Университет ИТМО

Факультет ФПИ и КТ

### Отчёт

### по лабораторной работе 3

**Системы Искуственного Интеллекта**

**Нечетный Вариант**

Студент:

Ляо Ихун

Преподаватель:  
Авдюшина Анна Евгеньевна

# Задача:

Создание дерева решений для классифкации студентов с алготримом C4.5.

# Вывод:

Для реализации дерева решений в программе сложно. Я делал следующим образом:

1. Выбирать корневой узел, то есть лучше признак.
2. Разделять данных на основы атрибутов выбранного признака( корневного узела ).
3. Проверять каждый лист, если процент успешного студента больше определённого значения, то считаем получается в этой ситуации результат yes, иначе опять мы выбираем лучший признак, и поставляем его в лист соответствующего ситуации. Потом повторяем 2-3

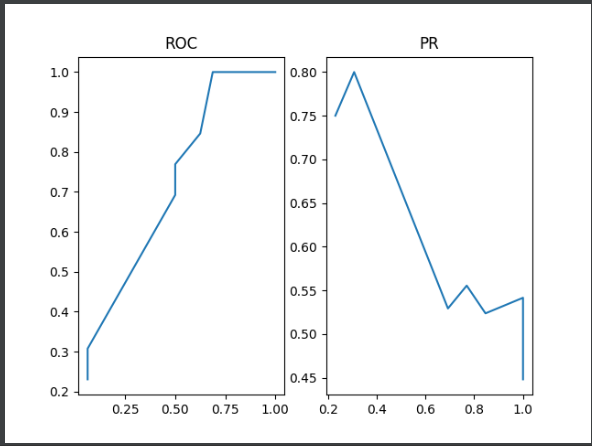
Таким образом хотя глубина дерево толко 1, но по сути

его выраженная глубина больше 1.

Пример:  
文本

描述已自动生成

Если лист является '', то есть студенты, которые пападает сюда, в группе 'не успешно'.Если лист является имнением признака, значит разделяем текущие данные на основе этого признака.

И результат в виде графики как:  


# Код:

import math  
import random  
import pandas as pd  
from collections import deque  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
  
ENTROPY\_NUMBERS=32  
YES = "YES"  
NO = "NO"  
Root = ""  
thesold = 0  
dict\_id\_to\_name = {  
 1:"Age",  
 2:"Sex",  
 3: "High-school",  
 4: "Scholarship",  
 5: "Additional work",  
 6: "Regular artist or sports",  
 7: "Want a partner",  
 8: "Total salary if available",  
 9: "Transportation to the university",  
 10: "Accommodation type in Cyprus",  
 11: "Mother edu",  
 12: "Father edu",  
 13: "Numbers of brothers or sisters",  
 14: "Parental status",  
 15: "Mother occupation",  
 16: "Father occupation",  
 17: "Weekly study hours",  
 18: "Reading frequency(not scientific)",  
 19: "Reading frequency(scientific)",  
 20: "Attendance to the seminars/conferences related to the department",  
 21: "Impact of your projects/activities on your success",  
 22: "Attendance to classes",  
 23: "Preparation to midterm exams(with others)",  
 24: "Preparation to midterm exams (time)",  
 25: "Taking notes in classes",  
 26: "Listening in classes",  
 27: "Discussion improves my interest and success in the course",  
 28: "Flip-classroom",  
 29: "Cumulative grade point average in the last semester",  
 30: "Expected Cumulative grade point average in the graduation",  
 31: "Course",  
 32: "OUTPUT Grade"  
}  
dictionary\_name\_to\_id = {  
 "Age":1,  
 "Sex":2,  
 "High-school":3,  
 "Scholarship":4,  
 "Additional work":5,  
 "Regular artist or sports":6,  
 "Want a partner": 7,  
 "Total salary if available": 8,  
 "Transportation to the university": 9,  
 "Accommodation type in Cyprus": 10,  
 "Mother edu": 11,  
 "Father edu": 12,  
 "Numbers of brothers or sisters":13,  
 "Parental status":14,  
 "Mother occupation": 15,  
 "Father occupation": 16,  
 "Weekly study hours":17,  
 "Reading frequency(not scientific)":18,  
 "Reading frequency(scientific)":19,  
 "Attendance to the seminars/conferences related to the department":20,  
 "Impact of your projects/activities on your success":21,  
 "Attendance to classes":22,  
 "Preparation to midterm exams(with others)":23,  
 "Preparation to midterm exams (time)":24,  
 "Taking notes in classes":25,  
 "Listening in classes":26,  
 "Discussion improves my interest and success in the course":27,  
 "Flip-classroom":28,  
 "Cumulative grade point average in the last semester":29,  
 "Expected Cumulative grade point average in the graduation":30,  
 "Course":31,  
 "OUTPUT Grade":32  
}  
  
def is\_success(out\_grade):  
 if out\_grade <= 2:  
 return False  
 else:  
 return True  
  
## succes as 1, fail as 0  
def normalization(all\_features):  
 for i in all\_features:  
 if int(i[32]) <= 2:  
 i[32] = 0  
 else:  
 i[32] = 1  
  
  
def print\_choices(features):  
 print("Program chooses features automatically:")  
 for i in features:  
 print(dict\_id\_to\_name[i])  
 print("")  
  
  
def get\_random\_features(set\_container, n):  
 while len(set\_container) < n:  
 random\_num = round(random.randint(1, 31))  
 set\_container.add(random\_num)  
  
## get information about the classification of k columns  
def split\_by\_k(data, k):  
 total = 0  
 dict\_diff = {}  
 for i in data:  
 total += 1  
 if i[k] in dict\_diff.keys():  
 dict\_diff[i[k]] = dict\_diff[i[k]] + 1  
 else:  
 dict\_diff[i[k]] = 1  
 dict\_diff["total"] = total  
 return dict\_diff  
  
## get info(entropy)  
def get\_info(over\_view):  
 result = 0  
 for key in over\_view.keys():  
 if key != "total":  
 per = over\_view[key]/over\_view["total"]  
 if per != 0:  
 result = result - per\*math.log(per, 2)  
 return result  
  
## split data by k features, and then count amount every group by success or not success  
## 例如对于sex返回[1：{"success":33,"fail":12,"total":45}, 2：{"success":30,"fail":41,"total":71},"total":116}  
def separation\_feature(data, k):  
 total = 0  
 dict\_diff = {}  
 for i in data:  
 total += 1  
 if i[k] in dict\_diff.keys():  
 if i[32] == 1:  
 dict\_diff[i[k]]["success"] = dict\_diff[i[k]]["success"] + 1  
 else:  
 dict\_diff[i[k]]["fail"] = dict\_diff[i[k]]["fail"] + 1  
 dict\_diff[i[k]]["total"] = dict\_diff[i[k]]["total"] + 1  
 else:  
 dict\_diff[i[k]] = {"success":0, "fail":0,"total":0}  
 if i[32] == 1:  
 dict\_diff[i[k]]["success"] = dict\_diff[i[k]]["success"] + 1  
 else:  
 dict\_diff[i[k]]["fail"] = dict\_diff[i[k]]["fail"] + 1  
 dict\_diff[i[k]]["total"] = dict\_diff[i[k]]["total"] + 1  
 dict\_diff["total"] = total  
 return dict\_diff  
  
  
def get\_info\_x(data):  
 result = dict()  
 for i in data.keys():  
 temp = 0  
 for k in data[i]:  
 if k != "total":  
 temp = temp + (data[i][k]["total"]/data[i]["total"]\*get\_info(data[i][k]))  
 result[i] = temp  
 return result  
  
  
def get\_max\_value(dictionary):  
 k = 0  
 v = -1  
 for i in dictionary.keys():  
 if dictionary[i] > v:  
 k = i  
 v = dictionary[i]  
 return k  
  
  
def get\_min\_value(dictionary):  
 k = 0  
 v = 999999999  
 for i in dictionary.keys():  
 if dictionary[i] < v:  
 k = i  
 v = dictionary[i]  
 return k  
  
## choices - chosen features  
def get\_split(choices, data):  
 dict\_result = dict()  
 for k in choices:  
 result = 0  
 diction = separation\_feature(data, k)  
 for i in diction:  
 if i!="total":  
 a = diction[i]["total"]  
 b = diction["total"]  
 result = result - (a / b \* math.log(a / b,2))  
 dict\_result[dict\_id\_to\_name[k]] = result  
 return dict\_result  
  
  
## 输入数据， 给出最适合用于分裂这个数据的特征  
def c45\_loop(data, choices):  
 split\_by\_score = split\_by\_k(data,ENTROPY\_NUMBERS)  
 entropy = get\_info(split\_by\_score)  
 ## get info\_x  
 groups\_data\_split\_by\_choices = dict()  
 for i in choices:  
 groups\_data\_split\_by\_choices[dict\_id\_to\_name[i]] = separation\_feature(data, i)  
 info\_x = get\_info\_x(groups\_data\_split\_by\_choices)  
 ## get gain  
 gain = dict()  
 for i in info\_x:  
 gain[i] = entropy - info\_x[i]  
 ## get H  
 h = get\_split(choices, data)  
 ## get IGR  
 IGR = dict()  
 for i in h.keys():  
 value = 0  
 if h[i] == 0:  
 value = 9999999  
 else:  
 value = gain[i]/h[i]  
 IGR[i] = value  
 best = get\_max\_value(IGR)  
 p = list([best,groups\_data\_split\_by\_choices[best]])  
 return p  
  
## 根据特征分裂数据  
def classification\_by\_feature(data, k):  
 result = dict()  
 for i in data:  
 if i[k] not in result.keys():  
 result[i[k]] = []  
 result[i[k]].append(i)  
 else:  
 result[i[k]].append(i)  
 return result  
  
def c45(data, choices:set,Tree:dict,Thesold):  
 start = True  
 re\_data = dict()  
 visited = deque()  
 while len(choices) != 0:  
 if start:  
 split\_feature = c45\_loop(data, choices)  
 choices.remove(dictionary\_name\_to\_id[split\_feature[0]])  
 global Root  
 Root = split\_feature[0]  
 Tree[split\_feature[0]] = dict()  
 for w in split\_feature[1]:  
 if w != "total":  
 k = split\_feature[1][w]  
 if k["success"]/k["total"] >= Thesold:  
 Tree[split\_feature[0]][w] = YES  
 else:  
 Tree[split\_feature[0]][w] = ""  
 ## 将数据根据特征种类分类，为接下来的else语句做准备  
 split\_group = classification\_by\_feature(data, dictionary\_name\_to\_id[split\_feature[0]])  
 next\_group = dict()  
 ##保证为已经有结果的叶子不再被分裂  
 for i in split\_group:  
 if Tree[split\_feature[0]][i] != YES and Tree[split\_feature[0]][i] != NO:  
 next\_group[i] = split\_group[i]  
 re\_data = {split\_feature[0]: next\_group}  
 start = False  
 all\_deceived = True  
 for i in Tree:  
 for l in Tree[i]:  
 if Tree[i][l] == "":  
 all\_deceived = False  
 if all\_deceived:  
 return  
 else:  
 temp\_data = dict()  
 for w in re\_data:  
 for k in re\_data[w]:  
 if len(choices) != 0 :  
 split\_feature = c45\_loop(re\_data[w][k], choices)  
 ## 添加节点  
 Tree[w][k] = split\_feature[0]  
 choices.remove(dictionary\_name\_to\_id[split\_feature[0]])  
 Tree[split\_feature[0]] = dict()  
 visited.append(split\_feature[0])  
 for t in split\_feature[1]:  
 if t != "total":  
 g = split\_feature[1][t]  
 if g["success"]/g["total"] >= Thesold:  
 Tree[split\_feature[0]][t] = YES  
 else:  
 Tree[split\_feature[0]][t] = ""  
 # print("-------------------------------------------------------")  
 # ## 将数据根据特征种类分类，为接下来的else语句做准备  
 split\_group = classification\_by\_feature(re\_data[w][k], dictionary\_name\_to\_id[split\_feature[0]])  
 next\_group = dict()  
 # ##保证为已经有结果的叶子不再被分裂  
 for p in split\_group:  
 if Tree[split\_feature[0]][p] != YES:  
 next\_group[p] = split\_group[p]  
 temp\_data[split\_feature[0]] = next\_group  
 ## 防止阙值太低，一下子全部找到  
 all\_deceived = True  
 for n in Tree:  
 for attr in Tree[n]:  
 if Tree[n][attr] == "":  
 all\_deceived = False  
 if all\_deceived:  
 return  
 else:  
 return  
 re\_data = temp\_data  
  
  
# ##get data  
data = pd.read\_csv("./data.csv", header=None)  
data\_origin = data.values[0:,0:]  
n = math.sqrt(32)  
n = round(n)  
numbers\_of\_features = set()  
##split data  
data\_train, data\_test = train\_test\_split(data\_origin, random\_state=0,test\_size=0.2)  
## success = 1 , fail = 0  
normalization(data\_test)  
normalization(data\_train)  
## computer chooses n features  
get\_random\_features(numbers\_of\_features,n)  
print\_choices(numbers\_of\_features)  
##all data prepared, start our c4.5  
marks = []  
## checking  
print("Result should be:")  
fact\_is = []  
proper\_ones = 0  
for i in data\_test:  
 fact\_is.append(i[32])  
 if i[32] == 1:  
 proper\_ones = proper\_ones + 1  
print(fact\_is)  
recalls = []  
precision = []  
TPRS = []  
FPRS = []  
PRECESIONS = []  
RECALLS = []  
while thesold <= 1:  
 result = []  
 global Tree  
 Tree = dict()  
 choices = numbers\_of\_features.copy()  
 c45(data\_train, choices, Tree, thesold)  
 for i in data\_test:  
 end = False  
 node = ""  
 start = True  
 while not end:  
 if start:  
 node = Root  
 start = False  
 id\_f = dictionary\_name\_to\_id[node]  
 if i[id\_f] not in Tree[node].keys():  
 result.append(0)  
 end = True  
 else:  
 if Tree[node][i[id\_f]] == YES:  
 result.append(1)  
 end = True  
 else:  
 if len(Tree[node][i[id\_f]]) == 0:  
 result.append(0)  
 end = True  
 else:  
 node = Tree[node][i[id\_f]]  
 else:  
 id\_f = dictionary\_name\_to\_id[node]  
 if i[id\_f] not in Tree[node].keys():  
 result.append(0)  
 end = True  
 else:  
 if Tree[node][i[id\_f]] == YES:  
 result.append(1)  
 end = True  
 else:  
 if len(Tree[node][i[id\_f]]) == 0:  
 result.append(0)  
 end = True  
 else:  
 node = Tree[node][i[id\_f]]  
 print("Result with threshold %.2f is:"%thesold)  
 thesold = thesold + 0.05  
 print(result)  
 ## 处理结果  
 index = 0  
 TP = 0  
 FP = 0  
 FN = 0  
 TN = 0  
 predicted\_true = 0  
 while index < len(data\_test):  
 if fact\_is[index] == 1 and result[index] == 1:  
 TP = TP + 1  
 if fact\_is[index] == 0 and result[index] == 1:  
 FP = FP + 1  
 if fact\_is[index] == 0 and result[index] == 0:  
 TN = TN + 1  
 if fact\_is[index] == 1 and result[index] == 0:  
 FN = FN + 1  
 index = index + 1  
 TPR = TP/(TP+FN)  
 FPR = FP/(FP+TN)  
 accuracy = (TP+TN)/(TP+TN+FP+FN)  
 recall = TP/(TP+FN)  
 precision = TP/(TP+FP)  
 TPRS.append(TPR)  
 FPRS.append(FPR)  
 PRECESIONS.append(precision)  
 RECALLS.append(recall)  
 print("recall is :%.5f"%recall)  
 print("precision is :%.5f"%precision)  
 print("")  
## graph  
#ROC  
fpr\_axis = np.array(FPRS)  
tpr\_axis = np.array(TPRS)  
plt.subplot(1,2,1)  
plt.title("ROC")  
plt.plot(fpr\_axis, tpr\_axis)  
##PR  
ps = np.array(PRECESIONS)  
rs = np.array(RECALLS)  
plt.subplot(1,2,2)  
plt.title("PR")  
plt.plot(rs, ps)  
  
plt.show()