|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.5.1

“Áp dụng mô hình học máy Random Forest, Decision Tree”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2023

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.5.1

“Áp dụng mô hình học máy Random Forest, Decision Tree”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2023

MỤC LỤC

[MỤC LỤC 1](#_Toc129729449)

[DANH MỤC HÌNH ẢNH 2](#_Toc129729450)

[1. Random Forests (RF) 3](#_Toc129729451)

[1.1. Tìm hiểu về Random Forest 3](#_Toc129729452)

[1.2. Cách thức hoạt động của thuật toán RF 4](#_Toc129729453)

[1.3. Ưu nhược điểm của thuật toán 6](#_Toc129729454)

[1.4. Các tính năng quan trọng 7](#_Toc129729455)

[1.5. Ứng dụng 7](#_Toc129729456)

[1.6. Thực nghiệm 8](#_Toc129729457)

[2. Decision Tree 11](#_Toc129729458)

[2.1. Tổng quan về Decision Tree 11](#_Toc129729459)

[2.2 Thuật toán ID3 11](#_Toc129729460)

[2.1.1 Ý tưởng 11](#_Toc129729461)

[2.1.2 Hàm số entropy 12](#_Toc129729462)

[2.1.3 Thuật toán ID3 14](#_Toc129729463)

[2.2. Điều kiện dừng 15](#_Toc129729464)

[2.3. Ưu nhược điểm của thuật toán 16](#_Toc129729465)

[2.4. Thực nghiệm 17](#_Toc129729466)

# DANH MỤC HÌNH ẢNH

[Hình 1.1. Mô hình hoá Random Forest 4](#_Toc129729435)

[Hình 1.2. Dự đoán kết quả trong RF 4](#_Toc129729436)

[Hình 1.3: Cách thứchoạt động của mô hình Random Forest 5](#_Toc129729437)

[Hình 1.4: Cài đặt thư viện scikit-learn 8](#_Toc129729438)

[Hình 1.5: Kết quả độ chính xác của mô hình 10](#_Toc129729439)

[Hình 2.1: Minh họa hàm số entropy 13](#_Toc129729440)

[Hình 2.2: Biểu đồ mô tả entropy tại ***p*** 13](#_Toc129729441)

[Hình 2.3: Công thức tính Entropy của node 14](#_Toc129729442)

[Hình 2.4: Công thức tính tổng trọng số của entropy 14](#_Toc129729443)

[Hình 2.5: Công thức tính Information Gain dựa trên ***x*** 15](#_Toc129729444)

[Hình 2.6: Công thức xác định thuộc tính tại mỗi node 15](#_Toc129729445)

[Hình 2.7: Mô hình Decision Tree bị overfitting với dữ liệu 15](#_Toc129729446)

[Hình 2.8: Cài đặt thư viện scikit-learn 17](#_Toc129729447)

[Hình 2.9: Kết quả độ chính xác của mô hình 18](#_Toc129729448)

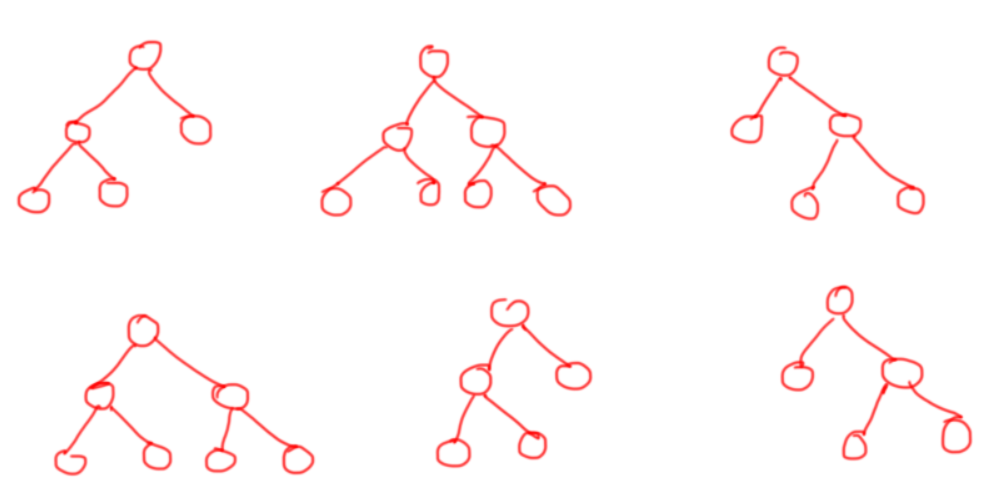
# 1. Random Forests (RF)

## 1.1. Tìm hiểu về Random Forest

Random Forests là thuật toán học có giám sát (supervised learning). Nó có thể được sử dụng cho cả bài toán phân lớp và hồi quy. Nó cũng là thuật toán linh hoạt và dễ sử dụng nhất. Một khu rừng bao gồm cây cối. Người ta nói rằng càng có nhiều cây thì rừng càng mạnh. Random forests tạo ra [cây quyết định](https://machinelearningcoban.com/2018/01/14/id3/) trên các mẫu dữ liệu được chọn ngẫu nhiên, được dự đoán từ mỗi cây và chọn giải pháp tốt nhất bằng cách bỏ phiếu. Nó cũng cung cấp một chỉ báo khá tốt về tầm quan trọng của tính năng. Random forests có nhiều ứng dụng, chẳng hạn như công cụ đề xuất, phân loại hình ảnh và lựa chọn tính năng. Nó có thể được sử dụng để phân loại các ứng viên cho vay trung thành, xác định hoạt động gian lận và dự đoán các bệnh. Nó nằm ở cơ sở của thuật toán Boruta, chọn các tính năng quan trọng trong tập dữ liệu.

Về mặt kỹ thuật, nó là một phương pháp tổng hợp (dựa trên cách tiếp cận phân chia và chinh phục) của các cây quyết định được tạo ra trên một tập dữ liệu được chia ngẫu nhiên. Bộ sưu tập phân loại cây quyết định này còn được gọi là rừng. Cây quyết định riêng lẻ được tạo ra bằng cách sử dụng chỉ báo chọn thuộc tính như tăng thông tin, tỷ lệ tăng và chỉ số Gini cho từng thuộc tính. Mỗi cây phụ thuộc vào một mẫu ngẫu nhiên độc lập. Trong bài toán phân loại, mỗi phiếu bầu chọn và lớp phổ biến nhất được chọn là kết quả cuối cùng. Trong trường hợp hồi quy, mức trung bình của tất cả các kết quả đầu ra của cây được coi là kết quả cuối cùng. Nó đơn giản và mạnh mẽ hơn so với các thuật toán phân loại phi tuyến tính khác.

Ở bước huấn luyện thì ta sẽ xây dựng nhiều cây quyết định, các cây quyết định có thể khác nhau.



Hình 1.1. Mô hình hoá Random Forest

A picture containing diagram

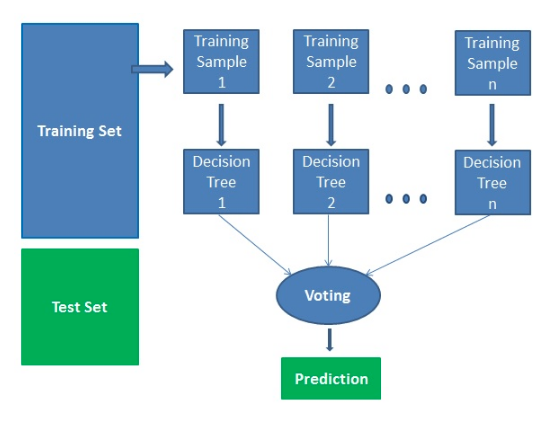
Description automatically generatedSau đó ở bước dự đoán, với một dữ liệu mới, thì ở mỗi cây quyết định ta sẽ đi từ trên xuống theo các node điều kiện để được các dự đoán, sau đó kết quả cuối cùng được tổng hợp từ kết quả của các cây quyết định.

Hình 1.2. Dự đoán kết quả trong RF

Ví dụ như trên, thuật toán Random Forest có 6 cây quyết định, 5 cây dự đoán 1 và 1 cây dự đoán 0, do đó cho ra dự đoán cuối cùng là 1.

## 1.2. Cách thức hoạt động của thuật toán RF

Random Forest hoạt động theo bốn bước:



Hình 1.3: Cách thứchoạt động của mô hình Random Forest

1. Chọn các mẫu ngẫu nhiên từ tập dữ liệu đã cho.
2. Thiết lập cây quyết định cho từng mẫu và nhận kết quả dự đoán từ mỗi quyết định cây.
3. Hãy bỏ phiếu cho mỗi kết quả dự đoán.
4. Chọn kết quả được dự đoán nhiều nhất là dự đoán cuối cùng.

Do quá trình xây dựng mỗi cây quyết định đều có yếu tố ngẫu nhiên (random) nên kết quả là các cây quyết định trong thuật toán Random Forest có thể khác nhau.

Thuật toán Random Forest sẽ bao gồm nhiều cây quyết định, mỗi cây được xây dựng dùng thuật toán Decision Tree trên tập dữ liệu khác nhau và dùng tập thuộc tính khác nhau. Sau đó kết quả dự đoán của thuật toán Random Forest sẽ được tổng hợp từ các cây quyết định.

Khi dùng thuật toán Random Forest, ta hay để ý các thuộc tính như: số lượng cây quyết định sẽ xây dựng, số lượng thuộc tính dùng để xây dựng cây. Ngoài ra, vẫn có các thuộc tính của thuật toán Decision Tree để xây dựng cây như độ sâu tối đa, số phần tử tối thiểu trong 1 node để có thể tách.

## 1.3. Ưu nhược điểm của thuật toán

* **Ưu điểm:**
* Trong thuật toán Decision Tree, khi xây dựng cây quyết định nếu để độ sâu tùy ý thì cây sẽ phân loại đúng hết các dữ liệu trong tập training dẫn đến mô hình có thể dự đoán tệ trên tập validation/test, khi đó mô hình bị overfitting, hay nói cách khác là mô hình có [high variance](https://viblo.asia/p/the-bias-variance-decomposition-eW65Gm3YZDO).
* Thuật toán Random Forest gồm nhiều cây quyết định, mỗi cây quyết định đều có những yếu tố ngẫu nhiên:
  + Lấy ngẫu nhiên dữ liệu để xây dựng cây quyết định.
  + Lấy ngẫu nhiên các thuộc tính để xây dựng cây quyết định.
* Do mỗi cây quyết định trong thuật toán Random Forest không dùng tất cả dữ liệu training, cũng như không dùng tất cả các thuộc tính của dữ liệu để xây dựng cây nên mỗi cây có thể sẽ dự đoán không tốt, khi đó mỗi mô hình cây quyết định không bị overfitting mà có thế bị underfitting, hay nói cách khác là mô hình có high bias. Tuy nhiên, kết quả cuối cùng của thuật toán Random Forest lại tổng hợp từ nhiều cây quyết định, thế nên thông tin từ các cây sẽ bổ sung thông tin cho nhau, dẫn đến mô hình có low bias và low variance, hay mô hình có kết quả dự đoán tốt.
* **Nhược điểm:**
* Random forest chậm tạo dự đoán bởi vì nó có nhiều cây quyết định. Bất cứ khi nào nó đưa ra dự đoán, tất cả các cây trong rừng phải đưa ra dự đoán cho cùng một đầu vào cho trước và sau đó thực hiện bỏ phiếu trên đó. Toàn bộ quá trình này tốn thời gian.
* Mô hình khó hiểu hơn so với cây quyết định, nơi bạn có thể dễ dàng đưa ra quyết định bằng cách đi theo đường dẫn trong cây.

## 1.4. Các tính năng quan trọng

Random forests cũng cung cấp một chỉ số lựa chọn tính năng tốt. Scikit-learn cung cấp thêm một biến với mô hình, cho thấy tầm quan trọng hoặc đóng góp tương đối của từng tính năng trong dự đoán. Nó tự động tính toán điểm liên quan của từng tính năng trong giai đoạn đào tạo. Sau đó, nó cân đối mức độ liên quan xuống sao cho tổng của tất cả các điểm là 1.

Điểm số này sẽ giúp bạn chọn các tính năng quan trọng nhất và thả các tính năng quan trọng nhất để xây dựng mô hình.

Random forests sử dụng tầm quan trọng của gini hoặc giảm tạp chất trung bình (MDI) để tính toán tầm quan trọng của từng tính năng. Gini tầm quan trọng còn được gọi là tổng giảm trong tạp chất nút. Đây là mức độ phù hợp hoặc độ chính xác của mô hình giảm khi bạn thả biến. Độ lớn càng lớn thì biến số càng có ý nghĩa. Ở đây, giảm trung bình là một tham số quan trọng cho việc lựa chọn biến. Chỉ số Gini có thể mô tả sức mạnh giải thích tổng thể của các biến. Random Forests và cây quyết định Random Forests là một tập hợp của nhiều cây quyết định. Cây quyết định sâu có thể bị ảnh hưởng quá mức, nhưng Random forests ngăn cản việc lấp đầy bằng cách tạo cây trên các tập con ngẫu nhiên. Cây quyết định nhanh hơn tính toán. Random forests khó giải thích, trong khi cây quyết định có thể diễn giải dễ dàng và có thể chuyển đổi thành quy tắc.

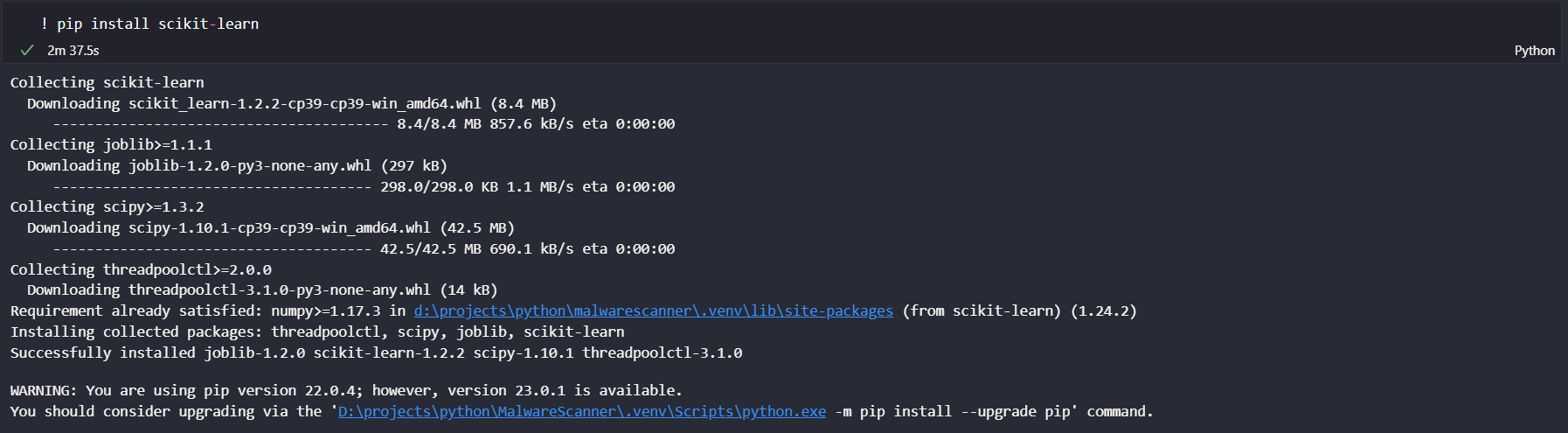
## 1.5. Ứng dụng

Thuật toán rừng ngẫu nhiên được sử dụng trong rất nhiều lĩnh vực khác nhau, như ngân hàng, thị trường chứng khoán, y học và thương mại điện tử.

* Ví dụ, trong tài chính, nó được sử dụng để phát hiện khách hàng có nhiều khả năng trả nợ đúng hạn hoặc sử dụng dịch vụ của ngân hàng thường xuyên hơn. Trong tên miền này, nó cũng được sử dụng để phát hiện những kẻ lừa đảo ra ngoài để lừa đảo ngân hàng. Trong giao dịch, thuật toán có thể được sử dụng để xác định hành vi trong tương lai của một cổ phiếu.
* Trong chăm sóc sức khỏe, nó được sử dụng để xác định sự kết hợp chính xác của các thành phần trong y học và phân tích lịch sử y tế của bệnh nhân để xác định bệnh.
* Rừng ngẫu nhiên được sử dụng trong thương mại điện tử để xác định xem khách hàng có thực sự thích sản phẩm hay không.

1.6. Thực nghiệm

Dựa trên các đặc điểm trên, chúng ta sẽ áp dụng mô hình học máy RF để chạy thực nghiệm việc phân loại thông qua *sklearn.neighbors.RandomForestClassifier* của thư viện *scikit-learn*. Để có thể sử dụng được thư viện, ta tiến hành cài đặt thư viện *scikit-learn* thông qua câu lệnh *pip install scikit-learn.*



Hình 1.4: Cài đặt thư viện scikit-learn

Sau khi cài đặt xong, ta tiến hành nạp dữ liệu sử dụng để huấn luyện và kiểm tra từ file có định dạng .csv vào chương trình thông qua thư viện *pandas*.

import pandas as pd

import numpy as np

train\_df=pd.read\_csv(r'dataset/train-0.csv',header=None,skiprows=1)

val\_df=pd.read\_csv(r'dataset/file-0.csv',header=None,skiprows=1)

test\_df=pd.read\_csv(r'dataset/file-1.csv',header=None,skiprows=1)

train\_x = np.array(train\_df.iloc[:, 3:])

train\_y = np.array(train\_df.iloc[:, 2])

val\_x = np.array(val\_df.iloc[:, 3:])

val\_y = np.array(val\_df.iloc[:, 2])

test\_x = np.array(test\_df.iloc[:, 3:])

test\_y = np.array(test\_df.iloc[:, 2])

Ở đây, chúng ta sử dụng bộ dữ liệu được tách ra thành 3 file với 3 mục đích khác nhau, cụ thể là sử dụng để huấn luyện, xác thực và kiểm tra. Ta tiến hành tách các đặc trưng và nhãn từ dữ liệu đã được nạp vào rồi sau đó biến đổi chúng thành đối tượng có dạng dữ liệu kiểu mảng bằng thư viện *numpy.* Tiếp đó, ta nạp thư viện scikit-learn đã cài đặt vào trong chương trình, thực hiện huấn luyện cho mô hình với dữ liệu vừa nạp vào và tiến hành cho mô hình phân loại mã độc.

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

model\_rfc10 = RandomForestClassifier(max\_depth=10)

history\_rfc10 = model\_rfc10.fit(train\_x, train\_y)

y\_pred\_vc = model\_rfc10.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred\_vc)

print("RFC10:", accuracy)

model\_rfc50 = RandomForestClassifier(max\_depth=50)

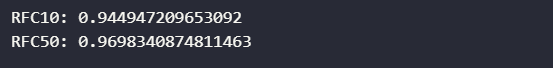
history\_rfc50 = model\_rfc50.fit(train\_x, train\_y)

y\_pred\_vc = model\_rfc50.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred\_vc)

print("RFC50:", accuracy)

Ở đây, ta huấn luyện lần lượt 2 mô hình với tham số chiều sâu của cây khác nhau, lần lượt là 10 và 50 dưới dạng tham số được truyền vào thông qua biến max\_depth. Sau khi tiến hành huấn luyện và cho mô hình phân loại mã độc, ta sử dụng hàm *sklearn.metrics.accuracy\_score* để đánh giá độ chính xác kết quả dự đoán của từng mô hình với dữ liệu kiểm tra có sẵn. Kết quả thu được từ các mô hình lần lượt có độ chính xác lần lượt đạt gần 0.94494 (~95%) đối với cây có chiều sâu là 10 và đạt gần 0.96983 (~97%) đối với cây có chiều sâu là 50.



Hình 1.5: Kết quả độ chính xác của mô hình

2. Decision Tree

2.1. Tổng quan về Decision Tree

Decision tree là một mô hình supervised learning, có thể được áp dụng vào cả hai bài toán classification và regression. Mỗi một nút trong (internal node) tương ứng với một biến; đường nối giữa nó với nút con của nó thể hiện một giá trị cụ thể cho biến đó. Mỗi nút lá đại diện cho giá trị dự đoán của biến mục tiêu, cho trước các giá trị của các biến được biểu diễn bởi đường đi từ nút gốc tới nút lá đó. Kỹ thuật học máy dùng trong cây quyết định được gọi là học bằng cây quyết định, hay chỉ gọi với cái tên ngắn gọn là cây quyết định.

Việc xây dựng một decision tree trên dữ liệu huấn luyện cho trước là việc đi xác định các câu hỏi và thứ tự của chúng. Một điểm đáng lưu ý của decision tree là nó có thể làm việc với các đặc trưng (trong các tài liệu về decision tree, các đặc trưng thường được gọi là thuộc tính – attribute) dạng categorical, thường là rời rạc và không có thứ tự. Ví dụ, mưa, nắng hay xanh, đỏ, v.v. Decision tree cũng làm việc với dữ liệu có vector đặc trưng bao gồm cả thuộc tính dạng categorical và liên tục (numeric). Một điểm đáng lưu ý nữa là decision tree ít yêu cầu việc chuẩn hoá dữ liệu.

2.2 Thuật toán ID3

ID3 là một thuật toán decision tree được áp dụng cho các bài toán classification mà tất cả các thuộc tính đều ở dạng *categorical*.

2.1.1 Ý tưởng

Trong ID3, chúng ta cần xác định thứ tự của thuộc tính cần được xem xét tại mỗi bước. Với các bài toán có nhiều thuộc tính và mỗi thuộc tính có nhiều giá trị khác nhau, việc tìm được nghiệm tối ưu thường là không khả thi. Thay vào đó, một phương pháp đơn giản thường được sử dụng là tại mỗi bước, một thuộc tính *tốt nhất* sẽ được chọn ra dựa trên một tiêu chuẩn nào đó. Với mỗi thuộc tính được chọn, ta chia dữ liệu vào các *child node* tương ứng với các giá trị của thuộc tính đó rồi tiếp tục áp dụng phương pháp này cho mỗi *child node*. Việc chọn ra thuộc tính *tốt nhất* ở mỗi bước như thế này được gọi là cách chọn *greedy (tham lam).* Cách chọn này có thể không phải là tối ưu, nhưng trực giác cho chúng ta thấy rằng cách làm này sẽ gần với cách làm tối ưu. Ngoài ra, cách làm này khiến cho bài toán cần giải quyết trở nên đơn giản hơn.

Sau mỗi câu hỏi, dữ liệu được phân chia vào từng child node tương ứng với các câu trả lời cho câu hỏi đó. Câu hỏi ở đây chính là một thuộc tính, câu trả lời chính là giá trị của thuộc tính đó. Để đánh giá chất lượng của một cách phân chia, chúng ta cần đi tìm một phép đo.

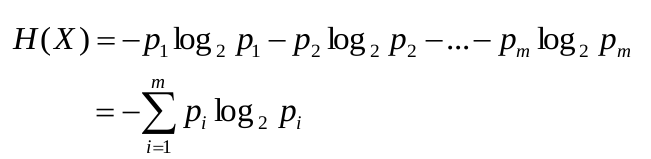
Trước hết, thế nào là một phép phân chia tốt? Bằng trực giác, một phép phân chia là tốt nhất nếu dữ liệu trong mỗi *child node* hoàn toàn thuộc vào một class–khi đó *child node* này có thể được coi là một leaf node, tức ta không cần phân chia thêm nữa. Nếu dữ liệu trong các child node vẫn lẫn vào nhau theo tỉ lệ lớn, ta coi rằng phép phân chia đó chưa thực sự tốt. Từ nhận xét này, ta cần có một hàm số đo *độ tinh khiết (purity)*, hoặc *độ vẩn đục (impurity)* của một phép phân chia. Hàm số này sẽ cho giá trị thấp nhất nếu dữ liệu trong mỗi child node nằm trong cùng một class (tinh khiết nhất), và cho giá trị cao nếu mỗi child node có chứa dữ liệu thuộc nhiều class khác nhau.

Một hàm số có các đặc điểm này và được dùng nhiều trong lý thuyết thông tin là hàm entropy.

2.1.2 Hàm số entropy

Giả sử ta có biến ngẫu nhiên rời rạc X:

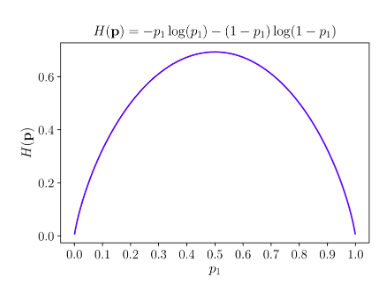
* Có không gian mẫu là {x1, x2, ..., xm} với xác suất P(X = x1) = p1, P(X = x2) = p2, ..., P(X = xm) = pm.
* Số bit trung bình nhỏ nhất để truyền một đơn vị dữ liệu theo phân phối P(X):



Hình 2.1: Minh họa hàm số entropy

Ta có, H(X) là entropy của X (0 <= H(X) <= log2m).

Xem xét m = 2, trong trường hợp X là tinh khiết nhất, tức là một trong hai pi bằng 0 và giá trị kia bằng 1, khi đó H(X) = 0. Khi X là vẩn đục nhất tức cả hai xác suất mang giá trị pi = 0.5, hàm entropy đạt giá trị cao nhất.



Hình 2.2: Biểu đồ mô tả entropy tại ***p***

Vậy với m > 2, hàm entropy đạt giá trị nhỏ nhất nếu có một giá trị pi = 1 (tức các giá trị pi còn lại mang giá trị 0), đạt giá trị lớn nhất nếu tất cả pi bằng nhau.

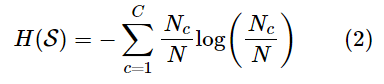
Giá trị entropy:

* Lớn: Phân phối P(X) gần với dạng phân phối đồng nhất (uniform distribution)
* Nhỏ: Phân phối P(X) xa dạng phân phối đồng nhất. Những giá trị này khiến nó được sử dụng trong việc đo độ vẩn đục của một pháp phân chia trong ID3. Nên ID3 còn được gọi là entropy-based descision tree.

2.1.3 Thuật toán ID3

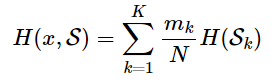
Trong ID3, tổng có trọng số của entropy tại các *leaf-node* sau khi xây dựng decision tree được coi là hàm mất mát của decision tree đó. Các trọng số ở đây tỉ lệ với số điểm dữ liệu được phân vào mỗi node. Công việc của ID3 là tìm các cách phân chia hợp lý (thứ tự chọn thuộc tính hợp lý) sao cho hàm mất mát cuối cùng đạt giá trị càng nhỏ càng tốt. Như đã đề cập, việc này đạt được bằng cách chọn ra thuộc tính sao cho nếu dùng thuộc tính đó để phân chia, entropy tại mỗi bước giảm đi một lượng lớn nhất. Bài toán xây dựng một decision tree bằng ID3 có thể chia thành các bài toán nhỏ, trong mỗi bài toán, ta chỉ cần chọn ra thuộc tính giúp cho việc phân chia đạt kết quả tốt nhất. Mỗi bài toán nhỏ này tương ứng với việc phân chia dữ liệu trong một *non-leaf node*. Chúng ta sẽ xây dựng phương pháp tính toán dựa trên mỗi node này.

Xét một bài toán với ***C*** class khác nhau. Giả sử ta đang làm việc với một *non-leaf node* với các điểm dữ liệu tạo thành một tập ***S*** với số phần tử là |***S***| = *N*. Giả sử thêm rằng trong số *N* điểm dữ liệu này, ***C*** điểm thuộc vào class ***c.*** Xác suất để mỗi điểm dữ liệu rơi vào một class ***c***  được xấp xỉ bằng *N*c/*N*. Như vậy, entropy tại node này được tính bởi:



Hình 2.3: Công thức tính Entropy của node

Tiếp theo, giả sử thuộc tính được chọn là ***x***. Dựa trên ***x***, các điểm dữ liệu trong ***S*** ược phân ra thành *K* child node ***S1, S2, …, SK*** với số điểm trong mỗi child node lần lượt là ***m1, m2, …, mK***. Ta định nghĩa



Hình 2.4: Công thức tính tổng trọng số của entropy

là tổng có trọng số entropy của mỗi child node–được tính tương tự như (2). Việc lấy trọng số này là quan trọng vì các node thường có số lượng điểm khác nhau.

Tiếp theo, ta định nghĩa *information gain* dựa trên thuộc tính ***x***:



Hình 2.5: Công thức tính Information Gain dựa trên ***x***

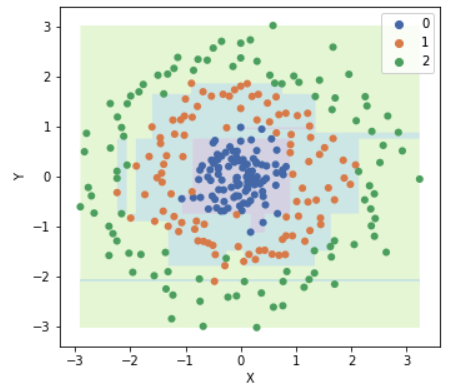
Trong ID3, tại mỗi node, thuộc tính được chọn được xác định dựa trên:



Hình 2.6: Công thức xác định thuộc tính tại mỗi node

2.2. Điều kiện dừng

Trong các thuật toán decision tree nói chung và ID3 nói riêng, nếu ta tiếp tục phân chia các node chưa tinh khiết, 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, tree 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.



Hình 2.7: Mô hình Decision Tree bị overfitting với dữ liệu

Để tránh overfitting, một trong số các phương pháp sau có thể được sử dụng. Tại một node, nếu một trong số các điều kiện sau đây xảy ra, ta không tiếp tục phân chia node đó và coi nó là một leaf node:

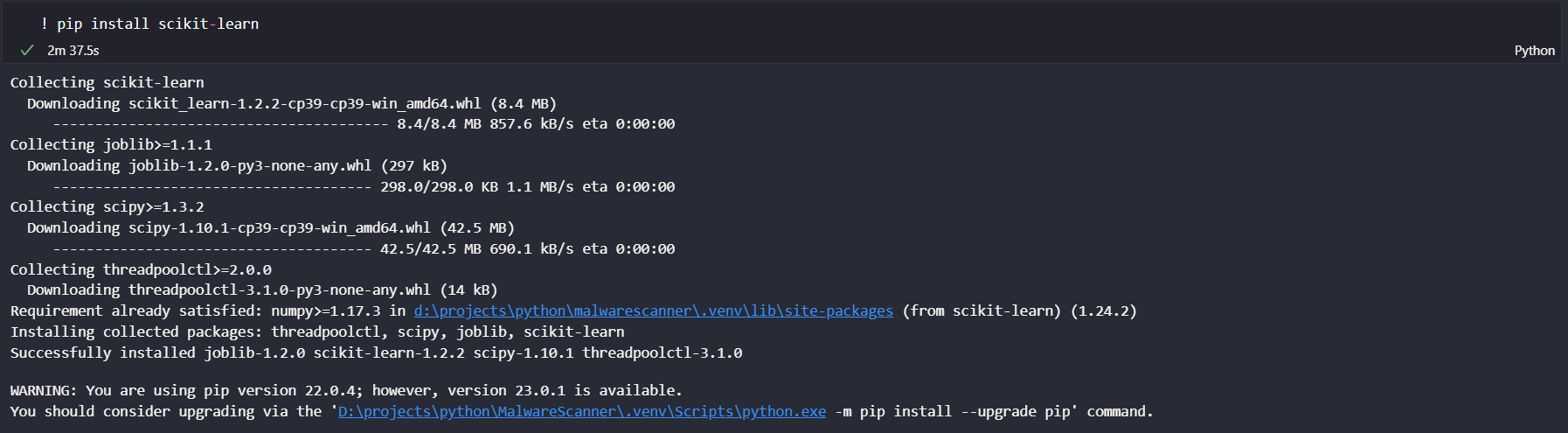
* Nếu node đó có entropy bằng 0, tức mọi điểm trong node đều thuộc một class.
* Nếu node đó có số phần tử nhỏ hơn một ngưỡng nào đó. Trong trường hợp này, ta chấp nhận có một số điểm bị phân lớp sai để tránh overfitting. Class cho leaf node này có thể được xác định dựa trên class chiếm đa số trong node.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

2.3. Ưu nhược điểm của thuật toán

* **Ưu điểm:**
* Mô hình dễ hiểu và dễ giải thích.
* Cần ít dữ liệu để huẩn luyện.
* Xử lý tốt với dữ liệu dạng số (rời rạc và liên tục) và dữ liệu hạng mục.
* Mô hình dạng white box rõ ràng.
* Xây dựng nhanh.
* Phân lớp nhanh.
* **Nhược điểm:**
* Không đảm bảo xây dựng được cây tối ưu.
* Có thể overfitting (tạo ra những cây quá khớp với dữ liệu huấn luyện hay quá phức tạp).
* Thường ưu tiên thuộc tính có nhiều giá trị (khắc phục bằng các sử dụng Gain Ratio).

2.4. Thực nghiệm

Dựa trên các đặc điểm trên, chúng ta sẽ áp dụng mô hình học máy DT để chạy thực nghiệm việc phân loại thông qua *sklearn.tree.DecisionTreeClassifier* của thư viện *scikit-learn*. Để có thể sử dụng được thư viện, ta tiến hành cài đặt thư viện *scikit-learn* thông qua câu lệnh *pip install scikit-learn.*



Hình 2.8: Cài đặt thư viện scikit-learn

Sau khi cài đặt xong, ta tiến hành nạp dữ liệu sử dụng để huấn luyện và kiểm tra từ file có định dạng .csv vào chương trình thông qua thư viện *pandas*.

import pandas as pd

import numpy as np

train\_df=pd.read\_csv(r'dataset/train-0.csv',header=None,skiprows=1)

val\_df=pd.read\_csv(r'dataset/file-0.csv',header=None,skiprows=1)

test\_df=pd.read\_csv(r'dataset/file-1.csv',header=None,skiprows=1)

train\_x = np.array(train\_df.iloc[:, 3:])

train\_y = np.array(train\_df.iloc[:, 2])

val\_x = np.array(val\_df.iloc[:, 3:])

val\_y = np.array(val\_df.iloc[:, 2])

test\_x = np.array(test\_df.iloc[:, 3:])

test\_y = np.array(test\_df.iloc[:, 2])

Ở đây, chúng ta sử dụng bộ dữ liệu được tách ra thành 3 file với 3 mục đích khác nhau, cụ thể là sử dụng để huấn luyện, xác thực và kiểm tra. Ta tiến hành tách các đặc trưng và nhãn từ dữ liệu đã được nạp vào rồi sau đó biến đổi chúng thành đối tượng có dạng dữ liệu kiểu mảng bằng thư viện *numpy.* Tiếp đó, ta nạp thư viện scikit-learn đã cài đặt vào trong chương trình, thực hiện huấn luyện cho mô hình với dữ liệu vừa nạp vào và tiến hành cho mô hình phân loại mã độc.

from sklearn.tree import DecisionTreeClassifier

tree\_clf = DecisionTreeClassifier(max\_depth=50)

history\_tree\_clf = tree\_clf.fit(train\_x, train\_y)

y\_pred\_vc = history\_tree\_clf.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred\_vc)

print("DTC:", accuracy)

Độ sâu của cây được sử dụng cho mô hình là 50 được truyền vào dưới dạng tham số thông qua biến max\_depth. Sau khi tiến hành huấn luyện và cho mô hình phân loại mã độc, ta sử dụng hàm sklearn.metrics.accuracy\_score để đánh giá độ chính xác kết quả dự đoán của mô hình với dữ liệu kiểm tra có sẵn. Kết quả thu được từ mô hình có độ chính xác đạt gần 0.94193 (~94%).



Hình 2.9: Kết quả độ chính xác của mô hình