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()
```

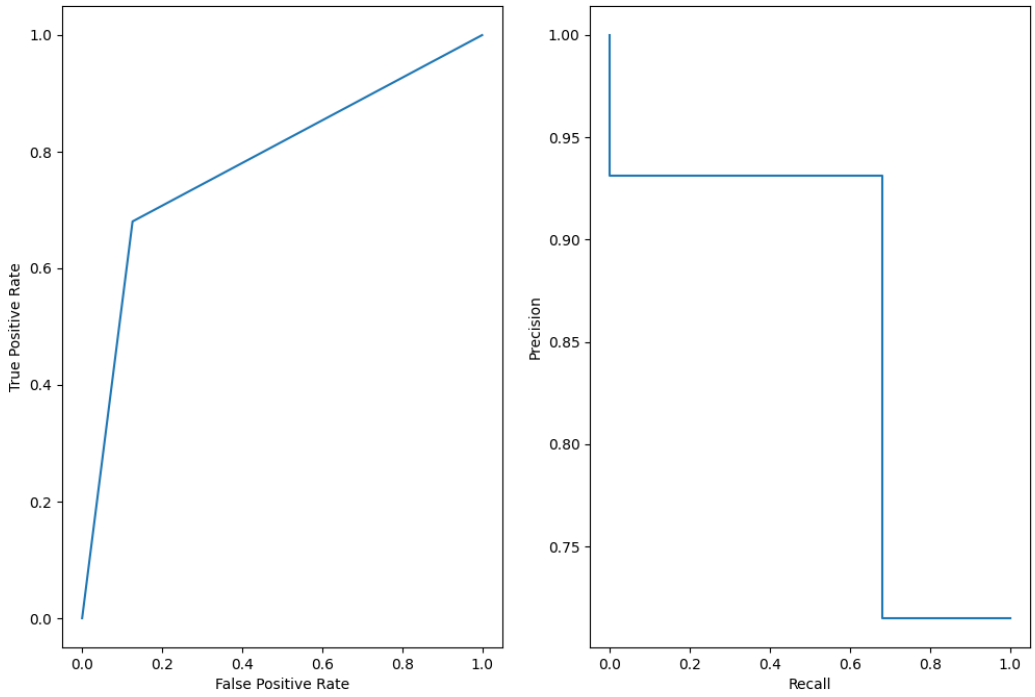
```
    auc_roc = auc(fpr, tpr)
    auc_pr = average_precision_score(y_true, y_score)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 8))
    roc_display.plot(ax=ax1)
    pr_display.plot(ax=ax2)
    plt.show()


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:

```
{'cap-color___6093': {'g___1404': {'stalk-color-above-ring___1404': {'g___139': 'e', 'p___314': 'p', 'b___177': 'p', 'w___620': {'stalk-root___62
Selected Features:
['gill-spacing', 'stalk-root', 'gill-attachment', 'stalk-color-above-ring', 'cap-color']
Predict:
['p', 'e', 'e', 'e', 'e', 'e', 'p', '?', 'e', 'p', 'e', '?', 'e', 'e', 'e', 'p', 'e', 'p', 'e', 'e', 'p', 'p', 'p', '?', 'p', 'p', 'p', 'e', 'e',
Expect:
['p', 'e', 'e', 'e', 'e', 'e', 'p', 'p', 'e', 'p', 'e', 'e', 'e', 'e', 'e', 'p', 'e', 'p', 'e', 'e', 'p', 'p', 'p', 'p', 'p', 'p', 'p', 'e', 'e',
accuracy_score: 0.6976858690300345
precision_score: 0.6976858690300345
recall_score: 0.6976858690300345
```



Вывод:

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