ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỌC MÁY TÍNH

KHAI THÁC DỮ LIỆU VÀ ỨNG DỤNG – CS313.N22

Bài tập 4.2: *Sử dụng ngôn ngữ Python triển khai bài toán phân lớp với các thuật toán cơ bản*

Nhóm 3:

Bùi Nguyễn Anh Trung (NT)	20520332
Hồ Thanh Tịnh	20520813
Nguyễn Trần Minh Anh	20520394
Lê Nguyễn Bảo Hân	20520174
Nguyễn Văn Đức Ngọc	20521666

Hồ Chí Minh, 19 tháng 04 năm 2023

Nội dung

Bảng phân công

Thành viên / Công Việc	Bùi Nguyễn Anh Trung	Hồ Thanh Tịnh	Nguyễn Trần Minh Anh	Lê Nguyễn Bảo Hân	Nguyễn Văn Đức Ngọc
Phân công, quản lý công việc chung			✓		
Tìm hiểu thuật toán SVM					✓
Tìm hiểu thuật toán Deep Learning	✓				
Tìm hiểu phương pháp phân chia dữ liệu thực nghiệm			✓		
Tìm hiểu độ đo đánh giá				✓	
Tổng hợp nội dung và định dạng báo cáo				✓	
Chuẩn bị demo			✓		
Làm slide		✓			
Mức độ hoàn thiện (%)	100%	100%	100%	100%	100%

1. Các thuật toán phân lớp

1.1. Thuật toán SVM

1.1.1. Support Vector Machine là gì?

Trong máy học, Support Vector Machine (SVM) là mô hình học có giám sát với các thuật toán học liên kết phân tích dữ liệu để phân loại và phân tích hồi quy. [1] [2]

Thuật toán được phát triển tại Phòng thí nghiệm AT&T Bell bởi Vladimir Vapnik cùng với các đồng nghiệp (*Boser* và cộng sự, 1992, *Guyon* và cộng sự, 1993, *Cortes* và Vapnik , 1995, Vapnik và cộng sự, 1997).

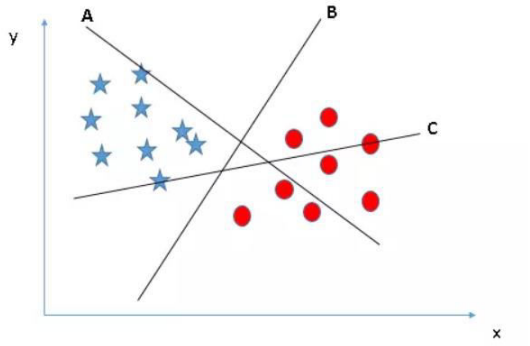
SVM là một trong những phương pháp dự đoán mạnh mẽ nhất, dựa trên khung học tập thống kê hoặc lý thuyết *Vapnik–Chervonenkis* do *Vapnik* (1982, 1995) và *Chervonenkis* (1974) đề xuất. Thuật toán đưa ra một tập hợp các mẫu huấn luyện, mỗi mẫu được đánh dấu là thuộc về một trong hai loại. Thuật toán huấn luyện SVM xây dựng một mô hình gán các mẫu mới cho loại này hay loại khác, làm cho nó trở thành một bộ phân loại tuyến tính nhị phân phi xác suất. SVM ánh xạ các ví dụ đào tạo tới các điểm trong không gian để tối đa hóa độ rộng của khoảng cách giữa hai loại. Các ví dụ mới sau đó được ánh xạ vào cùng một không gian đó và được dự đoán thuộc về một danh mục dựa trên việc chúng rơi vào phía nào của khoảng trống.

Ngoài việc thực hiện phân loại tuyến tính , các SVM có thể thực hiện phân loại phi tuyến tính một cách hiệu quả bằng cách sử dụng cái được gọi là thủ thuật nhân, ánh xạ hoàn toàn đầu vào của chúng vào các không gian đặc trưng nhiều chiều.

1.1.2. Ý tưởng và nền tảng toán học cho việc phân lớp

Ở đây, có 3 đường A, B, C, cần tìm đường nào là phân nhóm đúng cho nhóm ngôi sao và hình tròn. [3]

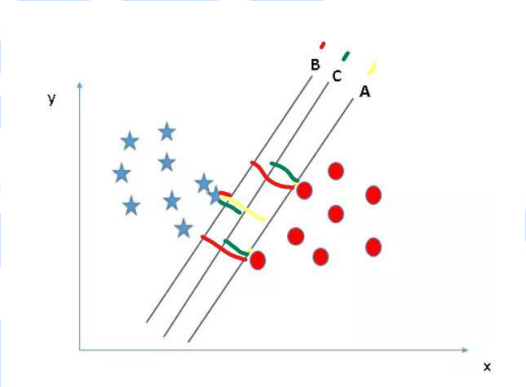
Ví dụ 1:



Hình 1.1 Minh họa 1 thuật toán SVM

B là lựa chọn hợp lí nhất vì phân chia chính xác hoàn toàn.

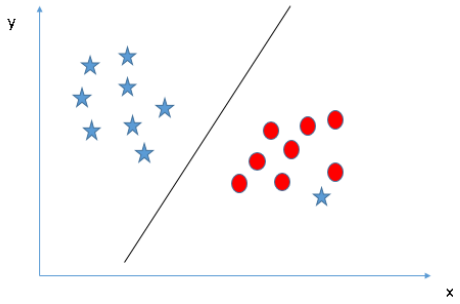
Ví dụ 2:



Hình 1.2 Minh họa 2 thuật toán SVM

C là lựa chọn hợp lí bởi theo trực quan ta thấy nó phân chia công bằng 2 lớp dựa trên khoảng cách biên giữa các điểm gần đường phân chia nhất với nó.

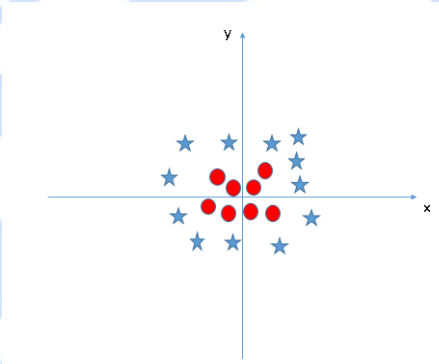
Ví dụ 3:



Hình 1.3 Minh họa 3 thuật toán SVM

Ở đây, ta bỏ qua ngôi sao ngoại lệ phía bên trái và chọn đường chia như trên.

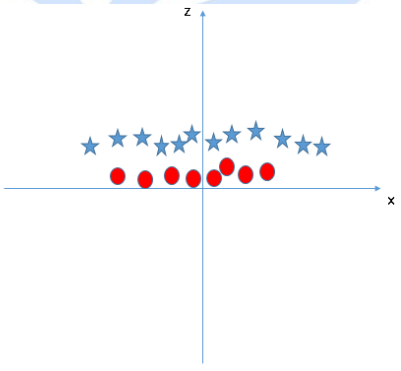
Ví dụ 4:



Hình 1.4 Minh họa 4 thuật toán SVM

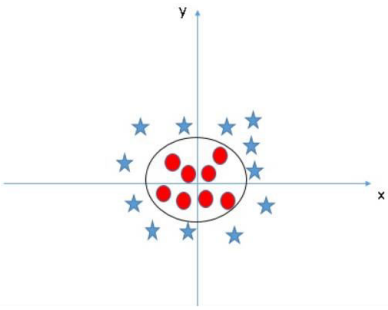
$z = x^2 + y^2$

Hình 4 ánh xạ qua z ta được hình *Hình 5*.



Hình 1.5 Minh họa 5 thuật toán SVM

Đến đây, có thể dễ dàng tìm được đường thẳng z = f(x) chia các điểm dữ liệu ra 2 lớp.

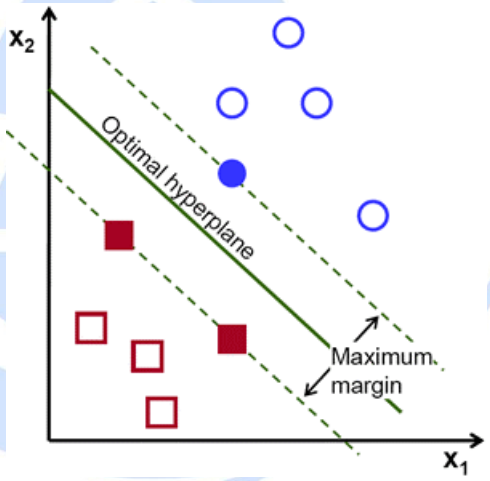


Hình 1.6 Minh họa 6 thuật toán SVM

Từ z ta có thể biến đổi thành đường phân chia 2 lớp như trên có phương trình dạng f(x,y) = 0, f(x,y) suy ra từ z = f(x).

1.1.2.1. Lề (Margin)

Margin là khoảng cách giữa siêu phẳng đến 2 điểm dữ liệu gần nhất tương ứng với các phân lớp.



Hình 1.7 Lề trong thuật toán SVM

Khoảng cách từ một điểm (vector) có toạ độ x_0 tới siêu mặt phẳng.

$w^T x + b = 0$ $(w, x \in R^d)$ xác định bởi công thức:

$$\frac{|w^T x_0 + b|}{||w||_2}$$

Trong đó $||w||_2 = \sqrt[2]{\sum_{i=1}^{d} w_i^2}$, d chính là số chiều của không gian.

1.1.2.2. Hard Margin SVM (SVM biên cứng)

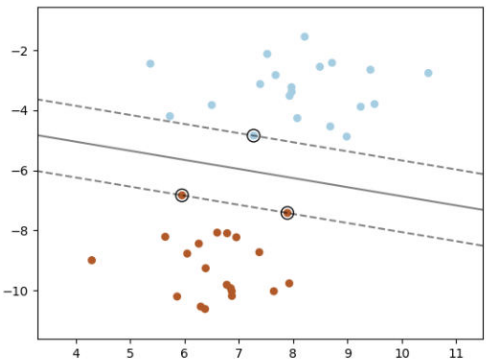
Giả sử rằng có hai class khác nhau được mô tả bởi các điểm trong không gian nhiều chiều, hai classes này linearly separable.

Các cặp dữ liệu của trainning set là $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ với $x_i \in R^d$ với d là số chiều của không gian, n là số điểm dữ liệu $y_i \in \{-1,1\}$.

Với cặp dữ liệu (x_n, y_n) bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

$$\frac{y_n(w^T x_n + b)}{||w||_2}$$

Công việc của chúng ta chính là tìm ra được đường phân chia (hay w, b) sao cho khoảng cách giữa đường (hay mặt phẳng phân cách 2 lớp) với các điểm gần nhất của 2 lớp là bằng nhau và là lớn nhất. Trên thực tế, có rất nhiều đường phân chia giúp lề giữa 2 lớp bằng nhau, nhưng ta cần tìm đường giúp cho margin lớn nhất. Việc margin giữa 2 lớp bằng nhau sẽ giúp tạo ra sự công bằng trong không gian phân cách 2 lớp.



Hình 1.8 Minh họa Hard Margin SVM

Margin được tính là khoảng cách gần nhất từ 1 điểm tới mặt đó (bất kể điểm nào trong hai classes):

$$margin = \min\left(\frac{y_n(w^T x_n + b)}{||w||_2}\right.$$

Bài toán tối ưu trong SVM chính là bài toán tìm w và b sao cho margin này đạt giá trị lớn nhất:

$$(w, b) = \arg\max(\frac{1}{||w||_2} * \min(y_n(w^T x_n + b))$$

Việc giải trực tiếp bài toán này sẽ rất phức tạp, ta cần đưa nó về bài toán đơn giản hơn.

Nhận xét quan trọng nhất là nếu ta thay vector hệ số $w = kw$ bởi $b = kb$ trong đó k là một hằng số dương thì mặt phân chia không thay đổi, tức khoảng cách từ từng điểm đến mặt phân chia không đổi, tức margin không đổi. Dựa trên tính chất này, ta có thể giả sử:

$$y_n(w^T x_n + b) \geq 1$$

Với điều kiện biến đổi trên, ta có bài toán tối ưu mới có ràng buộc là :

$$(w, b) = \arg\min \frac{1}{2}\left(||w||_2^2\right),$$

$$subject\ to\ y_n(w^T x_n + b) \geq 1\ , với\ mọi\ n \in \{1,2,..,n\}$$

Hay là:

$$(w, b) = \arg\min \frac{1}{2}\left(||w||_2^2\right),$$

$$subject\ to\ 1 - y_n(w^T x_n + b) \leq 0\ , với\ mọi\ n \in \{1,2,..,n\}$$

Giải bài toán trên ta được nghiệm w, b là duy nhất.

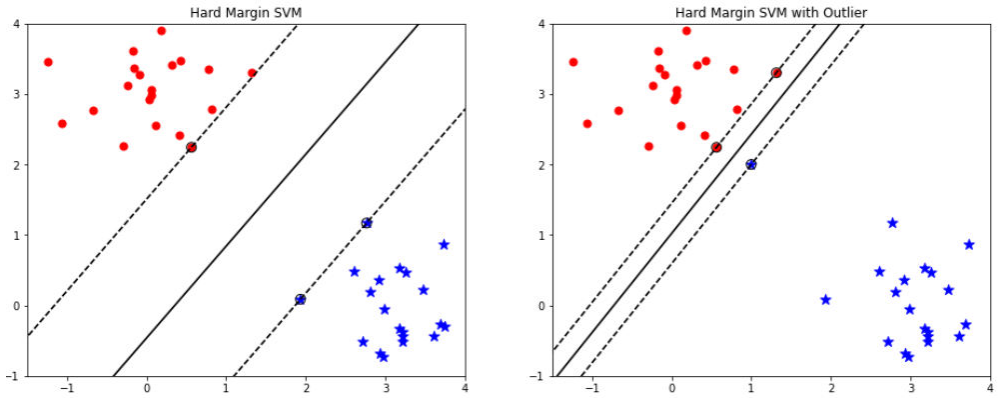
Sau khi tìm được mặt phân cách $w^T x_n + b = 0$, class của bất kỳ một điểm nào sẽ được xác định đơn giản bằng cách:

$$\text{class}(x_i) = \text{sgn}(w^T x_i + b)$$

Trong đó hàm sgn là hàm xác định dấu, nhận giá trị 1 nếu đối số là không âm và -1 nếu ngược lại.

1.1.2.3. Soft Margin SVM (SVM biên mềm)

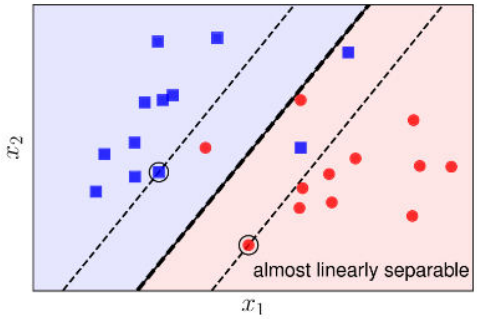
Hạn chế của SVM biên cứng



Hình 1.9 SVM biên cứng với dữ liệu ngoại lai

Hình bên trái là phân loại đường biên cứng (Hard margin SVM) đối với tập dữ liệu thông thường. Hình bên phải là phân loại đường biên cứng đối với dữ liệu chứa điểm ngoại lai (là điểm hình sao được khoanh tròn).

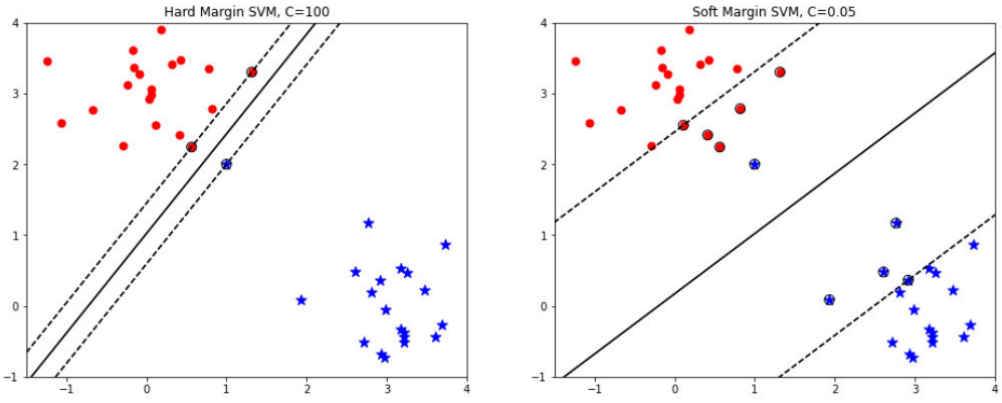
Phương pháp phân loại đường biên cứng buộc phải phân loại đúng mọi điểm dữ liệu, bao gồm cả điểm ngoại lai. Điều này khiến cho đường biên phân chia bị thu hẹp lại. Khi đó quy luật phân chia sẽ không còn giữ được yếu tố tổng quát và dẫn tới hiện tượng quá khớp (overfitting). Kết quả dự báo trên tập kiểm tra khi đó sẽ kém hơn so với tập huấn luyện. Đồng thời với nhưng bộ dữ liệu không linear separable thì Hard Margin SVM sẽ vô nghiệm.



Hình 1.10 Hạn chế của SVM biên cứng

Để khắc phục hạn chế của phân loại đường biên cứng, kỹ thuật phân loại đường biên mềm (Sorf Margin Classification) chấp nhận đánh đổi để mở rộng lề và cho phép phân loại sai các điểm ngoại lai. [4]

Cụ thể hơn, thuật toán sẽ chấp nhận một số điểm bị rơi vào vùng của lề (vùng nằm giữa hai đường nét đứt, vùng này còn được gọi là vùng không an toàn). Trái lại, chi phí cơ hội của sự đánh đổi đó là độ rộng lề lớn hơn. Đường biên phân chia được tạo ra từ kỹ thuật này thường nắm được tính tổng quát và hạn chế hiện tượng quá khớp.

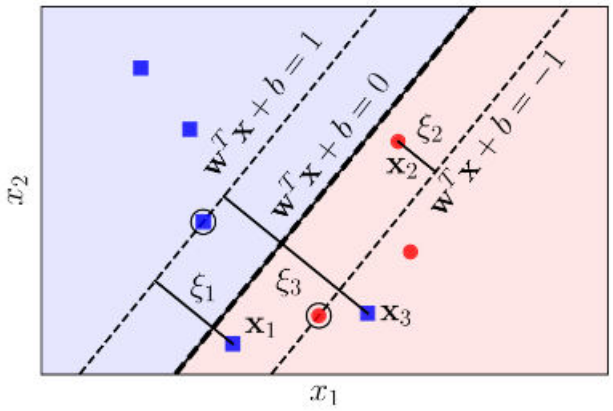


Hình 1.11 Hard Margin SVM và Soft Margin SVM

Ý tưởng của phân loại đường biên mềm là mở rộng lề và chấp nhận hi sinh một số điểm bị rơi vào vùng không an toàn để tạo ra một đường biên phân chia tổng quát hơn. Nhưng chúng ta không thể mở rộng lề ra vô cùng vì như vậy có nhiều điểm bị rơi vào vùng không an toàn hơn và dẫn tới sự hi sinh là quá lớn.

Quá trình mở rộng lề sẽ bị kìm hãm sao cho đối với những điểm bị rơi vào vùng không an toàn thì tổng khoảng cách của chúng tới mép của lề về phía mặt phẳng của nhãn ground truth của chúng là nhỏ nhất. Khoảng cách này được thể hiện qua biến slack (ký hiệu s_n):

$$s_i = |w^T x_i + b - y_i|$$



Giới thiệu các biến slack s_n. Với những điểm nằm trong vùng an toàn $s_n = 0$. Những điểm nằm trong vùng không an toàn nhưng vẫn đúng phía so với đường phân chia tương ứng với các $0 < s_n < 1$, ví dụ x_2 . Những điểm nằm ngược phía với class của chúng so với đường boundary ứng với các $s_n > 1$, ví dụ như x_1 , x_3.

Bài toán tối ưu trong Hard margin SVM:

$$(w, b) = \arg\min \frac{1}{2}\left(||w||_2^2\right),$$

$$subject\ to\ 1 - y_n(w^T x_n + b) \leq 0\ , với\ mọi\ n \in \{1,2,..,n\}$$

Với Soft Margin SVM, hàm mục tiêu sẽ có thêm một số hạng nữa giúp tối thiểu sự hy sinh. Từ đó ta có hàm mục tiêu:

$$(w, b) = \arg\min \frac{1}{2}\left(||w||_2^2\right) + C * \sum_{i=1}^{n} s_i$$

Hằng số C được dùng để điều chỉnh tầm quan trọng giữa margin và sự hy sinh. Hằng số này được xác định từ trước bởi người lập trình.

Điều kiện ràng buộc sẽ thay đổi một chút. Với mỗi cặp dữ liệu (x_n, y_n), thay vì ràng buộc cứng $1 - y_n(w^T x_n + b) \leq 0$, chúng ta sẽ có ràng buộc mềm:

$$y_n(w^T x_n + b) \geq 1 - s_n$$

$$\Rightarrow 1 - y_n(w^T x_n + b) - s_n \leq 0\ với\ mọi = 1,.,n$$

Tóm lại, ta sẽ có bài toán tối ưu ở dạng chuẩn cho Soft-margin SVM:

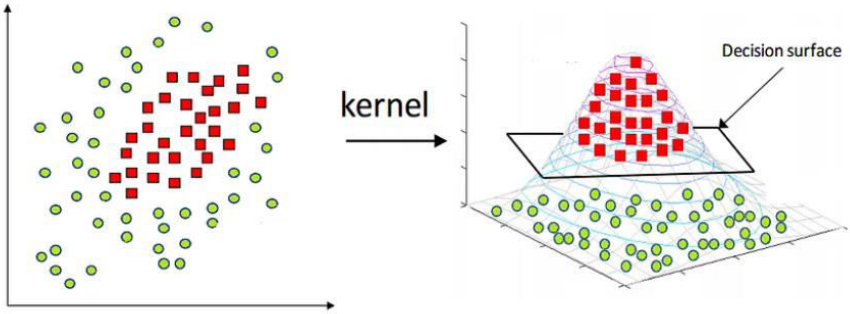
$$(w, b, s) = \arg\min \frac{1}{2}\left(||w||_2^2\right) + C * \sum_{i=1}^{n} s_i$$

$$subject\ to\ 1 - y_n(w^T x_n + b) - s_n \leq 0\ , với\ mọi\ n \in \{1,2,..,n\}$$

Hệ số C là một hệ số rất quan trọng thể hiện tỷ lệ đánh đối giữa sự mở rộng lề và và sự hi sinh. Một hệ số C lớn thì ảnh hưởng của $||w||_2^2$ sẽ không đáng kể lên giá trị của hàm mất mát. Giá trị của hàm mất mát chủ yếu đến từ tổng khoảng cách xâm lấn. Tối thiểu hàm mất mát đồng nghĩa với cần hạn chế bớt mức độ xâm lấn. Điều này dẫn tới độ rộng lề nhỏ hơn. Trong trường hợp tập dữ liệu là phân tuyến thì tồn tại một đường biên phân chia đúng mọi điểm dữ liệu. Như vậy $\sum_{i=1}^{n} s_i = 0$ và ta thu được bài toán phân loại đường biên cứng (Hard Margin SVM).

Trái lại trường hợp C nhỏ dẫn tới $||w||_2^2$ nhỏ. Do đó độ rộng lề là lớn hơn. Điều đó đồng nghĩa với mức độ xâm lấn cũng lớn theo. Sẽ tồn tại một số điểm xâm lấn tương ứng với và trường hợp này tương ứng $s_i > 0$ với bài toán phân loại đường biên mềm (Soft Margin SVM).

1.1.3. Kernal Trick trong SVM



Hình 1.12 Ánh xạ giữa không gian gốc 2 chiều tới không gian 3 chiều

Ta có thể thấy được không gian dữ liệu gốc có 2 lớp dữ liệu không có phân chia tuyến tính được, thuộc dạng phi tuyến tính, vì thế các thuật toán nói

trên là vô dụng. Trong SVM ta có 1 kĩ thuật gọi là Kernel Trick được dùng để tìm đường phân chia lớp bằng các hàm kernel. [5]

Ta thấy rằng trong hình trên, nếu chúng ta tìm cách ánh xạ dữ liệu từ không gian 2 chiều sang không gian 3 chiều, chúng ta sẽ có thể tìm thấy một bề mặt quyết định phân chia rõ ràng giữa các lớp khác nhau.

Tuy nhiên, khi ngày càng có nhiều chiều, việc tính toán trong không gian ngày càng trở nên đắt đỏ hơn. Đây là lúc Kernel Trick xuất hiện. Nó cho phép chúng ta vận hành trong không gian đối tượng ban đầu mà không cần tính toán tọa độ của dữ liệu trong không gian nhiều chiều hơn. Nó cho phép chúng ta hoạt động trong không gian đặc trưng ban đầu mà không cần tính toán tọa độ của dữ liệu trong không gian nhiều chiều hơn.

Ví dụ: $x = (x_1, x_2, x_{3)}\ , y = (y_1, y_2, y_3)$

Ở đây x và y là hai điểm dữ liệu trong 3 chiều. Giả sử rằng chúng ta cần ánh xạ x và y vào không gian 9 chiều. Chúng ta cần thực hiện các phép tính sau để có được kết quả cuối cùng, đó chỉ là một đại lượng vô hướng. Độ phức tạp tính toán, trong trường hợp này là O(n^2).

$$\phi(x) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$$

$$\phi(y) = (y_1^2, y_1y_2, y_1y_3, y_2y_1, y_2^2, y_2y_3, y_3y_1, y_3y_2, y_3^2)^T$$

$$\phi(x)^T \phi(y) = \sum_{i,j=1}^{3} x_i x_j y_i y_j$$

Tuy nhiên, nếu chúng ta sử dụng hàm nhân, được ký hiệu là k(x, y), thay vì thực hiện các phép tính phức tạp trong không gian 9 chiều, chúng ta sẽ đạt được kết quả tương tự trong không gian 3 chiều bằng cách tính tích vô hướng của x -transpose và y. Độ phức tạp tính toán trong trường hợp này là O(n).

$$k(x, y) = (x^T y)^2 = (x_1y_1 + x_2y_{2+}x_3y_3)\text{^}2 = \sum_{i,j=1}^{3} x_i x_j y_i y_j$$

Về bản chất, những gì thủ thuật hạt nhân cung cấp một cách hiệu quả hơn và ít tốn kém hơn để chuyển đổi dữ liệu sang các chiều cao hơn. Như đã nói, ứng dụng của thủ thuật kernel không chỉ giới hạn trong thuật toán SVM. Bất kỳ tính toán nào liên quan đến các tích vô hướng (x, y) đều có thể sử dụng thủ thuật kernel.

Các hàm hạt nhân (kernel function) phổ biến đã được tích hợp bên trong package sklearn:

- Kernel RBF: Kernel RBF dựa trên hàm Gaussian RBF. Hàm biến đổi phi tuyến của kernel này là hàm ẩn và tương đương với một đa thức với bậc vô hạn.

$$k(x, y) = e^{-gamma*||x-y||^2}, gamma > 0$$

- Kernel đa thức (poly): Tạo ra một đa thức bậc cao kết hợp giữa hai véc tơ.

$$k(x, y) = (gamma * x^T y + r)^d$$

  Với n đặc trưng ban đầu (số chiều dữ liệu ban đầu) ta mở rộng ra n*d chiều (n*d đặc trưng mở rộng).

- Kernel tuyến tính (linear): Đây là tích vô hướng giữa hai véc tơ.

$$k(x, y) = x^T y$$

Một điều quan trọng cần ghi nhớ là khi chúng ta ánh xạ dữ liệu sang một chiều cao hơn, có khả năng là chúng ta có thể điều chỉnh quá mức mô hình. Do đó, việc chọn đúng chức năng nhân (bao gồm các tham số phù hợp) và chính quy hóa có tầm quan trọng rất lớn.

Với danh sách các kernel được cung cấp ở mục trên thì chúng ta có thể điều chỉnh các tham số như sau:

- Kernel tuyến tính: tham số C.
- Kernel đa thức: tham số C, gamma, d
- Kernel RBF: tham số C, gamma

1.1.4. Ưu và nhược điểm

Ưu điểm:

- Xử lý trên không gian số chiều cao: SVM là một công cụ tính toán hiệu quả trong không gian chiều cao, trong đó đặc biệt áp dụng cho các bài toán phân loại văn bản và phân tích quan điểm nơi chiều có thể cực kỳ lớn.
- Tiết kiệm bộ nhớ: Do chỉ có một tập hợp con của các điểm được sử dụng trong quá trình huấn luyện và ra quyết định thực tế cho các điểm dữ liệu mới nên chỉ có những điểm cần thiết mới được lưu trữ trong bộ nhớ khi ra quyết định.
- Tính linh hoạt - phân lớp thường là phi tuyến tính. Khả năng áp dụng Kernel mới cho phép linh động giữa các phương pháp tuyến tính và phi tuyến tính từ đó khiến cho hiệu suất phân loại lớn hơn.

Nhược điểm:

- Bài toán số chiều cao: Trong trường hợp số lượng thuộc tính (p) của tập dữ liệu lớn hơn rất nhiều so với số lượng dữ liệu (n) thì SVM cho kết quả khá tồi.
- Chưa thể hiện rõ tính xác suất: Việc phân lớp của SVM chỉ là việc cố gắng tách các đối tượng vào hai lớp được phân tách bởi siêu phẳng SVM. Điều này chưa giải thích được xác suất xuất hiện của một thành viên trong một nhóm là như thế nào. Tuy nhiên hiệu quả của việc phân lớp có thể được xác định dựa vào khái niệm margin từ điểm dữ liệu mới đến siêu phẳng phân lớp mà chúng ta đã bàn luận ở trên.

1.2. Thuật toán Deep Learning

1.2.1. Logistic Regression

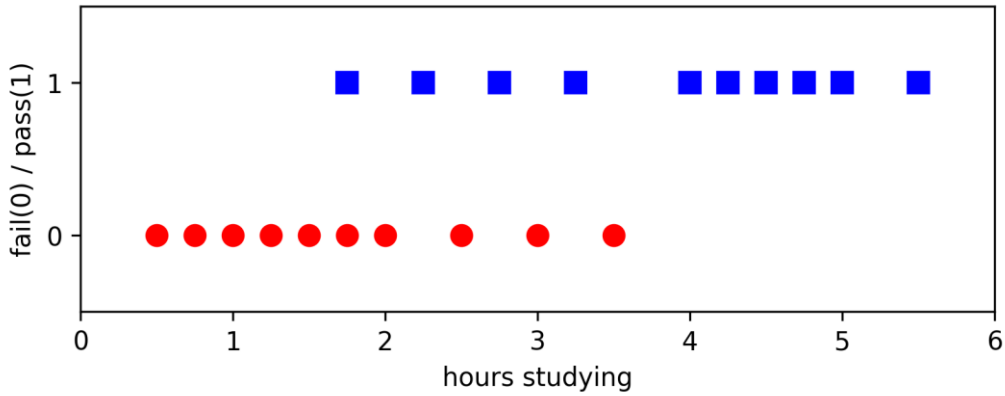
1.2.1.1. Giới thiệu vấn đề

Một nhóm 20 sinh viên dành thời gian trong khoảng từ 0 đến 6 giờ cho việc ôn thi. Thời gian ôn thi này ảnh hưởng đến xác suất sinh viên vượt qua kỳ thi như thế nào? [6]

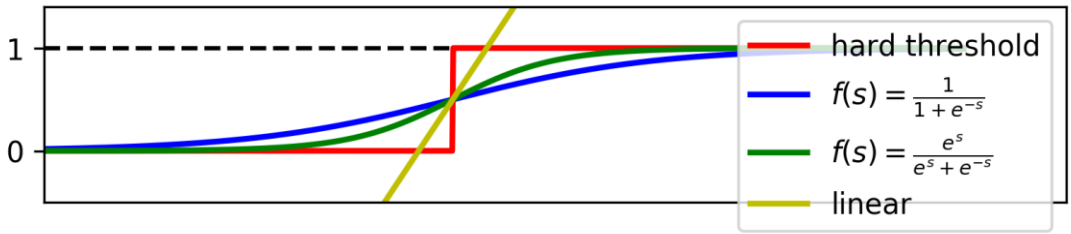
Kết quả thu được như sau:

HOURS	PASS	HOURS	PASS
.5	0	2.75	1
.75	0	3	0
1	0	3.25	1
1.25	0	3.5	0
1.5	0	4	1
1.75	0	4.25	1
1.75	1	4.5	1
2	0	4.75	1
2.25	1	5	1
2.5	0	5.5	1

Từ đó ta có kết quả minh họa:



Hình 1.13 Minh họa kết quả thi dựa vào số giờ ôn tập



Hình 1.14 Các activation function khác nhau.

1.2.1.2. Định nghĩa thuật toán

Thuật toán Logistic Regression là một phương pháp học máy được sử dụng rộng rãi trong các bài toán phân loại nhị phân, trong đó biến đầu ra chỉ có hai giá trị: 0 hoặc 1. Mục tiêu của thuật toán này là dự đoán xác suất của sự kiện xảy ra dựa trên một hoặc nhiều biến đầu vào.

Thuật toán Logistic Regression dựa trên kết quả của mô hình hồi quy tuyến tính và biến đầu ra được ánh xạ qua một hàm sigmoid để chuyển đổi giá trị đầu ra thành giá trị xác suất nằm trong khoảng từ 0 đến 1. Hàm sigmoid có dạng đường cong S, cho phép ánh xạ bất kỳ số thực nào thành giá trị xác suất.

Phương pháp Logistic Regression có thể được áp dụng cho cả biến đầu vào liên tục và rời rạc. Trong trường hợp biến đầu vào là rời rạc, chúng ta cần chuyển đổi chúng thành các biến giả trước khi sử dụng trong mô hình.

Mô hình Logistic Regression được huấn luyện bằng phương pháp ước lượng hàm likelihood tối đa (maximum likelihood estimation), trong đó chúng ta tìm kiếm các tham số mô hình để tối đa hoá xác suất quan sát

được dựa trên mô hình. Các tham số này thường được ước lượng bằng phương pháp gradient descent, là một thuật toán tối ưu hóa lặp đi lặp lại giúp tối thiểu hóa hàm chi phí.

1.2.1.3. Mô hình thuật toán

Đầu tiên, chúng ta cần xây dựng công thức cho mô hình Logistic Regression. Công thức chính của nó được định nghĩa như sau:

$$h_\theta(x) = g(\theta^T x)$$

Trong đó:

- $h_\theta(x)$: giá trị dự đoán của mô hình cho biến đầu ra
- $g(z)$: hàm sigmoid (còn được gọi là hàm logistic), được định nghĩa là $g(z) = \frac{1}{1+e^{-z}}$, với $z = \theta^T x$
- θ: vector các tham số của mô hình, cần được tìm ra thông qua quá trình huấn luyện
- x vector các biến đầu vào của mô hình

Hàm sigmoid là một hàm phi tuyến, có dạng giống với hàm tanh. Nó có giá trị giới hạn từ 0 đến 1, và được sử dụng để "biến đổi" giá trị của biến đầu vào z thành một giá trị xác suất $h_\theta(x)$.

Tiếp theo, chúng ta cần xây dựng hàm chi phí (cost function) để đánh giá mức độ "sai lệch" giữa giá trị dự đoán và giá trị thực tế. Hàm chi phí thường được định nghĩa là:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Trong đó:

- m: số lượng điểm dữ liệu trong tập huấn luyện
- $y^{(i)}$:giá trị thực tế của biến đầu ra cho điểm dữ liệu thứ *i*
- $x^{(i)}$:vector các biến đầu vào cho điểm dữ liệu thứ *i*

- λ: hệ số điều chuẩn (regularization parameter), được sử dụng để giảm thiểu overfitting. Nếu $\lambda = 0$, ta sẽ loại bỏ điều chuẩn khỏi công thức tính hàm chi phí

Hàm chi phí này có thể hiểu là tổng quát hóa của hàm entropy cross, đo lường sự khác biệt giữa xác suất dự đoán và xác suất thực tế. Nếu giá trị dự đoán gần với giá trị thực tế, hàm chi phí sẽ gần bằng 0.

Cuối cùng, chúng ta có thể sử dụng thuật toán Gradient Descent để tìm ra giá trị của các tham số θ tối ưu. Thuật toán này được sử dụng để tìm ra giá trị của θ sao cho hàm chi phí $J(\theta)$ đạt giá trị nhỏ nhất. Quá trình cập nhật tham số θ được thực hiện theo công thức sau:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

Trong đó:

- α: learning rate, là hằng số quyết định tốc độ học của thuật toán Gradient Descent.
- $\frac{\partial J(\theta)}{\partial \theta_j}$: đạo hàm riêng của hàm chi phí $J(\theta)$ theo tham số θ_j

Quá trình cập nhật tham số θ được tiếp tục cho đến khi giá trị của hàm chi phí $J(\theta)$ không thay đổi nhiều hoặc đạt đến giá trị nhỏ nhất. Sau đó, ta sử dụng các tham số θ đã tìm được để dự đoán giá trị cho các điểm dữ liệu mới.

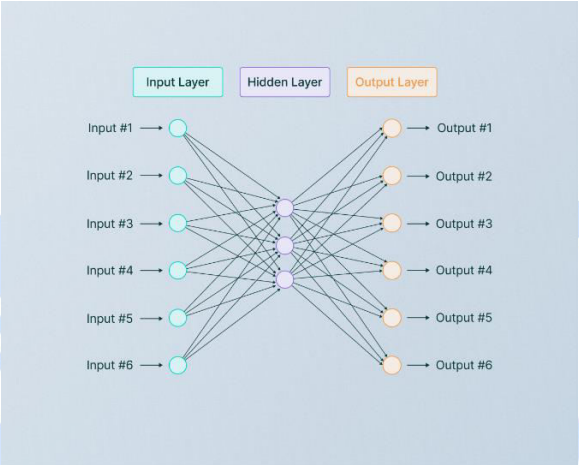
1.2.1.4. Nhận xét

Một trong những ưu điểm chính của Logistic Regression là tính đơn giản và khả năng giải thích rõ ràng. Mô hình cung cấp một hiểu biết rõ ràng về cách mỗi biến đầu vào ảnh hưởng đến biến đầu ra và có thể được giải thích dễ dàng bởi các bên liên quan không chuyên ngành.

Thuật toán Logistic Regression được sử dụng rộng rãi trong nhiều lĩnh vực như chăm sóc sức khỏe, tài chính, tiếp thị và khoa học xã hội để dự đoán kết quả liên quan đến hành vi của khách hàng, chẩn đoán bệnh, đánh giá rủi ro và nhiều ứng dụng khác.

Tóm lại, Logistic Regression là một mô hình đơn giản nhưng hiệu quả cho các bài toán phân loại nhị phân, cung cấp các kết quả dễ giải thích và được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau.

1.2.2. Neural Network



Hình 1.15 Mô hình Neural Netwok đơn giản

Neural Network là một mô hình học sâu được sử dụng rộng rãi để giải quyết các bài toán phân loại, dự đoán và nhận diện. Mô hình này được xây dựng từ nhiều lớp (layer) kết nối với nhau và sử dụng các hàm phi tuyến để giúp mô hình học được các đặc trưng phức tạp của dữ liệu.

Trong bài viết này, chúng ta sẽ phát triển neural network từ logistic regression, một thuật toán học có giám sát đơn giản.

Bước 1: Xây dựng mô hình Logistic Regression

Đầu tiên, chúng ta cần xây dựng mô hình Logistic Regression để làm cơ sở cho neural network. Mô hình này nhận đầu vào là vector đặc trưng của một quan sát và dự đoán xác suất của quan sát đó thuộc về một trong hai nhóm. Công thức của mô hình Logistic Regression là:

$$P(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}$$

Trong đó, θ là vector các trọng số của mô hình, x là vector đặc trưng của quan sát và y là nhãn của quan sát (1 hoặc 0).

Bước 2: Thêm lớp ẩn (hidden layer)

Sau khi xây dựng được mô hình Logistic Regression, chúng ta sẽ thêm một lớp ẩn vào neural network. Lớp ẩn này có thể giúp mô hình học được các đặc trưng phức tạp của dữ liệu. Các đặc trưng này sẽ được trích xuất từ vector đặc trưng ban đầu và truyền qua một hàm phi tuyến để tạo ra các giá trị mới cho lớp tiếp theo.

Bước 3: Xây dựng mô hình Neural Network

Mô hình Neural Network sẽ bao gồm hai lớp: lớp đầu vào và lớp đầu ra. Lớp đầu vào sẽ có số lượng neuron bằng với số chiều của vector đặc trưng của quan sát. Lớp đầu ra sẽ có hai neuron, tương ứng với hai nhóm trong bài toán phân loại.

Lớp ẩn sẽ có số lượng neuron tùy ý, tuy nhiên thường là lớn hơn hoặc bằng với số chiều của vector đặc trưng. Các giá trị đầu ra của lớp ẩn sẽ được truyền qua một hàm phi tuyến để tạo ra các giá trị mới cho lớp tiếp theo.

Bước 4: Huấn luyện mô hình Neural Network

Mô hình Neural Network sẽ được huấn luyện bằng cách tối ưu hàm chi phí (cost function) của mô hình. Hàm chi phí này được xác định bởi việc so sánh giữa dự đoán của mô hình và nhãn thực tế.

Công thức của hàm chi phí cho mô hình Neural Network là:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

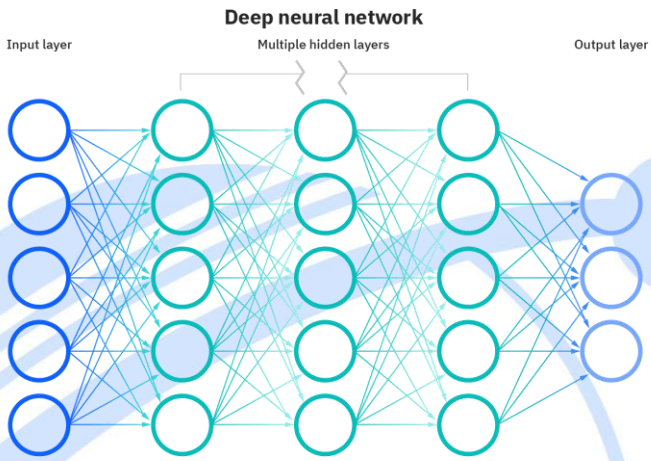
Trong đó, m là số lượng quan sát trong tập huấn luyện, $x^{(i)}$ là vector đặc trưng của quan sát thứ i, $y^{(i)}$ là nhãn của quan sát thứ i và $h_\theta\left(x^{(i)}\right)$ là kết quả dự đoán của mô hình cho quan sát thứ i

Bước 5: Sử dụng mô hình Neural Network để dự đoán

Sau khi huấn luyện mô hình Neural Network, chúng ta có thể sử dụng mô hình này để dự đoán nhãn của các quan sát mới. Các quan sát này sẽ được truyền qua mô hình và mô hình sẽ trả về xác suất của quan sát đó thuộc về mỗi nhóm. Chúng ta có thể chọn nhãn tương ứng với xác suất cao nhất để làm dự đoán cuối cùng.

Tóm lại, phát triển neural network từ logistic regression là một cách tiếp cận phổ biến trong học sâu. Việc này giúp chúng ta xây dựng các mô hình phân loại và dự đoán chính xác hơn, đặc biệt là khi dữ liệu có tính phức tạp.

1.2.3. Deep Neural Network



Hình 1.16 Mô hình Deep Neural Network

Deep Neural Network là một dạng của Neural Network được sử dụng rộng rãi trong các bài toán phân loại. Thuật toán Deep Neural Network cho bài toán phân lớp có thể được trình bày như sau:

1. **Chuẩn bị dữ liệu:** Dữ liệu đầu vào cần được chuẩn bị sao cho có thể đưa vào mô hình. Đối với bài toán phân lớp, dữ liệu cần được chia thành tập huấn luyện và tập kiểm tra.
2. **Khởi tạo trọng số:** Trọng số của các kết nối giữa các neuron được khởi tạo ngẫu nhiên với phân bố Gaussian hoặc Xavier.
3. **Feedforward:** Dữ liệu đầu vào được đưa vào lớp đầu tiên (lớp input) và được truyền qua các lớp ẩn (hidden layers) để tính toán ra dự đoán đầu ra (lớp output). Các kết nối giữa các neuron có trọng số được tính toán dựa trên đầu vào của từng neuron và trọng số của các kết nối đầu vào.
4. **Hàm kích hoạt:** Mỗi neuron ở các lớp ẩn và output sẽ sử dụng một hàm kích hoạt phi tuyến để tính toán đầu ra. Có nhiều loại hàm kích hoạt khác nhau như ReLU, Sigmoid, Tanh,...

5. **Tính toán lỗi:** Để đánh giá hiệu quả của mô hình, chúng ta cần tính toán lỗi giữa dự đoán của mô hình và nhãn thực tế. Hàm lỗi phổ biến được sử dụng là Cross-Entropy Loss.
6. **Backpropagation:** Sau khi tính toán lỗi, chúng ta cần điều chỉnh trọng số của các kết nối giữa các neuron để giảm thiểu lỗi. Việc này được thực hiện thông qua thuật toán backpropagation, trong đó đạo hàm của hàm lỗi được tính toán và lan truyền ngược lại để điều chỉnh trọng số.
7. **Cập nhật trọng số:** Sau khi tính toán đạo hàm và điều chỉnh trọng số, chúng ta cập nhật trọng số mới bằng cách sử dụng một thuật toán tối ưu như Gradient Descent hoặc Adam.
8. **Đánh giá mô hình:** Sau khi huấn luyện xong, chúng ta đánh giá hiệu suất của mô hình trên tập kiểm tra để xác định độ chính xác của mô hình.
9. **Lặp lại quá trình:** Quá trình feedforward, tính toán lỗi, backpropagation và cập nhật trọng số được lặp lại cho đến khi đạt được điều kiện dừng, chẳng hạn như số lượng vòng lặp tối đa hoặc độ chính xác mong muốn.

Deep Neural Network là một phương pháp hiệu quả để giải quyết bài toán phân loại với độ chính xác cao. Tuy nhiên, việc xây dựng và huấn luyện mô hình Deep Neural Network cần đòi hỏi nhiều thời gian và sức mạnh tính toán.

2. Phương pháp chia dữ liệu thực nghiệm

Khái niệm

Khi huấn luyện một mô hình máy học, một trong những điều quan trọng nhất mà ta cần chú ý là mô hình mà ta chọn phải phù hợp với bộ dữ liệu đang xét, và các tham số của mô hình đó phải được điều chỉnh cho phù hợp trong quá trình hoạt động trên dữ liệu, nhằm giúp mô hình đạt được kết quả cao.

Nhưng làm cách nào để ta biết được mô hình và tham số ta đang xét thực sự phù hợp với dữ liệu và có kết quả khả quan?

Đó chính là huấn luyên mô hình trên một phần của bộ dữ liệu đang xét, sau đó thử nghiệm mô hình đã được huấn luyện đó trên phần dữ liệu còn lại. Dựa vào

kết quả thử nghiệm mà cân đối, điều chỉnh các tham số hay thay đổi mô hình sao cho phù hợp với bộ dữ liệu. Quá trình chia dữ liệu ra thành phần huấn luyện và thử nghiệm được gọi là chia dữ liệu thực nghiệm.

Mục đích

Việc chia dữ liệu thành nhiều phần thường nhằm vào hai mục tiêu chính:

- Đánh giá mức độ hiệu quả của mô hình.
- Tránh xáy ra tình trạng: overfitting, underfitting,...

2.1. Phương pháp Holdout

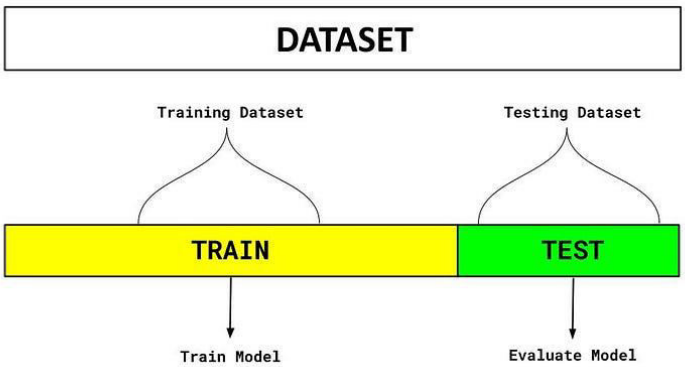
Là phương pháp chia bộ dữ liệu thành tập huấn luyện và thử nghiệm, holdout có nghĩa giữ lại, ý muốn nói dữ liệu sẽ được giữ lại một phần để mô hình đã được huấn luyện có thể thử nghiệm, từ đó đánh giá mức độ hiệu quả của mô hình. [7]

Có hai phương pháp Holdout chính: train_test_split và train_valid_test_split.

2.1.1. Phương pháp train_test_split

2.1.1.1. Khái niệm

Là phương pháp dùng để chia bộ dữ liệu thành hai phần huấn luyện và thử nghiệm (hay còn gọi là training set và testing set).



Hình 2.1 Minh họa cho phương pháp Holdout

2.1.1.2. Cách sử dụng

Để sử dụng phương pháp chia dữ liệu Holdout, cần sử dụng thư viện Scikit-Learn, đặc biệt là phương pháp train_test_split.

```
# Importing the dataset
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Then, loading the Boston Dataset
bhp = datasets.load_boston()

#Finally, creating the Training and Test Split
X_train, X_test, y_train, y_test =
train_test_split(bhp.data, bhp.target, random_state=42,
test_size=0.3)
```

Các tham số được sử dụng để tối ưu phương pháp chia dữ liệu thực nghiệm train_test_split:

- *test_size* (default=None): dữ liệu kiểu số thực hoặc số tự nhiên nhằm xác định kích thước của testing set.
- *train_size* (default=None): dữ liệu kiểu số thực hoặc số tự nhiên nhằm xác định kích thước của training set (nếu chỉ chia dữ liệu thanh 2 tập train và test thì chỉ cần khai báo 1 trong 2 chỉ số test_size, train_size là đủ).
- *random_state* (default=None): dữ liệu kiểu số tự nhiên, nhằm thiết lập sự xáo trộn của dữ liệu theo giá trị.
- *shuffle* (default=None): giá trị Boolean, nhằm xác định dữ liệu có xáo trộn hay không.
- *stratify* (default=None): giá trị kiểu mảng, biểu diễn chỉ số phân tầng của dữ liệu.

Dựa theo đoạn chương trình ở bên trên, ta có bộ dữ liệu Boston được chia thành hai tập *train:test* với tỉ lệ 7:3, chỉ số ngẫu nhiên là 42, và không có sự phân tầng dữ liệu.

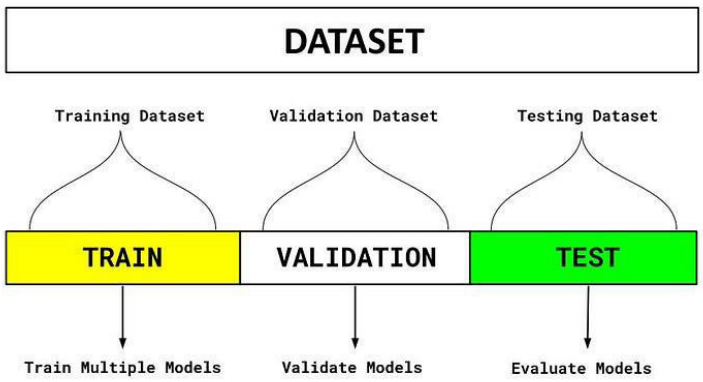
2.1.2. Phương pháp train_valid_test_split

2.1.2.1. Khái niệm

Ta biết được phương pháp Holdout thường chia dữ liệu thành hai tập train và test, khi đó tập test sẽ đảm nhận các vai trò: đánh giá mức độ hiệu quả, tối ưu hóa các tham số và lựa chọn mô hình, hiểu được cách thức mô hình hoạt động trên bộ dữ liệu.

Việc để tập thử nghiệm đảm nhận nhiều vai trò quan trọng có thể khá rủi ro trong quá trình huấn luyện, do nếu tập thử nghiệm không được phân chia một cách công bằng, thiếu khách quan có thể khiến quá trình huấn luyện trở nên kém hiệu quả.

Do đó nhằm phân tán sự phụ thuộc vào tập test, người ta sẽ chia dữ liệu thành 3 tập: training set, validation set và test set (hay còn được gọi là phương pháp train_val_test).



Hình 2.2 Minh họa cho train_valid_test_split

Qua đó tập validation sẽ đảm nhận vai trò là nền tảng tối ưu hóa tham số và mô hình, trong khi tập test để kiểm nghiệm mô hình đã qua huấn luyện ở tập train và tối ưu ở tập validation một cách khách quan.

2.1.2.2. Cách sử dụng

Để chia dữ liệu theo phương pháp train_val_test, ta có sử dụng phương pháp train_test_split như trên, hoặc sử dụng phương pháp train_valid_test_split đã được tích hợp sẵn trong thư viện Scikit-Learn.

- **Sử dụng phương pháp train_test_split:** ta sử dụng 2 lần phép chia train_test_split để chia dữ liệu thành 3 tập train, val và test với tỉ lệ 0.8:0.1:0.1. Bước đầu lấy tập train với tỉ lệ 80%, sau đó chia 20% còn lại cho 2 tập val và test với tỉ lệ 0.5:0.5.

```
# In the first step we will split the data in training
and remaining dataset
X_train, X_rem, y_train, y_rem = train_test_split(X, y,
train_size=0.8)

# Now since we want the valid and test size to be equal
(10% each of overall data).
# we have to define valid_size=0.5 (that is 50% of
remaining data)
X_valid, X_test, y_valid, y_test =
train_test_split(X_rem, y_rem, test_size=0.5)
```

- **Sử dụng phương pháp train_valid_test_split:** cũng tương tự như phương pháp train_test_split, nhưng phương pháp train_valid_test_split sẽ có thêm tham số valid_size, và ta phải khai báo 2 trong 3 tham số train_size, test_size, valid_size.

```
X_train, y_train,X_valid, y_valid, X_test, y_test =
train_valid_test_split(X, y, train_size = 0.8,
valid_size = 0.1, test_size = 0.1)


X_train, y_train,X_valid, y_valid, X_test, y_test =
```

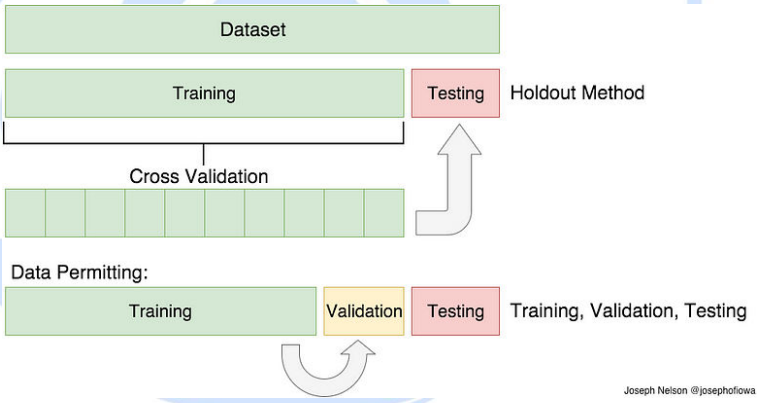
```
train_valid_test_split(X, y, train_size = 0.8,
valid_size = 0.1, test_size = 0.1)
```

2.2. Phương pháp Cross Validation

2.2.1. Khái niệm

Cross Validation (kiểm tra chéo) là phương pháp chia dữ liệu thành nhiều phần khác nhau nhằm thử nghiệm mô hình. [8]

Ý tưởng của Cross Vaidation giống phương pháp Holdout ở chỗ, bộ dữ liệu sẽ được chia thành hai tập train và test. Nhưng khác một điểm, phương pháp Cross Validation sẽ chia bộ dữ liệu thành nhiều phần khác nhau dựa trên số k nhất định, sau đó 1 phần ngẫu nhiên sẽ được dùng làm tập test và các phần còn lại dùng làm tập train. Quá trình này lặp lại k lần cho tới khi k tập con được chia ra đều đã từng là tập test.



Hình 2.3 Minh họa cho Cross Validation

Quá trình kiểm tra như vậy cho phép người dùng có cái nhìn công bằng hơn về bộ dữ liệu, đồng thời cũng đảm bảo không có sự bất công trong việc phân chia dữ liệu vì tất cả các mẫu dữ liệu dù khó hay dễ đều lần lượt trở thành tập train và test.

Phương pháp Cross Validation cũng rất hữu ích khi được sử dụng trên bộ dữ liệu nhỏ, có giới hạn vì nó tận dụng các thông tin trong quá trình huấn luyện, giúp hạn chế tình trạng quá khớp dữ liệu, đồng thời rất hiệu quả trong việc tối ưu tham số và lựa chọn mô hình phù hợp.

K-fold Cross Validation là tên gọi khác cụ thể và chi tiết hơn của Cross Validation, vì cách hoạt động của nó y hệt như định nghĩa có sẵn về Cross Validation. Ví dụ ta có một 10-fold cross validation, đây là cách nó chia bộ dữ liệu và hoạt động.

Hình 2.4 Phân chia bộ dữ liệu với 10-fold

Bộ dữ liệu sẽ được chia thành 10 phần khác nhau thực hiện bằng 10 bước, ở mỗi bước trong đó 1 phần là tập test và 9 phần còn lại sẽ là tập train tạm thời. Tiếp tục như vậy cho đến khi hết 10 bước và tất cả các phần dữ liệu đều là tập test.

2.2.2. Cách sử dụng

Ta sử dụng hàm KFold được tích hợp sẵn trong thư viện Scikit-Learn để áp dụng phương pháp này cho bộ dữ liệu.

Đầu tiên khai báo thư viện và gán giá trị k cho KFold.

```
# import KFold
from sklearn.model_selection import KFold
# Define the split - into 4 folds
KFold(n_splits=4, random_state=None, shuffle=False)
```

Sau đó áp dụng hàm KFold ta vừa khởi tạo được lên bộ dữ liệu.

```
for train_index, test_index in kf.split(X):
     print("TRAIN:", train_index, "TEST:", test_index)
     X_train, X_test = X[train_index], X[test_index]
```

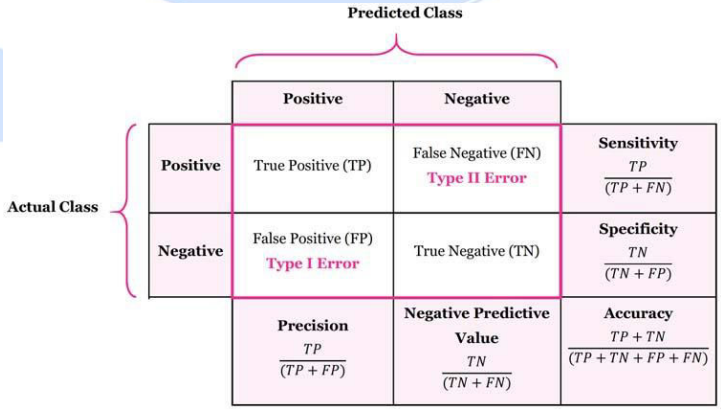
```
y_train, y_test = y[train_index], y[test_index]
```

2.3. So sánh giữa các phương pháp

	Holdout	Cross Validation
Ưu điểm	Ít tiêu tốn thời gian, bộ xử lý hơn so với phương pháp Cross Validation. Đồng thời khá hiệu quả với các bộ dữ liệu lớn.	Có thể thử nghiệm dữ liệu trên mô hình một cách toàn diện và công bằng, giúp ích cho quá trình tối ưu hóa mô hình.
Nhược điểm	Hoạt động kém hiệu quả khi bộ dữ liệu mất cân bằng hay thiếu thông tin, dễ dẫn tới tình trạng quá khớp dữ liệu.	Tiêu tốn nhiều tài nguyên, thời gian để xử lý trong quá trình thử nghiệm mô hình.

3. Độ đo đánh giá mô hình

Confusion Matrix thường được sử dụng cho bài toán phân loại để cung cấp cái nhìn tổng quan về hiệu quả phân lớp của mô hình. Nó chứa thông tin phân loại trên thực tế và dự đoán.



Hình 3.1 Confusion Matrix và những độ đo liên quan

Giả sử trong 100 bệnh nhân chỉ có 5 bệnh nhân được chẩn đoán có bệnh. Lúc này, nhóm đa số là không có bệnh với 95% và nhóm thiểu số là có bệnh với chỉ 5%.

Thông thường nếu chỉ sử dụng *Accuracy* để đánh giá $\left(\frac{0+95}{0+95+0+5} = 0.95 = 95\%\right)$, ta thấy độ chính xác của mô hình đạt 95% cho dù không xác định được nhóm thiểu số, điều này dẫn đến sự đánh giá sai lệch về độ hiệu quả của mô hình. Khi đó ta có thể cân nhắc tới một số metrics thay thế như:

- ***Precision:*** thể hiện độ tin cậy của kết quả khi mô hình xác định một mẫu thuộc về lớp đó (mức độ dự báo chính xác những trường hợp được dự báo là Positive)
- ***Recall:*** thể hiện khả năng mô hình có thể phát hiện được lớp đó (mức độ dự báo chính xác những trường hợp là Positive trong những trường hợp thực tế là Positive)
- ***F1-Score:*** cân bằng giữa *Precision* và *Recall*

Đây là chỉ số thay thế lý tưởng cho *Accuracy* khi mô hình có tỷ lệ mất cân bằng mẫu cao.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Nếu mô hình phân loại dự đoán sai nhóm thiểu số và số lượng False Positive tăng, *Precision* sẽ giảm và F_1 cũng giảm. Ngoài ra, nếu mô hình phân loại kém trong việc phát hiện ra các lớp thiểu số, tức nhiều lớp thiểu số này bị dự đoán sai thành lớp đa số, False Negative sẽ tăng lên và *Recall*, F_1 đều sẽ thấp. *F1-Score* chỉ tăng khi số lượng và chất lượng dự đoán cùng được cải thiện.

Ngoài ra, có thể sử dụng một số độ đo đánh giá khác:

- ***Kappa-Score:*** chỉ số đo lường mức độ liên kết tin cậy (inter-rater reliability) cho các categories.
- ***Gini:*** đo lường sự bất bình đẳng trong phân phối giữa Positive và Negative được dự báo từ mô hình.

- ***AUC:*** biểu diễn mối quan hệ giữa độ nhạy (sensitivity) và độ đặc hiệu (specificity), đánh giá khả năng phân loại good và bad được dự báo từ mô hình.
- ***ROC:*** thể hiện mối quan hệ, sự đánh đổi giữa độ nhạy và tỷ lệ cảnh báo sai; là đường cong biểu diễn tỷ lệ dự báo True Positive Rate (TPR) dựa trên tỷ lệ dự báo False Positive Rate (FPR) tại các ngưỡng Threshold khác nhau.

4. Tham khảo

[1] "Support Vector Machines," [Online]. Available: https://scikit-learn.org/stable/modules/svm.html#regression.

[2] "Chapter 2 : SVM (Support Vector Machine) — Theory," [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[3] "Giới thiệu về Support Vector Machine (SVM)," [Online]. Available: https://viblo.asia/p/gioi-thieu-ve-support-vector-machine-svm-6J3ZgPVElmB.

[4] "Soft Margin Support Vector Machine," [Online]. Available: https://machinelearningcoban.com/2017/04/13/softmarginsmv/.

[5] "Kernel Support Vector Machine," [Online]. Available: https://machinelearningcoban.com/2017/04/22/kernelsmv/.

[6] "Logistic Regression," [Online]. Available: https://machinelearningcoban.com/2017/01/27/logisticregression/.

[7] "Understanding Hold-Out Methods for Training Machine Learning Models," [Online]. Available: https://heartbeat.comet.ml/understanding-hold-out-methods-for-training-machine-learning-models-733f5179d716.

[8] "Train/Test Split and Cross Validation in Python," [Online]. Available: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6.