HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

KHO DỮ LIỆU VÀ KHAI PHÁ DỮ LIỆU

ĐỀ TÀI

XÂY DỰNG CÂY QUYẾT ĐỊNH

Giảng Viên: Nguyễn Quỳnh Chi

Nhóm môn học: D19-029

Nhóm sinh viên thực hiện: 05

Nguyễn Văn Khánh	-	**B19DCCN357**
Phan Quang Huy	-	**B19DCCN321**
Phạm Thu Hương	-	**B19DCCN339**
Trần Thị Thùy Dương	-	**B19DCCN159**
Trần Quang Hưng	-	**B19DCCN334**

Mục lục

I. Giới thiệu về bài toán phân loại - Classification

Bài toán phân lớp là quá trình phân lớp 1 đối tượng dữ liệu vào 1 hay nhiều lớp đã cho trước nhờ 1 mô hình phân lớp (model).

- Mô hình này được xây dựng dựa trên 1 tập dữ liệu được xây dựng trước đó có gán nhãn (hay còn gọi là tập huấn luyện).
- Quá trình phân lớp là quá trình gán nhãn cho đối tượng dữ liệu.

Như vậy, nhiệm vụ của bài toán phân lớp là cần tìm 1 mô hình phần lớp để khi có dữ liệu mới thì có thể xác định được dữ liệu đó thuộc vào phân lớp nào.

Đối với các bài toán phân lớp dữ liệu sử dụng các thuật toán học có giám sát (supervised learning) để xây dựng mô hình cho bài toán này.

Quá trình phân lớp dữ liệu

1. Chuẩn bị tập dữ liệu huấn luyện (dataset)

    Thông thường sẽ sử dụng cross-validation (kiểm tra chéo) để chia tập datasets thành 2 phần, 1 phần phục vụ cho training (training datasets) và phần còn lại phục vụ cho mục đích testing trên mô hình (testing dataset).

2. Xây dựng mô hình phân lớp (classifier model)

Mục đích của mô hình huấn luyện là tìm ra hàm f(x) và thông qua hàm f tìm được để gán nhãn cho dữ liệu, bước này thường được gọi là learning hay training.

$f(x) = y$

Trong đó:

x - các feature hay input đầu vào của dữ liệu

y - nhãn lớp hay output đầu ra

Thông thường để xây dựng mô hình phân lớp cho bài toán này cần sử dụng các thuật toán học giám sát (supervised learning) như k-nearest neighbors, Neural Network, SVM, Decision tree, Naive Bayes.

3. Kiểm tra dữ liệu với mô hình (making predictions)

    Sau khi đã tìm được mô hình phân lớp ở bước 2, thì ở bước này sẽ đưa vào các dữ liệu mới để kiểm tra trên mô hình phân lớp.

4. Đánh giá mô hình phân lớp và chọn ra mô hình tốt nhất

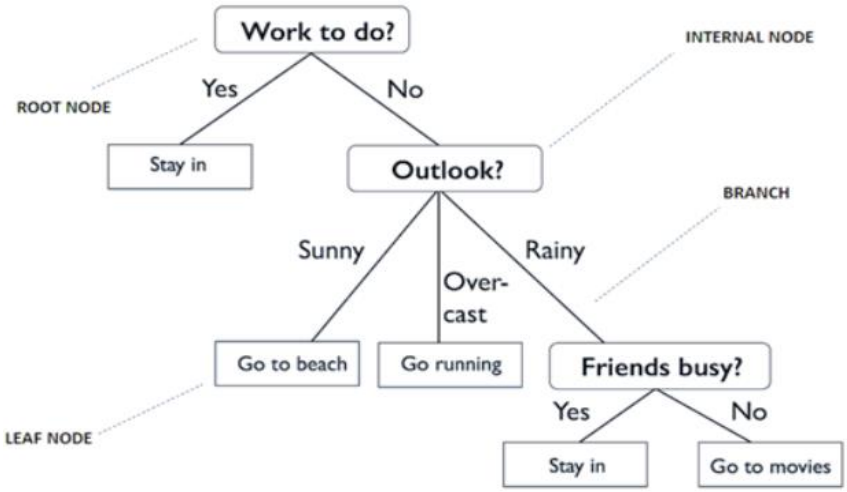
    Bước cuối cùng sẽ đánh giá mô hình bằng cách đánh giá mức độ lỗi của dữ liệu testing và dữ liệu training thông qua mô hình tìm được. Nếu không đạt được kết quả mong muốn thì phải thay đổi các tham số (turning parameters) của các thuật toán học

để tìm ra các mô hình tốt hơn và kiểm tra, đánh giá lại mô hình phân lớp, và cuối cùng chọn ra mô hình phân lớp tốt nhất cho bài toán.

II. Giới thiệu thuật toán Decision tree

Decision Tree là một thuật toán thuộc loại Supervised Learning, phương pháp học có giám sát, kết quả hay biến mục tiêu của Decision Tree chủ yếu là biến phân loại. Các thuật toán được xây dựng giống hình dạng một các cây có ngọn cây, thân cây, lá cây kết nối bằng các cành cây, và mỗi thành phần đều có ý nghĩa riêng của nó, như các yếu tố tác động lên quyết định sau cùng.

Decision Tree là thuật toán được xây dựng trên mô hình một các cây mục đích để thể hiện kết cấu, sự cấu trúc của một hệ thống ra quyết định, hay nói cách khác cách con người tư duy, logic ra sao để đi đến quyết định cuối cùng. Còn trong lĩnh vực dữ liệu, Decision Tree thể hiện mối quan hệ giữa các yếu tố (các biến độc lập), và sự tác động của chúng như thế nào đến biến mục tiêu. Ví dụ:



Nhìn trên hình có thể thấy một cây quyết định bao gồm:

- "Root node": điểm ngọn chứa giá trị của biến đầu tiên được dùng để phân nhánh.
- "Internal node": các điểm bên trong thân cây là các biến chứa các thuộc tính, giá trị dữ liệu được dùng để xét cho các phân nhánh tiếp theo
- "Leaf node": là các lá cây chứa giá trị của biến phân loại sau cùng.
- "Branch" là quy luật phân nhánh, nói đơn giản là mối quan hệ giữa giá trị của biến độc lập (Internal node) và giá trị của biến mục tiêu (Leaf node).

Quá trình phân chia nhánh cây trong mô hình cây quyết định đều dựa trên các công thức tính toán, định lượng rõ ràng, sao cho quá trình này sẽ đem lại kết quả tối ưu nhất.

Trước khi triển khai thuật toán Decision trees, thì quy trình phân tích hay mô hình dữ liệu phải thỏa mãn các yêu cầu sau:

- Tập dữ liệu phải đạt đủ chất lượng trước khi đưa vào phân tích, được chia thành các tập training và test sao cho phù hợp, với tập training thì phải có đầy đủ biến phân loại, biến mục tiêu (target variable), còn test data thì không có.
- Tập dữ liệu training phải dồi dào, đa dạng về các biến, thuộc tính dữ liệu để quá trình huấn luyện cho mô hình diễn ra tối ưu và kết quả phân loại chính xác.
- Các lớp, các nhóm hay giá trị của biến mục tiêu phải rời rạc, rõ ràng. Thông thường không thể áp dụng phân tích cây quyết định cho một biến mục tiêu liên tục (continuous variable). Thay vào đó, biến mục tiêu phải nhận các giá trị được phân định rõ ràng là thuộc về một lớp, nhóm cụ thể nào đó hoặc không thuộc về một lớp, nhóm cụ thể nào đó.

1. Decision tree – thuật toán C4.5

Thuật toán cây quyết định mới - C4.5, cho phép phân nhánh nhiều hơn 2 nhánh, và sử dụng công thức Entropy để tính mức độ tương đồng, "thuần nhất" — purity, của các đối tượng dữ liệu trong một node.

C4.5 được xem là phiên bản nâng cấp của ID3 với khả năng xử lý được cả dữ liệu định lượng dạng liên tục (continuous data) và cả dữ liệu định tính, và là thuật toán Decision tree tiêu biểu. C4.5 sử dụng thêm công thức Gain ratio để khắc phục các khuyết điểm của Information Gain của ID3 trong việc lựa chọn cách phân nhánh tối ưu.

C4.5 xem xét đến từng biến đầu vào, từng node trên cây quyết định, tìm ra các phân nhánh tối ưu cho đến khi không thể phân nhánh được nữa để phân loại chính xác cho đối tượng dữ liệu. Tuy nhiên C4.5 có một số điểm khác biệt sau:

- Thuật toán C4.5 không bị giới bởi số nhánh có thể phân trong mỗi lần và C4.5 thì có thể xây dựng cây quyết định với nhiều nhánh ở mỗi tầng.
- C4.5 thì sử dụng công thức Entropy và Information Gain, và Gain Ratio để tiến hành lựa chọn cách thức phân nhánh.

a) Công thức Entropy – cơ sở xây dựng thuật toán C4.5

Entropy là phương pháp lựa chọn cách phân nhánh tối ưu dựa trên cơ sở tối đa hóa lượng thông tin nhận vào "Information maximization" tức giảm thiểu tối đa độ hỗn độn và nhiễu loạn trong từng node "Entropy reduction". Các node phân nhánh được lựa chọn theo phương pháp này phải thể hiện tối đa thông tin cần thiết để cây quyết định có thể phân loại chính xác đối tượng dữ liệu vào các tập con có chứa nhãn, là giá trị/ thuộc tính của biến đầu vào.

Công thức tính Entropy

$$\text{Entropy(t)} = -\sum_j p(j|t) * \log_2 p(j|t)$$

Trong đó: p(j|t) là xác suất xuất hiện đối tượng dữ liệu mang thuộc tính j của biến mục tiêu trong node t. Khi p(j|t) càng lớn thì log2 của p(j|t) sẽ mang giá trị âm và tiến gần đến 0, nhân với giá trị âm p(j|t) ở đầu công thức luôn bé hơn 1, sẽ được giá trị dương tiến gần đến 0. Giá trị Entropy càng nhỏ thì tức node hay tập con chứa nhiều đối tượng dữ liệu có cùng thuộc tính j bất kỳ.

b) Công thức Information Gain

Information Gain - nếu dịch trực tiếp ra nghĩa tiếng Việt thì là "lượng thông tin có được", có thể hiểu đơn giản đây là công thức để xác định xem trong số các cách thức phân nhánh thì cách nào đem lại nhiều "thông tin nhất", "rõ ràng nhất", đầy đủ cơ sở nhất để chúng ta phân loại đối tượng dữ liệu theo các giá trị, các nhóm, các phân lớp có sẵn của biến mục tiêu.
Công thức tính:

$$GAIN_{split} = \ Entropy(p) - (\sum_{i=1}^{N} \frac{n_i}{n} * Entropy(i))$$

Tuy nhiên nhiều chuyên gia cho rằng Information Gain vẫn còn nhiều hạn chế nhất định ví dụ trong trường hợp biến cần xét có quá nhiều giá trị khác biệt (distinct values).
Information Gain có khuynh hướng thiên vị hay còn gọi "bias" cho các biến chứa nhiều giá trị dữ liệu khác nhau. Gain ratio do đó được ra đời như một bảng nâng cấp của Information Gain và hạn chế được các khuyết điểm của nó khi xét cả số lượng và quy mô của các nhánh trong khi tính toán:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

Với SplitINFO được tính

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

SplitInfo chính là "thông tin có được" về số lượng đối tượng dữ liệu trong mỗi nhánh.

c) Tiêu chuẩn dừng

Trong các thuật toán Decision tree, ta sẽ chia mãi các node nếu nó chưa tinh khiết. Như vậy, ta sẽ thu được một tree mà mọi điểm trong tập huấn luyện đều được dự đoán đúng (giả sử rằng không có hai input giống nhau nào cho output khác nhau). Khi đó, cây có thể sẽ rất phức tạp (nhiều node) với nhiều leaf node chỉ có một vài điểm dữ liệu. Như vậy, nhiều khả năng overfitting sẽ xảy ra.
Để tránh trường hợp này, ta có thể dừng cây theo một số phương pháp sau đây:

- Nếu node đó có entropy bằng 0, tức mọi điểm trong node đều thuộc một class.
- Nếu node đó có số phần tử nhỏ hơn một ngưỡng nào đó. Trong trường hợp này, ta chấp nhận có một số điểm bị phân lớp sai để tránh overfitting. Class cho leaf node này có thể được xác định dựa trên class chiếm đa số trong node.

- Nếu khoảng cách từ node đó đến root node đạt tới một giá trị nào đó. Việc hạn chế chiều sâu của tree này làm giảm độ phức tạp của tree và phần nào giúp tránh overfitting.
- Nếu tổng số leaf node vượt quá một ngưỡng nào đó.
- Nếu việc phân chia node đó không làm giảm entropy quá nhiều (information gain nhỏ hơn một ngưỡng nào đó).

2. Ưu điểm & khuyết điểm của thuật toán cây quyết định

Decision trees có các ưu điểm và khuyết điểm mà chúng ta phải quan tâm và nhìn lại, lấy đó làm cơ sở để áp dụng vào các dự án khai thác dữ liệu sao cho phù hợp, và hiệu quả.

Ưu điểm:

- Thuật toán Decision trees đơn giản, trực quan, không quá phức tạp để hiểu ngay lần đầu tiên. Đồng thời bộ dữ liệu training không nhất thiết phải quá lớn để tiến hành xây dựng mô hình phân tích.
- Một số thuật toán cây quyết định có khả năng xử lý dữ liệu bị missing và dữ liệu bị lỗi và ít bị ảnh hưởng bởi các dữ liệu ngoại lệ (outliers)
- Thuật toán cây quyết định là phương pháp không sử dụng tham số, "nonparametric", nên không cần phải có các giả định ban đầu về các quy luật phân phối như trong thống kê, và nhờ đó kết quả phân tích có được là khách quan, "tự nhiên" nhất.
- Thuật toán cây quyết định có thể áp dụng linh hoạt cho các biến target, biến mục tiêu là biến định tính (classification task), và cả định lượng (regression task)

Khuyết điểm

- Thuật toán cây quyết định hoạt động hiệu quả trên bộ dữ liệu đơn giản có ít biến dữ liệu liên hệ với nhau, và ngược lại nếu áp dụng cho bộ dữ liệu phức tạp, nhiều biến và thuộc tính khác nhau có thể dẫn đến mô hình bị overfitting, quá khớp với dữ liệu training dẫn đến vấn đề không đưa ra kết quả phân loại chính xác khi áp dụng cho dữ liệu test, và dữ liệu mới.
- Đối với thuật toán cây quyết định khi có sự thay đổi nhỏ trong bộ dữ liệu có thể gây ảnh hưởng đến cấu trúc của mô hình. Nghĩa là khi chúng ta điều chỉnh dữ liệu, cách thức phân nhánh, ngắt cây sẽ bị thay đổi, có thể dẫn đến kết quả sẽ khác so với ban đầu, phức tạp hơn.
- Thuật toán cây quyết định có khả năng "bias" hay thiên vị nếu bộ dữ liệu không được cân bằng.
- Thuật toán cây quyết định yêu cầu bộ dữ liệu training và test phải được chuẩn bị hoàn hảo, chất lượng tốt phải được cân đối theo các lớp, các nhóm trong biến mục tiêu. Ngoài ra biến mục tiêu phải có các giá trị "rời rạc" dễ nhận biết, không được quá đa dạng, và phải cụ thể để quá trình phân loại diễn ra dễ dàng hơn cho thuật toán.

- Thuật toán cây quyết định được hình thành trên các cách thức phân nhánh tại mỗi một thời điểm bất kỳ, ở một node hay biến dữ liệu bất kỳ và chỉ quan tâm duy nhất vào việc phân nhánh sao cho tối ưu tại thời điểm ấy, chứ không xét đến toàn bộ mô hình phải được thiết lập hiệu quả ra sao.

III. Cài đặt thuật toán

1. Các lý thuyết sử dụng cài đặt thuật toán

1.1. Độ đo

Sử dụng độ đo là GainRatio để chọn node tối ưu nhất phân chia

1.2. Điểm dừng

Các điều kiện để có thể dừng việc chia node:

- Tất cả các nhãn trong tập dữ liệu của node đó đều giống nhau
- Nếu không còn thuộc tính nào để xét nữa thì sẽ lấy nhãn chiếm đa số làm node lá
- Nếu tất cả các thuộc tính của node đều không thể chia nhánh(tức trong 1 cột chỉ có 1 giá trị), ta chọn node đấy làm node lá và lấy nhãn chiếm đa số gán cho node lá đó.

1.3. Cách chia nhánh

- Đối với thuộc tính phân loại: Ta chia node với số nhánh bằng số giá trị của thuộc tính, mỗi nhánh thuộc về một giá trị nào đó
- Đối với thuộc tính liên tục:
  + Sắp xếp giá trị của cột thuộc tính theo giá trị tăng dần
  + Lấy giá trị trung bình giữa 2 giá trị khác nhau liền kề để xét làm ngưỡng
  + Các giá trị liền kề khác nhau nhưng đều thuần nhất (chỉ có 1 nhãn) và giống nhau thì ta gộp 2 giá trị lại và bỏ ngưỡng giữa 2 giá trị đó đi.
  + Chọn số nhánh có thể chia, với từng cách chia nhánh kiểm tra xem cách chia nào tối ưu nhất (gainRatio lớn nhất) thì chọn làm cách chia nhánh cho thuộc tính đó.

1.4. Tỉa cây (Post pruning)

Sau khi xây dựng cây xong, sử dụng phương pháp Reduced Error Pruning để xem xét tỉa các nhánh cây không cần thiết

- Chia một tập dữ liệu validation để pruning các nhánh của cây.
- Tách tập validation thành các tập tương ứng với từng giá trị phân chia của node trong cây đã xây, lặp lại cho đến khi gặp node lá thì dừng.

- Xuất phát từ các node lá ở dưới cùng và chuyển node cha trực tiếp của các node lá đấy bằng cách lấy nhãn chiếm đa số làm nhãn.
- Tính độ chính xác trên tập validation trước và sau khi cắt tỉa nhánh.
- So sánh 2 độ chính xác, nếu độ chính xác sau khi cắt tỉa tăng lên thì ta gộp các node lá lại.

2. Các hàm trong thuật toán cài đặt

2.1. Khởi tạo đối tượng Node

Với các đối số truyền vào:

isLeaf(bool): Có phải lá hoặc không.

label(str): Thuộc tính của node. Nếu là node lá thì là nhãn kết quả

threshold(list): ngưỡng chia tối ưu tại Node.

gainRatio(float): Giá trị gainRatio tại Node.

category(list): chứa các giá trị phân loại của với thuộc tính categorical

2.2. Khởi tạo đối tượng C45

Đối số truyền vào: limitPartition(int): Giới hạn số khoảng chia.

reuseAttribute(bool): Xét lại các thuộc tính đã chọn

2.3. Huấn luyện bộ phân loại cây quyết định: fit()

Đối số truyền vào:

data(DataFrame): Tập dữ liệu loại bỏ cột nhãn.

labels(Series): Tập nhãn của data.

2.4. Tạo node cho cây:

Đối số truyền vào:

data(array): Mảng 2 chiều(numpy) chứa tập dữ liệu tại Node.

columns(list): Danh sách chứa các thuộc tính của data không bao gồm thuộc tính nhãn.

Giá trị trả về:

Node: Node đã được tạo thành

Mã giả:

- Kiểm tra tất cả có cùng nhãn không

  Nếu tất cả cùng nhãn

Trả về node hiện tại là một lá cùng với nhãn chung của tập dữ liệu

- Nếu không còn thuộc tính được xét

  Trả về Node hiện tại là một lá cùng với nhãn phổ biến của tập dữ liệu

- Nếu không

  Tìm thuộc tính tốt nhất, cách chia dữ liệu tốt nhất

  Nếu dữ liệu không được chia

  Trả về Node hiện tại là một lá cùng với nhãn phổ biến của tập dữ liệu

  Điền giá trị cho node hiện tại

  Nếu là giá trị rời rạc

    Gán tập giá trị rời rạc cho node

  Xóa thuộc tính đã được xét

  Tiếp tục tạo node tương ứng với các đoạn dữ liệu đã được chia

- Trả về node hiện tại sau khi đã tạo hết các con của nó

ll SameClass

2.5. Tìm thuộc tính tốt nhất, cách chia tốt nhất

Đối số truyền vào:

data(array): Mảng 2 chiều(numpy) chưa tập dữ liệu.

columns(list): danh sách các thuộc tính còn lại.

Giá trị trả về:

bestAttribute(str): Tên thuộc tính được chọn.

bestThreshold(list): Danh sách các giá trị ngưỡng.

bestPartitions(list): Danh sách các đoạn dữ liệu được chia tối ưu nhất.

gainRatio(float): Giá trị gainRatio tốt nhất cho tập dữ liệu.

Mã giả:

Khởi tạo các giá trị thể hiện cho chưa tìm được kết quả

Duyệt qua tất cả các thuộc tính cần lựa chọn

  Nếu là giá trị rời rạc

    Tạo danh sách chứa các giá trị phân biệt

    Tính gain ratio

Nếu gain ratio > max gain ratio

Thuộc tính hiện tại là tối ưu, max gain ratio = gain ratio

Nếu là giá trị liên tục

Sắp xếp lại dữ liệu theo giá trị của thuộc tính hiện tại

Tạo danh sách các ngưỡng: Giá trị của ngưỡng bằng trung bình cộng của hai giá trị khác nhau liền kề. Nếu tất cả các nhãn của hai giá trị liền kề đó giống nhau thì sẽ bỏ qua giá trị ngưỡng giữa 2 giá trị.

Nếu không có ngưỡng nào

Tính gain ratio

Thuộc tính hiện tại là tối ưu, max gain ratio = gain ratio

Khởi tạo số khoảng cần chia: partition = 2

Lặp đến khi số khoảng cần chia lớn hơn số khoảng tối đa

Tìm tất cả số cách chia có thể thành partition khoảng

Tính gain ratio của tất cả các cách chia

Chọn cách chia có gain ratio cao nhất

tăng partition lên 1

Trả về thuộc tính tốt nhất, cách chia tốt nhất

2.6. Tìm tất cả số cách chia dữ liệu thành partition đoạn

Đối số đầu vào:

lenThreshold(int): Độ dài của đoạn dữ liệu cần phải chia.

partition(int): Số đoạn chia.

Giá trị trả về:

list: Danh sách với mỗi phần tử là một danh sách chứa 1 cách chia - chứa thông tin vị trí của các giá trị ngưỡng được chọn.

Mã giả:

Hàm quay lui tìm cách chia

Đối số:

currIndexOfThreshold(int): chỉ mục của ngưỡng hiện tại

currPartition(int): số đoạn đã chia

Mã giả:

Nếu đã duyệt hết dữ liệu

Dừng

Nếu đã chia đủ đoạn(currPartition = partition)

Thêm vào kết quả cách chia hiện tại

Lời gọi quay lui với currIndexOfThreshold + 1, currPartion: Tức phần tử sau vẫn thuộc đoạn hiện tại

Nếu chưa chia đủ đoạn

Lời gọi quay lui với currIndexOfThreshold + 1,

currPartion + 1: Tức phần tử sau thuộc đoạn mới

Lời gọi quay lui với currIndexOfThreshold + 1,

currPartion: Tức phần tử sau vẫn thuộc đoạn hiện tại

Khởi tạo kết quả

Khởi tạo danh sách chứa cách chia hiện tại

Lời gọi quay lui với currIndexOfThreshold = 0, currPartition = 2: Tức bắt đầu chia từ phần tử đầu tiên với tối thiểu là chia dữ liệu được thành 2 đoạn

Trả về kết quả

2.7. Kiểm tra thuộc tính có phải rời rạc không

Đối số đầu vào:

attrValues(str): Tên của thuộc tính cần kiểm tra.

Giá trị trả về:

True: Thuộc tính rời rạc.

False: Thuộc tính liên tục.

2.8. Tính gainRatio

Đối số:

data(array): Tập dữ liệu cần phải tính.

partitions(list): Danh sách các đoạn dữ liệu được chia.

Trả về:

float: Giá trị gainRatio tương ứng.

2.9. Tính gainSplit

Đối số:

data(array): Tập dữ liệu cần phải tính.

partitions(list): Danh sách các đoạn dữ liệu được chia.

Trả về:

float: Giá trị gainSplit tương ứng.

2.10. Tính entropy

Đối số:

data(array): Tập dữ liệu cần phải tính.

Trả về:

float: Giá trị entropy tương ứng.

2.11. Tính splitInfo

Đối số:

data(array): Tập dữ liệu cần phải tính.

partitions(list): Danh sách các đoạn dữ liệu được chia.

Trả về:

float: Giá trị splitInfo tương ứng.

2.12. Kiểm tra tất cả bản ghi còn lại có cùng nhãn không

Đối số:

labels(list): Danh sách các nhãn.

Trả về:

int: Nhãn chung của tập giá trị.

None: Tồn tại nhãn khác nhau.

2.13. Tìm nhãn phổ biến nhất trong tất cả các bản ghi còn lại

Đối số:

labels(list): Danh sách các nhãn.

Trả về:

int: Giá trị nhãn xuất hiện nhiều nhất.

2.14. Dự đoán nhãn của một tập dữ liệu.

Đối số:

data(array): Tập dữ liệu cần dự đoán.

Trả về:

list: Danh sách chứa các nhãn dự đoán.

Mã giả:

Khởi tạo kết quả

Duyệt qua từng dòng:

Gọi hàm dự đoán nhãn cho dòng hiện tại

Lấy ra nhãn từ node tương đồng và lưu nhãn vào kết quả

Trả về kết quả

2.15. Tìm node thỏa mãn giá trị của dòng cần dự đoán hiện tại.

Đối số:

node(Node): Node hiện tại.

row(array): dòng cần dự đoán.

Kết quả: Nhãn dữ đoán

Mã giả:

- Thực hiện kiểm tra giá trị của dữ liệu trên cột tương ứng với nhánh phân loại của node hiện tại xem thuộc node con nào
    + Nếu là thuộc tính phân loại, duyệt qua các node con của node hiện tại và so sánh giá trị thuộc tính tương ứng với dữ liệu với giá trị thuộc tính của nhánh phân loại. Nếu 2 giá trị bằng nhau thì gọi đệ quy trên phân nhánh đó
    + Nếu là thuộc tính liên tục, duyệt qua các ngưỡng của node hiện tại, nếu giá trị tương ứng của dữ liệu nằm giữa khoảng giá trị được xác định bởi ngưỡng trong phân nhánh hiện tại thì thực hiện gọi đệ quy trên phân nhánh đó
- Nếu node của nhánh tiếp theo là node lá thì gán kết quả dự đoán là giá trị của node lá vào biến predict_label

2.16. Tỉa cây (Post pruning)

- Sử dụng phương pháp Reduced error pruning để cắt tỉa cây: Bỏ dần các node lá, chuyển các node cha của node lá thành node lá. So sánh Accuracy trên node cha trước và sau khi tỉa lá bằng tập dữ liệu validation. Nếu Accuracy tăng lên sau khi tỉa là thì ta bỏ các node lá cũ và chuyển node cha thành node lá.
- Đối số:

    + Node hiện tại
    + Tập dữ liệu validation
- Kết quả:
    + Cây sau khi được cắt tỉa
- Mã giả:
    + Chia tập dữ liệu validation thành các phân vùng tương ứng với các nhánh con của node. Sau đó nếu các node chưa phải node lá và phân vùng dữ liệu validation tương ứng với node con đó không rỗng thì sẽ gọi đệ quy trên nút con và tập dữ liệu validation đó.
    + Kiểm tra độ chính xác trước khi thực hiện pruning (accuracy_before_pruning) trên tập dữ liệu validation trên node hiện tại.
    + Chọn nhãn chiếm đa số trong node hiện tại, gán cho node hiện tại là node lá với giá trị nhãn là nhãn đa số. Sau đó kiểm tra độ chính xác sau khi pruning (accuracy_after_pruning)
    + Nếu accuracy_after_pruningg >= accuracy_before_pruning, thực hiện pruning chuyển node hiện tại thành node lá với nhãn là nhãn chiếm đa số
    + Nếu accuracy_after_pruningg < accuracy_before_pruning, giữ nguyên node hiện tại

IV. Kiểm định và đánh giá thuật toán

Bộ dữ liệu sử dụng: Heart Disease Dataset

Bộ dữ liệu này có từ năm 1988 và bao gồm bốn cơ sở dữ liệu: Cleveland, Hungary, Thụy Sĩ và Long Beach V. Nó chứa 76 thuộc tính, bao gồm thuộc tính dự đoán, nhưng tất cả các thí nghiệm được công bố đều đề cập đến việc sử dụng một tập hợp con gồm 14 trong số đó. Trường " target " đề cập đến sự hiện diện của bệnh tim ở bệnh nhân. Nó có giá trị số nguyên 0 = không có bệnh và 1 = bệnh.

Thông tin thuộc tính:

1. (age) – tuổi
2. (sex) - giới tính
3. (cp) – Loại đau ngực (4 giá trị)
4. (trestbps) – Huyết áp khi cơ thể ở trạng thái nghỉ
5. (chol) - Cholestoral huyết thanh trong Mg/DL
6. (fbs) - Lượng đường trong máu khi nhịn ăn> 120 mg/dl
7. (restecg) – Kết quả điện tâm đồ khi cơ thể ở trạng thái nghỉ (giá trị 0,1,2)
8. (thalach) - Nhịp tim tối đa
9. (exang) - Tập thể dục có gây đau thắt ngực không (1 – có, 0 – không)
10. (oldpeak) - Suy giảm ST do tập thể dục so với nghỉ ngơi
11. (slope) - độ dốc của đoạn ST đỉnh tập thể dục

12. (ca) - số mạch chính (0-3) được tô màu bằng fluoroscopy
13. (thal) - 0 = bình thường; 1 = khuyết tật cố định; 2 = khuyết tật đảo ngược
14. (target) (thuộc tính dự đoán) đề cập đến sự hiện diện của bệnh tim ở bệnh nhân. Nó có giá trị số nguyên 0 = không có bệnh và 1 = bệnh

Thống kê mô tả bộ dữ liệu

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	1025	1025	1025	1025	1025	1025	1025	1025	1025	1025	1025	1025	1025	1025
mean	54.43415	0.69561	0.942439	131.6117	246	0.149268	0.529756	149.1141	0.336585	1.071512	1.385366	0.754146	2.323902	0.513171
std	9.07229	0.460373	1.029641	17.51672	51.59251	0.356527	0.527878	23.00572	0.472772	1.175053	0.617755	1.030798	0.62066	0.50007
min	29	0	0	94	126	0	0	71	0	0	0	0	0	0
25%	48	0	0	120	211	0	0	132	0	0	1	0	2	0
50%	56	1	1	130	240	0	1	152	0	0.8	1	0	2	1
75%	61	1	2	140	275	0	1	166	1	1.8	2	1	3	1
max	77	1	3	200	564	1	2	202	1	6.2	2	4	3	1

Thư viện tham chiếu: C4point5 – PyPI

Cài đặt để sử dụng module: pip install C4point5

Kết quả khi chạy trên tập dữ liệu Heart Disease Dataset

Lần 1: Lấy ngẫu nhiên 70% tập dữ liệu cho việc huấn luyện và 30% cho việc test

Kết quả chạy với thuật toán cài đặt

Thuật toán cài đặt(tối đa 2 nhánh)	Thư viện C4point5

cp <= 0.5, gr = 0.218 :
  thalach <= 181.5, gr = 0.368 :
  ca <= 0.5, gr = 0.197 :
  trestbps <= 107.0 : 1
  trestbps > 107.0, gr = 0.219 :
  ca > 0.5, gr = 0.197 :
  trestbps <= 109.0, gr = 0.063 :
  chol <= 233.5 : 1
  chol > 233.5 : 0
  trestbps > 109.0, gr = 0.063 :
  age <= 63.5 : 0
  age > 63.5, gr = 0.069 :
cp > 0.5, gr = 0.218 :
  trestbps <= 179.0, gr = 0.278 :
  thalach <= 113.0, gr = 0.184 :
  thal <= 2.5 : 1
  thal > 2.5 : 0
  thalach > 113.0, gr = 0.184 :
  oldpeak <= 2.45, gr = 0.164 :
  age <= 56.5, gr = 0.117 :
  chol <= 153.0, gr = 0.234 :
	cp <= 0.5, gr=0.218 :
	  thalach <= 181.0, gr=0.368 :
	  ca <= 0.5, gr=0.197 :
	  oldpeak <= 0.7, gr=0.201 :
	  oldpeak > 0.7, gr=0.201 :
	  ca > 0.5, gr=0.197 :
	  trestbps <= 109.0, gr=0.063 :
	  chol <= 233.5 : 1
	  chol > 233.5 : 0
	  trestbps > 109.0, gr=0.063 :
	  thalach <= 123.0, gr=0.052 :
	  thalach > 123.0 : 0
	cp > 0.5, gr=0.218 :
	  trestbps <= 179.0, gr=0.278 :
	  thalach <= 113.0, gr=0.184 :
	  thal <= 2.5 : 1
	  thal > 2.5 : 0
	  thalach > 113.0, gr=0.184 :
	  oldpeak <= 2.45, gr=0.164 :
	  age <= 56.5, gr=0.117 :
	  chol <= 153.0, gr=0.234 :

ca <= 1.5 : 1 ca > 1.5 : 0 chol > 153.0, gr = 0.234 : thal <= 2.5, gr = 0.081 : ***restecg <= 0.5 : 1*** ***restecg > 0.5, gr = 0.009*** : thal > 2.5, gr = 0.081 : ***ca <= 2.0, gr = 0.466*** ***ca > 2.0 : 0*** age > 56.5, gr = 0.117 : ***chol <= 337.5, gr = 0.133 :*** ***chol > 337.5 : 1***	ca <= 1.5 : 1 ca > 1.5 : 0 chol > 153.0, gr=0.234 : thal <= 2.0, gr=0.081 : ***trestbps <= 109.0, gr=0.098 :*** ***trestbps > 109.0 : 1*** thal > 2.0, gr=0.081 : ***thalach <= 143.0 : 0*** ***thalach > 143.0, gr=0.545*** age > 56.5, gr=0.117 : ***age <= 69.5, gr=0.149 :*** ***age > 69.5 : 1***
Accuracy: 92.83%	Accuracy: 96.74%

So sánh kết quả dự đoán của 2 cây trên cùng tập dữ liệu test

	Nhan thuc te	Nhan du doan boi cay quyet dinh su dung thu vien C4point5	Nhan du doan boi cay quyet dinh xay dung duoc
1			
2	1	1	1
3	0	0	0
4	1	1	1
5	0	1	1
6	0	0	0
7	1	1	1
8	0	0	0
9	0	0	0
10	0	0	0
11	1	1	1
12	0	0	0
13	0	1	1
14	0	0	0
15	0	0	0
16	0	0	0
17	1	1	1
18	1	1	1
19	1	1	1
20	1	1	1
21	1	1	1
22	1	1	1
23	1	1	1
24	1	1	1
25	1	1	1
26	1	1	1
27	0	0	0
28	0	0	0

...

279	0	1	1
280	0	0	0
281	0	0	0
282	1	1	1
283	0	0	0
284	1	1	1
285	1	1	1
286	0	0	0
287	0	0	0
288	0	0	0
289	1	1	1
290	0	0	0
291	0	0	0
292	0	0	0
293	1	1	1
294	0	0	0
295	1	1	1
296	0	0	0
297	1	1	1
298	0	0	0
299	0	0	0
300	1	1	1
301	1	1	1
302	1	1	1
303	1	0	0
304	0	0	0
305	1	1	1
306	1	1	1
307	0	0	0
308	0	0	0
309	0	0	0

Tỷ lệ dự đoán giống nhau giữa 2 cây: 94.15584415584416 %

Ở cây quyết định có sự khác nhau giữa các node:

+ Các node ở thuật toán cài đặt có gainRatio cao hơn thư viện C4point5 là do cách chọn ngưỡng. Trong thuật toán cài đặt, các ngưỡng được chọn ở vị trí giữa các giá trị khác nhau. Còn trong thư viện C4point5, các ngưỡng chỉ được chọn tại vị trí các nhãn khác

nhau, sau đó lấy giá trị trung bình của 2 giá trị tương ứng với 2 nhãn đấy. Do đó các giá trị ngưỡng có thể có trong thuật toán cài đặt nhiều hơn, từ đó có thể tìm được giá trị ngưỡng tối ưu nhất

+ Các node ở thuật toán cài đặt có gainRatio thấp hơn thư viện C4point5 là do việc C4point5 xét lại các thuộc tính đã được xét ở các nhánh cha ở trên. Việc xét lại các giá trị thuộc tính đã được xét có thể tăng thông tin của nhánh dẫn đến gainRatio cao hơn. Tuy nhiên việc xét lại các thuộc tính có thể không đem lại kết quả tốt về mặt xây dựng cây khi phải xét lại liên tục thuộc tính. Từ đó có thể dẫn đến việc xây dưng cây quá lâu.
+ Kết quả Accuracy của thư viện C4point5 cao hơn thuật toán cài đặt, mặc dù có các node thuật toán cài đặt có gainRatio cao hơn, nhưng việc C4point5 xét lại các thuộc tính có thể đem giá trị tối ưu hơn nên cây có thể fit hơn với dữ liệu, từ đó kết quả dự đoán tốt hơn

Thuật toán cài đặt(tối đa 3 nhánh)	Thư viện C4point5
cp <= 0.5, gr = 0.218 : thalach <= 181.5, gr = 0.368 : ca <= 0.5, gr = 0.197 : ***trestbps <= 107.0 : 1*** ***trestbps > 107.0, gr = 0.219 :*** ca > 0.5, gr = 0.197 : ***trestbps <= 103.0 : 0*** ***103.0 < trestbps <= 107.0 : 1*** ***trestbps > 107.0, gr = 0.125 :*** thalach > 181.5 : 1 cp > 0.5, gr = 0.218 : trestbps <= 179.0, gr = 0.278 : ***thalach <= 96.5 : 1*** ***96.5 < thalach <= 113 : 0*** ***thalach > 113.0, gr = 0.262 :*** trestbps > 179.0 : 0	cp <= 0.5, gr=0.218 : thalach <= 181.0, gr=0.368 : ca <= 0.5, gr=0.197 : ***oldpeak <= 0.7, gr=0.201 :*** ***oldpeak > 0.7, gr=0.201 :*** ca > 0.5, gr=0.197 : ***trestbps <= 109.0, gr=0.063 :*** ***trestbps > 109.0, gr=0.063 :*** thalach > 181.0 : 1 cp > 0.5, gr=0.218 : trestbps <= 179.0, gr=0.278 : ***thalach <= 113.0, gr=0.184 :*** ***thalach > 113.0, gr=0.184 :*** trestbps > 179.0 : 0
Accuracy: 93.49%	Accuracy: 96.74%

+ Trong thuật toán cài đặt có thể chia tối đa 3 nhánh, các node cùng được chọn giống nhau như trestbps, thì thuật toán cài đặt có 3 ngưỡng 103, 107 với gainRatio 0.219 được chia tốt hơn so với thư viện C4point5 chỉ có 0.063 chỉ có 1 ngưỡng 109, hay node ‘thalach’ có 2 ngưỡng 96.5, 113 có gainRatio tốt hơn với 0.262 so với 0.184 chỉ có 1 ngưỡng 113 của C4point5.

+ Như vậy có thể thấy việc chia node với số nhánh nhiều hơn có thể đem lại kết quả tối ưu hơn trong việc chia Node với kết quả accuracy của cây có tối đa 3 nhánh hơn cây tối đa 2 nhánh với 93,49% so với 92,83%
+ Tuy nhiên, trong thư viện C4point5 có xét lại các thuộc tính đang xét ở các nhánh sau đó, cho nên cây C4point5 fit với dữ liệu hơn và kết quả dự đoán tốt hơn thuật toán cài đặt.

Lần 2: Kiểm định lại bộ dữ liệu với thuật toán cài đặt có xét lại thuộc tính

Thuật toán cài đặt(tối đa 2 nhánh)	Thư viện C4point5
cp <= 0.5, gr = 0.192 : thalach <= 181.5, gr = 0.327 : ca <= 0.5, gr = 0.184 : age <= 41.5 : 0 age > 41.5, gr = 0.221 : ***trestbps <= 117.5 : 1*** ***trestbps > 117.5, gr = 0.254 :*** ca > 0.5, gr = 0.184 : trestbps <= 109.0, gr = 0.222 : chol <= 233.5 : 1 chol > 233.5 : 0 trestbps > 109.0, gr = 0.222 : ***age <= 63.5 : 0*** ***age > 63.5, gr = 0.08*** : thalach > 181.5 : 1 cp > 0.5, gr = 0.192 : trestbps <= 179.0, gr = 0.247 : oldpeak <= 2.45, gr = 0.21 : thalach <= 109.0, gr = 0.159 : thal <= 2.5 : 1 thal > 2.5 : 0 thalach > 109.0, gr = 0.159 : age <= 44.5 : 1 age > 44.5, gr = 0.099 : chol <= 154.5, gr = 0.09 : thal <= 2.5 : 0 thal > 2.5 : 1 chol > 154.5, gr = 0.09 : sex <= 0.5, gr = 0.08 : ca <= 1.5, gr = 0.133 : thalach <= 173.0 : 1 thalach > 173.0, gr = 0.322 : ca <= 0.5 : 1	cp <= 0.5, gr=0.192 : thalach <= 181.0, gr=0.327 : ca <= 0.5, gr=0.184 : age <= 41.5 : 0 age > 41.5, gr=0.221 : ***fbs <= 0.5, gr=0.228 :*** ***fbs > 0.5 : 0*** ca > 0.5, gr=0.184 : trestbps <= 109.0, gr=0.222 : chol <= 233.5 : 1 chol > 233.5 : 0 trestbps > 109.0, gr=0.222 : ***thalach <= 123.0, gr=0.05 :*** ***thalach > 123.0 : 0*** thalach > 181.0 : 1 cp > 0.5, gr=0.192 : trestbps <= 179.0, gr=0.247 : oldpeak <= 2.45, gr=0.21 : thalach <= 109.0, gr=0.159 : thal <= 2.5 : 1 thal > 2.5 : 0 thalach > 109.0, gr=0.159 : age <= 44.5 : 1 age > 44.5, gr=0.099 : chol <= 154.5, gr=0.09 : thal <= 2.5 : 0 thal > 2.5 : 1 chol > 154.5, gr=0.09 : sex <= 0.5, gr=0.08 : ca <= 1.5, gr=0.133 : thalach <= 173.0 : 1 thalach > 173.0, gr=0.322 : ca <= 0.5 : 1

ca > 0.5 : 0 ca > 1.5, gr = 0.133 : restecg <= 0.5 : 0 restecg > 0.5 : 1 sex > 0.5, gr = 0.08 : thalach <= 121.5 : 0 thalach > 121.5, gr = 0.231 : chol <= 330.0, gr = 0.24 : thalach <= 174.5, gr = 0.116 : ***trestbps <= 106.5 : 1*** trestbps > 106.5, gr = 0.118 :	ca > 0.5 : 0 ca > 1.5, gr=0.133 : restecg <= 0.5 : 0 restecg > 0.5 : 1 sex > 0.5, gr=0.08 : thalach <= 121.5 : 0 thalach > 121.5, gr=0.231 : chol <= 330.0, gr=0.24 : thalach <= 174.0, gr=0.116 : ***oldpeak <= 2.25, gr=0.098 :***
Accuracy: 97.1%	Accuracy: 96,12%

Thuật toán cài đặt(tối đa 3 nhánh)	Thư viện C4point5
cp <= 0.5, gr = 0.192 : thalach <= 181.5, gr = 0.327 : ***chol <= 353.5, gr = 0.189 :*** ***chol < 353.5 <= 400.5: 1*** ***chol > 400.5 : 0*** cp > 0.5, gr = 0.192 : ***thalach <= 96.5 : 1*** ***thalach < 96.5 <= 113.0: 0*** ***thalach > 113.0, gr = 0.249*** :	cp <= 0.5, gr=0.192 : thalach <= 181.0, gr=0.327 : ***ca <= 0.5, gr=0.184 :*** ***ca > 0.5, gr=0.184 :*** cp > 0.5, gr=0.192 : ***trestbps <= 179.0, gr=0.247 :*** ***trestbps > 179.0 : 0***
Accuracy: 100%	Accuracy: 96,12%

Trong lần kiểm định này, so sánh cả 2 cây khi đều xét lại thuộc tính đã được chọn từ nhánh trên. Từ kết quả trên có thể thấy được sự cải thiện về độ chính xác của cây quyết định trong thuật toán cài đặt. Các node bắt đầu có sự khác nhau giữa 2 cây thì gainRatio của thuật toán cài đặt đều cao hơn thư viện C4point5. Có thể thấy được phương pháp chia ngưỡng của thuật toán cài đặt đem lại kết quả tối ưu hơn về việc tăng độ thông tin khi chia nhánh. Kết quả kiểm thử bộ test với Accuracy của thuật toán cài đặt là đều cao hơn thư viện C4point5,

với cây có tối đa 2 nhánh là 97% và cây có tối đa 3 nhánh là 100 % so với 96,12% của cây trong thư viên C4point5

Lần 3: Sử dụng bộ dữ liệu lớn hơn

Bộ dữ liệu sử dụng: Credit Card Classification - clean data

Tập dữ liệu này chứa phiên bản sạch của tập dữ liệu Credit Card Approval Prediction

Thông tin thuộc tính:

1. ID - ID của khách hàng
2. Gender - Giới tính (1=Nam, 0=Nữ)
3. Own_car - Khách hàng có sở hữu xe hơi không? (1=Có, 0=Không)
4. Own_property - Khách hàng có sở hữu tài sản không? (1=Có, 0=Không)
5. Work_phone - Khách hàng có sở hữu điện thoại cơ quan không? (1=Có, 0=Không)
6. Phone - Khách hàng có sở hữu điện thoại không? (1=Có, 0=Không)
7. Email - Khách hàng có địa chỉ email không? (1=Có, 0=Không)
8. Unemployed - Khách hàng đang thất nghiệp? (1=Có, 0=Không)
9. Num_children - Số lượng trẻ em
10. Num_family - số thành viên trong gia đình
11. Account_length - Số tháng thẻ tín dụng đã được sở hữu
12. Total_income - Tổng thu nhập
13. Age - Tuổi theo năm
14. Years_employed - Số năm làm việc
15. Income_type - loại thu nhập
16. Education_type - loại hình giáo dục
17. Family_status - Tình trạng gia đình
18. Housing_type - loại nhà ở
19. Occupation_type - Loại nghề nghiệp
20. Target - Biến mục tiêu (1=rủi ro cao, 0=rủi ro thấp)

Thống kê mô tả:

	ID	Gender	Own_car	Own_property	Work_phone	Phone	Email	Unemployed	Num_children	Num_family	Account_length	Total_income	Age	Years_employed	Target
count	9709	9709	9709	9709	9709	9709	9709	9709	9709	9709	9709	9709	9709	9709	9709
mean	5076105	0.348749	0.3677	0.671542	0.217427	0.287671	0.087548	0.174683	0.422804	2.182614	27.27006	181228.2	43.78409	5.66473	0.132145
std	40802.7	0.476599	0.482204	0.469677	0.412517	0.4527	0.28265	0.379716	0.767019	0.932918	16.64806	99277.31	11.62577	6.342241	0.338666
min	5008804	0	0	0	0	0	0	0	0	1	0	27000	20.50419	0	0
25%	5036955	0	0	0	0	0	0	0	0	2	13	112500	34.05956	0.92815	0
50%	5069449	0	0	1	0	0	0	0	0	2	26	157500	42.74147	3.761884	0
75%	5112986	1	1	1	0	1	0	0	1	3	41	225000	53.56715	8.200031	0
max	5150479	1	1	1	1	1	1	1	19	20	60	1575000	68.86384	43.02073	1

Với 1 tập dữ liệu khác có 10000 bản ghi bao gồm 20 thuộc tính, chia làm 7000-3000 cho tập train và test, việc xây dựng cây có sự khác biệt đáng kể về tốc độ,

- Đối với thuật toán cài đặt với cách chia tối đa 2 nhánh và không xét lại thuộc tính thì mất khoảng 30-40s để có thể xây dựng được cây
- Đối với thư viện C4point5 mất rất nhiều thời gian để thực hiện, có thể lên tới hàng chục phút. Do việc xét lại các thuộc tính đã được chọn để chia node, dẫn đến việc phải tính toán lại nhiều lần các tập dữ liệu.

Như vậy, việc xét lại thuộc tính đã được chọn có thể làm cho kết quả tốt hơn, nhưng đối với các bộ dữ liệu lớn việc xét lại có thể ảnh hưởng nhiều đến vấn đề thời gian. Vì vậy việc chọn lựa xét lại thuộc tính cần xem xét vào các yếu tố khác để chọn lựa phù hợp với bài toán

Kiểm tra kết quả với việc pruning cây

Lần 1: Kiểm định với bộ dữ liệu gồm hơn 1000 bản ghi, chia làm 3 tập dữ liệu gồm train, validation, test với số bản ghi tương ứng là 750, 150, 100

Trước khi cắt tỉa	Sau khi cắt tỉa
thal <= 2.5, gr = 0.39 : slope <= 1.5, gr = 0.199 : restecg <= 0.5 : 1 restecg > 0.5 : 0 slope > 1.5, gr = 0.199 : restecg <= 0.5 : 0 restecg > 0.5 : 1 thal > 2.5 : 0 sex <= 0.5, gr = 0.08 : ca <= 1.5, gr = 0.133 : slope <= 1.5, gr = 0.032 : restecg <= 0.5, gr = 0.036 : exang <= 0.5, gr = 0.031 : thal <= 2.5 : 1 thal > 2.5 : 1 exang > 0.5 : 1 restecg > 0.5 : 1 slope > 1.5 : 1 sex > 0.5, gr = 0.08 : ca <= 3.5, gr = 0.087 : slope <= 1.5, gr = 0.049 : thal <= 2.5, gr = 0.087 : restecg <= 0.5, gr = 0.02 : exang <= 0.5, gr = 0.01 :	thal <= 2.5, gr = 0.39 : slope <= 1.5 : 0 slope > 1.5, gr = 0.199 : restecg <= 0.5 : 0 restecg > 0.5 : 1 thal > 2.5 : 0 sex <= 0.5, gr = 0.08 : ca <= 1.5 : 1 sex > 0.5, gr = 0.08 : ca <= 3.5, gr = 0.087 : slope <= 1.5, gr = 0.049 : thal <= 2.5, gr = 0.087 : restecg <= 0.5, gr = 0.02 : exang <= 0.5 : 1

fbs <= 0.5 : 1 fbs > 0.5 : 1 exang > 0.5, gr = 0.01 : fbs <= 0.5 : 1 fbs > 0.5 : 0 restecg > 0.5, gr = 0.02 : fbs <= 0.5, gr = 0.092 : exang <= 0.5 : 1 exang > 0.5 : 1 fbs > 0.5, gr = 0.092 : exang <= 0.5 : 0 exang > 0.5 : 1 thal > 2.5, gr = 0.087 : fbs <= 0.5, gr = 0.296 : restecg <= 0.5, gr = 0.064 : exang <= 0.5 : 0 exang > 0.5 : 1 restecg > 0.5, gr = 0.064 : exang <= 0.5 : 0 exang > 0.5 : 0 fbs > 0.5 : 1 slope > 1.5, gr = 0.049 : fbs <= 0.5, gr = 0.112 : exang <= 0.5, gr = 0.104 : thal <= 2.5, gr = 0.01 : restecg <= 0.5 : 1 restecg > 0.5 : 1 thal > 2.5, gr = 0.01 : restecg <= 0.5 : 1 restecg > 0.5 : 1 exang > 0.5 : 1 fbs > 0.5 : 1	exang > 0.5, gr = 0.01 : fbs <= 0.5 : 1 fbs > 0.5 : 0 restecg > 0.5, gr = 0.02 : fbs <= 0.5 : 1 fbs > 0.5 : 0 thal > 2.5 : 0 slope > 1.5 : 1
Accuracy: 88.35%	Accuracy: 84.46%

Có thể thấy việc pruning giúp cây trở nên ngắn gọn, ít phức tạp và dễ hiểu hơn so với cây ban đầu. Việc pruning cũng giúp cây không quá overfitting với dữ liệu do dùng một tập validation để có thể giảm bớt những nhánh cây quá sát với dữ liệu, từ đó không có được cái nhìn tổng quát về tổng thể dữ liệu. Cây sau khi pruning ngắn hơn nên việc dự đoán kết quả có thể sẽ được thực hiện nhanh hơn.

Tuy nhiên khi nhìn vào kết quả Accuracy của 2 cây thì việc cây sau khi pruning có thể bị underfit so với dữ liệu. Nguyên nhân có thể là do việc bộ dữ liệu không có đủ nhiều dữ liệu để chia tập validation và train, dẫn đến không có đủ thông tin về toàn bộ dữ liệu. Từ đó tập validation khi sử dụng để pruning có thể làm mất mát thông tin.

Khi áp dụng với bộ dữ liệu có 10000 bản ghi bao gồm 15 thuộc tính, được chia làm 3 bộ dữ liệu train, validation, test với số bản ghi tương ứng là 7500, 1500, 1000 thì được kết quả như sau:

- Accuracy_after_pruning = 84,45%
- Accuracy_after_pruning = 86%

Như vậy việc áp dụng pruning trên bộ dữ liệu lớn có thể đem lại kết quả tốt hơn do giảm overfitting đối với dữ liệu, cải thiện khả năng tổng quát hóa. Từ đó có thể dự đoán kết quả chính xác hơn.