Министерство науки и высшего образования РФ

Национальный исследовательский университет ИТМО

Факультет Программной инженерии и компьютерных технологий

По дисциплине:

Системы искусственного интеллекта

Лабораторная работа №3.

«Древо решений»

Вариант: Нечетный

Выполнил: До Зыонг Мань

Группа: P33201

Санкт–Петербург

2022 год.

I. 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.

II. Решение

Полный код можно посмотреть по ссылке

1.Реализовать построения дерева решений

Я выбираю порог в 4 балла для оценки результатов обучения студента. То есть, если GRADE меньше или равен 4, он будет оцениваться как «not passed», а более 4 будет оцениваться как «passed».

```
data = pd.read_csv("DATA.csv",index_col=0, sep=';')
data['Decision'] = data['GRADE'].apply(lambda x:"not passed" if x<=4 else "passed")
X = data.iloc[:,:-2]
Y = data.iloc[:,-1]
```

Отобрать случайным образом sqrt(n) признаков**.**

Используйте библиотеку *train_test_split()*, чтобы разделить набор данных в соотношении 80:20. То есть 80% данных будет тренировочным набором, а остальные 20% — тестовым.

```
X = X.sample(n=6,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=0.2,random_state = 3)
```

2.Реализовать построения дерева решений

** Код для построения дерева решений*

```
class TreeNode:
    def __init__(self, feature_names, eps = 0.03,depth =10, min_leaf_size = 1):
        self.children = {}
        self.decision = None
        self.feature_names = feature_names
        self.split_feature_name = None
        self.split_feature_index = None
        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 information_gain(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 self.get_entropy(y) - entropy

    def fit(self,X,y):
        self._built_tree(X,y,0)

    def getMajorClass(self,y):
        sum = 0
        result = None
        for x in set(y):
            if np.sum(y == x) > sum:
                sum = np.sum(y == x)
                result = x
        return result
```

```
    def _built_tree(self,X,y,depth):
        if(len(set(y))) == 1:
            self.decision = y[0]
            return
        if len(X[0]) == 0:
            self.decision = self.getMajorClass(y)
            return
        if depth > self.depth:
            self.decision = self.getMajorClass(y)
            return
        if len(y) < self.min_leaf_size:
            self.decision = self.getMajorClass(y)
            return

        self.children = {}
        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:
            self.decision = self.getMajorClass(y)
            return

        self.split_feature_name = self.feature_names[best_feature_index]
        self.split_feature_index = cols.index(self.split_feature_name)
```

```
        for best_feature in set(X[:, best_feature_index]):
            index = X[:,best_feature_index] == best_feature
            sub_X = X[index]
            sub_X = np.delete(sub_X,best_feature_index,1)
            sub_col = np.delete(self.feature_names,best_feature_index)
            #print(sub_X.shape)
            if len(X[index]) > 0:
                self.children[best_feature] = TreeNode(feature_names=sub_col)
                self.children[best_feature]._built_tree(sub_X,y[index],depth+1)
            else:
                self.children[best_feature] = TreeNode(feature_names=sub_col,depth=1)
                self.children[best_feature]._built_tree(sub_X,y[index],depth+1)

    def predict(self, x):
        if self.decision is not None:
            return self.decision
        else:
            attr_val = x[self.split_feature_index]
            try:
                child = self.children[attr_val]
            except KeyError:
                return '?'
            return child.predict(x)

    def pretty_print(self,prefix=''):
        if self.split_feature_name is not None:
            for k,v in self.children.items():
                v.pretty_print(f"{prefix}:When {self.split_feature_name} is {k}")
        else:
            print(f"{prefix}:{self.decision}")
```

** Пример дерева решения*

```
:When COURSE ID is 1:When 13 is 1:not passed
:When COURSE ID is 1:When 13 is 2:When 25 is 2:not passed
:When COURSE ID is 1:When 13 is 2:When 25 is 3:When 22 is 1:When 5 is 1:not passed
:When COURSE ID is 1:When 13 is 2:When 25 is 3:When 22 is 1:When 5 is 2:not passed
:When COURSE ID is 1:When 13 is 2:When 25 is 3:When 22 is 2:not passed
:When COURSE ID is 1:When 13 is 3:When 30 is 1:not passed
:When COURSE ID is 1:When 13 is 3:When 30 is 2:When 22 is 1:When 25 is 2:passed
:When COURSE ID is 1:When 13 is 3:When 30 is 2:When 22 is 1:When 25 is 3:not passed
:When COURSE ID is 1:When 13 is 3:When 30 is 2:When 22 is 2:not passed
:When COURSE ID is 1:When 13 is 3:When 30 is 3:When 22 is 1:When 25 is 2:passed
:When COURSE ID is 1:When 13 is 3:When 30 is 3:When 22 is 1:When 25 is 3:passed
:When COURSE ID is 1:When 13 is 3:When 30 is 3:When 22 is 2:not passed
:When COURSE ID is 1:When 13 is 3:When 30 is 4:not passed
:When COURSE ID is 1:When 13 is 4:When 22 is 1:not passed
:When COURSE ID is 1:When 13 is 4:When 22 is 2:When 30 is 3:not passed
:When COURSE ID is 1:When 13 is 4:When 22 is 2:When 30 is 4:passed
:When COURSE ID is 1:When 13 is 5:When 30 is 1:not passed
:When COURSE ID is 1:When 13 is 5:When 30 is 2:passed
:When COURSE ID is 1:When 13 is 5:When 30 is 3:not passed
:When COURSE ID is 1:When 13 is 5:When 30 is 4:When 25 is 1:not passed
:When COURSE ID is 1:When 13 is 5:When 30 is 4:When 25 is 2:passed
:When COURSE ID is 1:When 13 is 5:When 30 is 4:When 25 is 3:passed
:When COURSE ID is 2:passed
:When COURSE ID is 3:passed
:When COURSE ID is 4:When 13 is 2:passed
:When COURSE ID is 4:When 13 is 3:not passed
:When COURSE ID is 4:When 13 is 4:not passed
:When COURSE ID is 5:When 13 is 2:not passed
:When COURSE ID is 5:When 13 is 3:passed
:When COURSE ID is 5:When 13 is 4:not passed
:When COURSE ID is 5:When 13 is 5:not passed
:When COURSE ID is 6:When 5 is 1:When 13 is 4:not passed
:When COURSE ID is 6:When 5 is 1:When 13 is 5:passed
:When COURSE ID is 6:When 5 is 2:passed
:When COURSE ID is 7:When 30 is 1:not passed
:When COURSE ID is 7:When 30 is 2:passed
:When COURSE ID is 7:When 30 is 3:passed
:When COURSE ID is 7:When 30 is 4:passed
:When COURSE ID is 8:not passed
```

* *Пример предсказания результата*

Поскольку мое дерево решений построено на дискретных переменных, будут случаи, когда оно не даст результата. В таких случаях я бы присвоил ему '?'

```
0     not passed
1     not passed
2     not passed
3     not passed
4     not passed
5         passed
6         passed
7     not passed
8     not passed
9     not passed
10    not passed
11    not passed
12    not passed
13        passed
14        passed
15    not passed
16    not passed
17    not passed
18    not passed
19        passed
20    not passed
21        passed
22        passed
23    not passed
24    not passed
25             ?
26        passed
27    not passed
28    not passed
```

3. Провести оценку реализованного алгоритма сиспользованием Accuracy, precision и recall.

```
def analyze(predict, expect):
    print("accuracy_score:", accuracy_score(expect,predict))
    print("precision_score:", precision_score(expect,predict, average="micro"))
    print("recall_score:", recall_score(expect,predict,average="micro"))
```

```
accuracy_score: 0.5862068965517241
precision_score: 0.5862068965517241
recall_score: 0.5862068965517241
```

4.Построить AUC-ROC и AUC-PR.

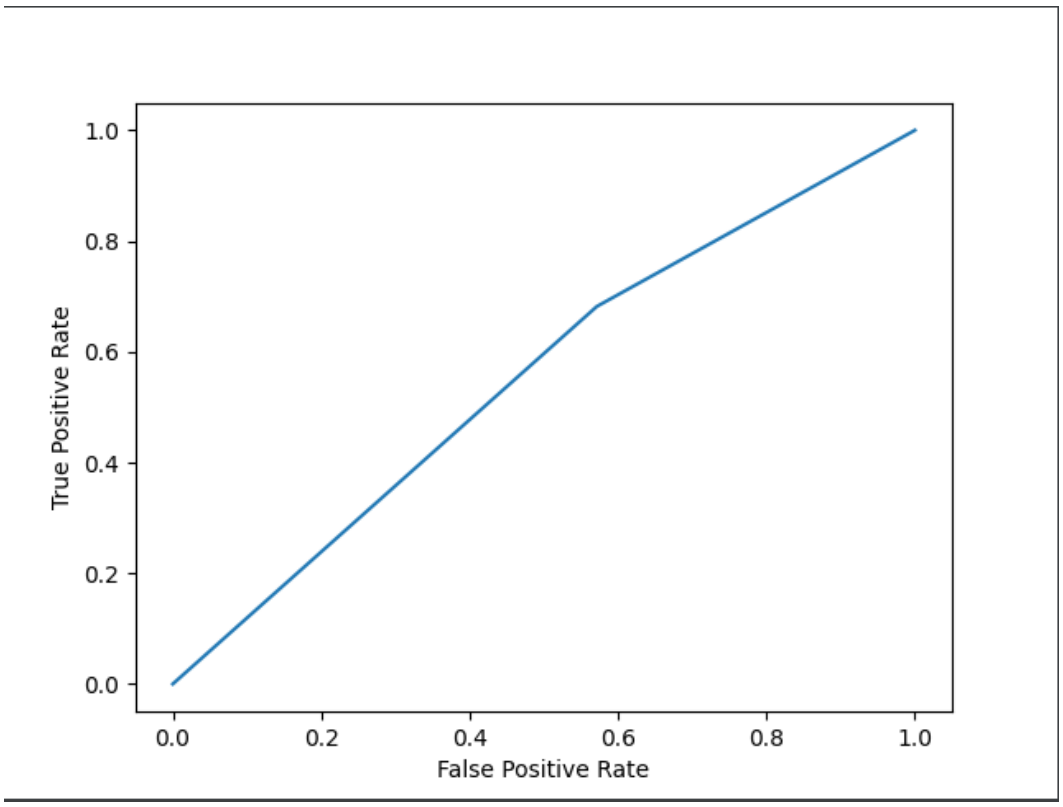
Чтобы визуализировать сравнение качества двух моделей дерева решений, я построил два графика AUC-ROC и AUC-PR.

* *Код для построения AUC-ROC и AUC-PR*

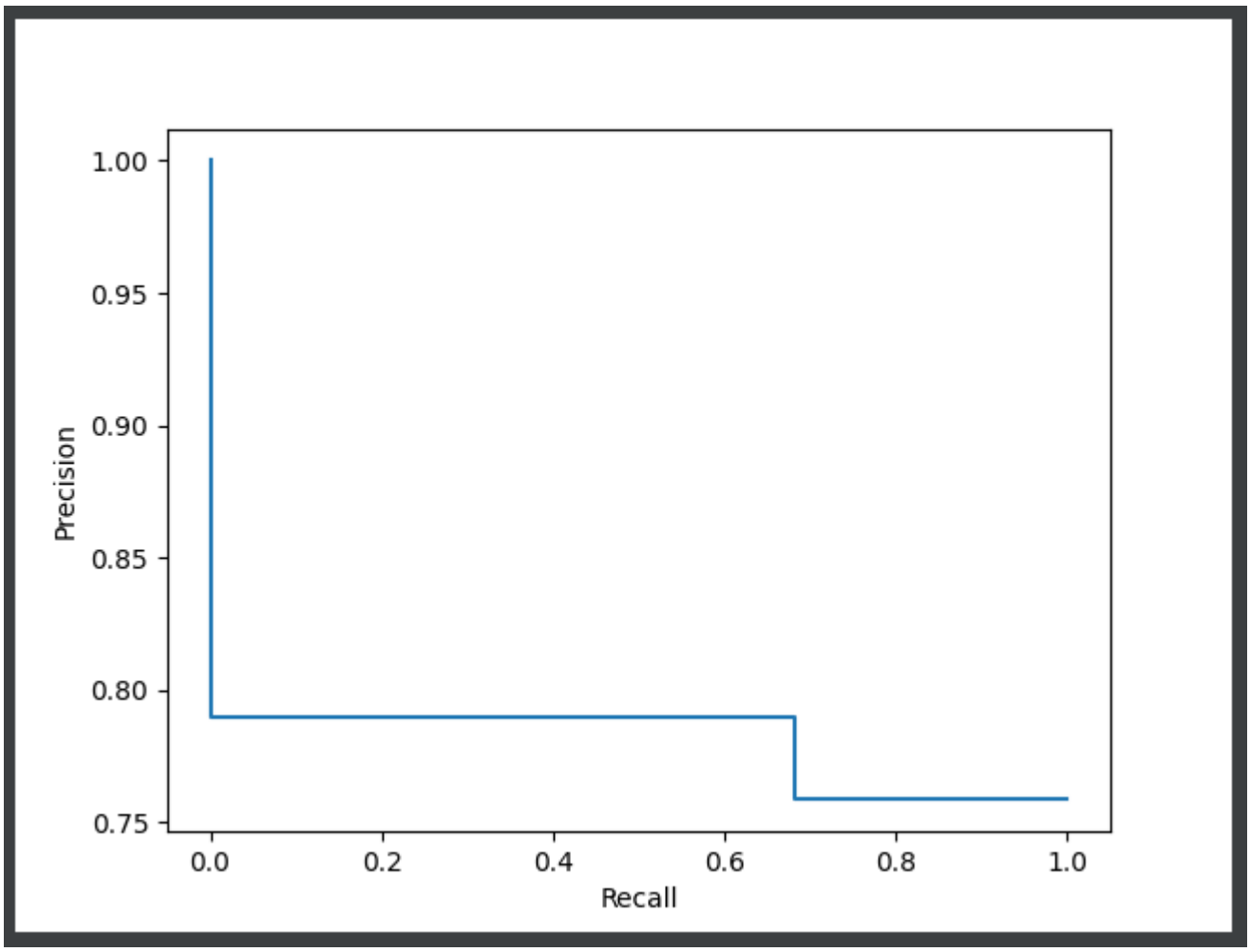
```
def draw_plt(predict_arr, expect_arr):
    y_true = np.array([0 if x == 'passed' else 1 for x in predict_arr])
    y_score = np.array([0 if x == 'passed' else 1 if x == 'not passed' else '-1' for x in expect_arr])

    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()
```

AUC-ROC Curve



AUC-PR



5.Вывод

Построение модели дерева решений для классификации данныхпо алгоритму C4.5.

Использование характеристических параметров, таких как Accuracy, Precision, Recall для оценки и сравнения качества двухмоделей дерева решений.

Построение графики AUC-ROC и AUC-PR.