# **LỜI MỞ ĐẦU**

Học máy (Machine Learning) là một lĩnh vực của trí tuệ nhân tạo (Artificial Intelligence - AI). Các thuật toán học máy cho phép máy tính đào tạo đầu vào dữ liệu và sử dụng phân tích thống kê để đưa ra các giá trị nằm trong một phạm vi cụ thể.

Ngày nay, những người sử dụng công nghệ đều được hưởng lợi từ việc học máy. Công nghệ nhận diện khuôn mặt giúp người dùng gắn thẻ và chia sẻ ảnh của bạn bè. Công nghệ nhận dạng ký tự quang học (OCR) chuyển đổi hình ảnh văn bản sang dạng di chuyển. Khi mà khả năng tính toán của máy tính được nâng lên một tầm cao mới cùng với lượng dữ liệu khổng lồ được thu thập, Machine Learning đã tiến thêm một bước dài và Deep Learning (DL) một lĩnh vực mới được ra đời.

Deep Learning được xây dựng từ mạng nơ ron sinh học và bao gồm nhiều lớp trong mạng nơ ron nhân tạo được tạo thành từ phần cứng và GPU. Deep Learning sử dụng một tầng các lớp đơn vị xử lý phi tuyến để trích xuất hoặc chuyển đổi các tính năng (hoặc biểu diễn) của dữ liệu. Đầu ra của một lớp phục vụ như là đầu vào của lớp kế tiếp. Deep learning tập trung giải quyết các vấn đề liên quan đến mạng thần kinh nhân tạo nhằm nâng cấp các công nghệ như nhận diện giọng nói, dịch tự động (machine translation), xử lý ngôn ngữ tự nhiên…

Trong số các thuật toán học máy hiện đang được sử dụng và phát triển, học sâu thu hút được nhiều nghiên cứu nhất và có thể đánh bại con người trong một số nhiệm vụ nhận thức. Do những đặc tính nổi bật và kết quả tối ưu, học tập sâu đã trở thành phương pháp tiếp cận được nghiên cứu và ứng dụng trong giải quyết nhiều bài toán thuộc lĩnh vực trí tuệ nhân tạo.

Chính vì vậy bọn em chọn đề tài:” Kỹ thuật BiLSTM, cài đặt của tensorflow và ứng dụng”….

# **CHƯƠNG 1: Kỹ thuật BiLSTM, cài đặt của tensorflow và ứng dụng**

## **1.Học sâu – Deep Learning**

### **1.1Mạng nơ-ron hồi quy RNN (Recurrent Neural Network)**

Mạng nơ ron hồi quy RNN (Recurrent Neural Network) được giới thiệu bởi John Hopfield năm 1982, là một trong những mô hình học sâu - Deep learning. Recurrent có nghĩa là thực hiện lặp lại cùng một tác vụ cho mỗi thành phần trong chuỗi. Trong đó, kết quả đầu ra tại thời điểm hiện tại phụ thuộc vào kết quả tính toán của các thành phần ở những thời điểm trước đó.

RNN là một mô hình có trí nhớ (memory), có khả năng nhớ được thông tin đã tính toán trước đó. Không như các mô hình Neural Network truyền thống trước đó là thông tin đầu vào (input) hoàn toàn độc lập với thông tin đầu ra (output).

Hầu hết RNN được thiết kế như là một chuỗi các module được lặp đi lặp lại, các môdun này thường có cấu trúc đơn giản chỉ có một lớp mạng tanh. Huấn luyện RNN tương tự như huấn luyện ANN truyền thống. Giá trị tại mỗi output không chỉ phụ thuộc vào kết quả tính toán của bước hiện tại mà còn phụ thuộc vào kết quả tính toán của các bước trước đó.

##### **Hình 1.1: Quá trình xử lý thông tin trong mạng RNN**

RNN có khả năng biểu diễn mối quan hệ phụ thuộc giữa các thành phần trong chuỗi (nếu chuỗi đầu vào có 6 từ thì RNN sẽ dàn ra thành 6 layer, mỗi layer ứng với mỗi từ, chỉ số mỗi từ được đánh từ 0 đến 5. Trong Hình ở trên, xt là input tại thời điểm thứ t, st là hidden state(memory) tại thời điểm thứ t, được tính dựa trên các hidden state trước đó kết hợp với input của thời điểm hiện tại với công thức:

𝑆𝑡 = tanh (𝑈𝑥𝑡 + 𝑊𝑠𝑡−1)

St-1 là hidden state được khởi tạo là 1 vector 0. Ot là output tại thời điểm thứ t, là một vector chứa xác xuất của toàn bộ các từ trong từ điển.

𝑂𝑡 = softmax(𝑉𝑠𝑡)

Không như ANN truyền thống, tại mỗi layer cần phải sử dụng một tham số khác, RNNs chỉ sử dụng một bộ parameters (U, V, W) cho toàn bộ các bước.

Ý tưởng ban đầu của RNN là kết nối những thông tin trước đó nhằm hỗ trợ cho các xử lý hiện tại. Nhưng đôi khi, chỉ cần dựa vào một số thông tin gần nhất để thực hiện tác vụ hiện tại. Ví dụ, chúng ta dự đoán từ cuối cùng trong câu “chuồn\_chuồn bay thấp thì mưa”, thì chúng ta không cần truy tìm quá nhiều từ trước đó, ta có thể đoán ngay từ tiếp theo sẽ là “mưa”. Trong trường hợp này, khoảng cách tới thông tin liên quan được rút ngắn lại, mạng RNN có thể học và sử dụng các thông tin quá khứ.

##### **Hình 1.2: RNN phụ thuộc short-term**

Trường hợp có nhiều thông tin hơn trong một câu, nghĩa là phụ thuộc vào ngữ cảnh. Ví dụ nhưng khi dự đoán từ cuối cùng trong đoạn văn bản “Tôi sinh ra và lớn lên ở Việt\_Nam … Tôi có\_thể nói thuần\_thục Tiếng\_Việt.” Từ thông tin gần nhất cho thấy rằng từ tiếp theo là tên một ngôn ngữ, nhưng khi chúng ta muốn biết cụ thể ngôn ngữ nào, thì cần quay về quá khứ xa hơn, để tìm được ngữ cảnh Việt\_Nam. Và như vậy, RNN có thể phải tìm những thông tin có liên quan và số lượng các điểm đó trở nên rất lớn.

Không được như mong đợi, RNN không thể học để kết nối các thông tin lại với nhau.

##### **Hình 1.3: RNN phụ thuộc long-term**

Về lý thuyết, RNN có thể nhớ được thông tin của chuỗi có chiều dài bất kì, nhưng trong thực tế mô hình này chỉ nhớ được thông tin ở vài bước trước đó.

RNN có các phiên bản mở rộng như: Bidirectional RNN (RNN hai chiều), Deep (Bidirectional) RNN, Long short-term memory networks (LSTM).

##### **Hình 1.4: Bidirectional RNN**

##### **Hình 1.5: Deep (Bidirectional) RNN**

### **1.2Mạng nơ-ron dài ngắn song song (BiLTSM)**

a. Giới thiệu

Một hướng tiếp cận với dữ liệu khác nữa, đó là sử dụng hai mạng nơ ron hồi quy theo hai chiều ngược nhau để xử lý (Bidirectional RNN). Một đơn vị RNN sẽ làm như thường lệ, tức là ta sẽ dùng nó để học các tín hiệu đầu vào từ thời điểm ban đầu tới thời điểm kết thúc (đi xuôi). Còn đơn vị RNN còn lại, ta sẽ đọc theo thứ tự thời điểm từ kết thúc trở lại ban đầu (đi ngược). Sau khi có cả hai kết quả, chúng sẽ được gom lại thành một để có thể dự đoán. Với ý tưởng như vậy, tại một thời điểm bất kỳ, mạng sẽ có được các thông tin trước và sau thời điểm ấy.

Do bản chất LSTM là cải tiến của RNN, cho nên ta có thể áp dụng nó và biến nó thành mạng nơ ron dài ngắn song song (BiLSTM). Mỗi LSTM sẽ vẫn có khả năng quên thông tin cũ (cổng quên), lọc thông tin mới (cổng đầu vào), hoặc giấu bớt kết quả (cổng đầu ra) như bình thường. Chính vì vậy, các thông tin từ quá khứ tới tương lai của mạng BiLSTM đều có thể tự học để tự điều chỉnh. Dẫn tới việc với các bài toán mà ta cần biết nhiều hơn về ngữ cảnh hiện tại của nó, thì mạng BiLSTM cho kết quả tốt hơn.

##### **Hình 1.6: Mạng Bi-RNN (có thể thế bằng BiLSTM) sau khi được “bung ra”.**

Ta thấy đơn vị mạng chính là mạng đi xuôi, và đơn vị mạng chính là mạng đi ngược.

b. Cách dự đoán kết quả của mạng BiLSTM

Như ý tưởng trên, ta sẽ có kết quả của 2 đơn vị mạng khác nhau (một cái chạy xuôi, và một cái chạy ngược). Chính vì vậy để có kết quả cuối cùng, ta phải gom được hai kết quả trên về một cái thống nhất. Một số cách đơn giản để làm điều này đó là:

● Tổng: Cộng từng tín hiệu đầu ra của hai mạng

● Tích: Nhân từng tín hiệu đầu ra của hai mạng

● Ghép: Hai vectơ tín hiệu đầu ra của hai mạng được ghép tiếp nhau (vectơ kết quả cuối sẽ có số chiều gấp đôi)

● Trung bình cộng: Trung bình cộng từng tín hiệu đầu ra của hai mạng

● Trung bình nhân: Trung bình nhân từng tín hiệu đầu ra

Thông thường, tùy bài toán mà ta dùng cách khác nhau. Tuy vậy, cách ghép hai vectơ là phổ biến nhất.

## **12.Tổng quan và cách cài đặt của Tensorflow**

### **2.1Giới thiệu**

Tensorflow là một thư viện mã nguồn mở do nhóm Google Drain nghiên cứu và phát triển. Sau đó được phát hành theo giấy phép mã nguồn mở Apache 2.0 vào ngày 9/11/2015. Nó được sử dụng nhiều trong xử lý trí tuệ nhân tạo – AI, machine learning,…

Tensorflow hỗ trợ CPU, GPU và đặt biệt là TPU – (Tensor Processing Unit : bộ xử lý Tensor được Google nghiên cứu và phát triển nhằm tối ưu cho machine learning nói riêng và xử lý trí tuệ nhân tạo nói chung). Nền tảng Google Cloud Platform đã hỗ trợ đầy đủ cho Tensorflow với cả CPU, GPU và TPU.

Tensorflow hỗ trợ cho cả PC và mobile, đồng thời nó hỗ trợ một số ngôn ngữ lập trình như ở bên dưới:

· Python

· C/C++

· Java

· Go

· JavaScript

Trong những ngôn ngữ lập trình mà Tensorflow hỗ trợ thì Python sẽ có tính tương thích cao nhất và có cả tương thích ngược giữa các phiên bản. Có nghĩa là code được implement với Tensorlfow 2.x có thể chạy tốt trên Tensorflow 1.x. Còn các ngôn ngữ còn lại thì không tương thích ngược.

### **2.2Cài đặt**

Bạn vào trang chủ của Python và chọn phiên bản Python 3.5 trở lên để tải về và cài đặt. Bởi vì hiện tại thì Tensorflow 2 chỉ hỗ trờ từ phiên bản Python 3.5 trở lên.

##### **Hình 1.6: Phiên bản tensorflow**

Sau khi đã cài xong Python thì mở command line lên gõ:

##### **Hình 1.7: Cài đặt tensorflow**

## **3.Bài toán phân loại trình tự sử dụng BiLSTM**

VD: Bài toán phân loại một tính năng cho mỗi bước chấm công

Đầu tiên chúng ta định nghĩa số bước chấm công

Sau đó chúng ta phải cập nhật hàm get\_sequence () để định hình lại các trình tự đầu vào và đầu ra thành 3 chiều để đáp ứng các mong đợi của LSTM.

Tiếp theo, chúng ta định nghĩa một LSTM. Lớp đầu vào sẽ có 10 bước thời gian với 1 tính năng là một mảnh, input\_shape = (10, 1).

Lớp ẩn đầu tiên sẽ có 20 đơn vị bộ nhớ, được bao bọc bằng một lớp 2 chiều và lớp đầu ra sẽ là một lớp được kết nối đầy đủ, xuất ra một giá trị trên mỗi bước thời gian. Một hàm kích hoạt sigmoid được sử dụng trên đầu ra để dự đoán giá trị nhị phân.

Một lớp bao bọc TimeDistributed được sử dụng xung quanh lớp đầu ra để có thể dự đoán một giá trị trên mỗi bước thời gian dựa trên trình tự đầy đủ được cung cấp làm đầu vào. Điều này yêu cầu lớp ẩn LSTM trả về một chuỗi giá trị (một giá trị mỗi bước) thay vì một giá trị duy nhất cho toàn bộ chuỗi đầu vào.

Cuối cùng, bởi vì đây là một vấn đề phân loại nhị phân, nên hàm mất mát nhị phân (binary\_crossentropy trong Keras) được sử dụng. Thuật toán tối ưu hóa ADAM hiệu quả được sử dụng để tìm trọng số và chỉ số độ chính xác được tính toán và báo cáo theo từng epoch

LSTM sẽ được huấn luyện trong 1.000 epoch. Một chuỗi đầu vào ngẫu nhiên mới sẽ được tạo mỗi kỷ nguyên để mạng phù hợp. Điều này đảm bảo rằng mô hình không ghi nhớ một chuỗi đơn lẻ và thay vào đó có thể tổng quát hóa một giải pháp để giải quyết tất cả các chuỗi đầu vào ngẫu nhiên có thể có cho vấn đề này.

Sau khi được đào tạo, mạng sẽ được đánh giá trên một chuỗi ngẫu nhiên khác. Các dự đoán sau đó sẽ được so sánh với chuỗi đầu ra dự kiến ​​để cung cấp một ví dụ cụ thể về kỹ năng của hệ thống.

Code đầy đủ:

# **CHƯƠNG 2: Bài toán học máy**

## **1 Bài toán phân loại bệnh cây đậu nành sử dụng kỹ thuật SVM**

### **1.1 Nguồn dữ liệu**

- Nguồn : https://archive.ics.uci.edu/

-Bộ dữ liệu: soybean-large.csv

-Gồm 19 lớp ,35 thuộc tính ,307 dòng

+35 thuộc tính gồm:

1. date

2. plant-stand

3. precip

4. temp

5. hail

6. crop-hist

7. area-damaged

8. severity

9. seed-tmt

10. germination

11. plant-growth

12. leaves

13. leafspots-halo

14. leafspots-marg

15. leafspot-size

16. leaf-shread

17. leaf-malf

18. leaf-mild

19. stem

20. lodging

21. stem-cankers

22. canker-lesion

23. fruiting-bodies

24. external decay

25. mycelium

26. int-discolor

27. sclerotia

28. fruit-pods

29. fruit spots

30. seed

31. mold-growth

32. seed-discolor

33. seed-size

34. shriveling

35. roots

+19 Lớp :

diaporthe-stem-canker: 10

charcoal-rot: 10

rhizoctonia-root-rot: 10

phytophthora-rot : 40

brown-stem-rot: 20

powdery-mildew: 10

downy-mildew: 10

brown-spot: 40

bacterial-blight: 10

bacterial-pustule: 10

purple-seed-stain: 10

anthracnose: 20

phyllosticta-leaf-spot:10

alternarialeaf-spot: 40

frog-eye-leaf-spot: 40

diaporthe-pod-&-stem-blight: 6

cyst-nematode: 6

herbicide-injury: 4

Tổng :307

### **1.2 Bài toán học máy**

Bài toán phân loại bệnh cây đậu nành sử dụng kỹ thuật SVM

Phương pháp đánh giá: Đánh giá mô hình học dựa trên kết quả dự đoán (với độ đo đơn giản Accuracy

#### **1.2.1 Xử lý dữ liệu**

-Bộ dữ liệu: soybean-large.csv

Dữ liệu mất mát : Có

Dùng hàm dropna() để xử lý :

+Hàm Dropna(***axis=0*, *how='any'*, *thresh=None*, *subset=None*, *inplace=False***):

Loại bỏ các giá trị bị thiếu.

**Thông số**

***Axis:*** {0 hoặc 'chỉ mục', 1 hoặc 'cột'}, mặc định là 0

-Xác định xem các hàng hoặc cột chứa các giá trị bị thiếu có bị xóa hay không.

-0, hoặc 'chỉ mục': Bỏ các hàng chứa các giá trị bị thiếu.

-1 hoặc 'cột': Bỏ các cột chứa giá trị bị thiếu.

***How:*** {‘any’, ‘all’}, mặc định ‘any’

-Xác định xem hàng hoặc cột có bị xóa khỏi DataFrame không khi chúng ta có ít nhất một NA hoặc tất cả NA.

-' any': Nếu có bất kỳ giá trị NA nào, hãy bỏ hàng hoặc cột đó.

-'all': Nếu tất cả các giá trị là NA, hãy bỏ hàng hoặc cột đó.

***Thresh:*** int, tùy chọn

-Yêu cầu nhiều giá trị không phải NA.

**Subset: *array-like, optional***

**-**Các nhãn dọc theo trục khác để xem xét, ví dụ: nếu bạn đang bỏ hàng, đây sẽ là danh sách các cột cần bao gồm.

**Inplace: *bool, default False***

**-**Nếu Đúng, thực hiện thao tác tại chỗ và trả về None.

Code : df = df.dropna()

Kết quả

+Dữ liệu trước xử lý :307

+Sau khi xử lý : 266

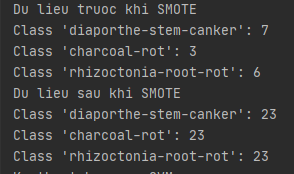


-Cân bằng dữ liệu SMOTE:

sm = SMOTE(random\_state = 2,k\_neighbors=1)

X\_train\_res, y\_train\_res = sm.fit\_sample(X\_train, y\_train)

+Kết quả



#### **1.2.2 Huấn luyện**

-Sử dụng kỹ thuật học máy SVM:

**THUẬT TOÁN SVM**

**-Giới thiệu**

Bài toán phân lớp (*Classification*) và dự đoán (*Prediction*) là hai bài toán cơ bản và có rất nhiều ứng dụng trong tất cả các lĩnh vực nhơ: học máy, nhận dạng, trí tuệ nhân tạo, .v.v . Trong khóa luận này, chúng em sẽ đi sâu nghiên cứu phương pháp Support Vector Machines (SVM), một phương pháp rất hiệu quả hiện nay.

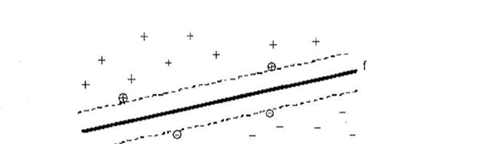
Phương pháp SVM được coi là công cụ mạnh cho những bài toán phân lớp phi tuyến tính được các tác giả Vapnik và Chervonenkis phát triển mạnh mẽ năm 1995. Phương pháp này thực hiện phân lớp dựa trên nguyên lý Cực tiểu hóa Rủi ro có Cấu trúc SRM (*Structural Risk Minimization*), được xem là một trong các phương pháp phân lớp giám sát không tham số tinh vi nhất cho đến nay. Các hàm công cụ đa dạng của SVM cho phép tạo không gian chuyển đổi để xây dựng mặt phẳng phân lớp

**- Định nghĩa**

Là phương pháp dựa trên nền tảng của lý thuyết thống kê nên có một nền tảng toán học chặt chẽ để đảm bảo rằng kết quả tìm được là chính xác

Là thuật toán học giám sát (*supervied learning*) được sử dụng cho phân lớp dữ liệu.

Là 1 phương pháp thử nghiệm, đưa ra 1 trong những phương pháp mạnh và chính xác nhất trong số các thuật toán nổi tiếng về phân lớp dữ liệu



SVM là một phương pháp có tính tổng quát cao nên có thể được áp dụng cho nhiều loại bài toán nhận dạng và phân loại

**- Ý tưởng của phương pháp**

Cho trước một tập huấn luyện, được biểu diễn trong không gian vector, trong đó mỗi tài liệu là một điểm, phương pháp này tìm ra một siêu phẳng quyết định tốt nhất có thể chia các điểm trên không gian này thành hai lớp riêng biệt tương ứng là lớp + và lớp -. Chất lượng của siêu phẳng này được quyết định bởi khoảng cách (gọi là biên) của điểm dữ liệu gần nhất của mỗi lớp đến mặt phẳng này. Khi đó, khoảng cách biên càng lớn thì mặt phẳng quyết định càng tốt, đồng thời việc phân loại càng chính xác.

Mục đích của phương pháp SVM là tìm được khoảng cách biên lớn nhất, điều này được minh họa như sau:

**-Nội dung phương pháp**

+Cơ sở lý thuyết

|  |  |
| --- | --- |
|  |  |
|  |  |

SVM thực chất là một bài toán tối ưu, mục tiêu của thuật toán này là tìm được một không gian F và siêu phẳng quyết định f trên F sao cho sai số phân loại là thấp nhất.

Cho tập mẫu (x1,y1), (x2, y2), … (xf, yf )} với xi  Rn , thuộc vào hai lớp nhãn: yi {-1,1} là nhãn lớp tương ứng của các xi (-1 biểu thị lớp I, 1 biểu thị lớp

II).

Ta có, phương trình siêu phẳng chứa vectơ xitrong không gian:

xi .w + b = 0

+1, Xi . W + b > 0

Đặt f(Xi) = sign (Xi . W + b) =

-1, Xi . W + b < 0

Như vậy, f(Xi) biểu diễn sự phân lớp của Xivào hai lớp như đã nêu. Ta nói yi= +1 nếu Xi € lớp I và yi = -1 nếu Xi€ lớp II . Khi đó, để có siêu phẳng f ta sẽ phải giải bài toán sau:

ur

Tìm min *w* với W thỏa mãn điều kiện sau:

yi(sin (Xi.W + b)) ≥ 1 với i € 1,n

Bài toán SVM có thể giải bằng kỹ thuật sử dụng toán tử Lagrange để biến đổi về thành dạng đẳng thức. Một đặc điểm thú vị của SVM là mặt phẳng quyết định chỉ phụ thuộc các Support Vector và nó có khoảng cách đến mặt phẳng quyết

ur

định là 1/ *w* . Cho dù các điểm khác bị xóa đi thì thuật toán vẫn cho kết quả giống như ban đầu. Đây chính là điểm nổi bật của phương pháp SVM so với các phương pháp khác vì tất cả các dữ liệu trong tập huấn luyện đều được dùng để tối ưu hóa kết quả.

***TÓM LẠI:*** trong trương hợp nhị phân phân tách tuyến tính, việc phân lớp được thực hiện qua hàm quyết định *f(x) = sign(<w.x> + b),* hàm này thu được bằng việc thay đổi vectơ chuẩn *w*, đây là vectơ để cực đại hóa viền chức năng

Việc mở rộng SVM để phân đa lớp hiện nay vẫn đang được đầu tư nghiên cứu. Có một phương pháp tiếp cận để giải quyết vấn để này là xây dựng và kết hợp nhiều bộ phân lớp nhị phân SVM (Chẳng hạn: trong quá trình luyện với SVM, bài toán phân m lớp có thể được biến đổi thành bài toán phân 2\*m lớp, khi đó trong mỗi hai lớp, hàm quyết định sẽ được xác định cho khả năng tổng quát hóa tối đa). Trong phương pháp này có thể đề cập tới hai cách là *một-đổi-một, một-đối-tất cả*

+Bài toán nhiều phân lớp với SVM

Để phân nhiều lớp thì kỹ thuật SVM nguyên thủy sẽ chia không gian dữ liệu thành 2 phần và quá trình này lặp lại nhiều lần. Khi đó hàm quyết định phân dữ liệu vào lớp thứ i của tập n , 2-Iớp sẽ là:

*fi(x)* = wiix + bi

Những phần tử *x* là support vector sẽ thỏa điều kiện

+1 nếu thuộc lớp i

*fi (x)* =

-1 nếu thuộc phần còn lại

Như vậy, bài toán phân nhiều lớp sử dụng phương pháp SVM hoàn toàn có thể thực hiện giống như bài toán hai lớp. Bằng cách sử dụng chiến lược *"một- đối- một”*(*one - against - one*).

Giả sử bài toán cần phân loại có k lớp (k > 2), chiến lược *"một-đối-một”*sẽ tiến hành k(k-l)/2 lần phân lớp nhị phân sử dụng phương pháp SVM. Mỗi lớp sẽ tiến hành phân tách với k-1 lớp còn lại để xác định k-1 hàm phân tách dựa vào bài toán phân hai lớp bằng phương pháp SVM.

**+**Các bước chính của phương pháp SVM

Phương pháp SVM yêu cầu dữ liệu được diễn tả nhớ các vector của các số thực. Như vậy nếu đầu vào chưa phải là số thì ta cần phải tìm cách chuyển chúng về dạng số của SVM

Tiền xử lý dữ liệu: Thực hiện biến đổi dữ liệu phù hợp cho quá trình tính toán, tránh các số quá lớn mô tả các thuộc tính. Thương nên co giãn (*scaling*) dữ liệu để chuyển về đoạn [-1, 1] hoặc [0, 1].

Chọn hàm hạt nhân: Lựa chọn hàm hạt nhân phù hợp tương ứng cho từng bài toán cụ thể để đạt được độ chính xác cao trong quá trình phân lớp.

Thực hiện việc kiểm tra chéo để xác định các tham số cho ứng dụng. Điều này cũng quyết định đến tính chính xác của quá trình phân lớp.

Sử dụng các tham số cho việc huấn luyện với tập mẫu. Trong quá trình huấn luyện sẽ sử dụng thuật toán tối ưu hóa khoảng cách giữa các siêu phẳng trong quá trình phân lớp, xác định hàm phân lớp trong không gian đặc trưng nhờ việc ánh xạ dữ liệu vào không gian đặc trưng bằng cách mô tả hạt nhân, giải quyết cho cả hai trường hợp dữ liệu là phân tách và không phân tách tuyến tính trong không gian đặc trưng.

Code:

from sklearn.svm import SVC

model2 = SVC().fit(X\_train,y\_train)

### **1.3 Đánh giá**

*-accuracy* (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.

-Với bài toán phân loại mà tập dữ liệu của các lớp là chênh lệch nhau rất nhiều, có một phép đó hiệu quả thường được sử dụng là Precision-Recall.

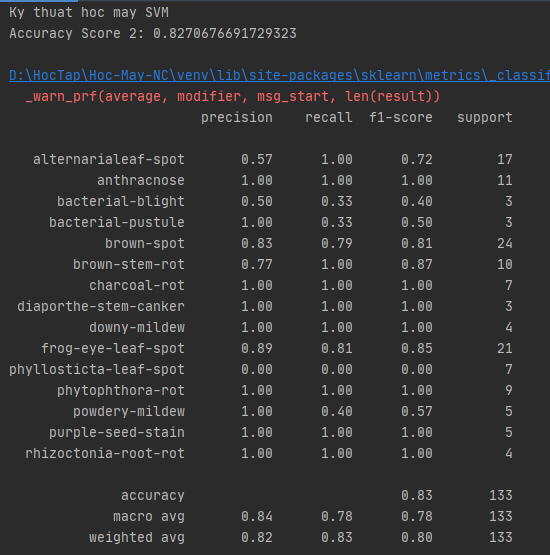
+**Precision** được định nghĩa là tỉ lệ số điểm **true positive** trong số những điểm **được phân loại là *positive*** (TP + FP).

**+Recall** được định nghĩa là tỉ lệ số điểm **true positive** trong số những điểm **thực sự là *positive*** (TP + FN).

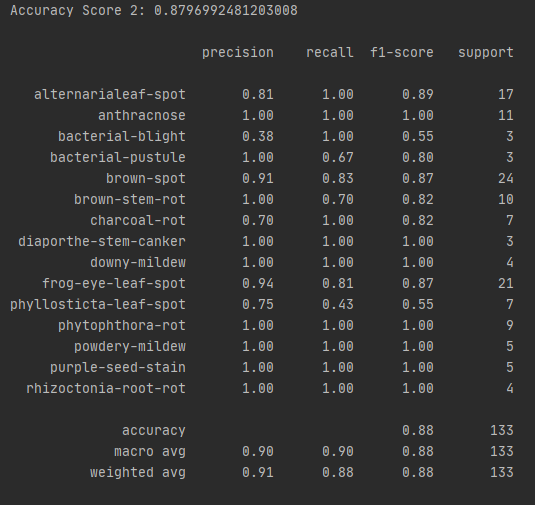
+Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. Recall cao đồng nghĩa với việc True Positive Rate cao, tức tỉ lệ bỏ sót các điểm thực sự là thấp.

-F1-score, là *harmonic mean* của precision và recall (giả sử rằng hai đại lượng này khác không)

\*Kết quả đánh giá



\*Kết quả đánh giá với SMOTE (cân bằng dữ liệu)



## **2. Bài toán phân lớp dùng để phân loại nấm độc hay nấm ăn được sử dụng kỹ thuật cây quyết định (Decision Trees)**

### **2.1 Mô tả bộ dữ liệu**

**Nguồn gốc:**

Hồ sơ về nấm được rút ra từ Hướng dẫn thực địa về nấm ở Bắc Mỹ của Hiệp hội Audubon (1981). GH Lincoff (Pres.), New York: Alfred A. Knopf

**Nhà tài trợ:**

Jeff Schlimmer ( Jeffrey.Schlimmer **'@'** a.gp.cs.cmu.edu )

**Thông tin tập dữ liệu:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Đặc điểm của Tập dữ liệu:** | Đa biến | **Số phiên bản:** | 8124 | **Khu vực:** | Đời sống |
| **Đặc điểm thuộc tính:** | Phân loại | **Số thuộc tính:** | 22 | **Ngày được tặng** | 1987-04-27 |
| **Nhiệm vụ liên quan:** | Phân loại | **Giá trị bị mất?** | Đúng | **Số lượt truy cập web:** | 592196 |

Bộ dữ liệu này bao gồm các mô tả về các mẫu giả định tương ứng với 23 loài nấm mang trong họ Agaricus và Lepiota (trang 500-525). Mỗi loài được xác định là chắc chắn ăn được, chắc chắn có độc, hoặc không rõ có thể ăn được và không được khuyến khích. Lớp sau này được kết hợp với lớp độc. Hướng dẫn nêu rõ rằng không có quy tắc đơn giản nào để xác định khả năng ăn được của nấm

**Thông tin thuộc tính (classes: edible=e, poisonous=p):**

1. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s

2. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s

3. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y

4. bruises: bruises=t, no=f

5. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s

6. gill-attachment: attached=a, descending=d, free=f, notched=n

7. gill-spacing: close=c, crowded=w, distant=d

8. gill-size: broad=b, narrow=n

9. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

10. stalk-shape: enlarging=e, tapering=t

11. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?

12. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s

13. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s

14. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

15. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

16. veil-type: partial=p, universal=u

17. veil-color: brown=n, orange=o, white=w, yellow=y

18. ring-number: none=n, one=o, two=t

19. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z

20. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y

21. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y

22. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

### **2.2 Mô tả bài toán**

Bài toán sử dụng bộ dữ liệu các thuộc tính vật lý nấm để phân loại ra 2 loại nấm ăn được và nấm độc. Sử dụng bài toán học máy phân lớp. Áp dụng ký thuật Logistic Regression

### **2.3 Mô tả chi tiết về cách xử lý dữ liệu**

Trong quá trình thu thật dữ liệu do dữ liệu quá lớn nên cũng không tránh khỏi mất mát 1 số dữ liệu để giải quyết vẫn đề này thì chúng ta sử dụng hàm dropna():

Vì bộ dữ liệu là dạng chữ nên 1 số chữ không thể chạy được trong học máy nên chúng ta cần đưa bộ dữ liệu về dạng số bằng hàm get\_dummies() trong thư viện pandas:

Kết quả trước và sau khi đưa về dạng số:

Sau khi chia train và test xong để tránh khỏi việc mất cân bằng dữ liệu thì chúng ta dùng hàm SMOTE của thư viện imblearn:

Kết quả:

### **2.4 Mô tả kỹ thuật Decision Trees**

#### **2.4.1 Giới thiệu**

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.

Chúng ta sẽ làm quen với một thuật toán xây dựng decision tree ra đời từ rất sớm và rất phổ biến: [Iterative Dichotomiser 3 (ID3)](https://en.wikipedia.org/wiki/ID3_algorithm).

ID3 là một thuật toán decision tree được áp dụng cho các bài toán classification mà tất cả các thuộc tính đều ở dạng categorical. Trong bài tiếp theo, chúng ta sẽ làm quen với một thuật toán khác có tên là Classification and Regression Tree (CART)–có thể được áp dụng vào cả hai loại classification và regression, như tên gọi của nó–làm việc với cả thuộc tính dạng categorical và liên tục.

#### **2.4.2 Ý tưởng**

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

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

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

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

#### **2.4.3. Hàm số entropy**

Cho một phân phối xác suất của một biến rời rạc xx có thể nhận nn giá trị khác nhau x1,x2,…,xnx1,x2,…,xn.

Giả sử rằng xác suất để xx nhận các giá trị này là pi=p(x=xi)pi=p(x=xi) với 0≤pi≤1,∑ni=1pi=10≤pi≤1,∑i=1npi=1. Ký hiệu phân phối này là p=(p1,p2,…,pn)p=(p1,p2,…,pn). Entropy của phân phối này được định nghĩa là

H(p)=−n∑i=1pilog(pi)(1)

trong đó loglog là logarit tự nhiên (*Một số tài liệu dùng logarit cơ số 2, nhưng giá trị của* H(p)H(p) *chỉ khác đi bằng cách nhân với một hằng số.*) và quy ước 0log(0) =0.

Xét một ví dụ với n=2n=2 được cho trên Hình 3. Trong trường hợp pp là *tinh khiết* nhất, tức một trong hai giá trị pipi bằng 1, giá trị kia bằng 0, entropy của phân phối này là H(p)=0H(p)=0. Khi pp là *vẩn đục* nhất, tức cả hai giá trị pi=0.5pi=0.5, hàm entropy đạt giá trị cao nhất.

|  |  |
| --- | --- |
|  | Hình 3: Đồ thị của hàm entropy với n=2n=2. |

Tổng quát lên với n>2n>2, hàm entropy đạt giá trị nhỏ nhất nếu có một giá trị pi=1pi=1, đạt giá trị lớn nhất nếu tất cả các pipi bằng nhau (việc này có thể được chứng minh bằng [phương pháp nhân tử Lagrange](https://machinelearningcoban.com/2017/04/02/duality/#--phuong-phap-nhan-tu-lagrange)).

Những tính chất này của hàm entropy khiến nó được sử dụng trong việc đo *độ vẩn đục* của một phép phân chia của ID3. Vì lý do này, ID3 còn được gọi là *entropy-based decision tree*.

#### **2.4.4. Thuật toán ID3**

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

Xét một bài toán với CC class khác nhau. Giả sử ta đang làm việc với một *non-leaf node* với các điểm dữ liệu tạo thành một tập SS với số phần tử là |S|=N|S|=N. Giả sử thêm rằng trong số NN điểm dữ liệu này, Nc,c=1,2,…,CNc,c=1,2,…,C điểm thuộc vào class cc. Xác suất để mỗi điểm dữ liệu rơi vào một class cc được xấp xỉ bằng NcNNcN (maximum likelihood estimation). Như vậy, entropy tại node này được tính bởi:

H(S)=−C∑c=1NcNlog(NcN)(2)

Tiếp theo, giả sử thuộc tính được chọn là xx. Dựa trên xx, các điểm dữ liệu trong SS được phân ra thành KK child node S1,S2,…,SKS1,S2,…,SK với số điểm trong mỗi child node lần lượt là m1,m2,…,mKm1,m2,…,mK. Ta định nghĩa

H(x,S)=K∑k=1mkNH(Sk)(3)

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

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

G(x,S)=H(S)−H(x,S)

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

x∗=argmaxxG(x,S)=argminxH(x,S)

tức thuộc tính khiến cho *information gain* đạt giá trị lớn nhất.

### **2.5. Phương pháp đánh giá Accuracy( Độ chính xác)**

Cách đơn giản và hay được sử dụng nhất là *accuracy* (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.

Đối với phân loại nhị phân, độ chính xác cũng có thể được tính theo mặt tích cực và tiêu cực như sau:

Trong đó *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

## **3. Bài toán dự đoán chất lượng rượu vang "Vinho Verde" của Bồ Đào Nha với biến thể đỏ**

### **3.1. Chi tiết dữ liệu sử dụng :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Đặc điểm của Tập dữ liệu:** | Đa biến | **Số phiên bản:** | 4898 | **Khu vực:** | Kinh doanh |
| **Đặc điểm thuộc tính:** | Thực tế | **Số thuộc tính:** | 12 | **Ngày được tặng** | 2009-10-07 |
| **Nhiệm vụ liên quan:** | Phân loại, hồi quy | **Giá trị bị mất?** | N / A | **Số lượt truy cập web:** | 1402897 |

**Nguồn:**

Paulo Cortez, Đại học Minho, Guimarães, Bồ Đào Nha, <http://www3.dsi.uminho.pt/pcortez>

A. Cerdeira, F. Almeida, T. Matos và J. Reis, Ủy ban trồng trọt vùng Vinho Verde (CVRVV) , Porto, Bồ Đào Nha

@ 2009

link :<https://archive.ics.uci.edu/ml/datasets/Wine+Quality>

Thông tin dữ liệu : Các bộ dữ liệu này có thể được xem như các nhiệm vụ phân loại hoặc hồi quy. Các lớp được sắp xếp theo thứ tự và không cân bằng (ví dụ: có nhiều loại rượu bình thường hơn loại xuất sắc hoặc kém). Các thuật toán phát hiện ngoại lệ có thể được sử dụng để phát hiện một số loại rượu ngon hoặc kém.

Bộ dữ liệu có 12 thuộc tính. 1600 dòng và 12 cột.

Dữ liệu có 12 cột gồm :

+ fixed acidity ( độ axit cố định )

+ volatile acidity (độ axit bay hơi )

+ citric acid (axit xitric)

+ residual sugar ( đường dư )

+ chlorides ( clorua )

+ free sulfur dioxide ( lưu huỳnh dioxit )

+ total sulfur dioxide ( tổng lưu huỳnh dioxit )

+ density ( tỷ trọng )

+ pH

+ sulphates ( sunfat )

+ alcohol ( rượu )

+ quality ( chất lượng )

### **3.2. Mô tả bài toán học máy :**

Bài toán sử dụng muốn dự đoán chất lượng của rượu

Sử dụng bài toán : phân loại

Kỹ thuật LogisticRegression

### **3.3. Chi tiết về cách xử lý dữ liệu**

- Dữ liệu thiếu, bị mất : Không

- Bài toán sử dụng kĩ thuật học máy : Logistic Regression

- Giới thiệu

Logistic Regression là một thuật toán phân loại được dùng để gán các đối tượng cho một tập giá trị rời rạc như 0,1,2…. Ví dụ điển hình là phân loại Email. Gồm có email công việc, email gia đình, email spam,…

Mô hình Logistic Regression

Đầu ra dự đoán của:

* Linear Regression:f(x)=wTxf(x)=wTx
* PLA:f(x)=sgn(wTx)f(x)=sgn(wTx)

Đầu ra dự đoán của logistic regression thường được viết chung dưới dạng:

f(x)=θ(wTx)f(x)=θ(wTx)

Trong đó θθ được gọi là logistic function. Một số activation cho mô hình tuyến tính được cho trong hình dưới đây:

Hình 2: Các activation function khác nhau.

* Đường màu vàng biểu diễn linear regression. Đường này không bị chặn nên không phù hợp cho bài toán này. Có một *trick* nhỏ để đưa nó về dạng bị chặn: *cắt* phần nhỏ hơn 0 bằng cách cho chúng bằng 0, *cắt* các phần lớn hơn 1 bằng cách cho chúng bằng 1. Sau đó lấy điểm trên đường thẳng này có tung độ bằng 0.5 làm điểm phân chia hai *class*, đây cũng không phải là một lựa chọn tốt. Giả sử có thêm vài bạn *sinh viên tiêu biểu* ôn tập đến 20 giờ và, tất nhiên, thi đỗ. Khi áp dụng mô hình linear regression như hình dưới đây và lấy mốc 0.5 để phân lớp, toàn bộ sinh viên thi trượt vẫn được dự đoán là trượt, nhưng rất nhiều sinh viên thi đỗ cũng được dự đoán là trượt (nếu ta coi điểm x màu xanh lục là *ngưỡng cứng* để đưa ra kết luận). Rõ ràng đây là một mô hình không tốt. Anh chàng sinh viên tiêu biểu này đã *kéo theo* rất nhiều bạn khác bị trượt.

Hình 3: Tại sao Linear Regression không phù hợp?

* Đường màu đỏ (chỉ khác với activation function của PLA ở chỗ hai class là 0 và 1 thay vì -1 và 1) cũng thuộc dạng *ngưỡng cứng* (hard threshold). PLA không hoạt động trong bài toán này vì dữ liệu đã cho không *linearly separable*.

· Các đường màu xanh lam và xanh lục phù hợp với bài toán của chúng ta hơn. Chúng có một vài tính chất quan trọng sau:

* + Là hàm số liên tục nhận giá trị thực, bị chặn trong khoảng (0,1)(0,1).
  + Nếu coi điểm có tung độ là 1/2 làm điểm phân chia thì các điểm càng xa điểm này về phía bên trái có giá trị càng gần 0. Ngược lại, các điểm càng xa điểm này về phía phải có giá trị càng gần 1. Điều này *khớp* với nhận xét rằng học càng nhiều thì xác suất đỗ càng cao và ngược lại.
  + *Mượt* (smooth) nên có đạo hàm mọi nơi, có thể được lợi trong việc tối ưu.

Một vài tính chất của Logistic Regression

Logistic Regression thực ra được sử dụng nhiều trong các bài toán Classification.

Mặc dù có tên là Regression, tức một mô hình cho fitting, Logistic Regression lại được sử dụng nhiều trong các bài toán Classification. Sau khi tìm được mô hình, việc xác định class yy cho một điểm dữ liệu xx được xác định bằng việc so sánh hai biểu thức xác suất:P(y=1|x;w); P(y=0|x;w)P(y=1|x;w); P(y=0|x;w)Nếu biểu thức thứ nhất lớn hơn thì ta kết luận điểm dữ liệu thuộc class 1, ngược lại thì nó thuộc class 0. Vì tổng hai biểu thức này luôn bằng 1 nên một cách gọn hơn, ta chỉ cần xác định xem P(y=1|x;w)P(y=1|x;w) lớn hơn 0.5 hay không. Nếu có, class 1. Nếu không, class 0.

Boundary tạo bởi Logistic Regression có dạng tuyến tính

Thật vậy, theo lập luận ở phần trên thì chúng ta cần kiểm tra:

P(y=1|x;w)>0.5 ⇔11+e−wTx>0.5 ⇔e−wTx<1 ⇔wTx>0P(y=1|x;w)>0.5 ⇔11+e−wTx>0.5 ⇔e−wTx<1 ⇔wTx>0

Nói cách khác, boundary giữa hai class là đường có phương trình wTxwTx. Đây chính là phương trình của một siêu mặt phẳng. Vậy Logistic Regression tạo ra boundary có dạng tuyến tính.

- Code thực hiện bài toán :

- Kết quả đạt được

Giải thích :

Accuracy : Là độ chính xác của mô hình học máy

Recall : Bao nhiêu cái được lấy ra là đúng. Chỉ số này còn được gọi là độ bao phủ tức là xét xem mô hình tìm được có khả năng tổng quát hóa thể nào.

F1-Score : Từ 2 yếu tố độ chính xác và độ bao phủ phía trên người ta đưa ra 1 chỉ số khác là F1-score. Đây được gọi là một trung bình điều hòa của các tiêu chí Precision và Recall và đồng thời nó có giá trị lớn nếu cả 2 giá trị Precision và Recall đều lớn.

Với kết quả đạt được như trên thì chúng ta thu được độ chính xác của mô hình học máy là 59%

### **3.4.Phương pháp đánh giá**

Cách đơn giản và hay được sử dụng nhất là *accuracy* (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.

Đối với phân loại nhị phân, độ chính xác cũng có thể được tính theo mặt tích cực và tiêu cực như sau:

Trong đó *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.