**Cài biên thuật toán C4.5 và bài toán phân loại dân cư.**

1. **Giới thiệu.**

Thuật toán C4.5 được coi như là một thuật toán tốt nhất bây giờ dùng để xây dựng cây quyết định trong việc giải quyết bài toán phân loại. C4.5 thực ra là sự cải tiến mở rộng của thuật toán ID 3. Đặc biệt trong các bước xử lý missing value, xử lý với các thuộc tính liên tục, tỉa cây, và biểu cây thành tập quy tắc và loại bỏ các điều kiện không làm tằng độ chính xác.

Tuy nhiên trong quá trình nghiên cứu do chưa lắm vững thuật toán của tác giả và việc cài đặt thuật toán tương đối phức táp nên em đã có sự thay đổi trong các bước triển khai thuật toán được khả thi.

1. **Các bước của thuật toán.**
2. **Lựa chọn thuộc tính để chia nhánh.**

Việc lựa chọn thuộc tính để phân loại thuật toán C4.5 cũng dựa trên phương pháp giống như thuật toán ID 3. Nghĩa là đều sử dụng cơ chế đánh giá dựa trên lượng tin để từ đó chọn thuột tính nào phân loại. Đại Gain của một thuộc tính X được tính theo công thức sau:

GainX= InfoT- InfoX (\*)

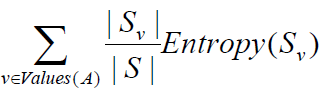
Trong đó InfoT là lượng tin trung bình trong mỗi trường hợp cụ thể. InfoX là lượng tin trung bình của một trường hợp khi chúng ta biết một thuộc tính nào đó. Tại sao công thức này lại có ý nghĩa đối với việc phân loại? Cơ sở lý thuyết nào đứng đằng sau đó?

Để hiểu được điều này chúng ta phải dựa trên cơ sở lý thuyết truyền tin. Theo đó lượng tin của một tin tỉ lệ nghịch với xác suất của tin đó.

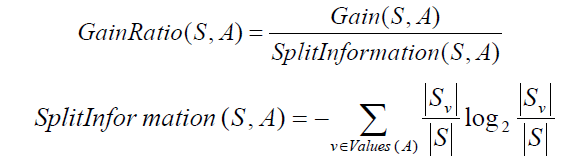
Trong bài toán phân loại chúng ta cần phân loại một trường hợp nào đó thuộc nhóm nào(>50K$ hay nhỉ <=50K$). Khi đó rõ ràng nếu một trường hợp nào đó khi chúng ta chưa biết một thông tin nào đó về trường hợp đấy (giá trị các thuộc tính) mà chúng ta có thể phân loại được nó thì hiển nhiên lượng tin nó là cao nhất. Do đó trong công thức (\*) InfoT luôn có giá trị cao nhất, GainX luôn luôn dương.

Thuộc tính X được lựa chọn là thuộc tính có GainX điều đó đồng nghĩa giá trị InfoX là nhỏ nhất. Tại sao thuộc tính có lượng tin nhỏ nhất lại là thuộc tính được lựa chọn? Nếu dựa trên lý thuyết truyền tin thì điều đó thể hiện là một trường hợp bất kỳ nếu ta biết thuộc tính X thì khả năng phân loại nào vào một trong các nhóm là dễ dàng nhất. Và điều đó dẫn đến lượng tin nó mang là nhỏ nhất.

Công thức để tính InfoX:



Từ công thức trên ta thấy, nếu tập subset Sv ứng với thuộc tính A có giá trị là v mà rất nhỏ thì lượng tin Entropy (Sv) sẽ trở nền gần bằng không. Đo đó thuộc tính A có lượng rất lớn các giá trị hay một subset có số lượng phần tử rất nhỏ thì sẽ dẫn đến InfoX gần bằng không, và rất bé. Điều này trong một số trường hợp việc chọn phần thuộc tính A để phân loại trong trường hợp đó không hợp lý. Do đó có một đại lượng khác được chọn để việc chọn thuộc tính phân loại sẽ đúng hơn là GainRatio được tính theo công thức:



Theo nhiều thống kê của nhiều tác giả thì việc sử dụng GainRation sẽ cho cây chính xác hơn và có số node ít hơn so với sử dụng Gain.

1. **Xử lý các thuộc tính liên tục.**

Các thuật toán cây quyết định chỉ có thể áp dụng với các thuộc tính rời rạc do đó các thuộc tính liên tục phải được rời rạc bằng cách chia khoảng. Vấn đề ở đây việc lựa chọn việc chia khoảng thế nào?

Nếu có N giá trị liên tục sắp xếp rời rạc thì sẽ có n-1 một khoảng tương ứng với n-1 midle point. Việc chọn MidlePoint nào để phân chia các thuộc tính liên tục thì có thể dựa trên Gain và Gain Ratio. Với bộ dữ liệu Adult thì việc sử dụng GainRatio lại không mang lại hiệu quả, thậm chí kết quả là tồi tệ! Vì thuộc tính Fnlwgt có khoảng biến thiên rất rộng, ững với mỗi khoảng chỉ có một vài giá trị. Do đó khi tính Split Info với mỗi midle point thì midle point có SplitInfo nhỏ nhất lại là midle point chia bộ dữ liệu thành hai phần : một subset có một phần tử và một subset có n-1 phần tử. Như vậy thì cây quyết định dựng lên sẽ có dạng bị lệch và tồi tệ hơn là do bộ dữ liệu lớn nên nó sẽ làm cho số lần gọi đệ quy quá nhiều và dẫn đến tràn bộ đệm.

Thay vào đó việc chọn threshold dựa trên Gain lại tỏ ra hiệu quả hơn. Khi threshold được chọn dựa vào Gain sẽ chia bộ dữ liệu ra thành hai phần tương đối cân bằng. Như thế sẽ tránh được việc bị tràn bộ đệm.

1. **Xử lý các trường hợp missing value.**

Vấn đề thuộc tính bị missing trong thuật toán C4.5 gốc được xử lý bằng việc xử lý xác suất và đánh trọng số. Nhưng trong phần mềm lần này nó được xử lý bằng cách đánh nhãn phổ biến nhất với các trường hợp thuộc cùng một lớp.

1. **Tỉa cây để xử lý overfitting data.**

Việc xây dựng cây lên từ bộ dữ liệu training sẽ luôn gây ra hiện tượng overfitting data. Nghĩa là sẽ có độ chính xác thấp hơn hiều so với khi test trên chính bộ dữ liệu training. Điều này được giải quyết trong C4.5 bằng chiến lược tỉa cây sau khi cây được xây dựng đầy đủ.

Đối với thuật toán C4.5 gốc tác giả sử sử dụng toàn bộ dữ liệu training để xây dựng cây. Rồi từ đó sử dụng một phương thức heuristic để đánh giá lỗi có thể mắc phải khi tỉa đi một node. Điều này nhằm làm tận dụng dữ liệu training, tránh chia nhỏ.

Nhưng trong thuật toán cài đặt của phần mềm này thì không sử dụng cách đó. Mà tách một phần dữ liệu training thành dữ liệu validate để thực hiện tỉa cây nhằm tăng độ chính xác.

1. **Biểu diễn cây thành luật và loại bỏ điều kiện để tăng độ chính xác.**

Cây quyết định được xây dựng lên thường có độ sâu khá lớn và số lượng node có thể lên tới hàng nghìn. Điều này làm cho nó không có ý nghĩa đối với người dùng vì quá khó để đọc hiểu. Do đó một cách để giải quyết là biểu diễn cây thành tập các luật. Không dừng ở đó mà còn tiến hành loại bỏ bớt các điều kiện. Và vẫn dùng phương thức ước lượng lỗi bằng phân phối xác suất để đánh giá độ chính xác.

Tuy nhiên trong chương trình thì phương thức này không được cài đặt.

1. **Các chú ý khi cài đặt.**

Việc cài đặt thuật toán có hai phần cần chú ý đó là: độ sâu của cây quyết định và hiệu năng của chương trình.

Đối vấn đề độ sâu của cây quyết định: về lý thuyết nếu tất cả các thuộc tính đều là thuộc tính rời rạc, thì nếu có n thuộc tính thì cây có độ sâu không quá n (ví dụ trong bài toán n=14). Nhưng do trong n thuộc tính luôn có các thuộc tính liên tục. Các thuộc tính liên tục với việc sử dụng chiến lượng tính gain để chọn khoảng, thì số độ sâu của cây sẽ lớn hơn rất nhiều. Nó phụ thuộc vào việc có thể chia khoảng tới khi nào chỉ còn thuộc một nhóm, hoặc không thể chia khoảng được nữa. Như vậy thì độ sâu của cây có thể rất lớn. Và dẫn đến lời gọi đệ quy sẽ quá nhiều và có thể gây tràn bộ đệm.

Vấn đề hiệu năng ảnh hướng tới rất nhiều thời gian training. Do đây là môt thuật toán có độ phức tạp cao. Chính vì thế khi tính toán cần phải tối ưu tối đa có thể. Một số đại lượng có thể tính toán trước. Một trong các thao tác tính toán tốn kém nhất là thao tác tính toán các threshold sau mỗi lần dữ liệu bị chia nhỏ. Qua kinh nghiệm quan sát thì nếu một thuộc tính liên tục như CapitalGain hay Fnlwgt được chọn thì sẽ có tới vài mức node con dưới cũng sẽ được chọn thuộc tính đó. Như vậy thì có thể sử dụng lại các khoảng threshold đã được tính từ node cha khi tìm threshold của thuộc tính đó. Một cái khác có thể làm tăng hiệu năng đáng kể là: sau mỗi lần cây chia nhánh thì đồng nghĩa với bộ dữ liệu được lọc thành các subset tương ứng với mỗi nhánh. Mỗi lần đệ quy ta có thể lưu lại một tập các điều kiện rồi lọc từ dữ liệu gốc. Nhưng như thế thì sẽ rất lâu vì phải duyệt trên cả tập dữ liệu gốc. Thay vào đó ta có thể truyền tham chiếu một tấp subset tới lần đệ quy tiếp theo. Rồi lọc từ tập nhỏ đó sẽ nhanh hơn rất nhiều. Điều này không làm tăng dung lượng bộ nhớ hay làm tràn bộ đệm. Vì thực ra nó vẫn dùng tham chiếu tới cùng một vùng nhớ.

1. **Kết quả.**

So với thuật toán gốc, biến thể thuật toán C45 có độ chính xác thấp hơn. Độ chính xác cao nhất là 82.00%. Trong khi thuật toán gốc có độ chính xác từ 83.5 tới 85.5% với cùng bộ dữ liệu Adult. Bảng dữ liệu dưới đây là độ chính xác của của trong một số lần test vởi tỉ lệ giữa dữ liệu training và dữ liệu validate.

|  |  |  |  |
| --- | --- | --- | --- |
| **Training Data** | **Validate** | **Trước khia tỉa** | **Sau khi tỉa** |
| 65% | 35% | 79.0% | 82.00% |
| 70% | 30% | 79.0% | 81.7% |
| 75% | 25% | 78.5% | 81.13% |
| 80% | 20% | 79% | 81.46% |

1. **Tài liệu tham khảo.**

*Tom Mitchell, Machine Learning, McGraw-Hill-1997.*

*J.Ross Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann-1993.*