Import thư viện và sử dụng 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.

**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))

sử dụng Confusion matrix để thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class:

**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())

dùng màu sắc để thể hiện confusion matrix:

**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()

sử dụng True/False Positive/Negative (đúng/sai, dương tính /âm tính) để phần tích trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác:

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)

mình hoa bài toán bằng biểu đồ:

**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')

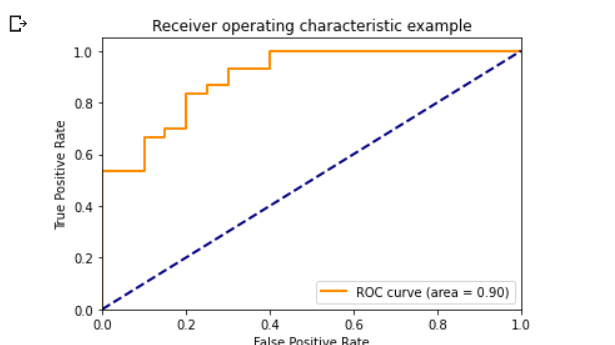
plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

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

plt.show()

kết quả:



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

Precision 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))

hai phép đánh giá dựa trên Precision-Recall được sử dụng là micro-average và macro-average.

Ví dụ về micro-average

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

**def** **PR**(tp, fp, fn):

P **=** float(tp)**/**(tp **+** fp)

R **=** float(tp)**/**(tp **+** fn)

**return** (P, R)

(P1, R1) **=** PR(tp1, fp1, fn1)

(P2, R2) **=** PR(tp2, fp2, fn2)

(P3, R3) **=** PR(tp3, fp3, fn2)

**print**('(P1, R1) = (%.2f, %.2f)'**%**(P1, R1))

**print**('(P2, R2) = (%.2f, %.2f)'**%**(P2, R2))

**print**('(P3, R3) = (%.2f, %.2f)'**%**(P3, R3))

total\_tp **=** tp1 **+** tp2 **+** tp3

total\_fp **=** fp1 **+** fp2 **+** fp3

total\_fn **=** fn1 **+** fn2 **+** fn3

micro\_ap **=** float(total\_tp)**/**(total\_tp **+** total\_fp)

micro\_ar **=** float(total\_tp)**/**(total\_tp **+** total\_fn)

**print**('(micro\_ap, micro\_ar) = (%.2f, %.2f)' **%** (micro\_ap, micro\_ar))

Ví dụ về macro-average

macro\_ap **=** (P1 **+** P2 **+** P3)**/**3

macro\_ar **=** (R1 **+** R2 **+** R3)**/**3

**print**('(macro\_ap, macro\_ar) = (%.2f, %.2f)' **%** (macro\_ap, macro\_ar))