TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN LẬP TRÌNH

TÌM HIỂU VỀ NGÔN NGỮ LẬP TRÌNH PYTHON VÀ CÁC THUẬT TOÁN CƠ BẢN

NGUYỄN DUY CƯỜNG - 20195845
LỚP: HỆ THỐNG THÔNG TIN VÀ TRUYỀN THÔNG K64

GIẢNG VIÊN HƯỚNG DẪN
TS. BÙI QUỐC TRUNG

TP. HÀ NỘI, 2022

Mục lục

Danh sách thuật ngữ

Tiếng Anh	Tiếng Việt
accuracy	độ chính xác
artificial intelligence	trí tuệ nhân tạo, trí thông minh nhân tạo
attribute	thuộc tính
child node	nút con
classification	phân loại
continuous	liên tục
decision tree	cây quyết định
discrete	rời rạc
entropy	độ hỗn loạn thông tin
gini index	chỉ số đo lường độ không sạch của dữ liệu
information gain	độ lợi thông tin
label	nhãn
leaf node	nút lá, nút không có con
machine learning	máy học, học máy
non-leaf node	nút trong, nút có con
overfitting	quá khớp
random variable	biến ngẫu nhiên
regression	hồi quy
root node	nút gốc
sibling node	các nút có cùng nút cha
supervised learning	học có giám sát
test data	dữ liệu kiểm tra
training score	độ chính xác khi huấn luyện
training data	dữ liệu huấn luyện
underfitting	không khớp
unsupervised learning	học không giám sát

MỞ ĐẦU

Lý do chọn đề tài

Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Gusso tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình và là ngôn ngữ lập trình dễ học.

Hiện nay, sự bùng nổ của kỷ nguyên số về các lĩnh vực như Trí Tuệ Nhân Tạo (Artificial Intelligence) và Dữ Liệu Lớn (Big Data) đã góp phần gia tăng không nhỏ nhu cầu sử dụng Python trong những năm gần đây và còn tiến xa hơn nữa trong tương lai.

Nội dung gồm 3 phần chính:

- **Tìm hiểu về ngôn ngữ lập trình python** - tìm hiểu một cách tổng quan về ngôn ngữ lập trình python và một số thư viện của nó.
- **Thuật toán tìm đường đi ngắn nhất** - phần này tìm hiểu và cài đặt 2 thuật thuật toán tìm đường đi ngắn nhất trên đồ thị vô hướng là thuật toán Dijkstra và Bellman-Ford.
- **Thuật toán cây quyết định** - thực hiện tìm hiểu về một thuật toán trong Machine Learning là thuật toán cây quyết định.

Mục đích thực hiện đề tài

Khi thực hiện đề tài, em mong muốn được tiếp cận và tìm hiểu ngôn ngữ lập trình Python và một số thư viện của python cũng như hiểu và cài đặt một số thuật toán cơ bản trên Python. Và từ đó vận dụng vào ngành đang học - Hệ thống thông tin và truyền thông.

Hai mục tiêu hướng đến là:

- Thứ nhất, sẽ có kiến thức cơ bản về Python và các thư viện trên Python.
- Thứ hai, tìm hiểu và có thể cài đặt một số thuật toán cơ bản đã được học hoặc thuật toán mới.

Chương 1

TÌM HIỂU VỀ NGÔN NGỮ LẬP TRÌNH PYTHON

1.1 Cú pháp Python cơ bản

1.1.1 Cài đặt môi trường

Giống như các ngôn ngữ lập trình khác như Java, C++,... khi muốn chạy một chương trình Python, trước hết cần cài đặt một môi trường có thể đọc và chạy được Python.

Hiện phiên bản mới nhất của Python được cập nhật và dowload miễn phí trên trang chủ python.org là Python 3.9.7.

Cùng với sự phát triển và phổ biến của Python, hiện có rất nhiều các editor có thể được sử dụng để lập trình Python. Dựa trên kết quả đánh giá chức năng, số lượng người dùng và phản hồi tích cực từ phía người dùng, có thể kể đến một số phần mềm lập trình như sau:

- **PyDev** - phát triển bởi Eclipse Foundation.
- **PyCharm** - phát triển bởi JetBrains.
- **Visual Studio Code** - phát triển bởi Mycrosoft.
- **Sublime Text** - phát triển bởi Sublime HQ.
- **Spyder** - phát triển bởi Pierre Raybaut, chuyên dùng để lập trình khoa học.

1.1.2 Biến

Biến chính xác như tên gọi của nó, tức là giá trị của nó có thể thay đổi. Các biến có thể giúp lưu trữ bất cứ cái gì nếu có thể định nghĩa được nó. Các biến chỉ là một phần của bộ nhớ máy tính nơi lưu trữ một số thông tin.

Xác định kiểu dữ liệu của biến bằng lệnh type(biến).

Có kiểu biến toàn cục và biến địa phương, biến được khai báo ngoài hàm là biến toàn cục còn biến được khai báo trong hàm là biến địa phương.

Ví dụ 1.1. Ví dụ về khai báo biến:

```
# Biến toàn cục
x = 5
def function():
    # biến cục bộ
    x = 10.5
    print(x)
```

1.1.3 Toán tử

Về cơ bản trong Python cũng có các toán tử số học, quan hệ, logic...tương tự như các ngôn ngữ khác.

- **Toán tử số học**

Toán tử	Mô tả	Ví dụ: a=20, b=10
+	Phép cộng	$a + b = 30$
	Phép trừ	$a - b = 10$
*	Phép nhân	$a * b = 200$
/	Phép chia lấy phần nguyên.	$a/b = 2$
%	Phép chia lấy phần dư.	$a \% b = 0$
**	Phép lấy số mũ	$a * * b = 10^{20}$
//	Floor Division	$9//2 = 4$

Bảng 1.1: Toán tử số học

- **Toán tử quan hệ**

Toán tử	Mô tả	Ví dụ: a=20, b=10
==	So sánh bằng	$(a == b)$ cho kết quả False
! =	So sánh không bằng.	$(a! = b)$ cho kết quả True.
<>	Không bằng	$(a <> b)$ cho kết quả True.
>	Lớn hơn.	$(a > b)$ cho kết quả True.
<	Nhỏ hơn.	$(a < b)$ cho kết quả False
>=	Lớn hơn hoặc bằng.	$(a >= b)$ cho kết quả True
<=	Nhỏ hơn hoặc bằng.	$(a <= b)$ cho kết quả False

Bảng 1.2: Toán tử quan hệ

- **Toán tử logic**
- **Toán tử gán**

Toán tử	Mô tả	Ví dụ: a=10, b=20
and	Nếu cả 2 biểu thức là True thì cho kết quả True.	$(a > 9 and b > 19) => True$
or	Nếu 1 trong 2 là True thì cho kết quả True.	$(a > 9 \, or \, b > 100) => True$
not	Được sử dụng phủ định logic của biểu thức.	$not \, (a > 9) => False$ cho kết quả True.

Bảng 1.3: Toán tử logic

Toán tử	Mô tả	Ví dụ
$=$	Phép gán bằng.	$c = a + b$ gán $a + b$ cho c
$+ =$	Phép gán cộng.	$c+ = a <=> c = c + a$
$- =$	Phép gán trừ.	$c - = a <=> c = c - a$
$* =$	Phép gán nhân.	$c * = a <=> c = c * a$
$/ =$	Phép gán chia lấy phần nguyên.	$c \, / = a <=> c = c/a$
$\% =$	Phép gán chia lấy phần dư.	$c \, \% = a <=> c = c\%a$
$** =$	Phép gán lũy thừa.	$c \, ** = a <=> c = c * *a$
$// =$	Phép gán chia floor.	$c \, // = a <=> c = c//a$

Bảng 1.4: Toán tử gán

- **Toán tử khai thác**

Toán tử	Mô tả
in	Trả về true nếu tìm thấy biến a trong tập các giá trị sau in, ngược lại trả về false.
not in	Trả về false nếu tìm thấy biến a trong tập các giá trị sau in, ngược lại trả về false.

Bảng 1.5: Toán tử khai thác

- **Toán tử nhận dạng** : is và not is

Toán tử	Mô tả
is	Kiểm tra xem hai giá trị có bằng nhau hay không. Toán tử này sẽ trả về True nếu a == b và ngược lại
not is	Ngược lại của is.

Bảng 1.6: Toán tử nhận dạng

- **Toán tử thao tác bit**

Toán tử	Mô tả
&	AND
|	OR
^	XOR
~	NOT
<<	Dịch trái nhị phân
>>	Dịch phải nhị phân

Bảng 1.7: Toán tử thao tác bit

Thứ tự ưu tiên của các toán tử: Việc áp dụng nhiều toán tử trong một câu lệnh cần được chú ý đến thứ tự ưu tiên của các toán tử đó. Theo thứ tự ưu tiên giảm dần như sau: toán tử số học, toán tử thao tác bit, toán tử quan hệ, toán tử gán, toán tử identify, toán tử membership, toán tử logic.

1.1.4 Các kiểu dữ liệu

Python cũng hỗ trợ các kiểu dữ liệu căn bản như:char, string, int, float, double... Ngoài ra Python cung cấp một loạt các dữ liệu phức hợp, thường được gọi là các chuỗi (sequence), sử dụng để nhóm các giá trị khác nhau như list, set , tuple và dictionary. Áp dụng hợp lý các kiểu dữ liệu này có thể khiến việc lập trình nhanh hơn rất nhiều.

- **List**: Đây là kiểu dữ liệu được đánh giá là đa năng nhất. List được biểu diễn bằng dãy các giá trị, được phân tách nhau bằng dấu phẩy, nằm trong dấu []. List không giới hạn số lượng mục, có thể có nhiều kiểu dữ liệu khác nhau trong cùng một list, như chuỗi, số nguyên, số thập phân,...

  Các giá trị trong list có chỉ số, có thứ tự, có thể thay đổi và cho phép lặp lại.

  Ví dụ 1.2. Khai báo và sử dụng kiểu dữ liệu danh sách:

```
list1 = [1, 2, 3.5, 10]
list2 = [1, 1, 'hello', 'my list', 10.1, True]
```

- **Tuple**: Tuple trong Python là một chuỗi các phần tử có thứ tự giống như list. Sự khác biệt giữa list và tuple là không thể thay đổi các phần tử trong tuple khi đã gán, nhưng trong list có thể thay đổi.

  Tuple thường được sử dụng cho các dữ liệu không cho phép sửa đổi và nhanh hơn list vì nó không thể thay đổi tự động.

  Ví dụ 1.3. Ví dụ về kiểu dữ liệu tuple:

```
mytuple = ("apple", "banana", "cherry")
mytuple = (1, 1, 2, 3.5, 5, "hello")
```

- **Set** : Set trong Python là tập hợp các phần tử duy nhất, không có thứ tự. Các phần tử trong set phân cách nhau bằng dấu phẩy và nằm trong dấu ngoặc nhọn . Các phần tử trong set có thể thay đổi.

  Ví dụ 1.4. Ví dụ về kiểu dữ liệu set:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

- **Dictionary**: Dictionary là tập hợp các cặp khóa giá trị không có thứ tự. Các dictionary được tối ưu hóa để trích xuất dữ liệu với điều kiện phải biết được khóa để lấy giá trị. Trong Python, dictionary được định nghĩa trong dấu ngoặc nhọn với mỗi phần tử là một cặp theo dạng key:value. Key và value này có thể là bất kỳ kiểu dữ liệu nào.

  Ví dụ 1.5. Ví dụ về kiểu dữ liệu dictionary:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
```

1.1.5 Hàm

Trong Python, hàm là một nhóm các lệnh có liên quan đến nhau được dùng để thực hiện một tác vụ, nhiệm vụ cụ thể nào đó. Hàm giúp chia chương trình thành những khối/phần/-module nhỏ hơn.

- Khi khởi tạo hàm không có kiểu dữ liệu trả về tuy nhiên ta vẫn có thể dùng lệnh return để trả về giá trị xác định.
- Các tham số mặc định của hàm là các tham số không bắt buộc phải cung cấp khi gọi hàm và khi đó giá trị của nó là giá trị mặc định. Sử dụng tham số mặc định bằng cách dùng toán tử =.
- Tham số tùy biến sử dụng khi ta không biết trước số lượng tham số sẽ truyền vào hàm. Khi đó biến có thể coi như một mảng các giá trị. Sử dụng tham số tùy biến bằng cách đặt dấu * trước tên tham số.

Ví dụ 1.6. Ví dụ về khai báo và định nghĩa hàm:

```
a = "Hello"

def Hello(a):
    print(a + "World")     # Hello World
```

1.1.6 Lập trình hướng đối tượng (OOP): Class

Python là một ngôn ngữ lập trình hỗ trợ hướng đối tượng tương tự như Java, C++, PHP... Nó cũng có các nguyên lý cơ bản của một ngôn ngữ hướng đối tượng đó là tính đóng gói, tính kế thừa, tính đa hình.

Hầu hết mọi thứ trong Python đều là một object, với tính chất và phương thức của riêng nó.

Class chính là một kiểu dữ liệu đặc biệt, dùng để tạo nên một đối tượng.

Ví dụ 1.7. Ví dụ về khai báo một class:

```
class MyClass:
    x = 5
```

Chúng ta có thể tạo một đối tượng từ class đã tạo:

Ví dụ 1.8. Tạo một đối tượng p1, và in ra giá trị của x:

```
p1 = MyClass()
print(p1.x)
```

Hàm __init__()

Để hiểu rõ hơn ý nghĩa của class ta cần hiểu về hàm __init__(). Mỗi classes đều có một hàm __init__(), hàm này sẽ được thực thi ngay khi class được bắt đầu.

Ta sử dụng hàm __init__() để gán gía trị, thuộc tính và các thao tác cần thiết khi đối tượng được tạo.

Ví dụ 1.9. Tạo một class Person, sử dụng hàm __inti__() để gán tên và tuổi:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)  # John
print(p1.age)   # 36
```

Phương thức của đối tượng

Mỗi đối tượng có thể có các phương thức. Phương thức là những hàm thuộc về đối tượng ấy.

Ví dụ 1.10. Tạo phương thức in ra lời chào, và sử dụng nó trên đối tượng p1:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

1.2 Một số thư viện của Python

Do hạn chế trong việc trình bày báo cáo nên em chỉ nêu một số ý chính của các thư viện và một số ví dụ. Các thao tác với thư viện đầy đủ hơn em đặt ở trong các file jupyter notebook của riêng từng thư viện.

1.2.1 Numpy

Numpy (viết tắt của "Numeric Python" hoặc "Numerical Python") là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

Để cài đặt numpy, ta có 2 cách:

- Sử dụng Command Prompt:

```
pip install numpy
```

- Hoặc sử dụng Anaconda Prompt:

```
conda install numpy
```

- **Khai báo thư viện**:

```
import numpy as np
```

- **Khởi tạo mảng** : Numpy cho phép ta tạo các mảng 1 chiều hay nhiều chiều với nhiều cách thức khác nhau.

  Ví dụ 1.11. Một số cách tạo mảng:

```
# Tạo mảng 1 chiều
a = np.array([1, 2, 3, 4])   # a =  [1 2 3 4]

# Mảng 2 chiều(nhiều chiều)
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])   # b =  [[1 2 3 4]
                                             #       [5 6 7 8]]
```

```
# Tạo mảng với các giá trị chưa biết nhưng đã biết kích thước
d = np.zeros((3, 4))
```

- **Các thao tác với mảng** : Numpy cho phép ta thực hiện các thao tác với mảng như: thực hiện các phép toán giữa các mảng, giữa các giá trị trên mảng, index, slicing, điều khiển kích thước của mảng, ...

  Ví dụ 1.12. Một vài ví dụ:

```
A = np.array([[1, 1],
              [0, 1]])

B = np.array([[2, 0],
              [3, 4]])

arr1 = A * B          # Phép nhân theo từng phần tử
arr2 = A @ B          # Phép nhân ma trận. Cách khác: A.dot(B)
A *= 3                # Nhân mỗi phần tử của A với 3
A.sum(axis=1)         # Tổng các phần tử của A theo hàng
```

1.2.2 Pandas

Thư viện pandas trong python là một thư viện mã nguồn mở, hỗ trợ đắc lực trong thao tác dữ liệu. Đây cũng là bộ công cụ phân tích và xử lý dữ liệu mạnh mẽ của ngôn ngữ lập trình python. Thư viện này được sử dụng rộng rãi trong cả nghiên cứu lẫn phát triển các ứng dụng về khoa học dữ liệu. Thư viện này sử dụng một cấu trúc dữ liệu riêng là Dataframe. Pandas cung cấp rất nhiều chức năng xử lý và làm việc trên cấu trúc dữ liệu này.

Để cài đặt pandas, ta có 2 cách:

- Sử dụng Command Prompt:

```
pip install pandas
```

- Hoặc sử dụng Anaconda Prompt:

```
conda install pandas
```

- **Khai báo thư viện**:

```
import pandas as pd
```

- **Khởi tạo** Pandas có 2 cấu trúc dữ liệu cơ bản là: Series (1 chiều) và DataFrame (2 chiều)

  Có nhiều cách để tạo một cấu trúc dữ liệu trên pandas như: truyền trực tiếp giá trị, lấy từ mảng trên Numpy, từ file, ...

  Ví dụ 1.13. Một số cách tạo một cấu trúc dữ liệu trên pandas:

```
# Tạo một series
series = pd.Series([1, 2, 4, 5, 7, 9, np.nan])

# Tạo DataFrame bằng cách truyền giá trị từ cấu trúc dữ liệu dictionary trong python
df2 = pd.DataFrame({
    "A" : [1, 2, 1, 2],
    "B" : pd.Timestamp("20130102"),
    "C" : pd.Series(1, index=list(range(4)), dtype="float32"),
    "D" : np.array([3]*4, dtype='int32'),
    "E" : pd.Categorical(["test", "train", "validation", "test"]),
    "F" : "Foo"
})
```

- **Xem xét, lấy dữ liệu từ cấu trúc dữ liệu** Pandas cho phép ta thao tác một cách trực quan, đơn giản với các cấu trúc dữ liệu. Ta có in ra dữ liệu, thêm, xóa, thay đổi, thống kê, chọn từng phần dữ liệu,... Ta cũng có thể lưu các dạng cấu trúc dữ liệu trên pandas thành các định dạng file khác nhau như csv, hdf5, ...

  Ví dụ 1.14. Một số thao tác với dữ liệu:

```
df.head()              # Xem 10 hàng đầu tiên của dữ liệu
df.index               # Xem các tên chỉ số
df.columns             # Xem các cột
df.to_numpy()          # chuyển từ DataFrame sang mảng trong Numpy
df.describe()          # Hiển thị thống kê của dữ liệu
```

1.2.3 Matplotlib

Matplotlib là một thư viện của Python được sử dụng để vẽ đồ thị với sự trợ giúp của các thư viện khác như Numpy và Pandas. Nó được sử dụng để tạo ra các nhiễu thống kê(statistical interferences, xuất hiện khi hai phân phối xác xuất trùng lặp lên nhau) và vẽ đồ thị 2D của các mảng. Matplotlib sử dụng pyplot để cung cấp giao diện tương tự như MATLAB và là một mã nguồn mở. Để cài đặt matplotlib, ta có 2 cách:

- Sử dụng Command Prompt:

```
pip install matplotlib
```

- Hoặc sử dụng Anaconda Prompt:

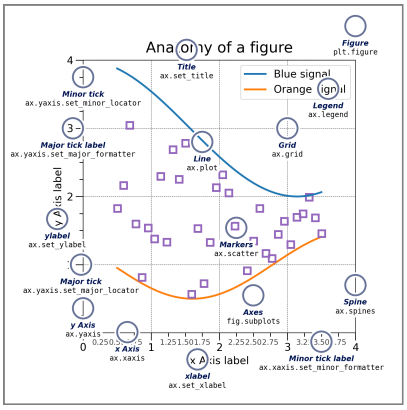
```
conda install matplotlib
```

Các thành phần trong đồ thị của Matplotlib:

- **Khai báo thư viện**:

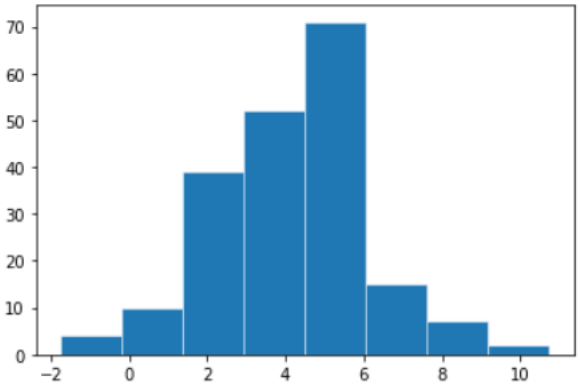
```
import numpy as np
```

Ví dụ 1.15. Một số dạng đồ thị:



Hình 1.1: Cấu trúc của đồ thị trong matplotlib

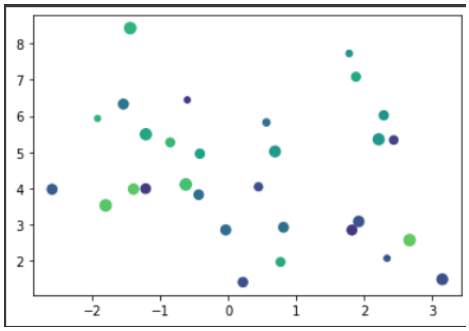
```
# hist(x)
x = 4 + np.random.normal(0, 2, 200)
fig, ax = plt.subplots()
ax.hist(x, bins=8, linewidth=0.5, edgecolor='white')

plt.show()
```



Hình 1.2: Đồ thị hist

Ví dụ 1.16. Một số dạng đồ thị:

```
# scatter(x, y)
x = np.random.normal(0, 2, 30)
y = 4 + np.random.normal(0, 2, 30)
# Chỉ định kích thước và màu sắc cho đồ thị được vẽ.
sizes = np.random.uniform(15, 80, 30)
colors =  np.random.uniform(15, 80, 30)
fig, ax = plt.subplots()
ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)
plt.show()
```



Hình 1.3: Đồ thị scatter

1.2.4 Seaborn

Seaborn cũng là một thư viện của Python giúp vẽ các đồ thị với sự trợ giúp của Matplotlib, Numpy và Pandas. Nó có thể được coi là một phần mở rộng của một thư viện Matplotlib vì nó được xây dựng dựa trên đó. Seaborn giúp quan sát các biến dữ liệu dạng đơn trị và đa trị. Nó còn có các đồ thị rất trực quan và đẹp mắt.

Cài đặt và khai báo seaborn:

- Sử dụng Command Prompt:

```
pip install seaborn
```

- Hoặc sử dụng Anaconda Prompt:

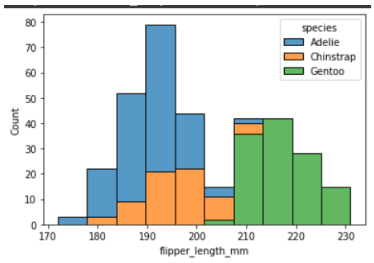
```
conda install seaborn
```

- Khai báo thư viện:

```
import seaborn as sns
```

Ví dụ 1.17. Ví dụ đồ thị:

```
penguins = sns.load_dataset('penguins')
sns.histplot(data=penguins, x='flipper_length_mm', hue='species', multiple='stack')
```



Hình 1.4: Đồ thị trong seaborn

Chương 2

THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT

2.1 Bài toán đường đi ngắn nhất

Cho đơn đồ thị có hướng $G = (V, E)$ với hàm trọng số $w : E \to R$ ($w(e)$ được gọi là độ dài hay trọng số của cạnh e).

Độ dài của đường đi $P = v_1 \to v_2 \to ... \to v_k$ là số:

$$w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

Đường đi ngắn nhất từ đỉnh u đến đỉnh v là đường đi có độ dài ngắn nhất trong số các đường đi nối u với v.

Độ dài của đường đi ngắn nhất từ u đến v còn được gọi là **Khoảng cách từ** u **tới** v và ký hiệu là $\delta(u, v)$.

Các dạng bài toán ĐĐNN

1. **Bài toán một nguồn một đích** : Cho hai đỉnh s và t, cần tìm đường đi ngắn nhất từ s đến t.
2. **Bài toán một nguồn nhiều đích** : Cho s là đỉnh nguồn, cần tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại.
3. **Bài toán mọi cặp** : Tìm đường đi ngắn nhất giữa mọi cặp đỉnh của đồ thị.

Ta thấy các bài toán được xếp theo thứ tự từ đơn giản đến phức tạp. Hễ có thuật toán hiệu

quả để giải một trong ba bài toán thì thuật toán đó cũng có thể sử dụng để giải hai bài toán còn lại.

Nếu đồ thị có chu trình âm thì độ dài đường đi giữa hai đỉnh nào đó có thể làm nhỏ tuỳ ý. Vậy để thực hiện bài toán tìm đường đi ngắn nhất ta giả thiết đồ thị không chứa chu trình âm.

Biểu diễn đường đi ngắn nhất : các thuật toán tìm đường đi ngắn nhất làm việc với hai mảng:

- $d(v)$: độ dài đường đi từ s đến v ngắn nhất hiện biết (cận trên cho độ dài đường đi ngắn nhất thực sự).
- $p(v)$: đỉnh đi trước v trong đường đi nói trên (sẽ sử dụng để truy ngược đường đi từ s đến v).

Giảm cận trên - Relaxation: sử dụng cạnh (u, v) để kiểm tra xem đường đi đến v đã tìm được có thể làm ngắn hơn nhờ đi qua u hay không. Các thuật toán tìm đđnn khác nhau ở số lần dùng các cạnh và trình tự duyệt chúng để thực hiện giảm cận.

Nhận xét chung

- Việc cài đặt các thuật toán được thể hiện nhờ **thủ tục gán nhãn**: Mỗi đỉnh v sẽ có nhãn gồm 2 thành phần $(d[v], p[v])$. Nhãn sẽ biến đổi trong quá trình thực hiện thuật toán.
- Nhận thấy rằng để tính khoảng cách từ s đến t, ở đây, ta phải tính khoảng cách từ s đến tất cả cách đỉnh còn lại của đồ thị.
- Hiện vẫn chưa biết thuật toán nào cho phép tìm đđnn giữa hai đỉnh làm việc thực sự hiệu quả hơn những thuật toán tìm đđnn từ một đỉnh đến tất cả các đỉnh còn lại.

2.2 Thuật toán Bellman-Ford

2.2.1 Ý tưởng thuật toán

Thuật toán Bellman-Ford tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị.

Thuật toán làm việc trong trường hợp trọng số của các cung là tùy ý. Giả thiết rằng đồ thị không có chu trình âm.

Đầu vào :

- Đồ thị $G = (V, E)$ với n đỉnh xác định bởi ma trận trọng số $W[u, v], u, v \in V$, đỉnh nguồn $s \in V$.

Đầu ra : Với mỗi $v \in V$

- $d[v] = \delta[s, v]$.
- $p[v]$ - Đỉnh đi trước v trong đđnn từ s đến v.

2.2.2 Cài đặt

Em thực hiện cài đặt thuật toán theo mã giả:

Algorithm 1 Bellman-Ford algorithm

```
1:  procedure BELLMAN-FORD
2:      for v ∈ V do                                          ▷ Khởi tạo
3:          d[v] ← w[s, v]
4:          p[v] ← s
5:      end for
6:      d[s] ← 0
7:      p[s] ← 0
8:      for k = 1 to n − 2 do
9:          for v ∈ V\{s} do
10:             for u ∈ V do
11:                 if d[v] > d[u] + w[u, v] then             ▷ Bước Relaxation
12:                     d[v] ← d[u] + w[u, v]
13:                     p[v] ← u
14:                 end if
15:             end for
16:         end for
17:     end for
18: end procedure
```

Tính đúng đắn của thuật toán có thể chứng minh trên cơ sở nguyên lý tối ưu của quy hoạnh động.

Độ phức tạp của thuật toán là $O(n^3)$

Đối với đồ thị thưa, sử dụng danh sách kề sẽ tốt hơn, khi đó vòng lặp theo u viết lại thành: **For** $u \in Ke(v)$ **do**

Trong trường hợp này ta thu được thuật toán với độ phức tạp $O(n.m)$

2.3 Thuật toán Dijkstra

2.3.1 Ý tưởng thuật toán

Trong trường hợp trọng số trên các cung là không âm, thuật toán do Dijkstra là hiệu quả hơn nhiều so với thuật toán Bellman-Ford.

Thuật toán được cây dựng dựa trên thủ tục gán nhãn. Đầu tiên nhãn của các đỉnh là tạm thời. Ở mỗi bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh u

trở thành cố định thì $d[u]$ sẽ cho ta độ dài của đđnn từ đỉnh s đến u. Thuật toán kết thúc khi nhãn của tất cả các đỉnh trở thành cố định.

Đầu vào :

- Đồ thị $G = (V, E)$ với n đỉnh xác định bởi ma trận trọng số $W[u, v], u, v \in V$, đỉnh nguồn $s \in V$.

Giả thiết: $w[u, v] \leqslant 0, u, v \in V$.

Đầu ra : Với mỗi $v \in V$

- $d[v] = \delta[s, v]$.
- $p[v]$ - Đỉnh đi trước v trong đđnn từ s đến v.

2.3.2 Cài đặt

Em thực hiện cài đặt thuật toán theo mã giả:

Algorithm 2 Dijkstra algorithm

```
procedure DIJKSTRA
    for v ∈ V do                                              ▷ Khởi tạo
        d[v] ← w[s, v]
        p[v] ← s
    end for
    d[s] ← 0
    S ← {s}                                                   ▷ S - Tập đỉnh có nhãn cố định
    T ← V\{s}                                                 ▷ T - Tập đỉnh có nhãn tạm thời
    while T ≠ ∅ do Tìm đỉnh u ∈ T thỏa mãn d[u] = min{d[z] : z ∈ T}
        T ← T\{u}
        S ← S ∪ {u}                                           ▷ Cố định nhãn của đỉnh u
        for v ∈ T do                                          ▷ Gán nhãn lại cho các đỉnh trong T
            if d[v] > d[u] + w[u, v] then                     ▷ Bước relaxation
                d[v] ← d[u] + w[u, v]
                p[v] ← u
            end if
        end for
    end while
end procedure
```

Thuật toán Dijkstra theo mã giả trên tìm được đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại trên đồ thị sau thời gian $\mathcal{O}(n^2)$

2.4 So sánh với thư viện có sẵn

Để kiểm tra tính đúng đắn và hiệu quả của thuật toán đã cài đặt, em sử dụng thư viện *Networkx* để so sánh kết quả và thời gian chạy.

2.4.1 Dữ liệu

Dữ liệu được tạo ngẫu nhiên từ thư viện *networkx*. Do việc tạo một đồ thị lớn có trọng số âm mà không có chu trình âm rất khó khăn nên em chỉ tạo đồ thị có trọng số dương.

Và em cũng thực hiện lưu đồ thị đã tạo ngẫu nhiên dưới dạng ma trận trọng số trên một mảng trên numpy để tiện kiểm tra thuật toán và tránh phải lưu trữ các file dữ liệu.

2.4.2 So sánh thời gian

Sử dụng 2 đồ thị được tạo ngẫu nhiên, em thực hiện tổng hợp thời gian chạy dưới bảng sau:

	100 đỉnh, 4000 cạnh	2000 đỉnh, 2.10^6 cạnh
Dijsktra	0.0068361759185791016	2.077575206756592
Thư viện: Dijkstra	0.005284309387207031	2.4850828647613525
Bellman-Ford	0.8282551765441895	Không chạy được
Thư viện: Bellman-Ford	0.0045166015625	2.7311718463897705

Bảng 2.1: So sánh thời gian chạy với thư viện(Đơn vị: giây(s))

2.4.3 Phân tích lý do

Thời gian chạy giữa thư viện và tự cài đặt còn chênh lệch nhiều, có thể do các lý do:

- Sử dụng cấu trúc dữ liệu để lưu trữ chưa tốt
- Sử dụng nhiều vòng lặp lồng nhau nên không được nhanh
- Đối với đồ thị thưa, sử dụng ma trận kề là không hiệu quả (đặc biệt với thuật toán Bellman-Ford)

Chương 3

THUẬT TOÁN CÂY QUYẾT ĐỊNH

3.1 Tổng quan thuật toán

3.1.1 Giới thiệu

Cây quyết định là một mô hình *supervised learning* trong *Machine Learning*, có thể được áp dụng vào cả hai bài toán *classification và regression*. Việc xây dựng một cây quyết định trên dữ liệu huấn luyện cho trước là việc 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, ... 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 cây quyết định ít yêu cầu việc chuẩn hoá dữ liệu.

3.1.2 Phân loại

Có 3 loại cây quyết định phổ biến sau:

- **Iterative Dichotomiser 3 (ID3)** - Tạo cây nhiều chiều, tìm cho mỗi node một đặc tính phân loại sao cho đặc 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.
- **Classification and Regression Trees (CART)** - 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 nút sử dụng một ngưỡng để đạt được "information gain" lớn nhất.

3.1.3 Ưu và nhược điểm của thuật toán

Tùy vào loại cây quyết định mà có ưu nhược điểm riêng. Nhưng nhìn chung thuật toán có những ưu nhược điểm chung như sau:

Về ưu điểm

- Cây quyết định thường mô phỏng cách suy nghĩ con người. Vì vậy mà đơn giản để hiểu và diễn giải dữ liệu.
- Giúp nhìn thấy được sự logic của dữ liệu.
- Có khả năng chọn được những đặc trưng 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

- Tỉ lệ tính toán tăng theo hàm số mũ.
- Dễ bị vấn đề overfitting [1] và underfitting [2] khi tập dữ liệu huấn luyện nhỏ.

Bài báo cáo này sử dụng loại **CART**. Do tính đơn giản và dễ tiếp cận cũng như những giá trị đặc trưng sử dụng là kiểu dữ liệu liên tục không phải phân loại nên không dùng *ID3* được. Và đây là loại cây quyết định được thư viện *scikit-learn* chọn sử dụng.

3.1.4 Một số khái niệm

Trước khi thực hiện xây dựng cây, ta cần giới thiệu một số hàm dùng để tính toán hiệu quả của từng bước phân chia dữ liệu trong quá trình xây dựng cây sau này.

1. **Mean Square Error - Sai số toàn phương trung bình**

   MSE là thước đo để đánh giá chất lượng của một ước lượng (ví dụ, một hàm toán học lập bản đồ mẫu dữ liệu của một tham số của dân số từ đó các dữ liệu được lấy mẫu) hoặc một yếu tố dự báo (ví dụ, một bản đồ chức năng có số liệu vào tùy ý để một mẫu của các giá trị của một số biến ngẫu nhiên). Định nghĩa của một MSE khác với những gì tương ứng cho dù là một trong những mô tả một ước lượng, hay một yếu tố dự báo.

   Nếu $\hat{Y}$ là một vector của n trị dự báo, và Y là vector các trị quan sát được, tương ứng với ngõ vào của hàm số phát ra dự báo, thì MSE của phép dự báo có thể ước lượng

[1] **Overfitting** là hiện tượng mô hình tìm được quá khớp với dữ liệu huấn luyện dẫn đến sự nhầm nhiễu.
[2] **Underfitting** là hiện tượng mô hình tìm được khác xa so với thực tế.

theo công thức:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Tức là MSE là trung bình $(\frac{1}{n}\sum_{i=1}^{n})$ của bình phương các sai số $((Y_i - \hat{Y}_i)^2)$. Đây là định lượng dễ dàng tính được cho một mẫu cụ thể (và do đó phụ thuộc mẫu).

2. **Mean Absolute Error - Sai số tuyệt đối trung bình**

Trong thống kê, **MSA**(sai số tuyệt đối trung bình) là thước đo sai số giữa các cặp giá trị cùng quan sát đối với cùng một hiện tượng. Ví dụ về sự tương quan giữa Y và X là giữa các giá trị dự đoán so với các giá trị quan sát được, giữa thời gian ban đầu so với thời gian sau khi xảy ra hiện tương hay giữa một phương pháp đo lường này với một phương pháp đo lường khác. Khi đó, MAE được tính bằng tổng sai số tuyệt đối chia cho cỡ mẫu của bộ dữ liệu:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

Trong đó y_i là giá trị dự đoán và x_i là giá trị đúng. Do đó **MAE** là trung bình của các giá trị sai số tuyệt đối $|y_i - x_i|$ của dữ liệu.

3. **Regression criterion**

Sau khi định nghĩa 2 hàm *MSE* và *MSA* như trên, tiếp theo ta sử dụng 2 hàm trên để định nghĩa hàm đo độ tinh khiết giữa cách bước phân chia của cây hồi quy.

Đối với bài toán hồi quy thì kết quả đầu ra là giá trị liên tục, vậy với nút m, ta tính giá trị *MSE* và *MSA* của nút đó như sau:

Min Squared Error(MSE)

$$\overline{y}_m = \frac{1}{N_m}\sum_{y \in Q_m} y$$

$$H(Q_m) = \frac{1}{N_m}\sum_{y \in Q_m}(y - \overline{y}_m)^2$$

Mean Absoluted Error(MAE) (chậm hơn nhiều so với MSE)

$$median(y)_m = \underset{y \in Q_m}{median(y)}$$

$$H(Q_m) = \frac{1}{N_m}|y - median(y)_m|$$

Cuối cùng, ta có *hàm tinh khiết*(*Impurity function*) được định nghĩa:

$$G_{Q_m,\theta} = H(Q_m) - (\frac{N_m^{left}}{N_m} H(Q_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(Q_m^{right}(\theta))) \tag{3.1}$$

4. **Entropy - Độ hỗn loạn**

   Entropy(độ hỗn loạn) 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 công thức tổng quát để tính giá trị entropy như sau:

$$E(\mathcal{S}) = -\sum_{i=1}^{n} p_i \log(p_i) \tag{3.2}$$

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

5. **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:

$$\mathbf{Gini}(\mathcal{S}) = 1 - \sum_{i=1}^{n} p_i^2 \tag{3.3}$$

   Với $\mathcal{S}$ là tập các dữ liệu, p_i là xác xuất 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 điều chung một nhãn.

6. **Classification criterion**

   Tương tự như hàm tinh khiết của bài toán hồi quy, với bài toán phân loại ta cũng có hàm tinh khiết tương tự:

   Bài toán phân loại có đầu ra là giá trị thuộc một trong K lớp, ở nút lá m, xác xuất của lớp chiếm đa số là:

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{3.4}$$

   Sử dụng công thức trên, ta có công thức tính giá trị *Entropy* và *Gini* ở các nút là:

Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \tag{3.5}$$

Entropy

$$H(Q_m) = -\sum_k p_{mk} log_2(p_{mk}) \tag{3.6}$$

Từ đó, ta có hàm tinh khiết cho bài toán phân loại là:

$$G_{Q_m,\theta} = H(Q_m) - (\frac{N_m^{left}}{N_m} H(Q_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(Q_m^{right}(\theta))) \tag{3.7}$$

3.1.5 Quá trình xây dựng cây

Cho N bộ dữ liệu $(x_i, y_i), i \in [1, 2, ...N]$ với:

- **Dữ liệu huấn luyện** : $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$, p là số thuộc tính(features) của từng điểm dữ liệu.
- **Nhãn** : $y \in R^N$

Thuật toán cây quyết định phân chia đệ quy các thuộc tính của dữ liệu sao cho các dữ liệu có nhãn giống nhau được gộp chung lại. Cây chia bộ dữ liệu thành M vùng $R_1, R_2, ..., R_M$.

Mỗi điểm dữ liệu sẽ được dự đoán như sau:

$$f(x) = \sum_i^M c_m I(x \in R_m)$$

Với bài toán hồi quy (Regression), c_m được tính là trung bình của giá trị ở nút lá của tất cả điểm dữ liệu. Với bài toán phân loại(Classification), c_m được tính dựa vào lớp có lượng đông nhất ở các nút lá.

Thuật toán cây quyết định chia dữ liệu thành các vùng để xây dựng nên cây bằng cách sử dụng thuật toán tham lam để chọn các bước chia tốt nhất. Ở mỗi bước chia, ta duyệt qua tất cả điểm dữ liệu và tất cả giá trị của dữ liệu để tìm ra cách chia tốt nhất.

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

1. Giả sử dữ liệu tại nút m được biểu diễn là Q_m và có N_m điểm dữ liệu tại nút đó. Với mỗi *attribute / feature* j, ta phân ra các ngưỡng T_m, từ các ngưỡng đó thực hiện chia dữ liệu thành các tập dữ liệu, trong mỗi tập dữ liệu bao gồm:
    - Set 1 $Q_m^{left}(\theta) = ((x, y)|x_j <= t_m)$

- Set 2 $Q_m^{right}(\theta) = Q_m \, Q_m^{left}(\theta)$

2. Chọn ngưỡng T_m sao cho các tập dữ liệu trở nên càng "tinh khiết" về mặt nhãn / lớp (*label* / *class*) càng tốt. Ta dùng *Hàm tinh khiết*(*impurity function*) đã được định nghĩa ở trên để đo độ tinh khiết(giá trị của hàm tinh khiết càng lớn nghĩa là độ tinh khiết của cây trước và sau khi chia là càng cao, nghĩa là phép chia của cây cho độ tinh khiết lớn) và tìm bước phân chia cây tốt nhất. Tùy thuộc vào bài toán là Regression hay Classification mà ta chọn các hàm để xây dựng hàm tinh khiết cho phù hợp.

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

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 dẫn khi gặp dữ liệu mới, dữ liệu chưa được học có thể quyết định sai mặc dù quá trình xây dựng cây có 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.
- Dừng khi độ sai số trên tập kiểm thử không giảm.
- Giới hạn phần tử nhỏ nhất trong lá.

3.2 Cài đặt thuật toán

Em đã thực hiện trực tiếp cài đặt thuật toán trên ngôn ngữ Python theo các bước và ý tưởng như đã trình bày ở trên và có gửi kèm mã nguồn vào một file riêng để dễ dàng chạy thử và kiểm tra kết quả. Về dữ liệu, em sử dụng bộ dữ liệu *titanic* trên trang *Kaggle*. Trước khi có thể dùng thuật toán để dự đoán, bộ dữ liệu đã được làm sạch, phân tích sử dụng các thư viện đã tìm hiểu ở chương 1. Bảng dưới là bảng so sánh thời gian chạy giữa 2 thuật toán(do giới hạn thời gian nên em chỉ kiểm tra với bài toán phân loại, và kết quả giữa tự cài đặt và thư viện giống nhau)

	Hàm entropy	**Hàm gini**
Tự cài đặt	0.31322360038757324	0.33696794509887695
Thư viện	0.0029256343841552734	0.003991127014160156

Bảng 3.1: So sánh thời gian chạy giữa tự cài đặt và thư viện (đơn vị: giây(s))

KẾT LUẬN

Python là một ngôn ngữ đơn giản dễ đọc và học. Nó không có cú pháp phức tạp như một số ngôn ngữ khác. Điều này tạo thêm cho Python lợi thế để sửa lỗi chương trình. Python có khả năng ứng dụng trong nhiều lĩnh vực.

Qua quá trình thực hiện Project, bản than em đã học tập cũng như rèn luyện được nhiều kĩ năng:

- Kỹ năng lập kế hoạch chi tiết một dự án nhỏ, kỹ năng quản lý thời gian và làm quen với việc hoàn thành deadline.
- Tìm hiểu thêm về Python và một số thư viện của nó. Cách để tự học và tìm hiểu một ngôn ngữ lập trình mới.
- Tìm hiểu và lập trình các thuật toán về đồ thị, so sánh và cải tiến làm cho thuật toán hiệu quả hơn.
- Kỹ năng viết báo cáo kĩ thuật ngắn.

Cuối cùng em xin cảm ơn thầy đã hướng dẫn và hỗ trợ em hoàn thành môn học một cách tốt nhất

Tài liệu tham khảo

[1] V. H. Tiệp, "Phân nhóm các thuật toán Machine Learning," 2016. **url**: https://goo.gl/S6Nwoy.

[2] B. Ricaud, "A simple explanation of entropy in decision trees," 2017. **url**: https://goo.gl/ogrMX6.

[3] Scikit-learn, "Decision Trees," **url**: https://scikit-learn.org/stable/modules/tree.html#.

[4] Coursera, "Machine Learning - Classification," **url**: https://www.coursera.org/learn/ml-classification.

[5] D. Institute, "Gini Index Explained," **url**: https://goo.gl/4h3GK8.

[6] C. M.Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] R. T.Hastie **and** J.Friedman, *Elements of Statistical Learning*. Springer, 2009.

[8] W3Schools, "Python tutorial," **url**: https://www.w3schools.com/python/python_classes.asp.