

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA ĐÀO TẠO CHẤT LƯỢNG CAO**

**🕯✡🕮🕮✡🕯**

****

**BÁO CÁO ĐỒ ÁN 2**

**DECISION TREE & THUẬT TOÁN C4.5**

**GVHD: Trần Nhật Quang**

**Sinh viên thực hiện:**

**Nguyễn Huy Cường \_ 16110028**

**Phạm Thanh Trung \_ 16110232**

***TP. Hồ Chí Minh, tháng 5 năm 2019***

NHẬN XÉT CỦA GIẢNG VIÊN

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

................................................................................................................................................

*TP.Hồ Chí Minh, ngày tháng 5 năm 2019*

**GIẢNG VIÊN HƯỚNG DẪN**

*(Ký tên)*

Trần Nhật Quang

**MỤC LỤC**

[I. DECISION TREE 1](#_Toc9541246)

[1. Giới thiệu 1](#_Toc9541247)

[2. Ưu điểm nhược điểm của cây quyết định 1](#_Toc9541248)

[3. Các bước xây dựng Decision Trees: 1](#_Toc9541249)

[4. Code ví dụ: 2](#_Toc9541250)

[II. THUẬT TOÁN C4.5: 3](#_Toc9541251)

[1. Giới thiệu: 3](#_Toc9541252)

[2. Mã giả của thuật toán C4.5: 4](#_Toc9541253)

[3. Các công thức và ví dụ: 5](#_Toc9541254)

[4. Các cơ chế xử lý: 7](#_Toc9541255)

[***4.1.*** ***Xử lý những giá trị thiếu:*** 7](#_Toc9541256)

[***4.2.*** ***Tránh “quá vừa” dữ liệu:*** 7](#_Toc9541257)

[5. Chuyển đổi sang luật: Cắt tỉa cây 8](#_Toc9541258)

[6. Code ví dụ: 8](#_Toc9541259)

[Bảng phân công công việc: 18](#_Toc9541260)

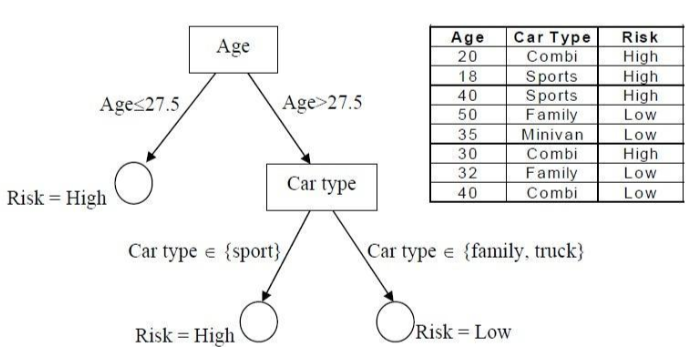
[Tài liệu tham khảo: 19](#_Toc9541261)

## DECISION TREE

## Giới thiệu

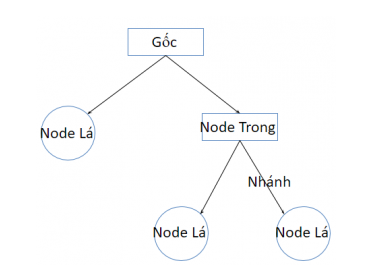
Cây quyết định là một phương pháp phổ biến trong việc khai phá dữ liệu. Cây quyết định là mô hình hỗ trợ ra quyết định dựa trên đồ thị các điều kiện. Tại mỗi nút, ta sẽ đối chiếu các điều kiện thực tế để quyết định rẽ nhánh nào. Nút lá là quyết định cuối cùng[1].

Ví dụ về cây quyết định[2]:



Decision Trees là một cây có cấu trúc[2]:

* Root (Gốc):  Là nút trên cùng của cây.
* Node trong: nút trung gian trên một thuộc tính đơn (hình Oval).
* Nhánh: Biểu diễn các kết quả của kiểm tra trên nút.
* Node lá: Biểu diễn lớp hay sự phân phối lớp (hình vuông hoặc chữ nhật)



Trong họ hàng nhà cây quyết định có 2 loại[1]:

* Cây hồi quy (Regression tree) ước lượng các hàm giá có giá trị là số thực thay vì được sử dụng cho các nhiệm vụ phân loại. (ví dụ: ước tính giá một ngôi nhà hoặc khoảng thời gian một bệnh nhân nằm viện)
* Cây phân loại (Classification tree), phân loại như: giới tính (nam hay nữ), kết quả của một trận đấu (thắng hay thua).

## Ưu điểm nhược điểm của cây quyết định

* + Ưu điểm[1]:

-Học được những đặc trưng từ nguồn data đầu vào, đầu ra là kết quả với hình dạng cây quyết định. Nghĩa là dễ dàng nhìn thấy đặc trưng của data đầu vào.

-Các loại phương pháp khác cần công việc tiền xử lý dữ liệu rất nhiều, còn với cây quyết định thì hầu như là không cần công đoạn tiền xử lý.

-Với các loại phân tích như Neural Network được coi như kiểu model hộp đen (nói đơn giản là bạn không hiểu trong hộp có gì, xử lý ra sao) thì Decision Tree giống như model hộp trắng.

-Hỗ trợ đánh giá độ chính xác của các model được tạo ra. Ở đây các model chính là các cây quyết định được tạo trong quá trình xử lý theo phương pháp này.

* + Nhược điểm[2]:

-Đối với các tập dữ liệu có nhiều thuộc tính thì cây quyết định sẽ lớn (về chiều sâu cả chiều ngang), vì vậy làm giảm độ dễ hiểu.

-Việc xếp hạng các thuộc tính để phân nhánh dựa vào lần phân nhánh trước đó và bỏ qua sự phụ thuộc lẫn nhau giữa các thuộc tính.

-Khi dùng độ lợi thông tin (Information Gain) để xác định thuộc tính rẽ nhánh, các thuộc tính có nhiều giá trị thường được ưu tiên chọn.

## Các bước xây dựng Decision Trees:

* Phát triển cây quyết định: đi từ gốc, đến các nhánh, phát triển quy nạp theo hình thức chia để trị. [2]

1. **Bước 1.** Chọn thuộc tính “tốt” nhất bằng một độ đo đã định trước
2. **Bước 2.**Phát triển cây bằng việc thêm các nhánh tương ứng với từng giá trị của thuộc tính đã chọn
3. **Bước 3.**Sắp xếp, phân chia tập dữ liệu đào tạo tới node con
4. **Bước 4.** Nếu các ví dụ được phân lớp rõ ràng thì dừng.
5. **Ngược lại:** lặp lại bước 1 tới bước 4 cho từng node con

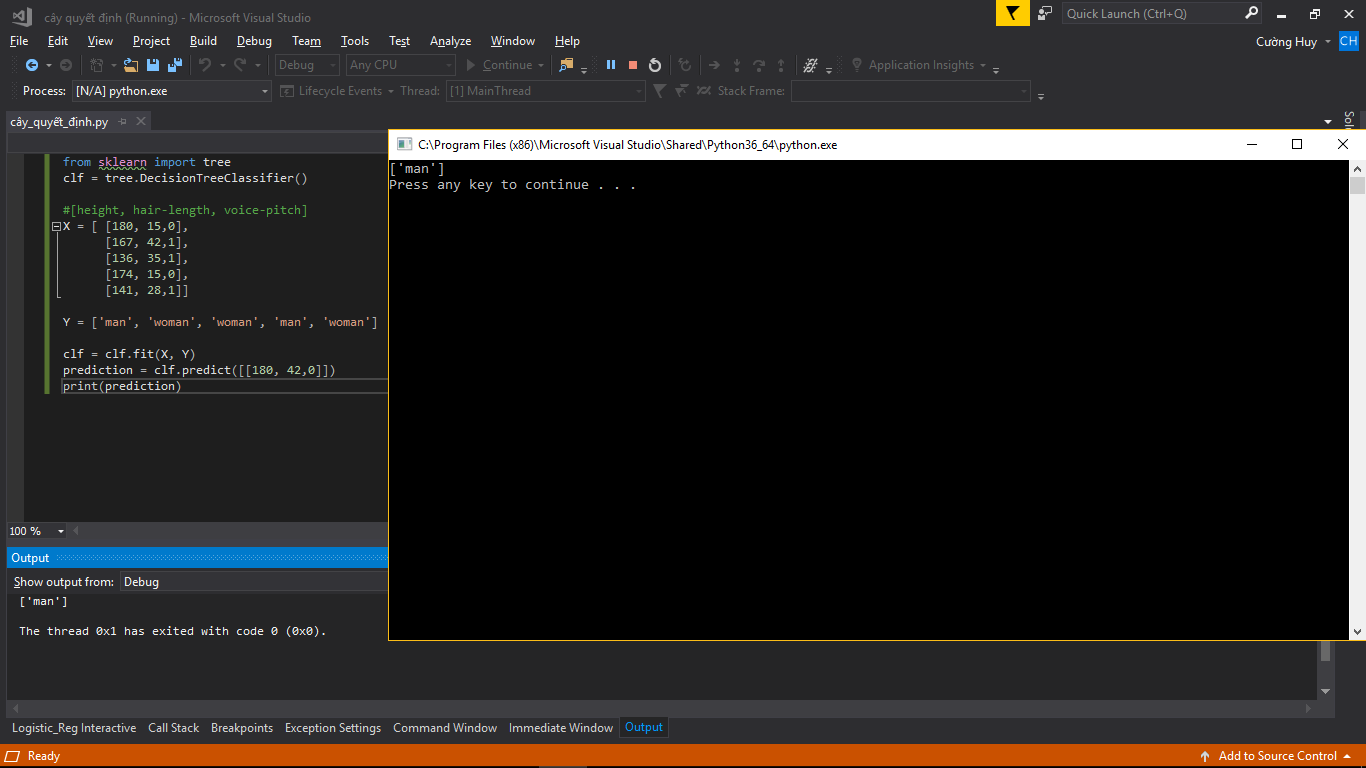
* Cắt tỉa cây: nhằm đơn giản hóa, khái quát hóa cây, tăng độ chính xác. [2]

## Code ví dụ:

Code: [3]

|  |
| --- |
| from sklearn import tree  clf = tree.DecisionTreeClassifier()  #[height, hair-length, voice-pitch]  X = [ [180, 15,0],  [167, 42,1],  [136, 35,1],  [174, 15,0],  [141, 28,1]]  Y = ['man', 'woman', 'woman', 'man', 'woman']  clf = clf.fit(X, Y)  prediction = clf.predict([[180, 42,0]])  print(prediction) |

Kết quả:



## THUẬT TOÁN C4.5:

1. **Giới thiệu:**

- Thuật toán C4.5 là thuật toán phân lớp dữ liệu dựa trên cây quyết định, hiệu quả và phổ biến trong những ứng dụng khai phá cơ sở dữ liệu có kích thước nhỏ. C4.5 là sự phát triển từ CLS và ID3. Năm 1993, J.Ross Quinlan phát triển thành C4.5 với 9000 dòng lệnh C. Phiên bản hiện tại của C4.5 là See5/C5.0. []

- Với những đặc điểm C4.5 là thuật toán phân lớp dữ liệu dựa trên cây quyết định hiệu quả và phổ biến trong những ứng dụng khai phá cơ sở dữ liệu có kích thước nhỏ. C4.5 sử dụng cơ chế lưu trữ dữ liệu thường trú trong bộ nhớ, chính đặc điểm này làm C4.5 chỉ thích hợp với những cơ sở dữ liệu nhỏ, và cơ chế sắp xếp lại dữ liệu tại mỗi node trong quá trình phát triển cây quyết định. C4.5 còn chứa một kỹ thuật cho phép biểu diễn lại cây quyết định dưới dạng một danh sách sắp thứ tự các luật if-then (một dạng quy tắc phân lớp dễ hiểu). Kỹ thuật này cho phép làm giảm bớt kích thước tập luật và đơn giản hóa các luật mà độ chính xác so với nhánh tương ứng cây quyết định là tương đương. [2]

- Tư tưởng phát triển cây quyết định của C4.5 là phương pháp Hunt:

S={S1,S2,…,Sn} là tập dữ liệu đào tạo

C={C1,C2,…,Cm} là tập các lớp

**Trường hợp 1:** Si (i=1…n) thuộc về Cj => Cây quyết định là 1 lá  
ứng Cj.

**Trường hợp 2:** S thuộc về nhiều lớp trong C.

Chọn 1 test trên thuộc ơnh đơn có nhiều giá trị O={O1,..Ok}  
(k thường bằng 2).

Test từ gốc của cây, mỗi Oi tạo thành 1 nhánh, chia S thành  
các tập con có giá trị thuộc tính = Oi. Đệ quy cho từng tập  
con => cây quyết định gồm nhiều nhánh, mỗi nhánh tương  
ứng với Oi. [2]

-Ưu nhược điểm của thuật toán C4.5:[4]

+ Chiếm thời gian sử dụng CPU và bộ nhớ lớn.

+ Sử dụng cơ chế lưu dữ liệu thường trú trong bộ nhớ => ứng dụng với database nhỏ (tần số lỗi lặp lại là 4% với database 20000 cases).

+ Có cơ chế xử lý thiếu, lỗi hoặc quá vừa dữ liệu.

+ Luật tạo ra đơn giản.

1. **Mã giả của thuật toán C4.5:**

**Pseudocode:** [2]

- Kiểm tra case cơ bản.

- Với mỗi thuộc tính A tìm thông tin nhờ việc tách thuộc tính A.

- Chọn a\_best là thuộc tính mà độ đo lựa chọn thuộc tính “tốt nhất”.

- Dùng a\_best làm thuộc tính cho node chia cắt cây.

- Đệ quy trên các danh sách phụ được tạo ra bởi việc phân chia theo a\_best, và thêm các node này như là con của node.

**Mã giả:** [2]

|  |
| --- |
| (1) ComputerClassFrequency(T); (2) if OneClass or FewCases return a leaf; Create a decision node N; (3) ForEach Attribute A ComputeGain(A); (4) N.test=AttributeWithBestGain; (5) if (N.test is continuous) find Threshold; (6) ForEach T’ in the splitting of T (7) If ( T’ is Empty ) Child of N is a leaf else (8) Child of N=FormTree(T’); (9) ComputeErrors of N; return N |

- C4.5 có những đăc điểm khác với các thuật toán khác, đó là: cơ chế chọn thuộc tính để kiểm tra tại mỗi node, cơ chế xử lý với những giá trị thiếu, việc tránh “quá vừa” dữ liệu, ước lượng độ chính xác và cơ chế cắt tỉa cây.

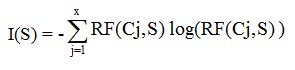
1. **Các công thức và ví dụ:**

- Hai độ đo được sử dụng trong C4.5 là **information gain** và **gain ratio**. RF(Cj,S) biểu diễn tần xuất (Relative Frequency) các case trong S thuộc về lớp Cj.

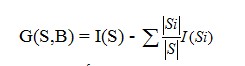
http://sinhvientot.net/wp-content/uploads/2017/05/image003.jpg

- Với |Sj| là kích thước tập các case có giá trị phân lớp là Cj và |S| là kích thước tập dữ liệu đào tạo.

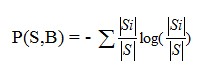
- Chỉ số thông tin cần thiết cho sự phân lớp: I(S) với S là tập cần xét sự phân phối lớp được tính bằng:



- Sau khi S được phân chia thành các tập con S1, S2,…, St  bởi test B thì **information gain** được tính bằng:



- Test B sẽ được chọn nếu có G(S,B) đạt giá trị lớn nhất.Tuy nhiên có một vấn đề khi sử dụng G(S,B) ưu tiên test có số lượng lớn kết quả, ví dụ G(S,B) đạt cực đại với test mà từng Si chỉ chứa một case đơn. Tiêu chuẩn gain ratio giải quyết được vấn đề này bằng việc đưa vào thông tin tiềm năng (potential information) của bản thân mỗi phân hoạch.

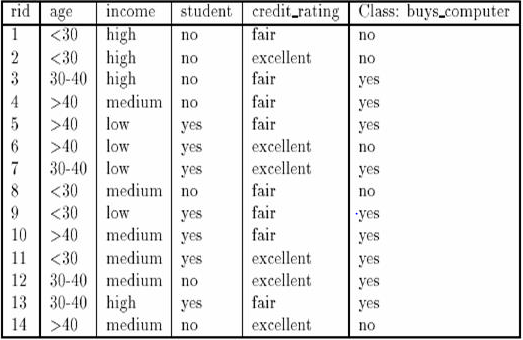


- Gain ratio=G(S,B)/P(S,B)lớn nhất => chọn test B

- Trong mô hình phân lớp C4.5, có thể dùng một trong hai loại chỉ số Information Gain hay Gain ratio để xác định thuộc tính tốt nhất. Trong đó Gain ratio là lựa chọn mặc định.[2]

**Ví dụ:**

* ***Với thuộc tính rời rạc:*** [2]



Trong tập dữ liệu trên: S1 là tập những bản ghi có giá trị phân lớp là **yes**, S2 là tập những bản ghi có giá trị phân lớp là **no**. Khi đó:

- I(S) = I(S1,S2) = I(9,5) = -9/14\*log29/14 – 5/14\*log25/14 = 0.940

- Tính G(S,A) với A lần lượt là từng thuộc tính:

A = age. Thuộc tính age đã được rời rạc hóa thành các giá trị <30, 30-40 và >40.

+ Với age = “<30”: I(S1) = (S11,S21) = **-**2/5\*log22/5 – 3/5\*log23/5 = 0.971  
     + Với age = “30-40”: I(S2) = (S12,S22) = 0  
     + Với age = “>40” : I(S3) = (S13,S23) = 0.971

Gain (S,age) = I(S1.S2)- ∑(|Si|/|S|)\*I(Si)=0.246

- Tính tương tự với các thuộc tính khác ta được:   
 + A = income: Gain (S, income) = 0.029  
 + A = student: Gain (S, student) = 0.151  
 + A = credit\_rating: Gain ( S, credit\_rating) = 0.048  
- Thuộc tính age là  thuộc tính có độ đo Information Gain lớn nhất.Do vậy age được chọn làm thuộc tính phát triển tại node đang xét.

* ***Với thuộc tính liên tục:*** [5]

- Xử lý thuộc tính liên tục đòi hỏi nhiều tài nguyên tính toán hơn thuộc tính rời rạc. Gồm các bước sau:

+ Kỹ thuật **Quick sort** được sử dụng để sắp xếp các case trong tập dữ liệu đào tạo theo thứ tự tăng dần hoặc giảm dần các giá trị của thuộc tính liên tục V đang xét. Được tập giá trị V = {v1, v2, …, vm}

+ Chia tập dữ liệu thành hai tập con theo ngưỡng θi = (vi + vi+1)/2 nằm giữa hai giá trị liền kề nhau vi và vi+1. Test để phân chia dữ liệu là test nhị phân dạng V <= θi hay V > θi. Thực thi test đó ta được hai tập dữ liệu con: V1 = {v1, v2, …, vi} và V2 = {vi+1, vi+2, …, vm}.

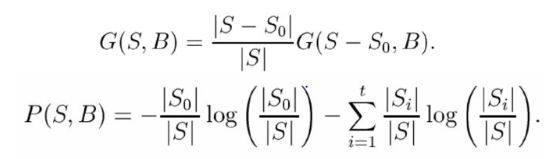
+ Xét (m-1) ngưỡng θi có thể có ứng với m giá trị của thuộc tính V bằng cách tính Information gain hay Gain ratio với từng ngưỡng đó. Ngưỡng có giá trị của Information gain hay Gain ratio lớn nhất sẽ được chọn làm ngưỡng phân chia của thuộc tính đó.

Việc tìm ngưỡng (theo cách tuyến tính như trên) và sắp xếp tập training theo thuộc tính liên tục đang xem xét đôi khi gây ra thắt cổ chai vì tốn nhiều tài nguyên tính toán.

1. **Các cơ chế xử lý:**
   1. ***Xử lý những giá trị thiếu:***

- Giá trị thiếu của thuộc tính là hiện tượng phổ biến trong dữ liệu, có thể do lỗi khi nhập các bản ghi vào cơ sở dữ liệu, cũng có thể do giá trị thuộc tính đó được đánh giá là không cần thiết đối với case cụ thể. [5]

- Trong quá trình xây dựng cây từ tập dữ liệu đào tạo S, B là test dựa trên thuộc tính Aa với các giá trị đầu ra là b1, b2, ..., bt. Tập S0 là tập con các case trong S mà có giá trị thuộc tính Aa không biết và Si biểu diễn các case với đầu ra là bi trong test B. Khi đó độ đo information gain của test B giảm vì chúng ta không học được gì từ các case trong S0. [5]

- Tương ứng với G(S, B), P(S, B) cũng thay đổi: 

- Hai thay đổi này làm giảm giá trị của test liên quan đến thuộc tính có tỉ lệ giá trị thiếu cao. Nếu test B được chọn, C4.5 không tạo một nhánh riêng trên cây quyết định cho S0. Thay vào đó, thuật toán có cơ chế phân chia các case trong S0 về vác tập con Si là tập con mà có giá trị thuộc tính test xác định theo trong số |Si|/ |S – S0|.[5]

* 1. ***Tránh “quá vừa” dữ liệu:***

- “Quá vừa” dữ liệu là một khó khăn đáng kể đối với học bằng cây quyết định và những phương pháp học khác. Quá vừa dữ liệu là hiện tượng: nếu không có các case xung đột (là những case mà giá trị cho mọi thuộc tính là giống nhau nhưng giá trị của lớp lại khác nhau) thì cây quyết định sẽ phân lớp chính xác toàn bộ các case trong tập dữ liệu đào tạo. Đôi khi dữ liệu đào tạo lại chứa những đặc tính cụ thể, nên khi áp dụng cây quyết định đó cho những tập dữ liệu khác thì độ chính xác không còn cao như trước. [5]

- Có một số phương pháp tránh “quá vừa” dữ liệu trong cây quyết định như:

+ Dừng phát triển cây sớm hơn bình thường, trước khi đạt tới điểm phân lớp hoàn hảo tập dữ liệu đào tạo. Với phương pháp này, một thách thức đặt ra là phải ước lượng chính xác thời điểm dừng phát triển cây.

+ Cho phép cây có thể “quá vừa” dữ liệu, sau đó sẽ cắt, tỉa cây

- Mặc dù phương pháp thứ nhất có vẻ trực quan hơn, nhưng với phương pháp thứ hai thì cây quyết định được sinh ra được thử nghiệm chứng minh là thành công hơn trong thực tế, vì nó cho phép các tương tác tiềm năng giữa các thuộc tính được khám phá trước khi quyết định xem kết quả nào đáng giữ lại. C4.5 sử dụng kỹ thuật thứ hai để tránh “quá vừa” dữ liệu. [5]

1. **Chuyển đổi sang luật: Cắt tỉa cây**

**Dạng luật:** if A and B and C… then class X. Không thỏa mãn điều kiện chuyển về lớp mặc định.

**Xây dựng luật:** 4 bước

* Bước 1:Mỗi đường đi từ gốc đến lá là một luật mẫu. Đơn giản luật mẫu bằng  
  cách bỏ dần điều kiện mà không ảnh hưởng tới độ chính xác của luật.
* Bước 2: Các luật đã cắt tỉa được nhóm lại theo giá trị phân lớp tạo ra các tập  
  Với mỗi tập con, xem xét để lựa chọn luật để tối ưu hóa độ chính  
  xác dự đoán của lớp gắn với tập luật đó.
* Bước 3:Sắp xếp các tập luật trên theo tần số lỗi. Lớp mặc định được tạo ra  
  bằng cách xác định các case trong tập S không chứa trong các luật hiện tại và chọn lớp phổ biến nhất trong các case đó làm lớp mặc định.
* Bước 4:Ước lượng đánh giá: các luật được ước lượng trên toàn tập S, loại bỏ  
  luật làm giảm độ chính xác của sự phân lớp.

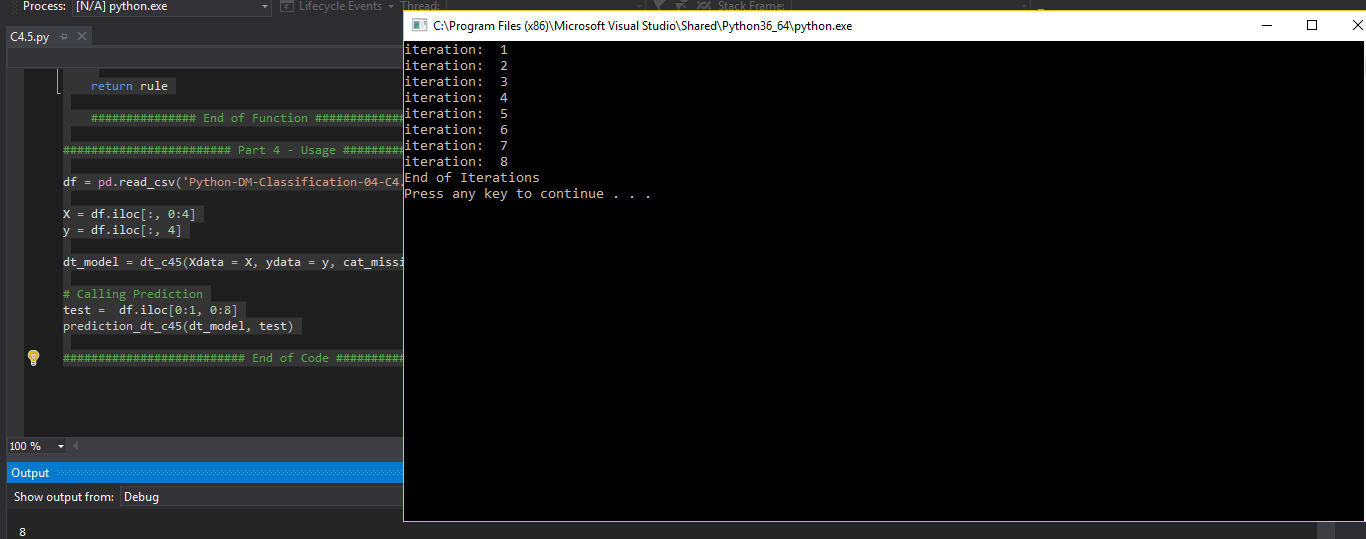
**Hoàn thành:** 1 tập các quy tắc đơn giản được lựa chọn cho mỗi lớp. [2]

1. **Code ví dụ:**

Code: [6]

|  |
| --- |
| # Installing Required Libraries  import pandas as pd  import numpy as np  from random import randint  from scipy import stats  from copy import deepcopy  # Function: Returns True, if a Column is Numeric  def is\_number(string):  for i in range(0, len(string)):  if pd.isnull(string[i]) == False:  try:  float(string[i])  return True  except ValueError:  return False  # Function: Returns True, if a Value is Numeric  def is\_number\_value(value):  if pd.isnull(value) == False:  try:  float(value)  return True  except ValueError:  return False  # Function: Performs a Chi\_Squared Test or Fisher Exact Test  def chi\_squared\_test(label\_df, feature\_df):  label\_df.reset\_index(drop=True, inplace=True)  feature\_df.reset\_index(drop=True, inplace=True)  data = pd.concat([pd.DataFrame(label\_df.values.reshape((label\_df.shape[0], 1))), feature\_df], axis = 1)  data.columns=["label", "feature"]  contigency\_table = pd.crosstab(data.iloc[:,0], data.iloc[:,1], margins = False)  m = contigency\_table.values.sum()  if m <= 10000 and contigency\_table.shape == (2,2):  p\_value = stats.fisher\_exact(contigency\_table)  else:  p\_value = stats.chi2\_contingency(contigency\_table, correction = False) # (No Yates' Correction)  return p\_value[1]  # Function: Prediction  def prediction\_dt\_c45(model, Xdata):  Xdata = Xdata.reset\_index(drop=True)  ydata = pd.DataFrame(index=range(0, Xdata.shape[0]), columns=["Prediction"])  for j in range(0, ydata.shape[1]):  if ydata.iloc[:,j].dropna().value\_counts().index.isin([0,1]).all():  for i in range(0, ydata.shape[0]):  if ydata.iloc[i,j] == 0:  ydata.iloc[i,j] = "zero"  else:  ydata.iloc[i,j] = "one"  data = pd.concat([ydata, Xdata], axis = 1)  rule = []    # Preprocessing - Boolean Values  for j in range(0, data.shape[1]):  if data.iloc[:,j].dtype == "bool":  data.iloc[:,j] = data.iloc[:, j].astype(str)    dt\_model = deepcopy(model)    for i in range(0, len(dt\_model)):  dt\_model[i] = dt\_model[i].replace("{", "")  dt\_model[i] = dt\_model[i].replace("}", "")  dt\_model[i] = dt\_model[i].replace(".", "")  dt\_model[i] = dt\_model[i].replace("IF ", "")  dt\_model[i] = dt\_model[i].replace("AND", "")  dt\_model[i] = dt\_model[i].replace("THEN", "")  dt\_model[i] = dt\_model[i].replace("=", "")  dt\_model[i] = dt\_model[i].replace("<", "<=")    for i in range(0, len(dt\_model) -2):  splited\_rule = [x for x in dt\_model[i].split(" ") if x]  rule.append(splited\_rule)    for i in range(0, Xdata.shape[0]):  for j in range(0, len(rule)):  rule\_confirmation = len(rule[j])/2 - 1  rule\_count = 0  for k in range(0, len(rule[j]) - 2, 2):  if is\_number\_value(data[rule[j][k]][i]) == False:  if (data[rule[j][k]][i] in rule[j][k+1]):  rule\_count = rule\_count + 1  if (rule\_count == rule\_confirmation):  data.iloc[i,0] = rule[j][len(rule[j]) - 1]  else:  k = len(rule[j])  elif is\_number\_value(data[rule[j][k]][i]) == True:  if rule[j][k+1].find("<=") == 0:  if data[rule[j][k]][i] <= float(rule[j][k+1].replace("<=", "")):  rule\_count = rule\_count + 1  if (rule\_count == rule\_confirmation):  data.iloc[i,0] = rule[j][len(rule[j]) - 1]  else:  k = len(rule[j])  elif rule[j][k+1].find(">") == 0:  if data[rule[j][k]][i] > float(rule[j][k+1].replace(">", "")):  rule\_count = rule\_count + 1  if (rule\_count == rule\_confirmation):  data.iloc[i,0] = rule[j][len(rule[j]) - 1]  else:  k = len(rule[j])    for i in range(0, Xdata.shape[0]):  if pd.isnull(data.iloc[i,0]):  data.iloc[i,0] = dt\_model[len(dt\_model)-1]    return data  # Function: Calculates the Information Gain Ratio  def info\_gain\_ratio(target, feature = [], uniques = []):  entropy = 0  denominator\_1 = feature.count()  data = pd.concat([pd.DataFrame(target.values.reshape((target.shape[0], 1))), feature], axis = 1)  for entp in range(0, len(np.unique(target))):  numerator\_1 = data.iloc[:,0][(data.iloc[:,0] == np.unique(target)[entp])].count()  if numerator\_1 > 0:  entropy = entropy - (numerator\_1/denominator\_1)\* np.log2((numerator\_1/denominator\_1))  info\_gain = float(entropy)  info\_gain\_r = 0  intrinsic\_v = 0  for word in range(0, len(uniques)):  denominator\_2 = feature[(feature == uniques[word])].count()  if denominator\_2[0] > 0:  intrinsic\_v = intrinsic\_v - (denominator\_2/denominator\_1)\* np.log2((denominator\_2/denominator\_1))  for lbl in range(0, len(np.unique(target))):  numerator\_2 = data.iloc[:,0][(data.iloc[:,0] == np.unique(target)[lbl]) & (data.iloc[:,1] == uniques[word])].count()  if numerator\_2 > 0:  info\_gain = info\_gain + (denominator\_2/denominator\_1)\*(numerator\_2/denominator\_2)\* np.log2((numerator\_2/denominator\_2))  if intrinsic\_v[0] > 0:  info\_gain\_r = info\_gain/intrinsic\_v  return float(info\_gain\_r)  # Function: Binary Split on Continuous Variables  def split\_me(feature, split):  result = pd.DataFrame(feature.values.reshape((feature.shape[0], 1)))  for fill in range(0, len(feature)):  result.iloc[fill,0] = feature.iloc[fill]  lower = "<=" + str(split)  upper = ">" + str(split)  for convert in range(0, len(feature)):  if float(feature.iloc[convert]) <= float(split):  result.iloc[convert,0] = lower  else:  result.iloc[convert,0] = upper  binary\_split = []  binary\_split = [lower, upper]  return result, binary\_split  # Function: C4.5 Algorithm  def dt\_c45(Xdata, ydata, cat\_missing = "none", num\_missing = "none", pre\_pruning = "none", chi\_lim = 0.1, min\_lim = 5):    ################ Part 1 - Preprocessing #############################  # Preprocessing - Creating Dataframe  name = ydata.name  ydata = pd.DataFrame(ydata.values.reshape((ydata.shape[0], 1)))  for j in range(0, ydata.shape[1]):  if ydata.iloc[:,j].dropna().value\_counts().index.isin([0,1]).all():  for i in range(0, ydata.shape[0]):  if ydata.iloc[i,j] == 0:  ydata.iloc[i,j] = "zero"  else:  ydata.iloc[i,j] = "one"  dataset = pd.concat([ydata, Xdata], axis = 1)    # Preprocessing - Boolean Values  for j in range(0, dataset.shape[1]):  if dataset.iloc[:,j].dtype == "bool":  dataset.iloc[:,j] = dataset.iloc[:, j].astype(str)  # Preprocessing - Missing Values  if cat\_missing != "none":  for j in range(1, dataset.shape[1]):  if is\_number(dataset.iloc[:, j]) == False:  for i in range(0, dataset.shape[0]):  if pd.isnull(dataset.iloc[i,j]) == True:  if cat\_missing == "missing":  dataset.iloc[i,j] = "Unknow"  elif cat\_missing == "most":  dataset.iloc[i,j] = dataset.iloc[:,j].value\_counts().idxmax()  elif cat\_missing == "remove":  dataset = dataset.drop(dataset.index[i], axis = 0)  elif cat\_missing == "probability":  while pd.isnull(dataset.iloc[i,j]) == True:  dataset.iloc[i,j] = dataset.iloc[randint(0, dataset.shape[0] - 1), j]  elif num\_missing != "none":  if is\_number(dataset.iloc[:, j]) == True:  for i in range(0, dataset.shape[0]):  if pd.isnull(dataset.iloc[i,j]) == True:  if num\_missing == "mean":  dataset.iloc[i,j] = dataset.iloc[:,j].mean()  elif num\_missing == "median":  dataset.iloc[i,j] = dataset.iloc[:,j].median()  elif num\_missing == "most":  dataset.iloc[i,j] = dataset.iloc[:,j].value\_counts().idxmax()  elif cat\_missing == "remove":  dataset = dataset.drop(dataset.index[i], axis = 0)  elif num\_missing == "probability":  while pd.isnull(dataset.iloc[i,j]) == True:  dataset.iloc[i,j] = dataset.iloc[randint(0, dataset.shape[0] - 1), j]    # Preprocessing - Unique Words List  unique = []  uniqueWords = []  for j in range(0, dataset.shape[1]):  for i in range(0, dataset.shape[0]):  token = dataset.iloc[i, j]  if not token in unique:  unique.append(token)  uniqueWords.append(unique)  unique = []    # Preprocessing - Label Matrix  label = np.array(uniqueWords[0])  label = label.reshape(1, len(uniqueWords[0]))    ################ Part 2 - Initialization #############################  # C4.5 - Initializing Variables  i = 0  impurity = 0  branch = [None]\*1  branch[0] = dataset  gain\_ratio = np.empty([1, branch[i].shape[1]])  lower = "0"  root\_index = 0  rule = [None]\*1  rule[0] = "IF "  skip\_update = False  stop = 2  upper = "1"    ################ Part 3 - C4.5 Algorithm #############################  # C4.5 - Algorithm  while (i < stop):  impurity = np.amax(gain\_ratio)  gain\_ratio.fill(0)  for element in range(1, branch[i].shape[1]):  if len(branch[i]) == 0:  skip\_update = True  break  if len(np.unique(branch[i][0])) == 1 or len(branch[i]) == 1:  if "." not in rule[i]:  rule[i] = rule[i] + " THEN " + name + " = " + branch[i].iloc[0, 0] + "."  rule[i] = rule[i].replace(" AND THEN ", " THEN ")  skip\_update = True  break  if i > 0 and is\_number(dataset.iloc[:, element]) == False and pre\_pruning == "chi\_2" and chi\_squared\_test(branch[i].iloc[:, 0], branch[i].iloc[:, element]) > chi\_lim:  if "." not in rule[i]:  rule[i] = rule[i] + " THEN " + name + " = " + branch[i].agg(lambda x:x.value\_counts().index[0])[0] + "."  rule[i] = rule[i].replace(" AND THEN ", " THEN ")  skip\_update = True  continue  if is\_number(dataset.iloc[:, element]) == True:  gain\_ratio[0, element] = 0.0  value = np.sort(branch[i].iloc[:, element].unique())  skip\_update = False  if branch[i][(branch[i].iloc[:, element] == value[0])].count()[0] > 1:  start = 0  finish = len(branch[i].iloc[:, element].unique()) - 2  else:  start = 1  finish = len(branch[i].iloc[:, element].unique()) - 2  if len(branch[i]) == 2 or len(value) == 1 or len(value) == 2:  start = 0  finish = 1  if len(value) == 3:  start = 0  finish = 2  for bin\_split in range(start, finish):  bin\_sample = split\_me(feature = branch[i].iloc[:, element], split = value[bin\_split])  if i > 0 and pre\_pruning == "chi\_2" and chi\_squared\_test(branch[i].iloc[:, 0], bin\_sample[0]) > chi\_lim:  if "." not in rule[i]:  rule[i] = rule[i] + " THEN " + name + " = " + branch[i].agg(lambda x:x.value\_counts().index[0])[0] + "."  rule[i] = rule[i].replace(" AND THEN ", " THEN ")  skip\_update = True  continue  igr = info\_gain\_ratio(target = branch[i].iloc[:, 0], feature = bin\_sample[0], uniques = bin\_sample[1])  if igr > float(gain\_ratio[0, element]):  gain\_ratio[0, element] = igr  uniqueWords[element] = bin\_sample[1]  if is\_number(dataset.iloc[:, element]) == False:  gain\_ratio[0, element] = 0.0  skip\_update = False  igr = info\_gain\_ratio(target = branch[i].iloc[:, 0], feature = pd.DataFrame(branch[i].iloc[:, element].values.reshape((branch[i].iloc[:, element].shape[0], 1))), uniques = uniqueWords[element])  gain\_ratio[0, element] = igr  if i > 0 and pre\_pruning == "min" and len(branch[i]) <= min\_lim:  if "." not in rule[i]:  rule[i] = rule[i] + " THEN " + name + " = " + branch[i].agg(lambda x:x.value\_counts().index[0])[0] + "."  rule[i] = rule[i].replace(" AND THEN ", " THEN ")  skip\_update = True  continue    if i > 0 and pre\_pruning == "impur" and np.amax(gain\_ratio) < impurity and np.amax(gain\_ratio) > 0:  if "." not in rule[i]:  rule[i] = rule[i] + " THEN " + name + " = " + branch[i].agg(lambda x:x.value\_counts().index[0])[0] + "."  rule[i] = rule[i].replace(" AND THEN ", " THEN ")  skip\_update = True  continue    if skip\_update == False:  root\_index = np.argmax(gain\_ratio)  rule[i] = rule[i] + str(list(branch[i])[root\_index])    for word in range(0, len(uniqueWords[root\_index])):  uw = uniqueWords[root\_index][word].replace("<=", "")  uw = uw.replace(">", "")  lower = "<=" + uw  upper = ">" + uw  if uniqueWords[root\_index][word] == lower:  branch.append(branch[i][branch[i].iloc[:, root\_index] <= float(uw)])  elif uniqueWords[root\_index][word] == upper:  branch.append(branch[i][branch[i].iloc[:, root\_index] > float(uw)])  else:  branch.append(branch[i][branch[i].iloc[:, root\_index] == uniqueWords[root\_index][word]])    rule.append(rule[i] + " = " + "{" + uniqueWords[root\_index][word] + "}")    for logic\_connection in range(1, len(rule)):  if len(np.unique(branch[i][0])) != 1 and rule[logic\_connection].endswith(" AND ") == False and rule[logic\_connection].endswith("}") == True:  rule[logic\_connection] = rule[logic\_connection] + " AND "  skip\_update = False  i = i + 1  print("iteration: ", i)  stop = len(rule)    for i in range(len(rule) - 1, -1, -1):  if rule[i].endswith(".") == False:  del rule[i]  rule.append("Total Number of Rules: " + str(len(rule)))  rule.append(dataset.agg(lambda x:x.value\_counts().index[0])[0])  print("End of Iterations")    return rule  ############### End of Function ##############  ######################## Part 4 - Usage ####################################  df = pd.read\_csv('Python-DM-Classification-04-C4.5.csv', sep = ';')  X = df.iloc[:, 0:4]  y = df.iloc[:, 4]  dt\_model = dt\_c45(Xdata = X, ydata = y, cat\_missing = "missing", num\_missing = "mean", pre\_pruning = "impur", chi\_lim = 0.1, min\_lim = 5)  # Calling Prediction  test = df.iloc[0:1, 0:8]  prediction\_dt\_c45(dt\_model, test)  ########################## End of Code ##################################### |

Kết quả:



# Bảng phân công công việc:

|  |  |  |
| --- | --- | --- |
| Tên SV | Đánh giá chung phần trăm đóng góp | Mô tả khái quát mảng công việc SV thực hiện trong đồ án |
| Nguyễn Huy Cường | 50% | -Tìm hiểu và viết báo cáo về Decision tree  -Code phần Decision tree và C4.5 |
| Phạm Thanh Trung | 50% | -Tìm hiểu và viết báo cáo về Thuật toán C4.5  -Chỉnh sửa và hoàn chỉnh báo cáo. |

# Tài liệu tham khảo:

[1] Phân tích cây quyết định với scikit-learn. (2017, September 3). Retrieved May 22, 2019, from <https://codetudau.com/phan-tich-cay-quyet-dinh-voi-scikit-learn/index.html>

[2] Người Chia sẻ. (2017, May 18). Thuật toán cây quyết định C4.5. Retrieved from <https://sinhvientot.net/thuat-toan-cay-quyet-dinh-c45/>

[3] Commento. (n.d.). Decision tree. Retrieved from <https://pythonprogramminglanguage.com/decision-tree/>

[4] Thuật Toán Cây Quyết Định C4.5 – SlidePlayer. Retrieved from

<https://slideplayer.com/slide/11186155/>

[5] NGHIÊN CỨU CÁC THUẬT TOÁN PHÂN LỚP DỮ LIỆU DỰA TRÊN CÂY QUYẾT ĐỊNH-VNU-UET. Retrieved from

https://bit.ly/2Hw4QlI

[6] Valdecy/C4.5. (2019, January 27). Retrieved from https://github.com/Valdecy/C4.5/blob/master/Python-DM-Classification-04-C4.5.py