Lựa chọn tính năng là một bước cơ bản trong nhiều quy trình học máy. Bạn loại bỏ một loạt các tính năng và bạn chỉ muốn chọn những tính năng có liên quan và loại bỏ những tính năng khác. Mục đích là đơn giản hóa vấn đề bằng cách loại bỏ các tính năng không sử dụng sẽ gây ra tiếng ồn không cần thiết (bạn đã từng nghe về [Occam](https://en.wikipedia.org/wiki/Occam's_razor) ?).

Boruta là một thuật toán khá thông minh có từ năm 2010 được thiết kế để tự động thực hiện lựa chọn tính năng trên một tập dữ liệu. Nó được sinh ra như một gói cho R ( [đây](https://www.jstatsoft.org/article/view/v036i11) là bài viết đầu tiên mô tả nó). Một phiên bản của Boruta dành cho Python - được gọi là BorutaPy - đã tồn tại và bạn có thể tìm thấy [tại đây](https://github.com/scikit-learn-contrib/boruta_py) .

Trong bài đăng này, chúng ta sẽ thấy một số mã Python đơn giản để triển khai Boruta từ đầu - tôi tin rằng xây dựng một thứ gì đó từ đầu là cách tốt nhất để thực sự hiểu nó - và, ở cuối bài đăng, chúng ta sẽ thấy cách sử dụng BorutaPy để làm cho cuộc sống của chúng tôi dễ dàng hơn.

1. Tất cả đều bắt đầu bằng X và y

Để thấy Boruta đang hoạt động, hãy xây dựng một tập dữ liệu đồ chơi với 3 đặc điểm ( *tuổi* , *chiều cao* và *cân nặng* ), một biến mục tiêu ( *thu nhập* ) và 5 quan sát.

import pandas as pd

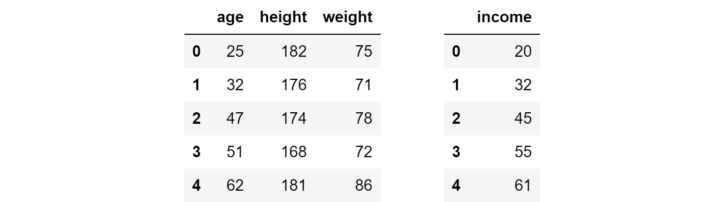
### make X and y

X = pd.DataFrame({'age': [25, 32, 47, 51, 62],

'height': [182, 176, 174, 168, 181],

'weight': [75, 71, 78, 72, 86]})

y = pd.Series([20, 32, 45, 55, 61], name = 'income')

X and y

Trong một tình huống thực tế, chúng ta sẽ phải xử lý nhiều hơn 3 tính năng (thường là từ hàng trăm đến hàng nghìn). Vì vậy, sẽ không khả thi khi xem xét từng người trong số họ và quyết định có nên giữ nó hay không. Hơn nữa, có những mối quan hệ (chẳng hạn như mối quan hệ và tương tác phi tuyến tính) không dễ dàng phát hiện bằng mắt người, thậm chí không thể phân tích chính xác.

Lý tưởng nhất, chúng tôi muốn tìm một thuật toán có thể tự quyết định xem bất kỳ đặc điểm nào đã cho của *X* có mang một giá trị dự đoán nào đó về *y hay không.*

2. Tại sao lại là Boruta?

Một thuật toán phổ biến để lựa chọn tính năng là SelectFromModel của [sklearn](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html#sklearn.feature_selection.SelectFromModel) . Về cơ bản, bạn chọn một mô hình thuận tiện - có khả năng nắm bắt các mối quan hệ và tương tác phi tuyến tính, ví dụ như một khu rừng ngẫu nhiên - và bạn phù hợp với nó trên *X* và *y* . Sau đó, bạn trích xuất tầm quan trọng của từng tính năng từ mô hình này và chỉ giữ lại những tính năng vượt quá ngưỡng quan trọng nhất định.

Điều này nghe có vẻ hợp lý, nhưng điểm yếu của cách tiếp cận như vậy là hiển nhiên: ai xác định ngưỡng và làm thế nào? Có rất nhiều sự tùy tiện trong đó.

Khi lần đầu tiên gặp phải vấn đề này, tôi đã bắt đầu tìm kiếm một giải pháp mạnh mẽ hơn. Cho đến khi tôi tìm thấy Boruta. Boruta là một thuật toán lựa chọn tính năng dựa trên cơ sở thống kê và hoạt động cực kỳ hiệu quả ngay cả khi người dùng không có bất kỳ đầu vào cụ thể nào. Làm sao nó có thể xảy ra?

Boruta dựa trên hai ý tưởng tuyệt vời.

2.1 Ý tưởng đầu tiên: các tính năng bóng

**Trong Boruta, các tính năng không cạnh tranh với nhau. Thay vào đó - và đây là ý tưởng tuyệt vời đầu tiên - họ cạnh tranh với một phiên bản ngẫu nhiên của chúng.**

Trên thực tế, bắt đầu từ *X* , một khung dữ liệu khác được tạo bằng cách trộn ngẫu nhiên từng tính năng. Các đặc trưng hoán vị này được gọi là **các đặc trưng bóng** . Tại thời điểm này, các dataframe bóng được gắn vào dataframe gốc để có được một dataframe mới (chúng tôi sẽ gọi nó là *X\_boruta* ), trong đó có hai lần số cột của *X* .

import numpy as np

### make X\_shadow by randomly permuting each column of X

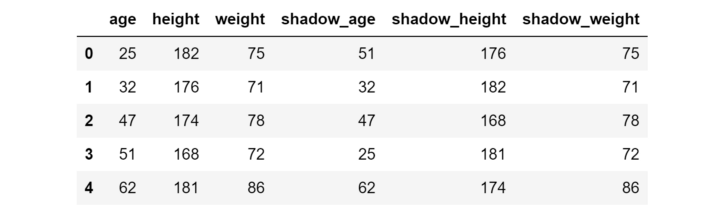
np.random.seed(42)

X\_shadow = X.apply(np.random.permutation)

X\_shadow.columns = ['shadow\_' + feat for feat in X.columns]

### make X\_boruta by appending X\_shadow to X

X\_boruta = pd.concat([X, X\_shadow], axis = 1)

X\_boruta

Sau đó, một khu rừng ngẫu nhiên được trang bị trên *X\_boruta* và *y* .

Bây giờ, chúng tôi xem xét tầm quan trọng của từng tính năng gốc và so sánh nó với một ngưỡng. Lần này, **ngưỡng được xác định là tầm quan trọng của đối tượng địa lý cao nhất được ghi lại trong số các đối tượng địa lý bóng** . Khi mức độ quan trọng của một đối tượng địa lý cao hơn ngưỡng này, đây được gọi là "lần truy cập". Ý tưởng là **một tính năng chỉ hữu ích nếu nó có khả năng hoạt động tốt hơn tính năng ngẫu nhiên tốt nhất** .

Mã để tái tạo quá trình này như sau

from sklearn.ensemble import RandomForestRegressor

### fit a random forest (suggested max\_depth between 3 and 7)

forest = RandomForestRegressor(max\_depth = 5, random\_state = 42)

forest.fit(X\_boruta, y)

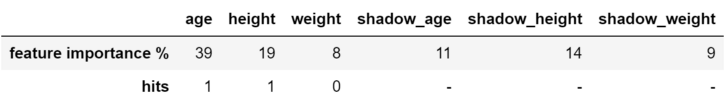
### store feature importances

feat\_imp\_X = forest.feature\_importances\_[:len(X.columns)]

feat\_imp\_shadow = forest.feature\_importances\_[len(X.columns):]

### compute hits

hits = feat\_imp\_X > feat\_imp\_shadow.max()

Kết quả của một lần chạy

Ngưỡng là 14% (tối đa là 11%, 14% và 8%), do đó 2 tính năng đã đạt được thành công, đó là *độ tuổi* và *chiều cao* (tương ứng là 39% và 19%), trong khi *cân nặng* (8%) ghi được dưới ngưỡng.

Rõ ràng, chúng ta nên giảm *cân* và *tăng dần* theo *tuổi tác* và *chiều cao* . **Nhưng chúng ta có nên tin tưởng vào cuộc chạy này không?**Điều gì sẽ xảy ra nếu đó chỉ là một cuộc chạy đua tăng cân không may mắn? Nếu thay vào đó nó chỉ là một cuộc chạy đua may rủi về tuổi tác và chiều cao thì sao?

Đây là lúc ý tưởng tuyệt vời thứ hai của Boruta phát huy tác dụng.

2.2 Ý tưởng thứ hai: phân phối nhị thức

Như thường xảy ra trong học máy (trong cuộc sống?), **Chìa khóa là sự lặp lại** . Không có gì ngạc nhiên khi 20 thử nghiệm đáng tin cậy hơn 1 thử nghiệm và 100 thử nghiệm đáng tin cậy hơn 20 thử nghiệm.

Ví dụ, hãy lặp lại 20 lần quá trình đã thấy ở trên.

### initialize hits counter

hits = np.zeros((len(X.columns)))

### repeat 20 times

for iter\_ in range(20):

### make X\_shadow by randomly permuting each column of X

np.random.seed(iter\_)

X\_shadow = X.apply(np.random.permutation)

X\_boruta = pd.concat([X, X\_shadow], axis = 1)

### fit a random forest (suggested max\_depth between 3 and 7)

forest = RandomForestRegressor(max\_depth = 5, random\_state = 42)

forest.fit(X\_boruta, y)

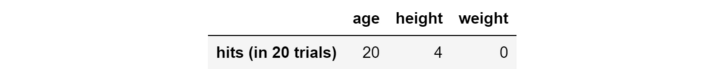
### store feature importance

feat\_imp\_X = forest.feature\_importances\_[:len(X.columns)]

feat\_imp\_shadow = forest.feature\_importances\_[len(X.columns):]

### compute hits for this trial and add to counter

hits += (feat\_imp\_X > feat\_imp\_shadow.max())

Kết quả của 20 lần chạy

Bây giờ, làm thế nào để chúng ta thiết lập một tiêu chí quyết định? Đây là ý tưởng tuyệt vời thứ hai có trong Boruta.

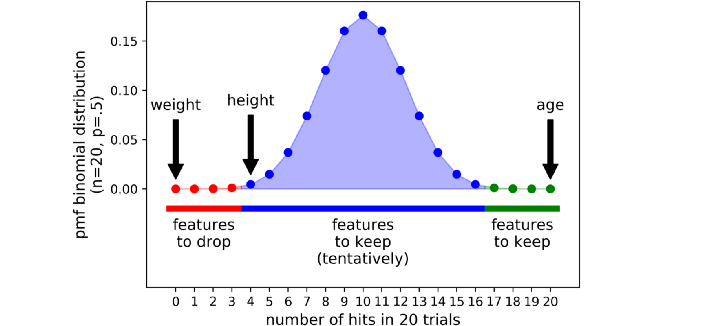
Hãy xem một tính năng (bất kể *tuổi tác* , *chiều cao* hay *cân nặng* ) và nói rằng chúng ta hoàn toàn không biết nó có hữu ích hay không. Xác suất mà chúng ta sẽ giữ nó là bao nhiêu? **Mức độ không chắc chắn tối đa về tính năng được thể hiện bằng xác suất 50%, giống như tung đồng xu** . Vì mỗi thử nghiệm độc lập có thể cho một kết quả nhị phân (lần truy cập hoặc không có lần truy cập), **một chuỗi *n* thử nghiệm tuân theo**[**phân phối nhị thức**](https://en.wikipedia.org/wiki/Binomial_distribution) .

Trong Python, hàm khối lượng xác suất của một phép chưng cất nhị thức có thể được tính như sau:

import scipy as sp

trials = 20

pmf = [sp.stats.binom.pmf(x, trials, .5) for x in range(trials + 1)]

Phân phối nhị thức và vị trí của các đối tượng

Ở Boruta, không có một ngưỡng khó nào giữa vùng từ chối và vùng chấp nhận. Thay vào đó, có 3 lĩnh vực:

* một **khu vực từ chối** (khu vực màu đỏ): các tính năng mà kết thúc ở đây được coi là tiếng ồn, vì vậy họ đang giảm;
* một **diện tích irresolution** (vùng màu xanh): Boruta là thiếu quyết đoán về các tính năng mà là trong lĩnh vực này;
* một **khu vực chấp nhận** (khu vực màu xanh lá cây): các đối tượng địa lý ở đây được coi là dự đoán, vì vậy chúng được giữ lại.

Vì vậy, chúng tôi đã thực hiện 20 lần lặp lại dữ liệu đồ chơi của mình và kết thúc với một số kết luận có cơ sở thống kê:

* Để dự đoán *thu nhập* của một người, *tuổi* được dự đoán và nên được giữ lại, *trọng lượng* chỉ là tiếng ồn và nên bỏ đi,
* Boruta không quyết đoán về *chiều cao* : sự lựa chọn là tùy thuộc vào chúng tôi, nhưng trong một khuôn khổ bảo thủ, thì nên giữ nó.

3. Sử dụng BorutaPy trong Python

Boruta có thể được cài đặt thông qua pip:

!pip install boruta

from boruta import BorutaPy

from sklearn.ensemble import RandomForestRegressor

import numpy as np

###initialize Boruta

forest = RandomForestRegressor(

n\_jobs = -1,

max\_depth = 5

)

boruta = BorutaPy(

estimator = forest,

n\_estimators = 'auto',

max\_iter = 100 # number of trials to perform

)

### fit Boruta (it accepts np.array, not pd.DataFrame)

boruta.fit(np.array(X), np.array(y))

### print results

green\_area = X.columns[boruta.support\_].to\_list()

blue\_area = X.columns[boruta.support\_weak\_].to\_list()

print('features in the green area:', green\_area)

print('features in the blue area:', blue\_area)

4. Kết luận

Lựa chọn tính năng là một phần quyết định của quá trình học máy: quá thận trọng có nghĩa là tạo ra tiếng ồn không cần thiết, trong khi quá tích cực có nghĩa là loại bỏ thông tin hữu ích.

Trong bài đăng này, chúng tôi đã xem cách sử dụng Boruta để thực hiện lựa chọn tính năng mạnh mẽ, có căn cứ về mặt thống kê trên tập dữ liệu của bạn. Thật vậy, việc đưa ra các quyết định quan trọng về các tính năng là rất quan trọng để đảm bảo sự thành công của mô hình dự đoán của bạn.