ĐỒ ÁN MÔN HỌC

Bộ môn : Phân tích dữ liệu bảo toàn tính riêng tư

MÁ CHẤN PHONG
19120616

Mục Lục

Nội Dung

I.Làm quen với với Differential Privacy

1. Đặt vấn đề?

Bạn đã bao giờ lộ thông tin cá nhân mình qua 1 phiếu khảo sát chưa ? Nếu thông tin ấy không phải là thông tin công khai thì sao ? Bạn hãy đọc qua những vấn đề dưới đây để hiểu thêm việc lộ thông tin riêng tư nguy hiểm thế nào ?

Vấn đề 1 : Có một phòng khám các bệnh nhân trĩ có 1 bảng thống kê số người bệnh trĩ đã khám ở trên cửa phòng khám bệnh. Trên bảng có ghi hôm nay có tổng cộng 100 bệnh nhân khám bị bệnh trĩ. Đến lượt bạn y tá kêu « Nguyễn Văn A » vào khám bất ngờ bạn lại bệnh trĩ thật khi bạn ra bảng thống kê lên 101 bệnh nhân khám bị bệnh trĩ và mọi người sẽ « a thì ra thằng Nguyễn Văn A đó bị bệnh trĩ tội nó ghê ». Và thế là bạn bị trĩ ai ai cũng biết .

Vấn đề 2 : Bây giờ giả sử nhà nước đưa cho bạn một survey về việc bạn có bị trĩ không nhé (an sinh xã hội mà). Do kết quả phải công bố công khai, họ hứa sẽ không lưu tên của bạn, mà chỉ có một số thông tin như bạn ở phường nào hay là chế độ ăn uống của bạn là gì. Đương nhiên rồi, họ muốn biết rằng việc bạn bị trĩ có phải do lý do địa lý, hay do bạn ăn nhiều thịt, hay do công việc bạn có ngồi nhiều hay không, vân vân và mây mây. Bạn thấy rằng các thông tin như tên tuổi số chứng minh nhân dân của bạn không được cho vào, nên bạn thấy việc cung cấp thông tin đó khá là an toàn. Bùm, ngay hôm sau bạn bè của bạn sẽ trêu bạn tới bến vì phát hiện ra bạn bị trĩ. Vì bạn là một con người yêu nước nên bạn không tin rằng nhà nước đã không thực hiện đúng cam kết bảo mật của họ. Vậy thì bằng cách nào mà thông tin của bạn bị lộ ra vậy?

Có rất nhiều phương pháp có thể tìm ra đúng entry của bạn trong cái database đó. Đơn giản nhất, có bao nhiêu người sống ở quận của bạn, làm công việc của bạn, có sở thích như bạn? Họ chỉ cần query đúng những thông số đó và họ sẽ lấy được ngay đúng entry họ cần biết. Đây gọi là linkage attack, khi kẻ tấn công biết được các thông tin liên quan về bạn từ các nguồn khác để query chính xác ra kết quả của bạn.

Vậy làm cách nào để tránh hiện tượng này bây giờ? Câu trả lời chính là Differential Privacy, topic của bài viết này. Tuy nhiên, trước khi chúng ta đến với định nghĩa cụ thể thì hãy bắt đầu với ý tưởng của phương pháp này nhé.

2. Làm thế nào để trả lời những câu hỏi nhạy cảm ?

Bây giờ chúng ta sẽ thu thập thông tin của họ bằng cách như sau: người tham gia sẽ tung một đồng xu. Nếu đồng xu là ngửa, họ sẽ trả lời thật lòng. Nếu đồng xu là úp, họ sẽ tung đồng xu một lần nữa, và nếu ngửa họ sẽ trả lời có, và úp họ sẽ trả lời không. Với một câu trả lời có, xác suất họ bị bệnh thật chỉ là 3/4, và họ hoàn toàn có thể chối bỏ bất cứ kết luận nào về họ dựa trên kết quả đó vì lý do hôm đó họ đen bạc. Thế là an toàn rồi nhỉ?

Giờ với kết quả họ cung cấp thì chúng ta tính trung bình ra sao? Thấy rằng, nếu họ bị bệnh, xác suất họ sẽ trả lời có là 3/4, và nếu không bị, xác suất trả lời có là 1/4.

Từ đó, chúng ta ra được công thức cho xác suất kết quả là có P so với xác suất thật một người bị trĩ P' :

$$P = \frac{3}{4}P' + \frac{1}{4}(1 - P') = \frac{1}{2}(P' + \frac{1}{2})$$

Đảo ngược lại công thức đó và bạn đã có được con số bạn mong muốn:

$$P' = 2P - \frac{1}{2}$$

Thật là vi diệu phải không? Bài học rút ra ở đây là nếu bạn muốn có privacy, bạn phải thêm tính ngẫu nhiên vào data của bạn.

3. Differential Privacy là gì ?

Differential privacy là kỹ thuật cung cấp khả năng bảo vệ quyền riêng tư mạnh mẽ. Cácphương pháp tiếp cận thường được sử dụng trong nhiều thập kỷ gần đây như de-identification (khử nhận dạng) và aggregation (tổng hợp) đã được chứng minh là bị phá vỡ bởi các cuộc tấn công một cách tinh vi và thậm chí là các kỹ thuật hiện đại hơn như k-Anonymity (k - ẩn danh) cũng dễ bị tấn công. Vì vậy Differential privacy đang nhanhchóng trở thành "tiêu chuẩn vàng" trong việc bảo vệ quyền riêng tư.

Differential Privacy là một thuộc tính của thuật toán chứ không phải thuộc tính của dữ liệu. Chúng ta có thể chứng minh dữ liệu đáp ứng differential privacy bằng cách chỉ ra rằng thuật toán tạo ra nó đáp ứng differential privacy.

Một hàm đáp ứng differential privacy thường được gọi là mechanism cơ chế. Chúng ta nói rằng một cơ chế F thỏa mãn differential privacy nếu tất cả các tập dữ liệu lân cận x và x' và tất cả các kết quả đầu ra có thể có S thỏa:

$$\frac{\Pr[F(x) = S]}{\Pr[F(x') = S]} \leq e^{\epsilon}$$

Hai tập dữ liệu được coi là lân cận nếu chúng khác nhau về dữ liệu của 1 cá nhân. F thường là 1 hàm ngẫu nhiên, do đó phân phối xác suất mô tả các kết quả đầu ra của nó không chỉ là một phân phối điểm.

Tính ngẫu nhiên được tích hợp trong F phải là "đủ" để một output quan sát được từ F sẽ không tiết lộ x hay x' là đầu vào. Tham số ε được gọi là tham số bảo mật. Giá trịεcàng nhỏ thì cung cấp mức độ riêng tư càng cao

II.Cơ Chế Laplace

1. Định nghĩa

Cơ chế Laplace, như tên gọi của nó, sử dụng phân phối Laplace để cộng một lượng noise vào kết quả của mỗi query.

Ta có hàm mật độ xác suất của phân phối Laplace(μ = 0, b) như sau:

$$p(x) = \frac{1}{2b}\exp\left(-\frac{|x|}{b}\right)$$

Gọi f: $X^n \rightarrow \mathbb{R}^k$

Δ là độ nhạy cảm của f.

Cơ chế Laplace được định nghĩa như sau:

$$M(X) = f(x) + Y(Y1, Y2, Y3, \dots Yk) \sim Lap(\frac{\Delta}{\varepsilon})$$

Cơ chế này thỏa ε-differential privacy.

Cách hoạt động của cơ chế Laplace là cộng vào kết quả một giá trị noise theo phân phối Laplace. Giá trị này phụ thuộc vào độ nhạy cảm của từng query.

Cơ chế Laplace là cơ chế cơ bản nhất và rất quan trọng trong diffenrential privacy.

2. Độ nhạy cảm

a) Định nghĩa :

Trong cơ chế Laplace thì lượng nhiễu cần thiết để đảm bảo diffirential privacy cho một query phụ thuộc vào độ nhạy cảm (sensitivity) của query đó. Độ nhảy cảm sẽ phản ánh: sự ảnh hưởng đến output khi input thay đổi. Ta có cơ chế Laplace được định nghĩa như sau

$$F(x) = f(x) + Lap(\frac{\Delta}{\varepsilon})$$

trong đó f(x) là hàm xác định (query), ϵ là tham số privacy, s là sensitivity của f.

b) Tính Sensitivity

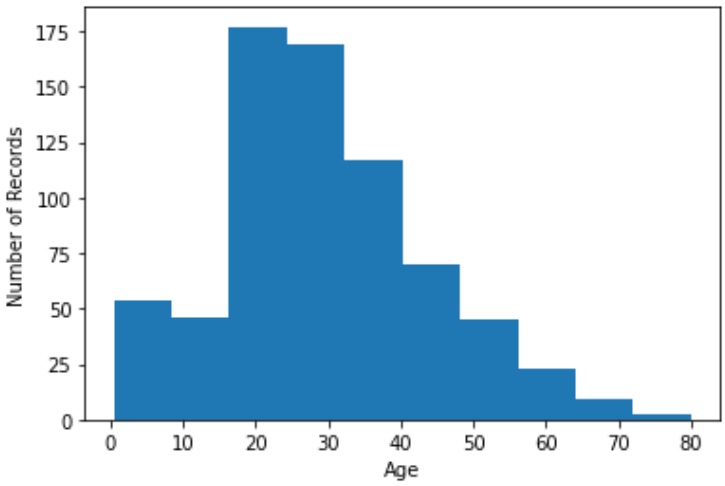
Đối với các hàm f trên tập số thực, độ nhạy cảm là dễ dàng tính được. Đối với những hàm ánh xạ tập dữ liệu vào số thực, ta có thể dùng một số phương pháp phân tích. Cụ thể ta sẽ xét đến các query tổng hợp dữ liệu: counts, sum, averages.

- Counting queries luôn có độ nhạy cảm bằng 1.
- Độ nhạy cảm của các summation query không đơn giản như counting query. Chúng phụ thuộc vào nội dung mà chúng ta thêm/ bớt. (cụ thể sẽ dùng thuật toán clipping ở phần sau)
- Cách đơn giản để trả lời average query là chia việc tính trung bình thành 2 giai đoạn: summation query và counting query. Với mỗi query ta sẽ tính toán như mô tả ở trên. Kết quả sau 2 query sẽ chia nhau để được kết quả của average query.

c) Clipping

Ý tưởng của clipping là "chặn" cho upperbound và lowerbound vào giá trị xác định. Ví dụ những người nào có tuổi trên 125 sẽ được cắt thành đúng 125. Và độ nhạy cảm của hàm sum lúc này sẽ bằng upperbound – lowerbound.

Ta có thể chọn cách nhìn vào dữ liệu và chọn ra upperbounds. Ví dụ trong tập dữ liệu Titanic. Không có dữ liệu nào có tuổi trên 80. Tuy nhiên, ta không thể chọn upperbound bằng 80. Vì như vậy, ta đã vi phạm vào nguyên tắc của differential privacy, đó là làm lộ phần thông tin của tập dữ liệu

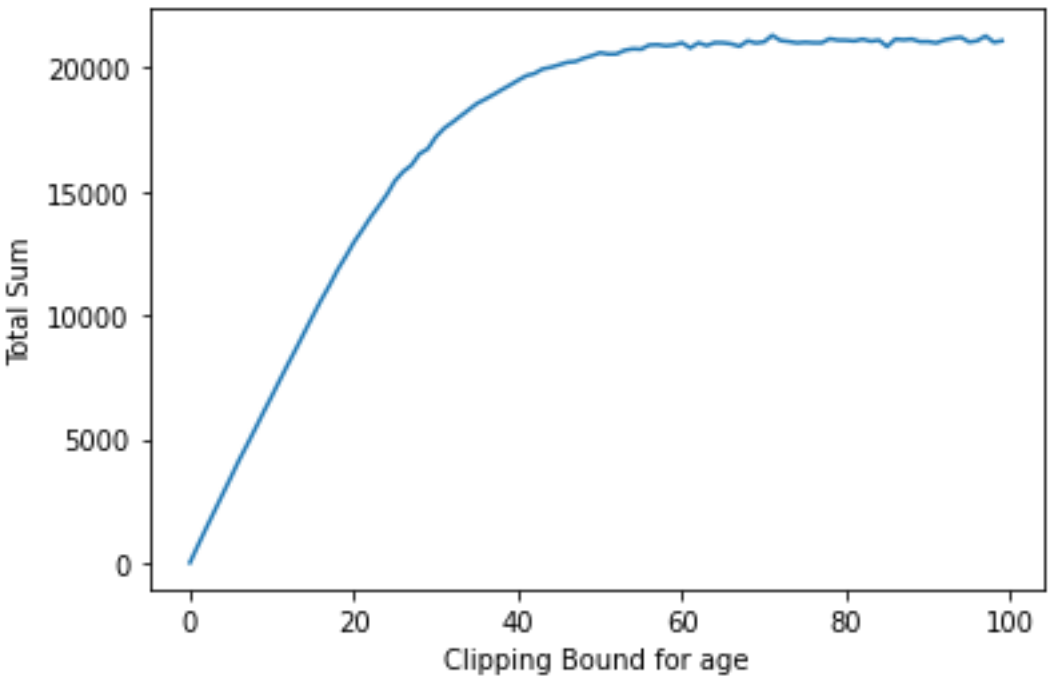


Vậy thì chúng ta sẽ vẽ một đồ thị thêm vào độ nhiễu và chọn upperbound. Một cách chọn khá tốt là tìm vùng của đồ thị đủ "trơn" (độ nhiễu thấp) và không tăng lên (clipping bound đủ tốt).

```
def laplace_mech(v, sensitivity, epsilon):
    return v + np.random.laplace(loc=0, scale=sensitivity/
epsilon)

epsilon_i = 1
plt.plot([laplace_mech(titanic_lite['Age']
     .clip(lower=0, upper=i).sum(), i, epsilon_i) for i in ran
ge(0,500)])
plt.xlabel('Clipping Bound for age')
plt.ylabel('Total Sum');
```

Hình vẽ :



III.Decision Tree (Cây Quyết Định)

1. Định nghĩa

Cây quyết định (Decision Tree) là một cây phân cấp có cấu trúc được dùng để phân lớp các đối tượng dựa vào dãy các luật. Các thuộc tính của đối tượngncó thể thuộc các kiểu dữ liệu khác nhau như Nhị phân (Binary) , Định danh (Nominal), Thứ tự (Ordinal), Số lượng (Quantitative) trong khi đó thuộc tính phân lớp phải có kiểu dữ liệu là Binary hoặc Ordinal.

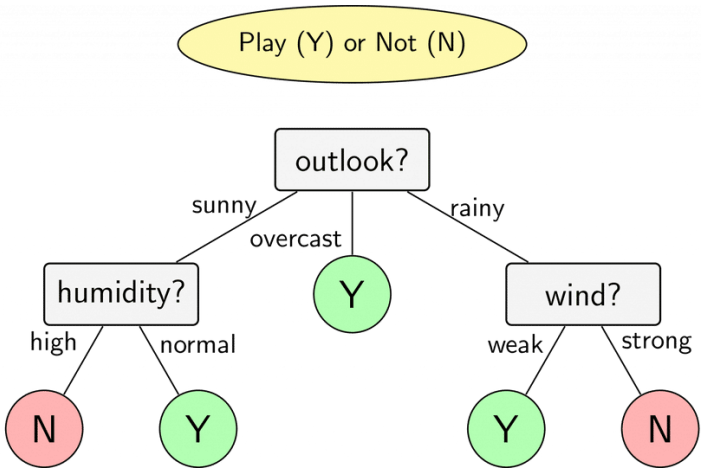
Tóm lại, cho dữ liệu về các đối tượng gồm các thuộc tính cùng với lớp (classes) của nó, cây quyết định sẽ sinh ra các luật để dự đoán lớp của các dữ liệu chưa biết.

Ta hãy xét một ví dụ 1 kinh điển khác về cây quyết định. Giả sử dựa theo thời tiết mà các bạn nam sẽ quyết định đi đá bóng hay không?

Những đặc điểm ban đầu là:

- Thời tiết
- Độ ẩm
- Gió

Dựa vào những thông tin trên, bạn có thể xây dựng được mô hình như sau:



Dựa theo mô hình trên, ta thấy:

Nếu trời nắng, độ ẩm bình thường thì khả năng các bạn nam đi chơi bóng sẽ cao. Còn nếu trời nắng, độ ẩm cao thì khả năng các bạn nam sẽ không đi chơi bóng.

2. Cách xây dựng Decision tree:

Có nhiều thuật toán để xây dưng cây quyết định (CART, ID3 ,…) cụ thể trong báo cáo này sẽ trình bài về ID3.

ID3 Iterative Dichotomiser 3 (J. R. Quinlan 1993) sử dụng phương pháp tham lam tìm kiếm từ trên xuống thông qua không gian của các nhánh có thể không có backtracking. ID3 sử dụng Entropy và Information Gain để xây dựng một cây quyết định.

Vậy thì Entropy là gì ?

Entropy là thuật ngữ thuộc Nhiệt động lực học, là thước đo của sự biến đổi, hỗn loạn hoặc ngẫu nhiên. Năm 1948, Shannon đã mở rộng khái niệm Entropy sang lĩnh vực nghiên cứu, thống kê với công thức như sau:

Với một phân phối xác suất của một biến rời rạc *x* có thể nhận n giá trị khác nhau *x1,x2,…,xn.*

Giả sử rằng xác suất để *x* nhận các giá trị này là *pi=p(x=xi).*

Ký hiệu phân phối này là *p=(p1 ,p2 ,…,pn)*. Entropy của phân phối này được định nghĩa là:

$$H(p) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Vậy thì Entropy có thể làm gì cho cây quyết định. Thì Information Gain nó là một hệ số để xây dựng cây quyết định. Nó dựa trên sự giảm của hàm Entropy khi tập dữ liệu được phân chia trên một thuộc tính.

Để xây dựng một cây quyết định, ta phải tìm tất cả thuộc tính trả về Infomation gain cao nhất.

Để xác định các nút trong mô hình cây quyết định, ta thực hiện tính Infomation Gain tại mỗi nút theo trình tự sau:

- Bước 1: Tính toán hệ số Entropy của biến mục tiêu S có N phần tử với Ni phần tử thuộc lớp C cho trước:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2 p_i \ \ với\ pi = \frac{Ni}{N}$$

- Bước 2: Tính hàm số Entropy tại mỗi thuộc tính: với thuộc tính x, các điểm dữ liệu trong S được chia ra K child node $S_1, S_2, S_3 \ldots S_k$ với số điểm trong mỗi child node lần lượt là $m_1, m_2, \ldots, m_k$, ta có:

$$H(x,S) = -\sum_{i=1}^{k} \frac{m_k}{N} H(S_k)$$

- Bước 3: Chỉ số Gain Information được tính bằng:

$$Information_{gain} = H(s) - H(x,S)$$

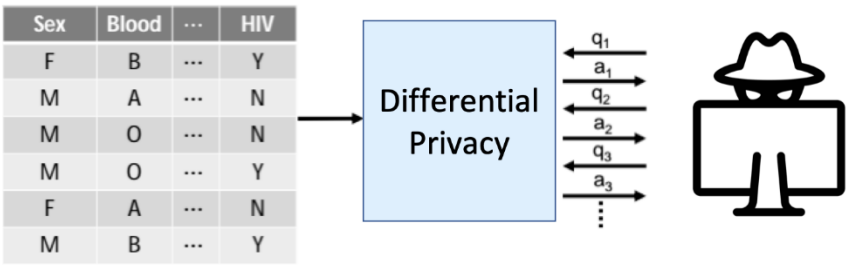
Sau đó chúng ta dựa vào hệ số Infomation gain cho từng thuộc tích để chọn thuộc tính và split các giá trị thích hợp.

IV.Kết hợp Differential Privacy và Decision Tree

1. Sử dụng Differential Privacy khi nào ?

Việc sử dụng DP (Differential Privacy) vào cây decision tree để nhằm mục đích tăng cường tính riêng tư cho dữ liệu và không ảnh hưởng tới mọi dự đoán của cây quyết định.

Vậy việc sử dụng DP nhằm mục đích làm nhiễu kết quả của dữ liệu. Khi dữ liệu được truy vấn từ database chúng ta sẽ thả noise làm kết quả truy vấn lệch đi về kết quả nhưng vẫn bảo vệ được tính riêng tư người dùng.



2. Cách xây dựng

Đầu tiên chúng ta cần xây dựng hàm nhiễu để làm nhiễu kết quả truy vấn. Ở đồ án mình dùng dữ liễu từ file csv nên việc thả nhiễu sẽ được thêm vào khi truy vấn vào dữ liệu.

Hàm nhiễu sẽ chia làm 2 hàm 1 hàm sẽ thả nhiễu khi truy vấn 1 phần tử và 1 hàm sẽ thả nhiễu khi truy vấn các hàm sum, count, mean ,… trên data.

```
# nhiễu cho các phần tử truy vấn đơn
def laplace_mech(v, sensitivity, epsilon):
    return v + np.random.laplace(loc=0, scale=sensitivity/epsilon)
# nhiễu cho các phần tử truy vấn hàm
def noise_laplace(colunm,method,epsilon):
  if method != "count":
    sensitivity = float(Sensitivities[colunm][method])
  else:
    sensitivity = 1
  return np.random.laplace(0, sensitivity*1.0/epsilon)
```

Hàm để thực hiện các phép truy vấn :

```
def use_query_DP(data, colunm, query_type, epsilon):
        if (query_type == "mean"):
            partial_epsilon = epsilon/2
            noisy_sum = data[colunm].sum()+ noise_laplace(colunm, "sum"
, partial_epsilon)
            noisy_count = data[colunm].count()+ noise_laplace(colunm, "
count", partial_epsilon)
            noisy_mean = noisy_sum/noisy_count
            return noisy_mean
        elif (query_type == "max"):
```

```
            return data[colunm].max()+ noise_laplace(colunm, "sum", eps
ilon)
        elif (query_type == "min"):
            return data[colunm].min()+ noise_laplace(colunm, "sum", eps
ilon)
        elif (query_type == "median"):
            return data[colunm].median()+ noise_laplace(colunm, "sum",
epsilon)
        elif (query_type == "mode"):
            return data[colunm].mode()+ noise_laplace(colunm, "sum", ep
silon)
        elif (query_type == "sum"):
            return data[colunm].sum()+ noise_laplace(colunm, "sum", eps
ilon)
        elif (query_type == "count"):
            return data[colunm].count()+ noise_laplace(colunm, "count",
 epsilon)
        elif query_type == "variance":
          partial_epsilon = epsilon/3
          square_sum = sum((data[colunm][i])**2 for i in range(data[col
unm].count()))
          noisy_square_sum = square_sum +  noise_laplace(colunm, "sum_s
quare", partial_epsilon)
          noisy_sum = data[colunm].sum()+ noise_laplace(colunm, "sum",
partial_epsilon)
          noisy_count = data[colunm].count()+ noise_laplace(colunm, "co
unt", partial_epsilon)
          noisy_mean = noisy_sum/noisy_count
          noisy_var = (1/noisy_count)*(noisy_square_sum - 2*noisy_mean*
noisy_sum) + noisy_mean*noisy_mean
          return noisy_var
        else :
          raise "Not implemented"
```

Tiếp theo là hàm Entropy có sử dụng DP :

```
def entropy(data ,y):
  if isinstance(data[y], pd.Series):
      set_column = data[y].unique()
      maxtrix = dict()
      for item in set_column:
        rel = use_query_DP(data[:][data[y] == item],y,"count",EPSILON)
        maxtrix[item] = int(rel)
      val_count = pd.Series(maxtrix)
      shape = use_query_DP(data,y,"count",EPSILON)
      a = val_count/shape
      entropy = np.sum(-a*np.log2(a+1e-9))
      return(entropy)
  else:
```

```
    raise('Object must be a Pandas Series.')
```

Hàm information_gain

```
def information_gain(y,data, mask, func=entropy):
  try:
    a = sum(mask)
    b = mask.shape[0] - a
    label = y.name
    col , col_label= pd.DataFrame(y[mask]) , y[mask].name
    col_2, col_2_label= pd.DataFrame(y[-mask]),y[-mask].name
    if(a == 0 or b ==0):
      ig = 0
    else:
      ig = entropy(data= data,y = label)-
a/(a+b)*entropy(col,col_label)-b/(a+b)*entropy(col_2,col_2_label)
    return ig
  except Exception as e:
    raise(e)
```

Xây dựng cây bằng các hàm như **max_information_gain_split** , **make_prediction**, **make_split**, **get_best_split**, ….Và tạo cây

```
def train_tree(data,y, target_factor, max_depth = None,
min_samples_split = None, min_information_gain = 1e20,
counter=0, max_categories = 20):
  # Check  max_categories đã đúng chưa
  if counter==0:
    types = data.dtypes
    check_columns = types[types == "object"].index
    for column in check_columns:
      var_length = len(data[column].value_counts())
      if var_length > max_categories:
        raise ValueError('The variable ' + column + ' has '+
                         str(var_length)
       + ' unique values, which is more than the accepted ones: '
                        +  str(max_categories))

  #check độ sâu
  if max_depth == None:
    depth_cond = True

  else:
    if counter < max_depth:
      depth_cond = True

    else:
      depth_cond = False
```

```
  # check điều kiện mẫu
  if min_samples_split == None:
    sample_cond = True

  else:
    if data.shape[0] > min_samples_split:
      sample_cond = True

    else:
      sample_cond = False

  # check đk ig
  if depth_cond & sample_cond:

    var,val,ig,var_type = get_best_split(y, data)

    # If đk ig  đúng  thực hiện split
    if ig is not None and ig >= min_information_gain:

      counter += 1

      left,right = make_split(var, val, data,var_type)

      # khởi tạo sub-tree
      split_type = "<=" if var_type else "in"
      question =   "{} {}  {}".format(var,split_type,val)
      # question = "\n" + counter*" " + "|-
>" + var + " " + split_type + " " + str(val)
      subtree = {question: []}


      #(recursion)
      yes_answer = train_tree(left,y, target_factor, max_depth,
min_samples_split,min_information_gain, counter)

      no_answer = train_tree(right,y, target_factor, max_depth,
min_samples_split,min_information_gain, counter)

      if yes_answer == no_answer:
        subtree = yes_answer

      else:
        subtree[question].append(yes_answer)
        subtree[question].append(no_answer)

    #nếu không đúng với điều kiện của ig thực hiện dự đoán
    else:
      pred = make_prediction(data,y,target_factor)
      return pred
```

```
    #ngung khi ko đúng độ sâu và điều kiện
  else:
    pred = make_prediction(data,y,target_factor)
    return pred

  return subtree
```

Và cuối cùng dự đoán :

```
titanic_prediction = []
num_obs = 20

for i in range(num_obs):
  obs_pred = clasificar_datos(titanic_lite.iloc[i,:], titanic_tree)
  titanic_prediction.append(obs_pred)

print("Predictions: ",titanic_prediction,
"\n\nReal values:", titanic_lite.Survived[:num_obs].to_numpy())
```

Kết quả sẽ được so sách với 20 điểm dữ liệu đầu tiên của data titanic,

Và source code và dataset nằm ở đường link sau : DP_Decision_tree

Tài Liệu Tham Khảo :

[1] A. F. Jauregui and A. Legal, "How to program a desision tree in Python from 0," 2018. [Online]. Available: https://anderfernandez.com/en/blog/code-decision-tree-python-from-scratch/.

[2] "Cây Quyết Định (Decision Tree)," 6 6 2019. [Online]. Available: https://trituenhantao.io/kien-thuc/decision-tree/.

[3] N. Dat, “Sử dụng cơ chế Laplace trong Differential Privacy để bảo vệ tính riêng tư của dữ liệu,” 20 9 2021. [Trực tuyến]. Available: https://viblo.asia/p/su-dung-co-che-laplace-trong-differential-privacy-de-bao-ve-tinh-rieng-tu-cua-du-lieu-LzD5dM8WKjY.

[4] J. P.Near and C. Abuah, Programming Differential Privacy.