall\_curve, roc\_curve, auc, average\_precision\_score, \  
 precision\_score, recall\_score, RocCurveDisplay, PrecisionRecallDisplay  
  
dict\_num\_name = {'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 drawing\_graph(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("accuracy\_score:", accuracy\_score(expect\_arr, predict\_arr))  
 print("precision\_score:", precision\_score(expect\_arr, predict\_arr, average='micro'))  
 print("recall\_score:", recall\_score(expect\_arr, predict\_arr, average='micro'))  
  
  
class DecisionTree:  
 def \_\_init\_\_(self, feature\_names, eps=0.03, depth=10, min\_leaf\_size=1):  
 self.tree = dict()  
 self.feature\_names = feature\_names  
 self.eps = eps  
 self.depth = depth  
 self.min\_leaf\_size = min\_leaf\_size  
  
 def get\_entropy(self, x):  
 entropy = 0  
 for x\_value in set(x):  
 p = x[x == x\_value].shape[0] / x.shape[0]  
 entropy -= p \* np.log2(p)  
 return entropy  
  
 def get\_condition\_entropy(self, x, y):  
 entropy = 0  
 for x\_value in set(x):  
 sub\_y = y[x == x\_value]  
 tmp\_ent = self.get\_entropy(sub\_y)  
 p = sub\_y.shape[0] / y.shape[0]  
 entropy += p \* tmp\_ent  
 return entropy  
  
 def information\_gain(self, x, y):  
 return 1 - self.get\_condition\_entropy(x, y) / (self.get\_entropy(x) + 0.00000000001)  
  
 def fit(self, X, y):  
 self.tree = self.\_built\_tree(X, y)  
  
 def \_built\_tree(self, X, y, depth=1):  
 if len(set(y)) == 1:  
 return y[0]  
 label\_1, label\_2 = set(y)  
 max\_label = label\_1 if np.sum(y == label\_1) > np.sum(y == label\_2) else label\_2  
 if len(X[0]) == 0:  
 return max\_label  
 if depth > self.depth:  
 return max\_label  
 if len(y) < self.min\_leaf\_size:  
 return max\_label  
 best\_feature\_index = 0  
 max\_gain = 0  
 for feature\_index in range(len(X[0])):  
 gain = self.information\_gain(X[:, feature\_index], y)  
 if max\_gain < gain:  
 max\_gain = gain  
 best\_feature\_index = feature\_index  
 if max\_gain < self.eps:  
 return max\_label  
 T = {}  
 sub\_T = {}  
 for best\_feature in set(X[:, best\_feature\_index]): # Берем список уникальных значений столбца с лучшим gain  
 sub\_y = y[X[:, best\_feature\_index] == best\_feature]  
 sub\_X = X[X[:, best\_feature\_index] == best\_feature]  
 sub\_X = np.delete(sub\_X, best\_feature\_index, 1)  
 sub\_T[best\_feature + "\_\_\_" + str(len(sub\_X))] = self.\_built\_tree(sub\_X, sub\_y, depth + 1)  
 T[self.feature\_names[best\_feature\_index] + "\_\_\_" + str(len(X))] = sub\_T  
 return T  
  
 def predict(self, x, tree=None):  
 if x.ndim == 2:  
 res = []  
 for x\_ in x:  
 res.append(self.predict(x\_))  
 return res  
 if not tree:  
 tree = self.tree  
 tree\_key = list(tree.keys())[0]  
 x\_feature = tree\_key.split("\_\_\_")[0]  
 try:  
 x\_index = self.feature\_names.index(x\_feature)  
 except ValueError:  
 return '?'  
 x\_tree = tree[tree\_key]  
 for key in x\_tree.keys():  
 if key.split("\_\_\_")[0] == x[x\_index]:  
 tree\_key = key  
 x\_tree = x\_tree[tree\_key]  
 if type(x\_tree) == dict:  
 return self.predict(x, x\_tree)  
 else:  
 return x\_tree  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data = pd.read\_csv("agaricus-lepiota.data", header=None)  
 Y = data.iloc[:, 0]  
 X = data.iloc[:, 1:]  
 X = X.sample(n=5, axis=1)  
 cols = X.columns.tolist()  
 cols\_new = [dict\_num\_name.get(str(x)) for x in cols]  
 X = X.values  
 Y = Y.values  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3,

random\_state=0)  
 clf = DecisionTree(feature\_names=cols\_new, eps=0.03)  
 clf.fit(X\_train, Y\_train)  
 predict = clf.predict(X\_test)  
 print(clf.tree)  
 print("Selected Features:")  
 print(cols\_new)  
 print("Predict:")  
 print(predict)  
 print("Expect:")  
 print(Y\_test.tolist())  
 analyze(predict, Y\_test)  
 drawing\_graph(predict, Y\_test)

**Вывод программы:**

**Изображение выглядит как текст

Автоматически созданное описание**

****

**Вывод:**

В ходе данной лабораторной работы я познакомился с Python-библиотеками для классификации, а также научился строить дерево разбиения на классификации.