**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 ]

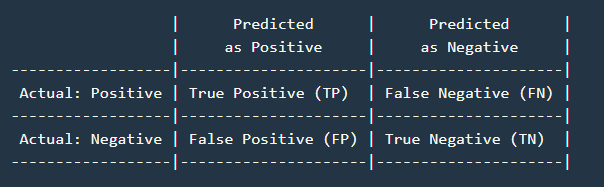
[ 0.33333333 0.66666667 0. ]

[ 0. 0.33333333 0.66666667]]

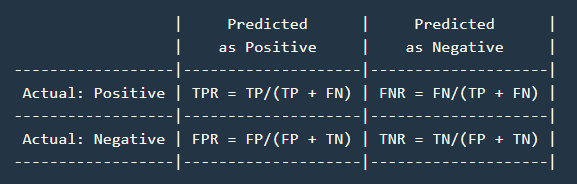
**True/False Positive/Negative**

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**.

**Precision và Recall**