|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 2.6

“Nghiên cứu các phương pháp thử nghiệm đánh giá mô hình học máy, học sâu”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2022

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 2.6

“Nghiên cứu các phương pháp thử nghiệm đánh giá mô hình học máy học sâu”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2022

MỤC LỤC

[MỤC LỤC 1](#_Toc116022833)

[DANH MỤC HÌNH ẢNH 2](#_Toc116022834)

[CÁC PHƯƠNG PHÁP THỬ NGHIỆM VÀ ĐÁNH GIÁ 3](#_Toc116022835)

[1. Lý do cần đánh giá mô hình 3](#_Toc116022836)

[2. Phương pháp thử nghiệm phục vụ cho đánh giá mô hình 3](#_Toc116022837)

[3. Các độ đo thường được dùng trong mô hình 4](#_Toc116022838)

[3.1. Accuracy 5](#_Toc116022839)

[3.2. Confusion matrix 5](#_Toc116022840)

[3.3. True/False Positive/Negative 10](#_Toc116022841)

[3.4. Receiver Operating Characteristic curve 11](#_Toc116022842)

[3.5. Area Under the Curve 15](#_Toc116022843)

[3.6. Precision và Recall 15](#_Toc116022844)

[3.7. F1-score 17](#_Toc116022845)

[TÀI LIỆU THAM KHẢO 19](#_Toc116022846)

# DANH MỤC HÌNH ẢNH

[Hình 1.1: *unnormalized confusion matrix*và *normalized confusion matrix* 9](#_Toc116022847)

[Hình 1.2: Receiver Operating Characteristic curve và Area Under the Curve. 14](#_Toc116022848)

[Hình 1.3: Cách tính Precision và Recall. 15](#_Toc116022849)

# CÁC PHƯƠNG PHÁP THỬ NGHIỆM VÀ ĐÁNH GIÁ

## 1. Lý do cần đánh giá mô hình

Khi đã xây dựng một mô hình machine learning và huấn luyện nó trên một tập dữ liệu, điều nên làm là đánh giá hiệu năng của mô hình trên tập dữ liệu mới.

Việc đánh giá mô hình giúp giải quyết những vấn đề sau:

* Mô hình đã được huấn luyện thành công hay chưa?
* Mức độ thành công của mô hình tốt đến đâu?
* Khi nào nên dừng quá trình huấn luyện?
* Khi nào nên cập nhật mô hình?

Trả lời được 4 câu hỏi trên, có thể quyết định mô hình này có thực sự phù hợp cho bài toán hay không.

Đánh giá một mô hình có tốt hay không thường được thực hiện trên dữ liệu mà mô hình chưa được huấn luyện. Tỷ lệ thường thấy của một tập dữ liệu huấn luyện so với tập dữ liệu thử nghiệm là 70% và 30%.

Ta sử dụng dữ liệu mới khi đánh giá mô hình nhằm giảm thiểu khả năng quá khớp (overfitting) đối với tập huấn luyện. Đôi khi sẽ hữu ích khi đánh giá mô hình và cùng lúc huấn luyện nó để tìm ra các chỉ số tốt nhất của một mô hình. Tuy nhiên, không thể sử dụng bộ thử nghiệm để thực hiện đánh giá này. Có thể sẽ phải chọn các thông số hoạt động tốt nhất trên dữ liệu thử nghiệm, nhưng có thể không phải là các tham số bao quát nhất.

## 2. Phương pháp thử nghiệm phục vụ cho đánh giá mô hình

Để có thể đánh giá mô hình trong khi xây dựng và chuẩn hóa nó, chúng ta có thể tạo ra một tập dữ liệu con - được biết đến là tập dữ liệu đánh giá. Một tập dữ liệu điển hình sẽ có tỷ lệ 60% cho tập huấn luyện, 20% cho tập đánh giá và 20% cho tập thử nghiệm. Phương pháp thường dùng là sử dụng k-fold để phân chia bộ dữ liệu. Người sử dụng thường sử dụng 5-fold hoặc 10-fold.

Một điều rất quan trọng là độ hỗn loạn của tập dữ liệu có thể được tăng lên đáng kể khi bạn xáo trộn dữ liệu trước khi chia tập dữ liệu ra thành nhiều phần nhỏ. Điều này giúp cho mỗi tập dữ liệu con có thể thể hiện rõ ràng hơn tính chất của tập dữ liệu lớn.

Ngoài ra, có các phương pháp chia dataset khác như Hold-out/Repeated Hold-out (thường được sử dụng khi dataset lớn), Stratified sampling (dùng cho imbalanced dataset), Cross-Validation, Bootstrap Sampling, v.v.

## 3. Các độ đo thường được dùng trong mô hình

Khi xây dựng một mô hình Machine Learning, chúng ta cần một phép đánh giá để xem mô hình sử dụng có hiệu quả không và để so sánh khả năng của các mô hình. Trong bài viết này, tôi sẽ giới thiệu các phương pháp đánh giá các mô hình classification.

Hiệu năng của một mô hình thường được đánh giá dựa trên tập dữ liệu kiểm thử (test data). Cụ thể, giả sử đầu ra của mô hình khi đầu vào là tập kiểm thử được mô tả bởi vector y\_pred - là vector dự đoán đầu ra với mỗi phần tử là class được dự đoán của một điểm dữ liệu trong tập kiểm thử. Ta cần so sánh giữa vector dự đoán y\_pred này với vector class thật của dữ liệu, được mô tả bởi vector y\_true.

Ví dụ với bài toán có 3 lớp dữ liệu được gán nhãn là 0, 1, 2. Trong bài toán thực tế, các class có thể có nhãn bất kỳ, không nhất thiết là số, và không nhất thiết bắt đầu từ 0. Chúng ta hãy tạm giả sử các class được đánh số từ 0 đến C-1 trong trường hợp có C lớp dữ liệu. Có 10 điểm dữ liệu trong tập kiểm thử với các nhãn thực sự được mô tả bởi y\_true = [0, 0, 0, 0, 1, 1, 1, 2, 2, 2]. Giả sử bộ phân lớp chúng ta đang cần đánh giá dự đoán nhãn cho các điểm này là y\_pred = [0, 1, 0, 2, 1, 1, 0, 2, 1, 2].

Có rất nhiều cách đánh giá một mô hình phân lớp. Tuỳ vào những bài toán khác nhau mà chúng ta sử dụng các phương pháp khác nhau. Các phương pháp thường được sử dụng là: accuracy score, confusion matrix, ROC curve, Area Under the Curve, Precision and Recall, F1 score, Top R error, etc.

### 3.1. Accuracy

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.

Trong ví dụ này, ta có thể đếm được có 6 điểm dữ liệu được dự đoán đúng trên tổng số 10 điểm. Vậy ta kết luận độ chính xác của mô hình là 0.6 (hay 60%). Để ý rằng đây là bài toán với chỉ 3 class, nên độ chính xác nhỏ nhất đã là khoảng 1/3, khi tất cả các điểm được dự đoán là thuộc vào một class nào đó.

**from** \_\_future\_\_ **import** print\_function

**import** numpy **as** np

**def** **acc**(y\_true, y\_pred):

correct **=** np.sum(y\_true **==** y\_pred)

**return** float(correct)**/**y\_true.shape[0]

y\_true **=** np.array([0, 0, 0, 0, 1, 1, 1, 2, 2, 2])

y\_pred **=** np.array([0, 1, 0, 2, 1, 1, 0, 2, 1, 2])

**print**('accuracy = ', acc(y\_true, y\_pred))

accuracy = 0.6

Để sử dụng thư viện có thể dùng:

**from** sklearn.metrics **import** accuracy\_score

**print**('accuracy = ',accuracy\_score(y\_true, y\_pred))

accuracy = 0.6

### 3.2. Confusion matrix

Cách tính sử dụng accuracy như ở trên chỉ cho chúng ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, chúng ta sử dụng một ma trận được gọi là confusion matrix.

Về cơ bản, confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class. Để hiểu rõ hơn, hãy xem bảng dưới đây:

Total: 10 | Predicted | Predicted | Predicted |

| as: 0 | as: 1 | as: 2 |

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

True: 0 | 2 | 1 | 1 | 4

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

True: 1 | 1 | 2 | 0 | 3

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

True: 2 | 0 | 1 | 2 | 3

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

Có tổng cộng 10 điểm dữ liệu. Chúng ta xét ma trận tạo bởi các giá trị tại vùng 3x3 trung tâm của bảng.

Ma trận thu được được gọi là confusion matrix. Nó là một ma trận vuông với kích thước mỗi chiều bằng số lượng lớp dữ liệu. Giá trị tại hàng thứ i, cột thứ j là số lượng điểm **lẽ ra thuộc vào class i nhưng lại được dự đoán là thuộc vào class j**. Như vậy, nhìn vào hàng thứ nhất (0), ta có thể thấy được rằng trong số bốn điểm thực sự thuộc lớp 0, chỉ có hai điểm được phân loại đúng, hai điểm còn lại bị phân loại nhầm vào lớp 1 và lớp 2.

***Chú ý:*** Có một số tài liệu định nghĩa ngược lại, tức giá trị tại ***cột*** thứ *i*, ***hàng*** thứ *j* là số lượng điểm lẽ ra thuộc vào class *i* nhưng lại được dự đoán là thuộc vào class *j*. Khi đó ta sẽ được confusion matrix là ma trận chuyển vị của confusion matrix như cách tôi đang làm.

Ta có thể suy ra ngay rằng tổng các phần tử trong toàn ma trận này chính là số điểm trong tập kiểm thử. Các phần tử trên đường chéo của ma trận là số điểm được phân loại đúng của mỗi lớp dữ liệu. Từ đây có thể suy ra accuracy chính bằng tổng các phần tử trên đường chéo chia cho tổng các phần tử của toàn ma trận. Đoạn code dưới đây mô tả cách tính confusion matrix:

**def** **my\_confusion\_matrix**(y\_true, y\_pred):

N **=** np.unique(y\_true).shape[0] *# number of classes*

cm **=** np.zeros((N, N))

**for** n **in** range(y\_true.shape[0]):

cm[y\_true[n], y\_pred[n]] **+=** 1

**return** cm

cnf\_matrix **=** my\_confusion\_matrix(y\_true, y\_pred)

**print**('Confusion matrix:')

**print**(cnf\_matrix)

**print**('\nAccuracy:', np.diagonal(cnf\_matrix).sum()**/**cnf\_matrix.sum())

Confusion matrix:

[[ 2. 1. 1.]

[ 1. 2. 0.]

[ 0. 1. 2.]]

Accuracy: 0.6

Cách biểu diễn trên đây của confusion matrix còn được gọi là unnormalized confusion matrix, tức ma confusion matrix chưa chuẩn hoá. Để có cái nhìn rõ hơn, ta có thể dùng normalized confuion matrix, tức confusion matrix được chuẩn hoá. Để có normalized confusion matrix, ta lấy mỗi hàng của unnormalized confusion matrix sẽ được chia cho tổng các phần tử trên hàng đó. Như vậy, ta có nhận xét rằng tổng các phần tử trên một hàng của normalized confusion matrix luôn bằng 1. Điều này thường không đúng trên mỗi cột. Dưới đây là cách tính normalized confusion matrix:

normalized\_confusion\_matrix **=** cnf\_matrix**/**cnf\_matrix.sum(axis **=** 1, keepdims **=** True)

**print**('\nConfusion matrix (with normalizatrion:)')

**print**(normalized\_confusion\_matrix)

Confusion matrix (with normalizatrion:)

[[ 0.5 0.25 0.25 ]

[ 0.33333333 0.66666667 0. ]

[ 0. 0.33333333 0.66666667]]

Sử dụng thư viện:

**from** sklearn.metrics **import** confusion\_matrix

cnf\_matrix **=** confusion\_matrix(y\_true, y\_pred)

**print**('Confusion matrix:')

**print**(cnf\_matrix)

Confusion matrix:

[[2 1 1]

[1 2 0]

[0 1 2]]

Confusion matrix thường được minh hoạ bằng màu sắc để có cái nhìn rõ ràng hơn. Đoạn code dưới đây giúp hiển thị confusion matrix ở cả hai dạng:

**import** matplotlib.pyplot **as** plt

**import** itertools

**def** **plot\_confusion\_matrix**(cm, classes,

normalize**=**False,

title**=**'Confusion matrix',

cmap**=**plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

**if** normalize:

cm **=** cm.astype('float') **/** cm.sum(axis**=**1, keepdims **=** True)

plt.imshow(cm, interpolation**=**'nearest', cmap**=**cmap)

plt.title(title)

plt.colorbar()

tick\_marks **=** np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation**=**45)

plt.yticks(tick\_marks, classes)

fmt **=** '.2f' **if** normalize **else** 'd'

thresh **=** cm.max() **/** 2.

**for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment**=**"center",

color**=**"white" **if** cm[i, j] **>** thresh **else** "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

*# Plot non-normalized confusion matrix*

class\_names **=** [0, 1, 2]

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes**=**class\_names,

title**=**'Confusion matrix, without normalization')

*# Plot normalized confusion matrix*

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes**=**class\_names, normalize**=**True,

title**=**'Normalized confusion matrix')

plt.show()

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

Hình 1.1: unnormalized confusion matrix và normalized confusion matrix

Với các bài toán với nhiều lớp dữ liệu, cách biểu diễn bằng màu này rất hữu ích. Các ô màu đậm thể hiện các giá trị cao. Một mô hình tốt sẽ cho một confusion matrix có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ. Nói cách khác, khi biểu diễn bằng màu sắc, đường chéo có màu càng đậm so với phần còn lại sẽ càng tốt. Từ hai hình trên ta thấy rằng confusion matrix đã chuẩn hoá mang nhiều thông tin hơn. Sự khác nhau được thấy ở ô trên cùng bên trái. Lớp dữ liệu 0 được phân loại không thực sự tốt nhưng trong unnormalized confusion matrix, nó vẫn có màu đậm như hai ô còn lại trên đường chéo chính

### 3.3. True/False Positive/Negative

Cách đánh giá này thường được áp dụng cho các bài toán phân lớp có hai lớp dữ liệu. Cụ thể hơn, trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác. Ví dụ, trong bài toán xác định có bệnh ung thư hay không thì việc không bị sót (miss) quan trọng hơn là việc chẩn đoán nhầm âm tính thành dương tính. Trong bài toán xác định có mìn dưới lòng đất hay không thì việc bỏ sót nghiêm trọng hơn việc báo động nhầm rất nhiều. Hay trong bài toán lọc email rác thì việc cho nhầm email quan trọng vào thùng rác nghiêm trọng hơn việc xác định một email rác là email thường.

Trong những bài toán này, người ta thường định nghĩa lớp dữ liệu quan trọng hơn cần được xác định đúng là lớp Positive (P-dương tính), lớp còn lại được gọi là Negative (N-âm tính). Ta định nghĩa True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) dựa trên confusion matrix chưa chuẩn hoá như sau:

| Predicted | Predicted |

| as Positive | as Negative |

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

Actual: Positive | True Positive (TP) | False Negative (FN) |

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

Actual: Negative | False Positive (FP) | True Negative (TN) |

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

Người ta thường quan tâm đến TPR, FNR, FPR, TNR (R - Rate) dựa trên normalized confusion matrix như sau:

| Predicted | Predicted |

| as Positive | as Negative |

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

Actual: Positive | TPR = TP/(TP + FN) | FNR = FN/(TP + FN) |

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

Actual: Negative | FPR = FP/(FP + TN) | TNR = TN/(FP + TN) |

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

False Positive Rate còn được gọi là False Alarm Rate (tỉ lệ báo động nhầm), False Negative Rate còn được gọi là Miss Detection Rate (tỉ lệ bỏ sót). Trong bài toán dò mìn, thà báo nhầm còn hơn bỏ sót, tức là ta có thể chấp nhận False Alarm Rate cao để đạt được Miss Detection Rate thấp.

**Chú ý:**

* Việc biết một cột của confusion matrix này sẽ suy ra được cột còn lại vì tổng các hàng luôn bằng 1 và chỉ có hai lớp dữ liệu.
* **Với các bài toán có nhiều lớp dữ liệu**, ta có thể xây dựng bảng True/False Positive/Negative cho **mỗi lớp** nếu coi lớp đó là lớp Positive, các lớp còn lại gộp chung thành lớp Negative, giống như cách làm trong [one-vs-rest](https://machinelearningcoban.com/2017/02/11/binaryclassifiers/#one-vs-rest-hay-one-hot-coding).

### 3.4. Receiver Operating Characteristic curve

Trong một số bài toán, việc tăng hay giảm FNR, FPR có thể được thực hiện bằng việc thay đổi một ngưỡng (threshold) nào đó. Lấy ví dụ khi ta sử dụng thuật toán [Logistic Regression](https://machinelearningcoban.com/2017/01/27/logisticregression/), đầu ra của mô hình có thể là các lớp cứng 0 hay 1, hoặc cũng có thể là các giá trị thể hiện xác suất để dữ liệu đầu vào thuộc vào lớp 1. Khi sử dụng thư viện [sklearn Logistic Regression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html), ta có thể lấy được các giá trị xác xuất này bằng phương thức predict\_proba(). Mặc định, ngưỡng được sử dụng là 0.5, tức là một điểm dữ liệu x sẽ được dự đoán rơi vào lớp 1 nếu giá trị predict\_proba(x) lớn hơn 0.5 và ngược lại.

Nếu bây giờ ta coi lớp 1 là lớp Positive, lớp 0 là lớp Negative, câu hỏi đặt ra là làm thế nào để tăng mức độ báo nhầm (FPR) để giảm mức độ bỏ sót (FNR)? Chú ý rằng tăng FNR đồng nghĩa với việc giảm TPR vì tổng của chúng luôn bằng 1.

Một kỹ thuật đơn giản là ta thay giá trị threshold từ 0.5 xuống một số nhỏ hơn. Chẳng hạn nếu chọn threshold = 0.3, thì mọi điểm được dự đoán có xác suất đầu ra lớn hơn 0.3 sẽ được dự đoán là thuộc lớp Positive. Nói cách khác, tỉ lệ các điểm được phân loại là Positive sẽ tăng lên, kéo theo cả False Positive Rate và True Positive Rate cùng tăng lên (cột thứ nhất trong ma trận tăng lên). Từ đây suy ra cả FNR và TNR đều giảm.

Ngược lại, nếu ta muốn bỏ sót còn hơn báo nhầm, tất nhiên là ở mức độ nào đó, như bài toán xác định email rác chẳng hạn, ta cần tăng threshold lên một số lớn hơn 0.5. Khi đó, hầu hết các điểm dữ liệu sẽ được dự đoán thuộc lớp 0, tức Negative, và cả TNF và FNR đều tăng lên, tức TPR và FPR giảm xuống.

Như vậy, ứng với mỗi giá trị của threshold, ta sẽ thu được một cặp (FPR, TPR). Biểu diễn các điểm (FPR, TPR) trên đồ thị khi thay đổi threshold từ 0 tới 1 ta sẽ thu được một đường được gọi là Receiver Operating Characteristic curve hay ROC curve. (Chú ý rằng khoảng giá trị của threshold không nhất thiết từ 0 tới 1 trong các bài toán tổng quát. Khoảng giá trị này cần được đảm bảo có trường hợp TPR/FPR nhận giá trị lớn nhất hay nhỏ nhất mà nó có thể đạt được).

Dưới đây là một ví dụ với hai lớp dữ liệu. Lớp thứ nhất là lớp Negative có 20 điểm dữ liệu, 30 điểm còn lại thuộc lớp Positive. Giả sử mô hình đang xét cho các đầu ra của dữ liệu (xác suất) được lưu ở biến scores.

*# generate simulated data*

n0, n1 **=** 20, 30

score0 **=** np.random.rand(n0)**/**2

label0 **=** np.zeros(n0, dtype **=** int)

score1 **=** np.random.rand(n1)**/**2 **+** .2

label1 **=** np.ones(n1, dtype **=** int)

scores **=** np.concatenate((score0, score1))

y\_true **=** np.concatenate((label0, label1))

**print**('True labels:')

**print**(y\_true)

**print**('\nScores:')

**print**(scores)

True labels:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1]

Scores:

[ 0.16987517 0.27608323 0.10851568 0.13395249 0.24878687 0.29100097

0.21036182 0.48215779 0.01930099 0.30927599 0.26581374 0.15141354

0.26298063 0.10405583 0.30773121 0.39830016 0.04868077 0.17290186

0.28717646 0.3340749 0.4174846 0.27292017 0.68740357 0.62108568

0.20781968 0.43056031 0.67816027 0.47037842 0.23118192 0.68862749

0.24559788 0.58645887 0.69637251 0.5247967 0.24265087 0.60485646

0.54800088 0.69565411 0.20509934 0.39638029 0.30860676 0.6267616

0.42360257 0.5507021 0.50313701 0.67614457 0.60108083 0.25201502

0.27830655 0.58669514]

Nhìn chung, các điểm thuộc lớp 1 có score cao hơn. Thư viện sklearn sẽ giúp chúng ta tính các thresholds cũng như FPR và TPR tương ứng:

**from** sklearn.metrics **import** roc\_curve, auc

fpr, tpr, thresholds **=** roc\_curve(y\_true, scores, pos\_label **=** 1)

**print**('Thresholds:')

**print**(thresholds)

Thresholds:

[ 0.69637251 0.50313701 0.48215779 0.4174846 0.39830016 0.39638029

0.30927599 0.30860676 0.28717646 0.27830655 0.27608323 0.27292017

0.26298063 0.25201502 0.24878687 0.23118192 0.21036182 0.20509934

0.01930099]

**print**('False Positive Rate:')

**print**(fpr)

False Positive Rate:

[ 0. 0. 0.05 0.05 0.1 0.1 0.2 0.2 0.35 0.35 0.4 0.4

0.5 0.5 0.55 0.55 0.6 0.6 1. ]

**print**('True Positive Rate:')

tpr

True Positive Rate:

array([ 0.03333333, 0.53333333, 0.53333333, 0.66666667, 0.66666667,

0.7 , 0.7 , 0.73333333, 0.73333333, 0.76666667,

0.76666667, 0.8 , 0.8 , 0.83333333, 0.83333333,

0.93333333, 0.93333333, 1. , 1. ])

Như vậy, ứng với threshold = 0.69637251, fpr = 0 và tpr = 0.03. Đây không phải là một ngưỡng tốt vì mặc dụ False Positive Rate thấp, True Positive Rate cũng rất thấp. Chúng ta luôn muốn rằng FPR thấp và TPR cao.

ROC cho bài toán này được minh hoạ như dưới đây:

**import** matplotlib.pyplot **as** plt

**from** itertools **import** cycle

plt.figure()

lw **=** 2

plt.plot(fpr, tpr, color**=**'darkorange',

lw**=**lw, label**=**'ROC curve (area = %0.2f)' **%** auc(fpr, tpr))

plt.plot([0, 1], [0, 1], color**=**'navy', lw**=**lw, linestyle**=**'--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc**=**"lower right")

plt.show()

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

Hình 1.2: Receiver Operating Characteristic curve và Area Under the Curve.

### 3.5. Area Under the Curve

Dựa trên ROC curve, ta có thể chỉ ra rằng một mô hình có hiệu quả hay không. Một mô hình hiệu quả khi có FPR thấp và TPR cao, tức tồn tại một điểm trên ROC curve gần với điểm có toạ độ (0, 1) trên đồ thị (góc trên bên trái). Curve càng gần thì mô hình càng hiệu quả.

Có một thông số nữa dùng để đánh giá mà tôi đã sử dụng ở trên được gọi là Area Under the Curve hay AUC. Đại lượng này chính là diện tích nằm dưới ROC curve màu cam. Giá trị này là một số dương nhỏ hơn hoặc bằng 1. Giá trị này càng lớn thì mô hình càng tốt.

**Chú ý:** [Cross validation](https://machinelearningcoban.com/2017/03/04/overfitting/#-cross-validation) cũng có thể được thực hiện bằng cách xác định ROC curve và AUC lên [validation set].

### 3.6. Precision và Recall

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.

Trước hết xét bài toán phân loại nhị phân. Ta cũng coi một trong hai lớp là *positive*, lớp còn lại là *negative*.

Xét Hình 1.3 dưới đây:



Hình 1.3: Cách tính Precision và Recall.

Với một cách xác định một lớp là *positive*, **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).

Một cách toán học, Precison và Recall là hai phân số có tử số bằng nhau nhưng mẫu số khác nhau:

Precision=TPTP+FP Recall=TPTP+FNPrecision=TPTP+FP Recall=TPTP+FN

Bạn đọc có thể nhận thấy rằng TPR và Recall là hai đại lượng bằng nhau. Ngoài ra, cả Precision và Recall đều là các số không âm nhỏ hơn hoặc bằng một.

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 *positive* là thấp.

Ví dụ nhỏ dưới đây thể hiện cách tính Precision và Recall dựa vào Confusion Matrix cho bài toán phân loại nhị phân.

**from** \_\_future\_\_ **import** print\_function

**import** numpy **as** np

*# confusion matrix to precision + recall*

**def** **cm2pr\_binary**(cm):

p **=** cm[0,0]**/**np.sum(cm[:,0])

r **=** cm[0,0]**/**np.sum(cm[0])

**return** (p, r)

*# example of a confusion matrix for binary classification problem*

cm **=** np.array([[100., 10], [20, 70]])

p,r **=** cm2pr\_binary(cm)

**print**("precition = {0:.2f}, recall = {1:.2f}".format(p, r))

precition = 0.83, recall = 0.91

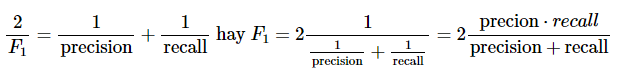
Khi Precision = 1, mọi điểm tìm được đều thực sự là *positive*, tức không có điểm *negative* nào lẫn vào kết quả. Tuy nhiên, Precision = 1 không đảm bảo mô hình là tốt, vì câu hỏi đặt ra là liệu mô hình đã tìm được tất cả các điểm *positive* hay chưa. Nếu một mô hình chỉ tìm được đúng một điểm *positive* mà nó chắc chắn nhất thì ta không thể gọi nó là một mô hình tốt.

Khi Recall = 1, mọi điểm *positive* đều được tìm thấy. Tuy nhiên, đại lượng này lại không đo liệu có bao nhiêu điểm *negative* bị lẫn trong đó. Nếu mô hình phân loại mọi điểm là *positive* thì chắc chắn Recall = 1, tuy nhiên dễ nhận ra đây là một mô hình cực tồi.

Một mô hình phân lớp tốt là mô hình có cả Precision và Recall đều cao, tức càng gần một càng tốt. Có hai cách đo chất lượng của bộ phân lớp dựa vào Precision và Reall: Precision-Recall curve và F-score.

### 3.7. F1-score

$F\_1$ score, hay F1-score, là *harmonic mean* của precision và recall (giả sử rằng hai đại lượng này khác không):

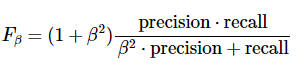


F1-score có giá trị nằm trong nửa khoảng (0,1]. F1 càng cao, bộ phân lớp càng tốt. Khi cả recall và precision đều bằng 1 (tốt nhất có thể), F1=1. Khi cả recall và precision đều thấp, ví dụ bằng 0.1, F1=0.1. Dưới đây là một vài ví dụ về F1

| **precision** | **recall** | F1F1 |
| --- | --- | --- |
| 1 | 1 | 1 |
| 0.1 | 0.1 | 0.1 |
| 0.5 | 0.5 | 0.5 |
| 1 | 0.1 | 0.182 |
| 0.3 | 0.8 | 0.36 |

Như vậy, một bộ phân lớp với precision = recall = 0.5 tốt hơn một bộ phân lớp khác với precision = 0.3, recall = 0.8 theo cách đo này.

Trường hợp tổng quát của F1 score là Fβ score:



F1 chính là một trường hợp đặc biệt của Fβ khi β=1. Khi β>1, recall được coi trọng hơn precision, khi β<1, precision được coi trọng hơn. Hai đại lượng β thường được sử dụng là β=2 và β=0.5.

# TÀI LIỆU THAM KHẢO

1. Thuynt, Các phương pháp đánh giá một mô hình phân lớp dữ liệu, http://hoctructuyen123.net/cac-phuong-phap-danh-gia-mot-mo-hinh-phan-lop-du-lieu/.
2. Quy Nguyen, Bài toán phân lớp và các phương pháp đánh giá, <https://ndquy.github.io/posts/bai-toan-phan-lop-va-danh-gia/>
3. X. Su, W. Shi, X. Qu, Y. Zheng and X. Liu, “DroidDeep: using Deep Belief Network to characterize and detect android malware,” Soft Computing, vol. 24, pp. 6017-6030, 2020.
4. X. Qin, F. Zeng and Y. Zhang, “MSNdroid: the Android malware detector based on multi-class features and deep belief network,” in Proceedings of the ACM Turing Celebration Conference, New York, NY, USA, no. 44, pp. 1-5, 2019.
5. M. K. Alzaylaee, S. Y. Yerima and S. Sezer, “DL-Droid: Deep learning based android malware detection using real devices,” Computers & Security, vol. 89, pp. 101663, 2020.
6. F. Mercaldo and A. Santone, “Deep learning for image-based mobile malware detection,” Computer Virology and Hacking Techniques, vol. 16, pp. 157-171, 2020.
7. Z. Ma, H. Ge, Z. Wang, Y. Liu and X. Liu, “Droidetec: Android malware detection and malicious code localization through deep learning,” ArXiv, vol. abs/2002.0394, 2020.
8. W. Y. Lee, J. Saxe and R. Harang, “SeqDroid: obfuscated Android malware detection using stacked convolutional and recurrent neural networks,” In book: Deep Learning Applications for Cyber Security, pp. 197-210, 2019.
9. W. Wang, M. Zhao and J. Wang, “Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network,” Journal of Ambient Intelligence and Humanized Computing, vol. 10, pp. 3035-3043, 2019.
10. D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei et al., “IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture,” Computer Networks, vol. 171, pp. 107-138, 2020.
11. D. Zhu, T. Xi, P. Jing, D. Wu, Q. Xia et al., “A transparent and multimodal malware detection method for Android Apps,” in Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, New York, NY, United States, pp. 51-60, 2019.