

Decision_Tree

October 2, 2023

1 Introduction

1.1 Decision Tree Model

- **Decision Tree** là một mô hình thuật toán máy học phổ biến được sử dụng cho cả bài toán **classification** và **regression**. Nó là một cấu trúc **tree** trong đó một **node** biểu thị một **feature**, nhánh biểu thị một **decision-rule** và mỗi **leaf node** biểu thị một kết quả dự đoán của mô hình. Decision Tree chủ yếu được sử dụng để ra quyết định theo hình thức phân cấp và có cấu trúc.

```
[2]: HEIGHT = 121
```











1.1.1 Structure

Xét ví dụ phân loại chó và mèo sau:

```
[3]: display.Image('fig1.png', height = HEIGHT)
```

```
[3]:
```

Cat classification example

	Ear shape	Face shape	Whiskers	Cat
	Pointy	Round	Present	1
	Floppy	Not round	Present	1
	Floppy	Round	Absent	0
	Pointy	Not round	Present	0
	Pointy	Round	Present	1
	Pointy	Round	Absent	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0

Categorical (discrete values) X y

Figure 1: Ví dụ phân loại chó và mèo

- Một decision tree sẽ có cấu trúc là một *binary tree* với *root node*, *decision node* và *leaf node*:
 - **Root node**: node trên cùng của cây, cũng là feature đầu tiên được dùng để phân loại.
 - **Decision node**: các node trong của cây, là các feature được dùng để phân loại tại mỗi node.
 - **Leaf node**: các node dưới cùng của cây, là các kết quả dự đoán của mô hình.
- Đặc điểm:
 - Tại mỗi node (không phải leaf node) sẽ có **đúng 2 nhánh**.
 - Mỗi nhánh đại diện cho một giá trị của feature tại node đó (hoặc một **rule**) dùng để phân loại.

```
[4]: display.Image('fig2.png', height = HEIGHT)
```

[4]:

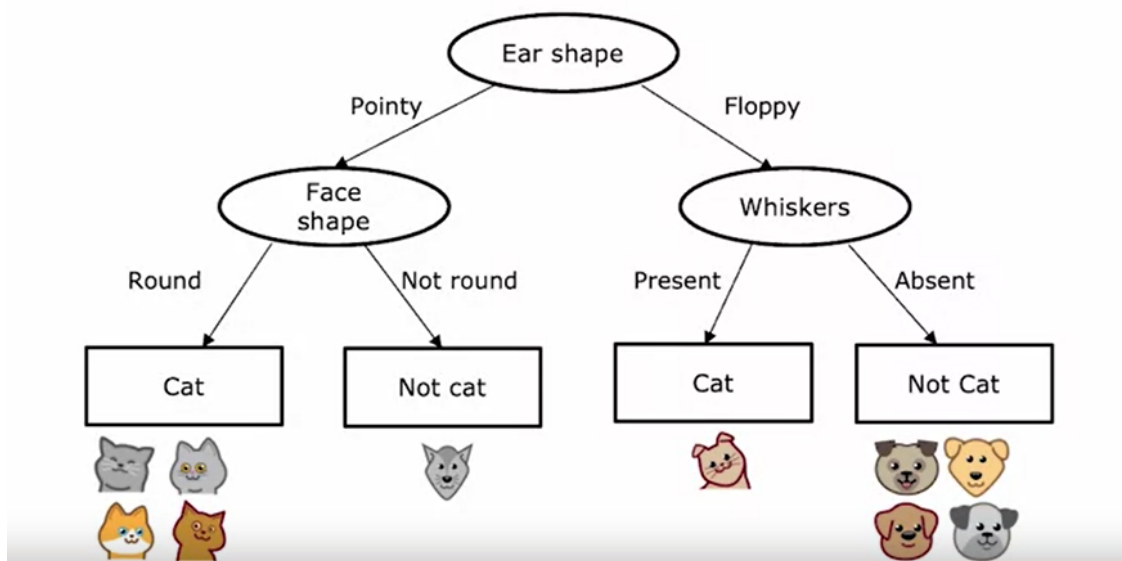


Figure 2: Một decision tree để phân loại chó và mèo

1.1.2 Prediction Method

- Với mỗi điểm dữ liệu cần dự đoán, mô hình decision tree sẽ bắt đầu tại root node, so sánh giá trị của điểm dữ liệu tại feature được quy định tại root node với rule để quyết định rằng điểm dữ liệu đó nên đi về nhanh bên trái hay bên phải?
- Tương tự như vậy cho các decision node, cho đến khi dừng lại tại một leaf node.
- **Kết quả của leaf node** đó sẽ là kết quả dự đoán của mô hình cho điểm dữ liệu.

```
[5]: display.Image('fig3.png', height = HEIGHT)
```

[5]:

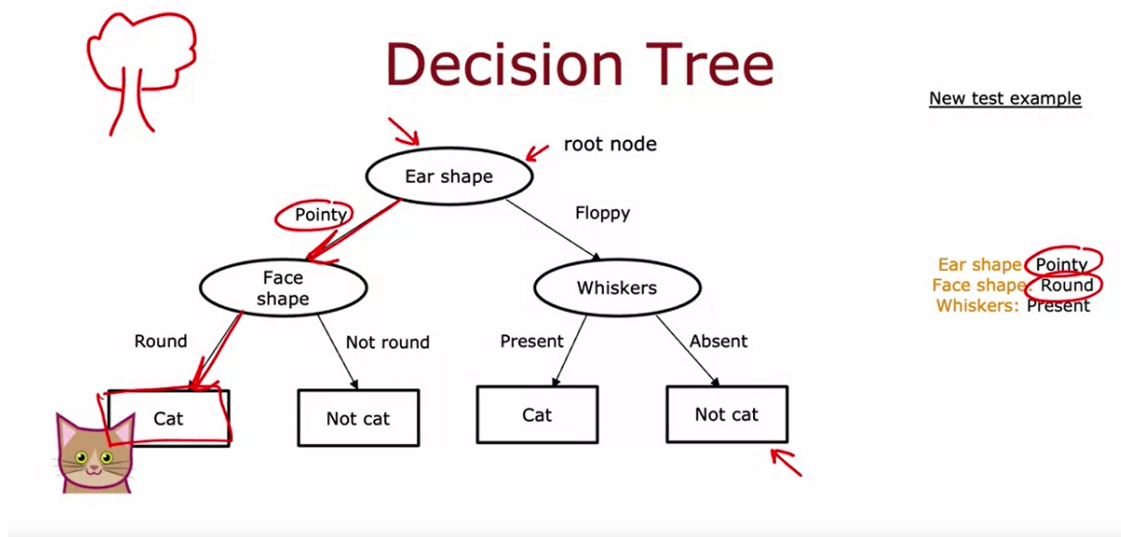


Figure 3: Cách dự đoán một điểm dữ liệu

1.1.3 Purpose

- Sử dụng để giải quyết các bài toán **classification**. Ví dụ: phân loại email có spam hay không spam.
- Decision tree cũng có thể được sử dụng để giải quyết các bài toán **regression**. Ví dụ: dự đoán giá nhà dựa vào các thông tin về ngôi nhà.
- Sử dụng trong các mô hình **Tree Ensembles** chẳng hạn như: **Random Forest**, **XGBoost** như là một thành phần đơn vị của mô hình lớn.
- **Phân tích dữ liệu**: Dựa vào quá trình *learning* của decision tree (sẽ được trình bày bên dưới) có thể giúp phân tích mối quan hệ giữa các biến đầu vào và đầu ra, xem feature nào có ảnh hưởng lớn đến việc dự đoán đầu ra giúp ta rút trích các feature quan trọng và các đặc điểm của tập dữ liệu.
- Các ứng dụng thực tế khác như: **Decision Support Systems**, **Recommendation Systems**, **Troubleshooting and Diagnostics**, **Supply Chain Management**, y tế và khoa học sức khỏe, tài chính và quảng cáo trực tuyến.

2 Decision Tree Learning Process

2.1 Case Study

- **Problem Abstraction**:
 - Xét bài toán phân loại nhị phân - **Binary Classification** với tập dữ liệu - **dataset** được cho: $D = (X - inputs, y - outputs)$.
 - Trong đó đặc điểm của tập **inputs** chỉ bao gồm các feature có giá trị rời rạc là 0 hoặc 1 (**binary values**) và **outputs** chỉ thuộc 2 class với 2 nhãn là 0 hoặc 1.
- **Problem Goal**:
 - Với dataset đã cho, tìm cách xây dựng một decision tree để có thể phân loại tốt nhất.

2.2 Tutorial

- Với cấu trúc của decision tree đã trình bày bên trên, một ý tưởng đơn giản mà ta có thể nghĩ tới khi sử dụng dataset để xây một decision tree phù hợp đó là:
 - Xuất phát từ root node với toàn bộ điểm dữ liệu trong dataset: ta **chọn một feature** để gán cho root node và thiết lập rule tương ứng để **chia các điểm dữ liệu thành 2 nhánh trái và phải theo rule đã quy định**.
 - Điều cần mong đợi ở đây là sự “tạp chất” - *impurity* của tập dữ liệu **phải giảm đi** sau khi chia thành 2 nhánh trái và phải.
 - Lặp lại quá trình này cho các node con của root node và các decision node khác cho đến khi ta quyết định dừng việc phân chia tại một leaf node.
 - Khi đó, **nhãn của leaf node** sẽ là nhãn của đa số các điểm dữ liệu ở leaf node đó (*voting method*).
 - Tham khảo figure 2.

2.3 Problem Posed

- Làm thế nào để **quyết định feature được chọn** để phân chia tập dữ liệu tại mỗi node:
 - Ta hiện thực hóa mong đợi **maximize purity (minimize impurity)** như thế nào?
- Khi nào thì nên **dừng việc phân chia**:
 - Khi các điểm dữ liệu tại một node là 100% có cùng nhãn. [1]
 - Khi depth của tree vượt quá một ngưỡng *max_depth* nào đó. [2]
 - Khi **số lượng điểm dữ liệu tại một node** là nhỏ hơn một ngưỡng nào đó. [3]
 - Khi việc phân chia tại một node **không cải thiện đáng kể purity score**. [4]
- Dừng việc phân chia theo các tiêu chí [2], [3], [4] là cần thiết để tránh hiện tượng *overfitting* xảy ra trong quá trình *learning*.

2.4 Measuring Purity

- Làm thế nào để định nghĩa và đo lường purity của một tập dữ liệu một cách toán học?
- Có nhiều cách để định nghĩa điều này chẳng hạn như sử dụng công thức *gini*, *entropy*,
- Trong bài báo cáo này chúng ta sẽ chỉ tìm hiểu về công thức entropy, các công thức khác ta hoàn toàn có thể có cách tiếp cận tương tự.

2.4.1 Entropy

- Đặt p_1 là phân số biểu thị số lượng điểm dữ liệu có nhãn $y = 1$ so với số lượng điểm dữ liệu.
- p_0 là phân số biểu thị số lượng điểm dữ liệu có nhãn $y = 0$ so với số lượng điểm dữ liệu.
- $p_0 + p_1 = 1$
- *Entropy* :
$$H(p_1, p_0) = H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$
$$= -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$
- Lưu ý:
 - Đây là bài toán Binary Classification nên \log ở đây sẽ là \log_2 .
 - Để công thức không sai về mặt toán học, ta quy ước $0 \log_2(0) = 1$, xảy ra khi $p_1 = 0$ hoặc $p_1 = 1$.

Entropy Function Intuition

```
[6]: display.Image('fig4.png', height = HEIGHT)
```

[6]:

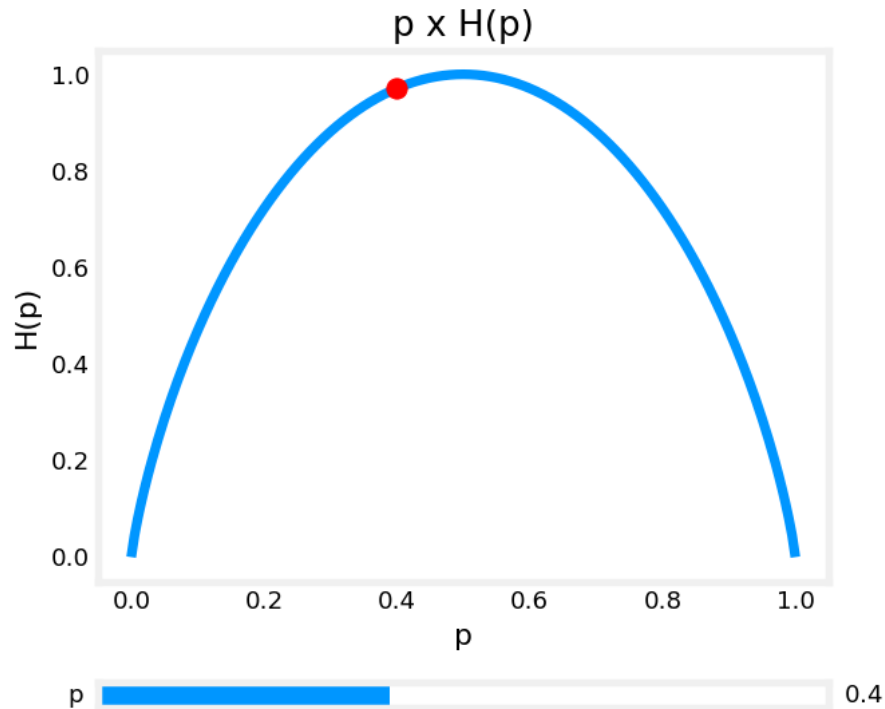


Figure 4: Hàm entropy

- Hàm số trên đạt giá trị **càng lớn** khi p càng tiến gần về 0.5 và bằng 1 khi $p = 0.5$. Hay nói cách khác khi đó tập dữ liệu sẽ là 50 – 50 \rightarrow dữ liệu chứa nhiều “tạp chất”.
- Hàm số trên đạt giá trị **càng nhỏ** khi p càng tiến gần về 0 hoặc 1. Hay nói cách khác khi đó tập dữ liệu sẽ gần như chỉ chứa một loại nhãn - class duy nhất \rightarrow dữ liệu “tinh khiết”.

2.5 Choosing a split: Information Gain

- **Information Gain**: lượng giảm của entropy (*reduce impurity*) khi phân chia tập dữ liệu được gọi là Information Gain.
- Giả sử rằng ta quyết định chọn 1 feature bất kỳ và gán cho root node để tiến hành phân chia tập dữ liệu thành 2 nhánh trái và phải:
 - Ta tiến hành **tính entropy cho cả 2 tập ở 2 nhánh trái và phải**.
 - Rõ ràng rằng: ta muốn **entropy ở cả 2 nhánh trái và phải là càng nhỏ càng tốt**. Vì khi đó ta đã chia dữ liệu một cách “tinh khiết” hơn.
- Có đến 2 giá trị entropy, nếu ta muốn chọn ra feature tốt nhất thì ta nên so sánh 2 giá trị entropy này giữa các feature như thế nào?

- Ta có thể lấy **trung bình của 2 giá trị entropy** này rồi đi so sánh giữa các feature với nhau và quyết định chọn ra feature tốt nhất.
- Tuy nhiên, để ý rằng 2 giá trị entropy này có phụ thuộc vào số lượng điểm dữ liệu ở mỗi nhánh, vì vậy ta sẽ phải lấy **trung bình có trọng số** với trọng số cho mỗi nhánh là **tỉ lệ giữa số điểm dữ liệu ở mỗi nhánh so với tổng số điểm dữ liệu ban đầu**.

[7]: `display.Image('fig5.png', height = 308)`

[7]:

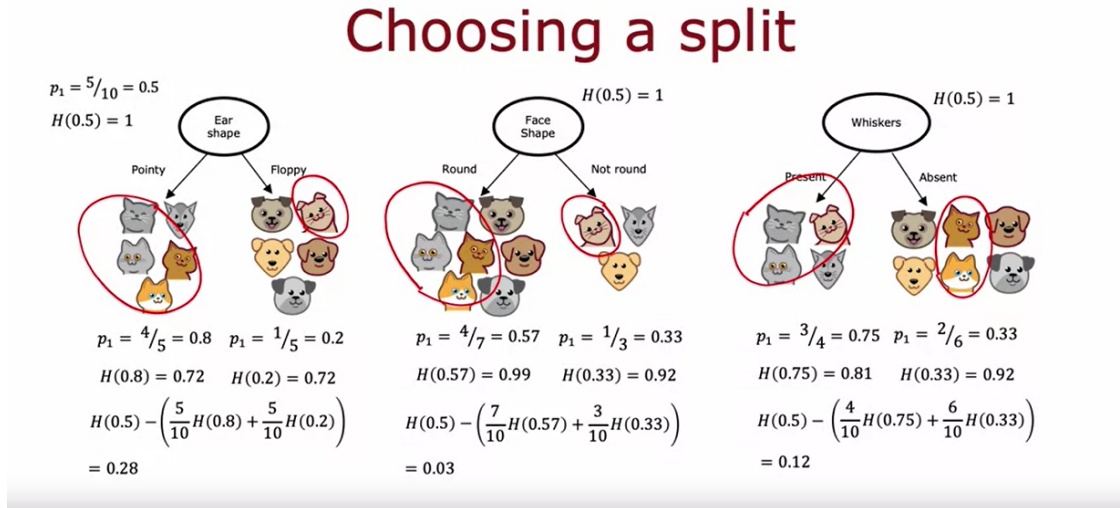


Figure 5: Chọn một feature để phân chia tập dữ liệu - Information Gain

- Nhưng để tổng quát hơn, ta quan tâm đến **lượng giảm entropy** giữa tập dữ liệu ban đầu và sau khi phân chia hơn là trung bình entropy có trọng số ở 2 nhánh, khi đó đại lượng Information Gain được định nghĩa như sau:
- **Information Gain:** là hiệu giữa entropy của tập dữ liệu ban đầu với trung bình entropy có trọng số ở 2 nhánh trái và phải.

$$H(p_1^{node}) - w^{left} * H(p_1^{left}) - w^{right} * H(p_1^{right})$$

- Trong đó:
 - $p_1^{node}, p_1^{left}, p_1^{right}$: là phân số biểu thị giữa số lượng điểm dữ liệu có nhãn $y = 1$ với tổng số lượng điểm dữ liệu của tập dữ liệu ban đầu/nhánh trái/nhánh phải.
 - w^{left}, w^{right} : là trọng số của nhánh trái và nhánh phải.
- Tại đây, ta chỉ cần tính information gain cho các feature có thể và quyết định **chọn ra feature có information gain lớn nhất**. Các decision node khác cũng được chọn feature hoàn toàn tương tự.
- Ngoài ra, đại lượng information gain cũng là một đại lượng giúp ta ngừng việc phân chia theo tiêu chí [4]: khi mà ta thấy information gain nhỏ hơn một ngưỡng nhất định nào đó.

2.6 Algorithm

- Với toàn bộ ý tưởng và lý thuyết đã trình bày bên trên, ta hoàn toàn có thể xây dựng một thuật toán **decision tree learning** hoàn chỉnh.
- **Thuật toán:**
 - Bắt đầu với tất cả điểm dữ liệu tại root node
 - Tính information gain cho tất cả các feature có thể và chọn cái có information gain cao nhất
 - Phân chia tập dữ liệu theo feature đã được chọn và tạo ra 2 nhánh trái và phải của cây
 - Lặp lại quá trình cho đến khi dừng lại bởi một trong các tiêu chí sau:
 - * Khi tất cả các điểm dữ liệu tại một node là 100% thuộc về một class duy nhất.
 - * Khi chiều cao của cây vượt quá một ngưỡng max depth nào đó.
 - * Khi số lượng điểm dữ liệu tại một node là nhỏ hơn một ngưỡng nào đó.
 - * Khi information gain nhận được từ việc phân chia là nhỏ hơn một ngưỡng nào đó.
- Thuật toán trên là một thuật toán **chia để trị - phân hoạch và có tính đệ quy**.

2.6.1 Complexity

- Độ phức tạp cho việc dự đoán 1 điểm dữ liệu là: $O(h)$
 - Trong đó h là chiều cao của cây hay max_depth
- Độ phức tạp cho quá trình learning là khá khó để tính toán khi đây là một thuật toán phân hoạch dữ liệu tại mỗi node.
 - Tuy nhiên, tại mỗi node độ phức tạp sẽ là $O(n_{features} * N * \log_2(N))$
 - Trong đó:
 - * $n_{features}$: là số lượng feature có thể của tập dữ liệu.
 - * N : là số lượng điểm dữ liệu tại node đó
 - * Lý do vì sao độ phức tạp có xuất hiện $\log_2(N)$ sẽ được trình bày bên dưới.
- Để ý rằng, số lượng node cần xây dựng của decision tree không quá lớn do chiều cao của cây luôn bị giới hạn bởi số lượng feature có thể cũng như các tiêu chí nghiêm ngặt trong việc dừng phân chia. \rightarrow Vì vậy decision tree learning là một thuật toán có chi phí learning tốt.

2.7 Problem Expansion

2.7.1 Continuous Valued Features

- Trong thực tế, các feature không thể chỉ bao gồm các giá trị rời rạc 0 hoặc 1 mà thường sẽ là các giá trị số thực liên tục, vậy làm sao để xử lý điều này?
- Cách đơn giản nhất để giải quyết điều này là ta sẽ sắp xếp lại các giá trị của feature thành một dãy tăng dần rồi **chọn các mốc để chia các giá trị này thành 2 phần \leq và $>$** (đưa continuous valued feature về binary feature).
- Ví dụ với 10 điểm dữ liệu ta có thể chọn ra 9 mốc để phân chia như hình bên dưới. Tính information gain cho mỗi mốc và quyết định chọn mốc tốt nhất.
- Cách làm này tưởng chừng sẽ làm tăng độ phức tạp cho việc tính toán và xây dựng decision tree, tuy nhiên do ta đã sắp xếp lại các giá trị nên ta hoàn toàn có thể sử dụng các kỹ thuật như **prefix sum** để tính nhanh các yếu tố $p_1^{node}, p_1^{left}, p_1^{right}, w^{left}, w^{right}$ trong công thức. Đó là lý do mà độ phức tạp của quá trình learning tại mỗi node có kết quả như bên trên.

```
[8]: display.Image('fig6.png', height = HEIGHT)
```

[8]:

Splitting on a continuous variable

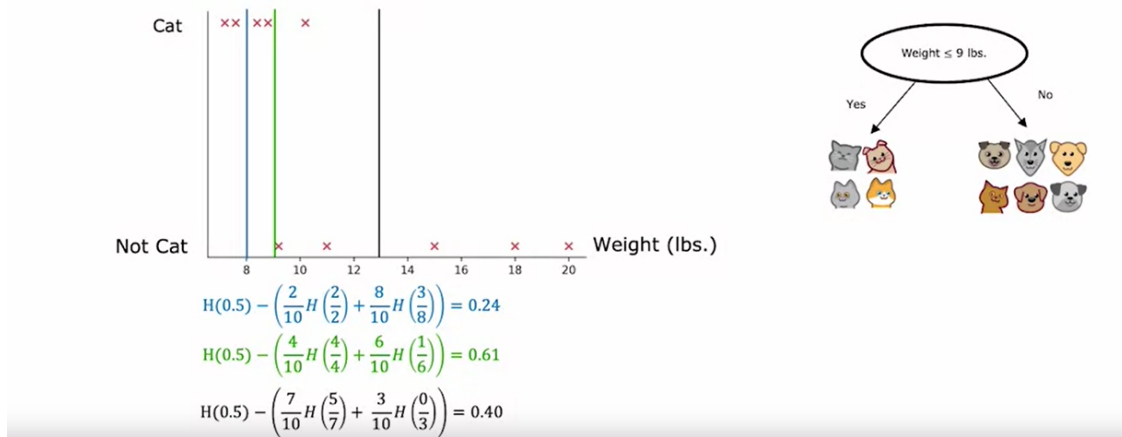


Figure 6: Xử lý continuous variable

2.7.2 Categorical Features

- Tương tự với các feature có dạng *categorical*, ta hoàn toàn có thể mã hóa chúng thành các giá trị rời rạc 0, 1, 2, và thực hiện tính toán như cách chúng ta làm với *continuous valued features*.
- Tuy nhiên có một cách tốt hơn, sử dụng **OneHot Encoding** (bài báo cáo này sẽ không trình bày chi tiết về kỹ thuật OneHot Encoding).
- Hạn chế của cách làm này là OneHot Encoding sẽ tạo ra thêm k feature với categorical feature có k danh mục, vì vậy sẽ làm tăng số lượng feature của tập dữ liệu nhưng lại đảm bảo rằng hiệu suất của thuật toán sẽ cao hơn.

2.7.3 Multiclass Classification

- Như đã trình bày về công dụng của decision tree, ta hoàn toàn có thể sử dụng để giải quyết bài toán Multiclass Classification.
- Thật vậy, thứ duy nhất chúng ta cần điều chỉnh sẽ chỉ là công thức tính entropy của tập dữ liệu.
- Một cách tổng quát, entropy của tập dữ liệu có nhãn y thuộc k class sẽ có công thức là:

$$H(p_0, p_1, \dots, p_{k-1}) = -p_0 * \log_k(p_0) - p_1 * \log_k(p_1) - \dots - p_{k-1} * \log_k(p_{k-1})$$

- Trong đó:
 - p_i là phân số biểu thị giữa số lượng điểm dữ liệu có nhãn $y = i$ với tổng số điểm dữ liệu
 - $p_0 + p_1 + p_2 + \dots + p_{k-1} = 1$
 - log bây giờ sẽ là \log_k**
- Các tính chất của hàm số hoàn toàn tương tự với hàm số khi chỉ có 2 class.

2.7.4 Regression

- Tương tự như Multiclass Classification để có thể giải quyết được bài toán Regression chúng ta cũng điều chỉnh lại công thức entropy:
 - Khi đó thay entropy bằng phương sai - **variance** của các biến y của tập dữ liệu.
 - **Kết quả của các leaf node** sẽ là trung bình cộng của các biến y của tập dữ liệu tại leaf node đó (*mean method*).

3 Using Decision Tree with sklearn

- Import thư viện

```
[9]: import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, confusion_matrix,
↳ConfusionMatrixDisplay, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV,
↳StratifiedKFold
```

- Chuẩn bị tập dữ liệu $dataset = (X, y)$
- Trong đó:
 - shape of X : (n_examples, n_features), shape of y : (n_examples,)
 - X bắt buộc phải là số, y có thể không bắt buộc phải là số

```
[10]: from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

- Có thể sử dụng hàm **train_test_split** để tạo ra tập train và tập test, phục vụ cho việc huấn luyện và kiểm tra hiệu suất của mô hình

```
[11]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

- **Tạo một estimator** để dự đoán với các hyperparameter như `max_depth` [2], `min_samples_split` [3]

```
[12]: clf = DecisionTreeClassifier(criterion='entropy', max_depth=3,
↳min_samples_split=3)
```

- Thực hiện learning bằng phương thức **fit**

```
[ ]: clf.fit(X_train, y_train)
```

- Dự đoán kết quả cho các điểm dữ liệu bằng phương thức **predict**

```
[14]: y_pred = clf.predict(X_test)
```

- Đánh giá hiệu suất của mô hình bằng phương thức **score** hoặc hàm **accuracy_score**

```
[15]: accuracy_test = accuracy_score(y_test, y_pred)
accuracy_train = clf.score(X_train, y_train)
```

- **Trực quan hóa mô hình** bằng hàm `tree.plot` để thấy cách mà mô hình được xây dựng và phân loại

```
[ ]: tree.plot_tree(clf)
```

- Vẽ **confusion_matrix** để thực hiện phân tích lỗi và xem xét hiệu suất chính xác hơn

```
[ ]: cfs_mtx = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cfs_mtx)
disp.plot()
```

- **Tạo một bản báo cáo** chi tiết về các thông số đánh giá hiệu suất của mô hình phân loại

```
[ ]: print(classification_report(y_test, y_pred))
```

- Tune các hyperparameter bằng **GridSearchCV**

```
[ ]: # parameters for GridSearch with Cross
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': list(range(1, 10)),
    'min_samples_split': list(range(2, 10))
}
# search fitting
GSCross = GridSearchCV(estimator=clf, param_grid=param_grid, cv=3)
GSCross.fit(X_train, y_train)
```

```
[ ]: GSCross.best_params_
GSCross.best_score_
```

```
[ ]: # receive best_estimator
best_est = GSCross.best_estimator_
best_est.score(X_test, y_test)
```

4 Discussion

4.1 The Characteristics and Observations about Decision Tree

4.1.1 Decision Tree là một mô hình học không tham số có giám sát - non-parametric supervised learning

- Khác với các mô hình khác như *linear regression*, *logistic regression*, decision tree không sử dụng các tham số - **parameters** cho mô hình.
- Ta có thể xem feature và các rule được chọn ở mỗi node như là parameters cho mô hình.

- Các **hyperparameters** phổ biến của decision tree như: *max_depth*, *min_samples_split* có thể được sử dụng cho việc dừng quá trình phân chia để tránh *overfitting*. Ta cũng có thể xem *max_depth* như là **bậc của mô hình**.
- Cả tiêu chí ***gini*** và ***entropy*** đều có thể được xem xét làm tiêu chí để chọn feature và rule. Tuy nhiên *gini* có thể dễ dàng tính hơn *entropy* (do không chứa logarit), nhưng các đặc điểm của 2 hàm số đều gần như nhau.
- Công thức *gini*:

$$G(p_1, p_0) = 1 - p_1^2 - p_0^2$$

- Trong đó: p_1, p_0 như đã định nghĩa với *entropy*.

4.1.2 Ranh giới phân loại của Decision Tree chỉ có thể là các đường thẳng (siêu phẳng) song song với các trục tọa độ

- Thật vậy, tại mỗi node ta chỉ **chọn duy nhất một feature và một mốc tương ứng (*threshold*)** để phân chia nếu là continuous valued features nên các đường ranh giới phân loại của decision tree chỉ có thể là các đường thẳng song song với trục tọa độ.

[22]: `display.Image('fig7a.png', height = HEIGHT)`

[22]:

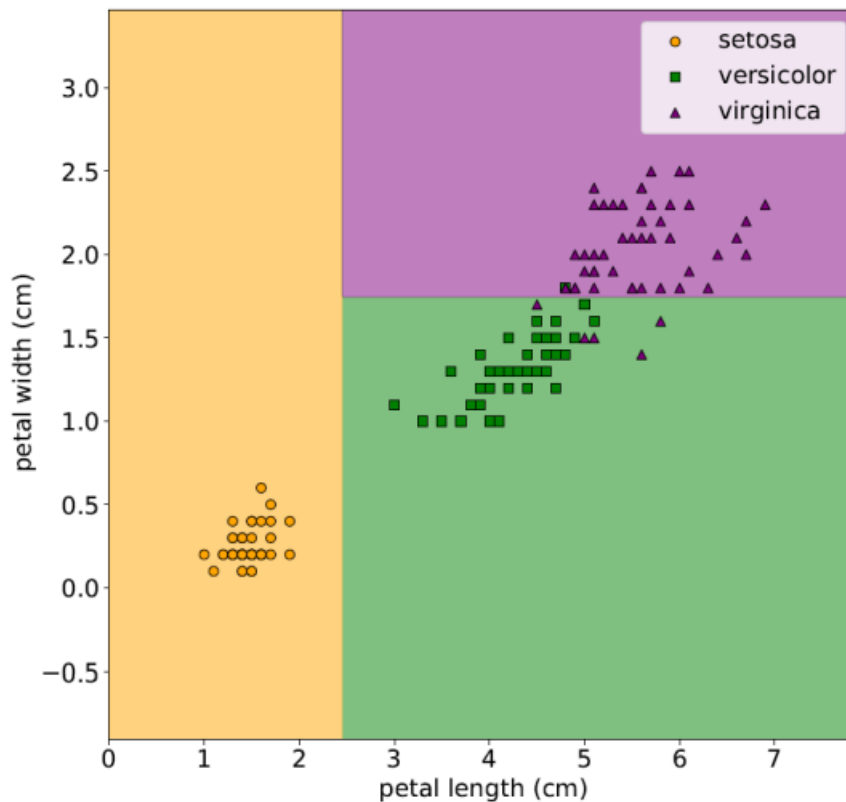


Figure 7a: Đường ranh giới phân loại của decision tree

- Điều này có thể là một điểm hạn chế của decision tree, bởi lẽ nhiều bộ dữ liệu các đường ranh giới nên là các đường chéo thay vì chỉ là các đường thẳng song song với trục tọa độ.

```
[23]: display.Image('fig7b.png', height = 308)
```

[23]:

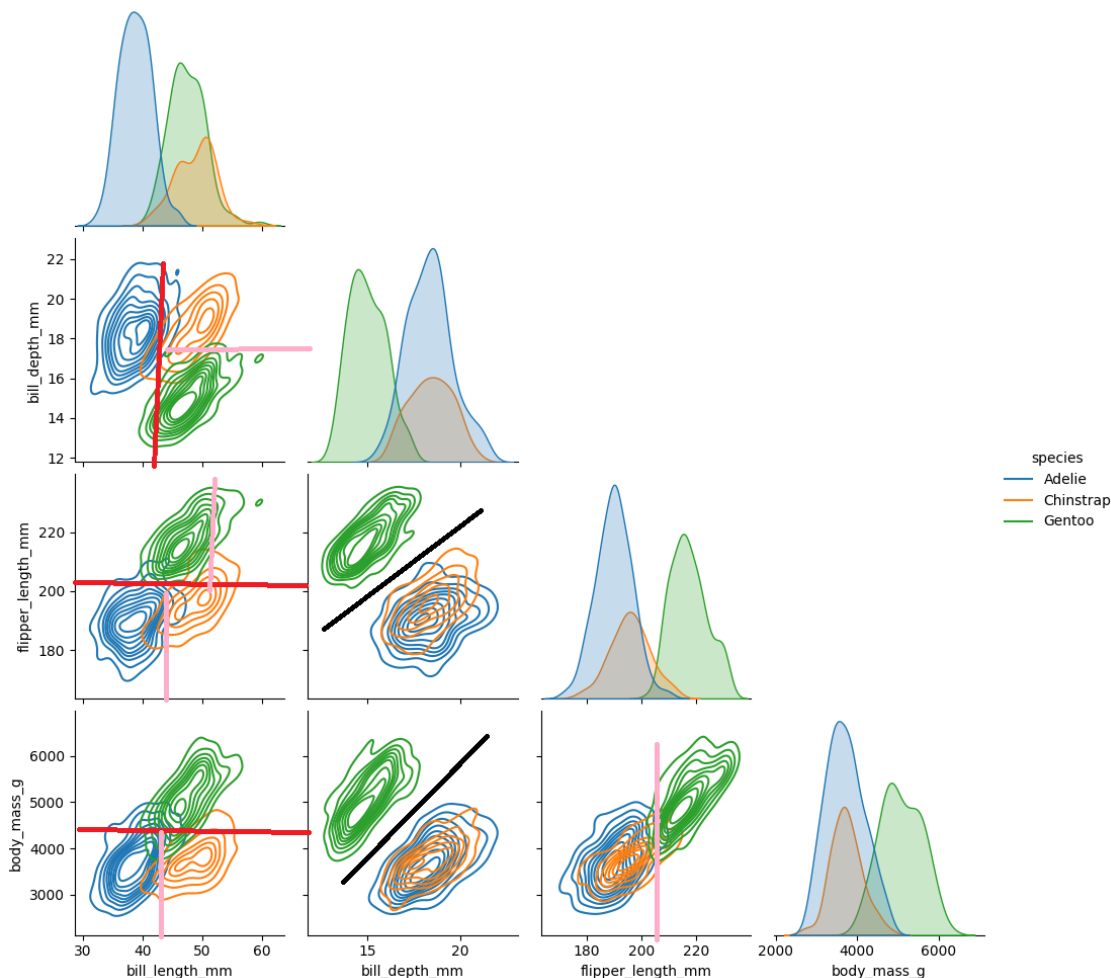


Figure 7b: Đường ranh giới phân loại của decision tree

- Rõ ràng trong hình ảnh trên, các đường ranh giới phân loại nên là các đường chéo màu đen thay vì là sự kết hợp của các đường thẳng song song với trục tọa độ.
- Ta có thể khắc phục điều này bằng cách tại mỗi node ta sẽ chọn 2 feature (hoặc thậm chí là nhiều hơn) và chọn một đường thẳng trong không gian của 2 feature đó để làm đường phân chia tập dữ liệu thay vì chỉ là một mốc (threshold ứng với 1 feature):
 - Tuy nhiên câu hỏi đặt ra ở đây là làm sao để chọn đường thẳng phân chia đó?
 - Ta cần sự hỗ trợ của một vài thuật toán phức tạp khác, cũng như tiêu tốn thêm chi phí về thời gian xây dựng cây.

4.1.3 Decision Tree là một cây nhị phân

- Cần lưu ý rằng theo cấu trúc trình bày tương ứng với thuật toán bên trên (cũng là cấu trúc và thuật toán đang được sử dụng phổ biến nhất với mô hình decision tree) thì tại mỗi node, sẽ **chỉ phân chia thành 2 nhánh trái và phải** tương ứng với các giá trị 1 hoặc 0 (yes hoặc no).
- Các phiên bản cấu trúc tương ứng với thuật toán khác có thể khiến decision tree trở thành cây k phân, tuy nhiên điều cần cân nhắc khi áp dụng các phiên bản đó là:
 - Chi phí về mặt lưu trữ lớn
 - Thời gian tính toán, xây dựng lớn
 - Kiểm soát chiều cao của cây, cắt tỉa nhánh
 - Tránh overfitting

4.1.4 Không phải tất cả các feature của tập dữ liệu đều được sử dụng để phân loại

- Nhắc lại, tại mỗi node ta **chỉ xem xét 1 feature duy nhất một cách độc lập** để quyết định chọn nó hay không mà không quan tâm đến sự ảnh hưởng của các feature khác đến nó. Hơn nữa chiều cao của cây là bị giới hạn bởi max_depth cũng như các tiêu chí nghiêm ngặt khác.
- -> Điều này dẫn đến rằng, **không phải tất cả các feature đều được chọn để phân loại và đều ảnh hưởng đến việc phân loại**.
- Vậy nên các feature được chọn để phân loại sau quá trình learning thường là các feature quan trọng, có tác động lớn đến biến y.
- -> Ta có thể vận dụng điều này để sử dụng decision tree nhằm rút trích các feature quan trọng hay tìm các đặc điểm của tập dữ liệu như đã trình bày về công dụng của decision tree.
- Để khắc phục và cải thiện việc không phải tất cả các feature đều được sử dụng để phân loại của decision tree ta có thể nghĩ đến ý tưởng **lấy mẫu các điểm dữ liệu từ tập dữ liệu với tập con của tập feature thay vì là toàn bộ feature để tiến hành learning trên từng decision tree một rồi kết hợp kết quả của các decision tree đó lại để dự đoán kết quả cuối cùng**. Đây cũng chính là ý tưởng ban đầu của các mô hình *Tree Ensembles* lớn hơn như *Random Forest* hay *XGBoost*.

4.1.5 Decision không hiệu quả với các loại dữ liệu không cấu trúc: image, text, audio,

- Như đã đề cập decision tree chỉ xem xét các feature để phân loại một cách độc lập tại mỗi node nên nó không phù hợp để giải quyết các bài toán mà dữ liệu không phải dạng bảng như image, text, audio,

4.2 When to use Decision Trees

- **Decision Tree (non-parametric) vs Neural Network (parametric):**
- **Decision Tree:**
 - Decision Tree hoạt động trên tập dữ liệu có cấu trúc (dạng bảng)
 - Không được khuyến khích với tập dữ liệu không có cấu trúc như image, text, audio
 - Yêu cầu ít điểm dữ liệu để huấn luyện do không có quá nhiều tham số mô hình.
 - Nhanh: chi phí cho việc dự đoán và quá trình learning đều khá nhỏ và phù hợp.
 - Được hiểu và giải thích dễ dàng bởi con người - *human interpretable* do chúng ta có thể trực quan hóa cấu trúc tree.
- **Neural Network:**

- Hoạt động tốt trên tất cả các loại dữ liệu: có cấu trúc và không có cấu trúc.
- Chứa nhiều tham số mô hình nên có lẽ là cần nhiều điểm dữ liệu hơn để huấn luyện và chậm hơn decision tree.
- Nhưng có thể sử dụng kỹ thuật ***transfer learning*** nên sẽ tiết kiệm chi phí kinh tế hơn trong công nghiệp so với xây dựng các mô hình như XGBoost.
- Khó khăn để có thể được hiểu và giải thích về cấu trúc và trực quan hóa mô hình.

5 Reference Documents

- Advanced Learning Algorithms, Coursera, Andrew Ng:
 - <https://www.coursera.org/learn/advanced-learning-algorithms>
- sklearn: <https://scikit-learn.org/stable/modules/tree.html>
- geeksforgeeks: <https://www.geeksforgeeks.org/decision-tree/>
- **algorithm demonstration sourcecode:**
 - colab:
 - <https://colab.research.google.com/drive/1oFhCdeEUeVT21CSJTo5KT00ACeNxCnqL?usp=sharing>
 - github:
 - https://github.com/QuanHoangNgoc/All_Decision_Tree_Hoang_Ngoc_Quan_22521178
- **Author: Hoàng Ngọc Quân - 22521178**