**Các Phương Pháp Đánh Giá Một Hệ Thống Phân Lớp(Model Evaluation)**

**Giới Thiệu:**

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.

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.

**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.

**import** numpy **as** np *#numpy để làm việc với mảng***from** sklearn.naive\_bayes **import** GaussianNB *#thư viện dùng naive-bayes***from** sklearn.metrics **import** accuracy\_score *#hàm accuracy\_score trong sklearn  
# Dữ liệu huấn luyện đầu vào*training\_points = [[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]] *#các điểm*training\_labels = [1, 1, 1, 2, 2, 2] *# các nhãn*X = np.array(training\_points) *#*Y = np.array(training\_labels)  
*# Tạo Naive Bayes classifier*clf = GaussianNB()  
clf.fit(X, Y) *#train model với naive bayes  
# dữ liệu test*test\_points = [[1, 1], [2, 2], [3, 3], [4, 3]]  
test\_labels = [2, 2, 2, 1]  
predicts = clf.predict(test\_points) *#dựa vào model sau khi đã huấn luyện, dự đoán dữ liệu test  
# Kết quả Accuracy tính bằng tay*count = len([**"ok" for** idx, label **in** enumerate(test\_labels) **if** label == predicts[idx]])  
print (**"Kết quả Accuracy được tính bằng tay là: %f"** % (float(count) / len(test\_labels)))  
*# Calculate Accuracy Rate by using accuracy\_score()*print (**"Kết quả Accuracy dùng hàm accuracy\_score() trong sklearn là: %f"** % accuracy\_score(test\_labels, predicts))

*Kết quả:*

Kết quả Accuracy được tính bằng tay là: 0.750000

Kết quả Accuracy dùng hàm accuracy\_score() trong sklearn là: 0.750000

**Confution Matrix (CM)**

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:



Có tổng cộng **165** điểm dữ liệu. Chúng ta xét ma trận tạo bởi các giá trị tại vùng **2x2** 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 **60** điểm thực sự thuộc lớp **0**, chỉ có một điểm được phân loại đúng, điểm còn lại bị phân loại nhầm vào lớp **1**.

Chúng 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:

**import** numpy **as** np  
**def** confusion\_matrix(y\_true,y\_pred): *# xây dựng hàm tính ma trận* N= np.unique(y\_true).shape[0] *#tính số lớp của ma trận* cm = np.zeros((N,N)) *#khởi tạo ma trận 0 NxN lớp* **for** i **in** range(y\_true.shape[0]): *# lấy từng phần tử trong ma trận y\_true* cm[y\_true[i],y\_pred[i]] +=1  
 **return** cm  
  
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])  
cnf\_matrix = confusion\_matrix(y\_true, y\_pred) *#gọi hàm*print(**'Confusion matrix là :'**)  
print(cnf\_matrix)  
*#Độ chính xác Accuracy được tính bằng tổng các thành phần trên đường chéo chính chia cho tổng các thành phần trên ma trận*print(**'\nĐộ chính xác Accuracy: {}%'**.format(int(np.diagonal(cnf\_matrix).sum()/cnf\_matrix.sum()\*100)))

*Kết quả:*

Confusion matrix là :

[[ 2. 1. 1.]

[ 1. 2. 0.]

[ 0. 1. 2.]]

Độ chính xác Accuracy: 60%

**True/False Positive/Negative**

Cách biểu diễn trên đây của **confusion matrix** còn được gọi là **unnormalized confusion matrix**, tức **CM** chưa chuẩn hoá. Để có cái nhìn rõ hơn, ta có thể dùng **normalized confuion matrix**, tức **CM** 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(**"Normalized confusion matrix là:\n {}"**.format(normalized\_confusion\_matrix))

*Kết quả:*

Normalized confusion matrix là:

[[ 0.5 0.25 0.25 ]

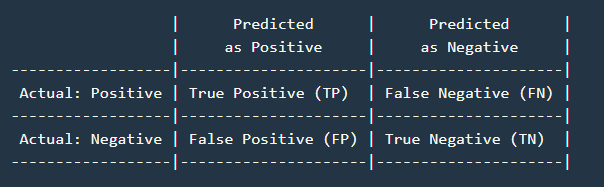
[ 0.33333333 0.66666667 0. ]

[ 0. 0.33333333 0.66666667]]

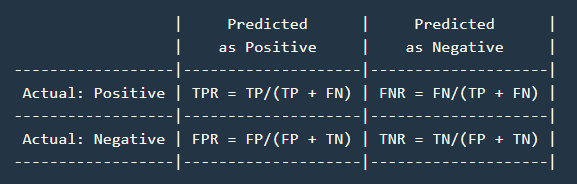
**True/False Positive/Negative**

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:



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



**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**.

**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**, 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***, 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*.

**import** numpy **as** np  
**from** sklearn.metrics **import** roc\_curve, auc  
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))  
fpr, tpr, thresholds = roc\_curve(y\_true, scores, pos\_label = 1)  
print(**"True lables: \n{}\nScores: \n{}\nThresholds:\n{}\nFPR:\n{}\nTPR: \n{}\n"**.format(y\_true,scores,thresholds,fpr,tpr))

*Kết quả:*

True lables:

[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:

[ 0.00399151 0.25826574 0.16858168 0.03329907 0.37570748 0.20519163

0.14683173 0.33808756 0.30511688 0.34295643 0.42260856 0.10433163

0.2062915 0.04314333 0.22564941 0.30344634 0.41291515 0.14160504

0.30188107 0.27875212 0.2734653 0.49132647 0.3026426 0.30907854

0.52039594 0.64291785 0.56946258 0.651503 0.68220519 0.47964084

0.63036443 0.45834015 0.22527505 0.24574169 0.47388309 0.23264289

0.33615167 0.46375379 0.45881991 0.55189626 0.49818998 0.34390935

0.5318267 0.65937194 0.27529899 0.20633514 0.47168178 0.6369788

0.61766553 0.26678961]

Thresholds:

[ 0.68220519 0.45834015 0.37570748 0.34390935 0.33808756 0.30907854

0.30344634 0.3026426 0.27875212 0.26678961 0.25826574 0.23264289

0.22564941 0.20633514 0.00399151]

FPR:

[ 0. 0. 0.15 0.15 0.25 0.25 0.35 0.35 0.45 0.45 0.5 0.5

0.55 0.55 1. ]

TPR:

[ 0.03333333 0.63333333 0.63333333 0.66666667 0.66666667 0.73333333

0.73333333 0.76666667 0.76666667 0.86666667 0.86666667 0.93333333

0.93333333 1. 1. ]

***Nhận xét:***

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** (**FPR**) thấp, **True Positive Rate (TPR)** cũng rất thấp. Chúng ta luôn muốn rằng **FPR** thấp và **TPR** cao.

Dựng hình đường **ROC:**

**import** matplotlib.pyplot **as** plt  
**from** itertools **import** cycle  
plt.figure()  
lw = 2  
plt.plot(fpr, tpr, color=**'darkorange'**,  
 lw=lw, label=**'Đường cong ROC (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(**'Ví dụ về đường cong Receiver operating characteristic'**)  
plt.legend(loc=**"lower right"**)  
plt.show()



***Hình 1: Mô tả đường ROC***

**Area Under the Curve**

Dựa trên đường cong **ROC**, 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 đường cong **ROC** 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.

**Precision và Recall**

**Định nghĩa:**

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 3 dưới đây:



***Hình 2: 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:

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))

*Kết quả:*

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.

Khi kích thước các lớp dữ liệu là chênh lệch (imbalanced data hay skew data), precision và recall thường được sử dụng.