**Binary classifiers cho Multi-class Classification**

Có nhiều thuật toán khác được xây dựng riêng cho các bài toán với nhiều classes (multi-class classification problems), tôi sẽ giới thiệu sau. Còn bây giờ, chúng ta vẫn có thể sử dụng các binary classifiers để thực hiện công việc này, với một chút thay đổi.

Có ít nhất bốn cách để áp dụng binary classifiers vào các bài toán multi-class classification:

**ONE-VS-ONE**

Xây dựng rất nhiều bộ ***binary classifiers*** cho từng cặp ***classes****.* Bộ thứ nhất phân biệt *class 1* và *class 2*, bộ thứ hai phân biệt *class 1* và *class 3…*Khi có một dữ liệu mới vào, đưa nó vào toàn bộ các bộ ***binary classifiers*** trên. Kết quả cuối cùng có thể được xác định bằng cách xem *class* nào mà điểm dữ liệu đó được phân vào nhiều nhất *(major voting)*. Hoặc với ***Logistic Regression*** thì ta có thể tính tổng các xác suất tìm được sau mỗi bộ ***binary classifier***.

Như vậy, nếu cóclasses thì tổng số **binary classifiers** phải dùng là . Đây là một con số lớn, cách làm này không lợi về tính toán. Hơn nữa, nếu một chữ số thực ra là chữ số 1, nhưng lại được đưa vào bộ phân lớp giữa các chữ số 5 và 6, thì cả hai khả năng tìm được (là 5 hoặc 6) đều không hợp lý.

**HIERARCHICAL (Phân Tầng)**

Các làm như **one-vs-one** sẽ mất rất nhiều thời gian training vì có quá nhiều bộ phân lớp cần được xây dựng. Một cách khác giúp tiết kiệm số ***binary classifiers*** hơn đó là hierarchical. Ý tưởng như sau:

Ví dụ với bài toán phân biệt chữ số viết tay, với 4 chữ số 4, 5, 6, 7. Vì ta thấy chữ số 4 và 7 khá giống nhau, chữ số 5 và 6 khá giống nhau nên trước tiên chúng ta xây dựng bộ phân lớp [4, 7] vs [5, 6]. Sau đó xây dựng thêm hai bộ 4 vs 7 và 5 vs 6 nữa. Tổng cộng, ta cần 3 bộ binary classifiers. Chú ý rằng có nhiều cách chia khác nhau, ví dụ [4, 5, 6] vs 7, [4, 5] vs 6, rồi 4 vs 5.

Ưu điểm của phương pháp này là sử dụng ít bộ binary classifiers hơn one-vs-one. Hạn chế lớn nhất của nó là việc nếu chỉ một binary classifier cho kết quả sai thì kết quả cuối cùng chắc chắn sẽ sai. Ví dụ, nếu 1 ảnh chứa chữ số 5, nhưng ngay bước đầu tiên đã bị ***misclassifed (sai lớp)*** sang nhánh [4, 7] thì kết quả cuối cùng sẽ là 4 hoặc 7, cả hai đều sai.

**BINARY CODING**

Có một cách giảm số binary classifiers hơn nữa là binary coding, tức mã hóa output của mỗi class bằng một số nhị phân. Ví dụ, nếu có 4 *classes* thì class thứ nhất được mã hóa là **00**, ba class kia được mã hóa lần lượt là **01**, **10** và **11**. Với cách làm này, số bộ binary classifiers phải thực hiện chỉ là trong đó, **C** là số lượng class, [**a**] là số lượng *class,* [**a**] là số nguyên *không nhỏ hơn* ***a.*** *Class* thứ nhất sẽ đi tìm bit của output(đã được mã hóa nhị phân), *class* thứ hai sẽ đi tìm bit thứ hai…

Cách làm này sử dụng một số lượng nhỏ nhất các bộ ***binary classifiers***. Nhưng nó có một hạn chế rất lớn là chỉ cần một bit bị phân loại sai sẽ dẫn đến dữ liệu bị phân loại sai. Hơn nữa, nếu số classes không phải là lũy thừa của hai, mã nhị phân nhận được có thể là một giá trị không tương ứng với class nào.

**ONE HOT ENCODING:**

**One hot encoding:** là sự biểu diễn các biến phân loại dưới dạng véc tơ nhị phân.

**from** numpy **import** argmax  
*# khởi tạo chuỗi đầu vào*data = **'xin chào'***# định nghĩa bộ chữ cái*alphabet=**'aăàáâbcdđeêghiklmnoôơpqrxtuưvsy '**print(**"Chuỗi ban đầu: {}"**.format(data))  
*# mapping chuỗi sang integer*char\_to\_int = dict((c, i) **for** i, c **in** enumerate(alphabet))  
*# mapping chuỗi sang integer*int\_to\_char = dict((i, c) **for** i, c **in** enumerate(alphabet))  
integer\_encoded = [char\_to\_int[char] **for** char **in** data]  
print(**"Giá trị sau khi chuyển sang kiểu số : {}"**.format(integer\_encoded))  
*# one hot encoding*onehot\_encoded = list()  
**for** value **in** integer\_encoded:  
 letter = [0 **for** \_ **in** range(len(alphabet))]  
 letter[value] = 1  
 onehot\_encoded.append(letter)  
print(**"Encoding chữ cái đầu: {}"**.format(onehot\_encoded[0]))  
*# trả lại giá trị sau mã hóa, ở đây ta sẽ ví dụ ta muốn lấy chữ i ở vị trí =1 trong chuỗi "xin chào" ban đầu*inverted = int\_to\_char[argmax(onehot\_encoded[0])]  
print(**"Giải mã chữ cái đầu: {}"**.format(inverted))

*Kết quả:*

Chuỗi ban đầu: xin chào

Giá trị sau khi chuyển sang kiểu số: [24, 13, 17, 31, 6, 12, 2, 18]

Encoding chữ cái đầu: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

Giải mã chữ cái đầu: x

**One-hot-encoding sử dụng sklearn:**

**from** numpy **import** array  
**from** numpy **import** argmax  
**from** sklearn.preprocessing **import** LabelEncoder *#import từ thư viện sklearn dùng LabelEncoder***from** sklearn.preprocessing **import** OneHotEncoder *#import từ thư viện sklearn dùng OneHotEncoder  
# tạo một mảng mẫu*data = [**'lạnh'**, **'lạnh'**, **'ấm'**, **'lạnh'**, **'nóng'**, **'nóng'**, **'ấm'**, **'lạnh'**, **'ấm'**, **'nóng'**]  
values=array(data) *#chuyển sang mảng n chiều*print(**"Mảng ban đầu: {}"**.format(values))  
*# integer encode*label\_encoder= LabelEncoder() *#gán biến label\_encoder cho hàm LabelEncoder() để mã hóa labels*integer\_encoded = label\_encoder.fit\_transform(values) *#thực thi mã hóa các labels*print(**"Mảng sau khi đã mã hóa: {}"**.format(integer\_encoded))  
*# binary encode*onehot\_encoder = OneHotEncoder(sparse=**False**)  
integer\_encoded = integer\_encoded.reshape(len(integer\_encoded), 1)  
onehot\_encoded = onehot\_encoder.fit\_transform(integer\_encoded)  
print(onehot\_encoded)  
*# invert first example*inverted =label\_encoder.inverse\_transform([argmax(onehot\_encoded[0, :])])  
print(inverted)

*Kết quả:*

Mảng ban đầu: ['lạnh' 'lạnh' 'ấm' 'lạnh' 'nóng' 'nóng' 'ấm' 'lạnh' 'ấm' 'nóng']

Mảng sau khi đã mã hóa: [0 0 2 0 1 1 2 0 2 1]

[[ 1. 0. 0.]

[ 1. 0. 0.]

[ 0. 0. 1.]

[ 1. 0. 0.]

[ 0. 1. 0.]

[ 0. 1. 0.]

[ 0. 0. 1.]

[ 1. 0. 0.]

[ 0. 0. 1.]

[ 0. 1. 0.]]

['lạnh']

**One-vs-Rest/One-hot coding**

Phương pháp được sử dụng nhiều nhất là One-vs-Rest (một số tài liệu gọi là **ove-vs-all**, **one-against-rest** hoặc **one-against-all**). Cụ thể, nếu có class thì ta sẽ xây dựng ***classifiers***, mỗi ***classifier*** tương ứng với một class. ***Classifier*** thứ nhất giúp phân biệt *class\_1* vs *not\_class\_1,* tức xem một điểm có thuộc class 1 hay không, hoặc xác suất để một điểm rơi vào class 1 là bao nhiêu. Tương tự thế, ***classifier*** thứ 2 sẽ phân biệt *class\_2 vs not\_class\_2…*Kết quả cuối cùng có thể được xác định bằng cách xác định class mà một điểm rơi vào với xác suất cao nhất.

Phương pháp này còn có tên là **one-hot coding** vì với cách mã hóa trên, giả sử có 4 classes, class **1, 2, 3, 4** sẽ lần lượt được mã hóa dưới dạng nhị phân bởi **1000, 0100, 0010** hoặc **0001**. **One-hot** vì chỉ có **one bit** là **hot** (bằng 1).

Hàm ***Logistic Regression*** trong thư viện sklearn có thể được dùng trực tiếp để áp dụng vào các bài toán **multi-class classification** với phương pháp **one-vs-rest**. Phương pháp mặc định cho các bài toán **multi-class** của hàm này được xác định bởi biến **multi\_class**. Có hai lựa chọn cho biến này, trong đó lựa chọn mặc định là ***ovr*** tức **one-vs-rest.**