Dealing with missing data

August 8, 2020

1 Introduction

Có thể có nhiều lỗi trong quá trình xử lý tập data, một vài chỗ có thể ko có sẵn, hoặc bị bỏ qua khi khảo sát. Chúng ta thường thấy những bảng data bị NaN (Not a number), NULL (cho những unknown value trong các relational DB). Ko may thay, những công cụ tính toán ko thể tự xử lý các missing data và sẽ tạo ra các kết quả ko đoán trước được nếu chúng ta bỏ qua nó. Vậy nên, một điều rất quan trọng là chúng ta phải xử lý missing data trước khi các bước phân tích diễn ra. Trong phần này, chúng ta giới thiệu qua các kĩ thuật testing để xóa bỏ những thứ bị thiếu, hoặc input vào những chỗ thiếu

2 Reasons why data goes missing

Missing at Random (MAR): Thiếu ngẫu nhiên có nghĩa là xu hướng thiếu một điểm dữ liệu không liên quan đến dữ liệu bị thiếu, nhưng nó liên quan đến một số dữ liệu được quan sát. VD: thường thì học sinh (tuổi dưới 18) sẽ chưa có điểm đo trình độ tiếng Anh theo các chứng chỉ như TOEIC, IELTS..., nếu bạn thu thập điều này thì sẽ xảy ra MAR

Missing Completely at Random (MCAR): sự thiếu dữ liệu kiểu này là hoàn toàn ngẫu nhiên, ko liên quan đến dữ liệu dc quan sát

Missing not at Random (MNAR): dữ liệu thiếu có phụ thuộc gì đó với dữ liệu quan sát dc. VD: thường nếu bạn hỏi tuổi của các cô các chị (giới tính nữ), bạn sẽ ko nhận dc câu trả lời (thiếu thuộc tính tuổi, rõ ràng nó phụ thuộc vào giới tính)

2 trường hợp đầu thì remove ổn, nhưng MNAR thì có khả năng cao gây ra bias cho model của bạn.

3 Identifying missing values in tabular data

[47]:
```
import pandas as pd
from io import StringIO
```

[48]:
```
csv_data = \
'''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''
```

```
# If you are using Python 2.7, you need
# to convert the string to unicode:
# csv_data = unicode(csv_data)
df = pd.read_csv(StringIO(csv_data))
df
```

[48]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

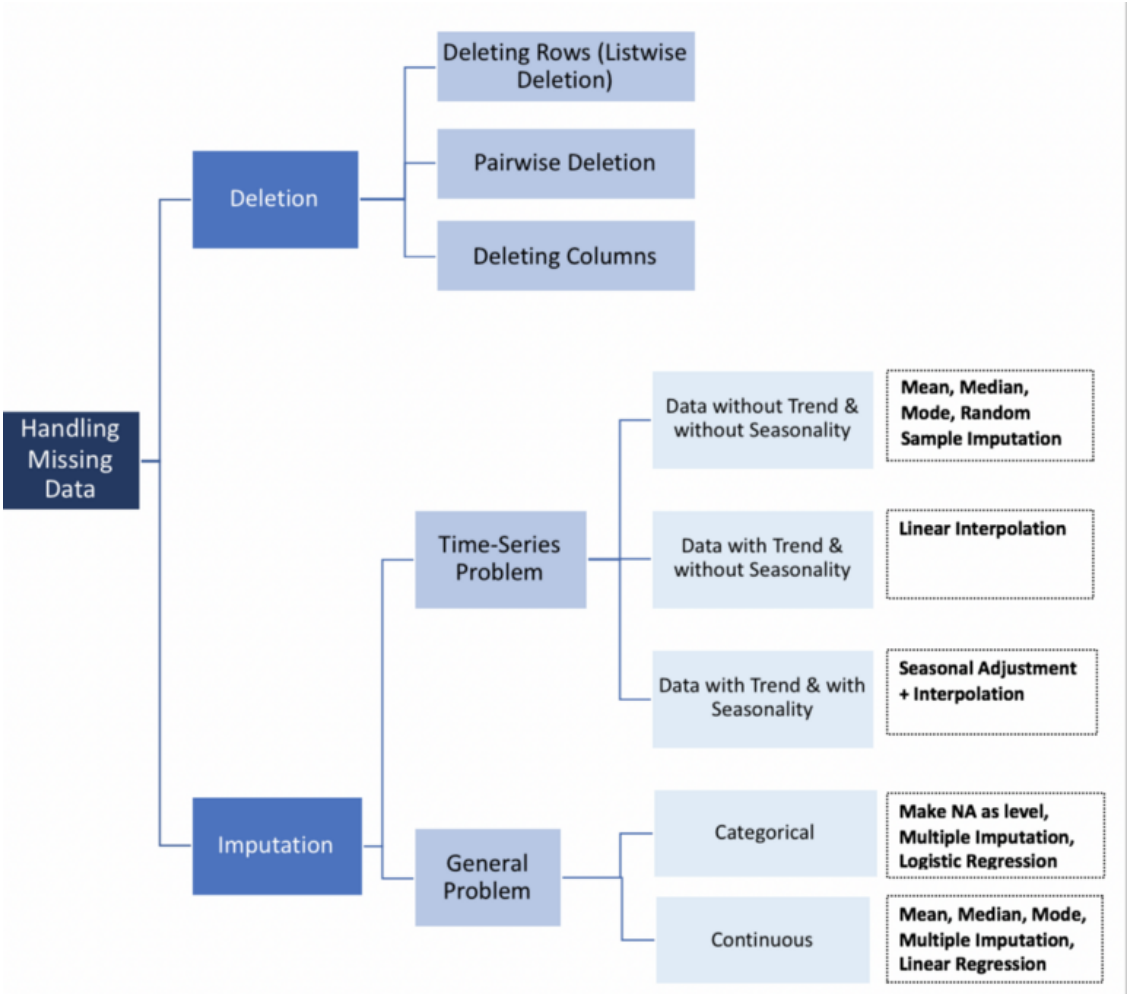
[49]:
```
df.isnull().sum()
```

[49]:
```
A    0
B    0
C    1
D    1
dtype: int64
```

Scikit-learn được thiết kế chỉ để làm việc với NumPy arrays. Và chính Numpy array cũng dễ handle tốt, vậy nên thường chúng ta được khuyến khích sử dụng NumPy array khi có thể. Trong DataFrame, chúng ta vẫn có thể truy cập NumPy array bởi values attribute

4 Solution

4.1 Vanilla mindmap for dealing with missing data



4.2 Deletion

4.2.1 Listwise

Loại bỏ training example có một hoặc nhiều giá trị bị thiếu

[50]:
```
csv_data = \
'''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''
# If you are using Python 2.7, you need
# to convert the string to unicode:
# csv_data = unicode(csv_data)
```

```
df = pd.read_csv(StringIO(csv_data))
df
```

[50]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

[51]:
```
df.dropna(axis=0)
```

[51]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

Similarly, we can drop columns that have at least one NaN in any row by setting the axis argument to 1:

[52]:
```
df.dropna(axis=1)
```

[52]:

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

Phương thức dropna hỗ trợ một số tham số bổ sung

[53]:
```
# chỉ loại những hàng với tất cả các features = NaN
df.dropna(how='all')
```

[53]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

[54]:
```
df = df.append({'A' : 30. , 'C' : 40., 'D' : 50.} , ignore_index=True)
df
```

[54]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN
3	30.0	NaN	40.0	50.0

[55]:
```
# only drop rows where NaN appear in specific columns (here: 'C')
# chỉ loại bỏ các dòng có NaN ở cột liệt kê trong "subset"
df.dropna(subset=['B'])
```

[55]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

```
1   5.0   6.0   NaN  8.0
2  10.0  11.0  12.0  NaN
```

4.2.2 Pairwise

Phân tích tất cả các trường hợp có các biến quan tâm và do đó tối đa hóa tất cả dữ liệu có sẵn trên cơ sở phân tích (training ex nào có đủ biến thì lấy). Một điểm mạnh của kỹ thuật này là nó tăng sức mạnh trong phân tích của bạn nhưng nó có nhiều nhược điểm. Nó giả định rằng dữ liệu bị thiếu là MCAR. Nếu bạn dùng pairwise thì bạn sẽ nhận được các tập con khác nhau đóng góp vào các phần khác nhau của model, điều này có thể gây khó khăn cho việc giải thích.

```
     ageNA DV1 DV2
[1,]    18  NA   9
[2,]    NA   1   4
[3,]    27   5   2
[4,]    22  -3   7
```

Cases 3 and 4 will be used to find covariance between ageNa & DV1

Cases 2,3 and 4 will be used to find covariance between DV1 and DV2

Giả định MCAR là rất quan trọng, nếu missing data ko thực sự là MCAR, bạn sẽ có các tập con rất phiến diện!

4.2.3 Dropping Variables

Đôi khi bạn có thể bỏ training ex nếu thiếu dữ liệu cho hơn 60% chiều dc quan sát (nhưng chỉ khi bạn biết chúng ko quan trọng)

[56]:
```
# giữ lại các hàng mà mỗi hàng có ít nhất 3 chiều có data
df.dropna(thresh=3)
```

[56]:
```
     A     B     C     D
0  1.0   2.0   3.0   4.0
1  5.0   6.0   NaN   8.0
2 10.0  11.0  12.0   NaN
3 30.0   NaN  40.0  50.0
```

4.3 Imputation

Thông thường việc loại bỏ 1 mẫu huấn luyện hoặc việc loại bỏ tất cả feature đơn giản là không khả thi. Bởi vì chúng ta có thể mất quá nhiều dữ liệu có giá trị. Trong trường hợp này chúng ta sẽ áp dụng các kĩ thuật nội suy khác nhau để ước tính các giá trị còn thiếu từ các mẫu huấn luyện khác trong tập dữ liệu của chúng ta. Một trong những kĩ thuật nội suy phổ biến nhất là đặt giá trị trung bình cho các mẫu huấn luyện bị thiếu bằng trung bình của toàn bộ các mẫu khác trong tập dữ liệu. Một cách thuận tiện để đạt được điều này là dùng SimpleImputer class từ scikit-learn. Được minh họa qua đoạn code sau:

[57]:
```
from sklearn.impute import SimpleImputer
import numpy as np
import pandas as pd

# Dùng bộ dữ liệu về bệnh tiểu đường
dataset = pd.read_csv('pima-indians-diabetes.csv', header=None)
dataset.describe()
```

[57]:

	0	1	2	3	4	5	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

	6	7	8
count	768.000000	768.000000	768.000000
mean	0.471876	33.240885	0.348958
std	0.331329	11.760232	0.476951
min	0.078000	21.000000	0.000000
25%	0.243750	24.000000	0.000000
50%	0.372500	29.000000	0.000000
75%	0.626250	41.000000	1.000000
max	2.420000	81.000000	1.000000

[58]:
```
dataset.head(10)
```

[58]:

	0	1	2	3	4	5	6	7	8
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

[59]:
```
# Đếm số lượng data bị miss ở mỗi feature
print((dataset==0).sum())
```

```
0    111
1      5
```

```
2     35
3    227
4    374
5     11
6      0
7      0
8    500
dtype: int64
```

[60]:
```
from numpy import nan

# Thay thế các giá trị 0 bằng NaN
dataset = dataset.replace(0, nan)
# Đếm số lượng data bị NaN ở mỗi feature
print((dataset.isnull()).sum())
```

```
0    111
1      5
2     35
3    227
4    374
5     11
6      0
7      0
8    500
dtype: int64
```

[61]:
```
dataset.head(10)
```

[61]:

	0	1	2	3	4	5	6	7	8
0	6.0	148.0	72.0	35.0	NaN	33.6	0.627	50	1.0
1	1.0	85.0	66.0	29.0	NaN	26.6	0.351	31	NaN
2	8.0	183.0	64.0	NaN	NaN	23.3	0.672	32	1.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21	NaN
4	NaN	137.0	40.0	35.0	168.0	43.1	2.288	33	1.0
5	5.0	116.0	74.0	NaN	NaN	25.6	0.201	30	NaN
6	3.0	78.0	50.0	32.0	88.0	31.0	0.248	26	1.0
7	10.0	115.0	NaN	NaN	NaN	35.3	0.134	29	NaN
8	2.0	197.0	70.0	45.0	543.0	30.5	0.158	53	1.0
9	8.0	125.0	96.0	NaN	NaN	NaN	0.232	54	1.0

[62]:
```
# Thay thế các giá trị NaN bằng giá trị trung bình của tất cả các mẫu huấn␣
↪luyện khác NaN của feature
sim_imr = SimpleImputer(missing_values=np.nan, strategy='mean')
sim_imr = sim_imr.fit(dataset.values)
imputed_data = sim_imr.transform(dataset.values)
print(imputed_data[:3,:])
```

```
[[  6.         148.          72.          35.         155.54822335
   33.6          0.627       50.           1.        ]
 [  1.          85.          66.          29.         155.54822335
   26.6          0.351       31.           1.        ]
 [  8.         183.          64.          29.15341959 155.54822335
   23.3          0.672       32.           1.        ]]
```

[63]:
```
dataset = dataset.fillna(dataset.mean())
dataset.head(10)
```

[63]:

	0	1	2	3	4	5	6	7	\
0	6.000000	148.0	72.000000	35.00000	155.548223	33.600000	0.627	50	
1	1.000000	85.0	66.000000	29.00000	155.548223	26.600000	0.351	31	
2	8.000000	183.0	64.000000	29.15342	155.548223	23.300000	0.672	32	
3	1.000000	89.0	66.000000	23.00000	94.000000	28.100000	0.167	21	
4	4.494673	137.0	40.000000	35.00000	168.000000	43.100000	2.288	33	
5	5.000000	116.0	74.000000	29.15342	155.548223	25.600000	0.201	30	
6	3.000000	78.0	50.000000	32.00000	88.000000	31.000000	0.248	26	
7	10.000000	115.0	72.405184	29.15342	155.548223	35.300000	0.134	29	
8	2.000000	197.0	70.000000	45.00000	543.000000	30.500000	0.158	53	
9	8.000000	125.0	96.000000	29.15342	155.548223	32.457464	0.232	54	

	8
0	1.0
1	1.0
2	1.0
3	1.0
4	1.0
5	1.0
6	1.0
7	1.0
8	1.0
9	1.0

4.3.1 Time-Series Problems

Last Observation Carried Forward (LOCF) & Next Observation Carried Backward (NOCB) Đây là một cách tiếp cận thống kê phổ biến để phân tích dữ liệu đo lường lặp lại theo thời gian mà một số quan sát tiếp theo có thể bị thiếu. Dữ liệu dọc theo dõi cùng một mẫu tại các thời điểm khác nhau. Cả hai phương pháp này đều có thể gây ra sai lệch trong phân tích và hoạt động kém khi dữ liệu có chứa trend nào đó bất thường

Linear Interpolation Phương pháp này hoạt động tốt cho time series analysis với trend nhưng không phù hợp với dữ liệu thay đổi mang tính chu kì, cứ lặp lại hoài như thế

Seasonal Adjustment + Linear Interpolation Phương pháp này hoạt động tốt cho dữ liệu có cả trend và tính chu kì

Mean, Median and Mode Bạn có thể tính các đại lượng này và thay thế vào chỗ thiếu

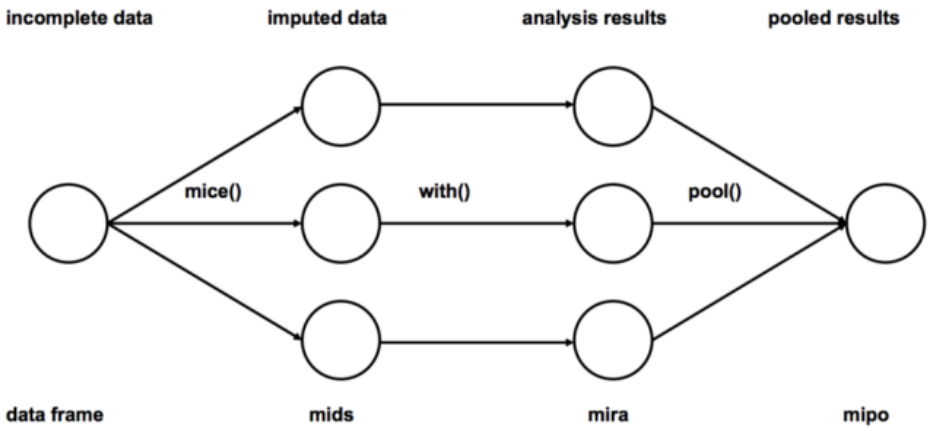
Linear regression Đầu tiên, vài predictor (với các chiều thuộc tính nào đó) cho missing value dc chọn bằng correlation matrix. Predictor tốt nhất sẽ dc lấy ra, các chiều có tương quan sẽ là các biến nhận vào của một hàm tuyến tính (independent variables), thuộc tính chứa missing value sẽ là kết quả (dependent variable). Cách này có thể dự đoán cho các vector chưa đầy đủ (nhưng phải có đủ data để tính ra chỗ missing).

Bằng cách tạo ra nhiều hàm regression, ta có thể tiến hành các thao tác lặp đi lặp lại để điền đủ vào các chỗ bị thiếu.

Có điều, dùng hàm regression để input missing data sẽ làm data trở nên "fit too well" - khác với thực tế, điều đó ko tốt.

Đặc biệt, khi bạn dùng cách này là bạn giả định có một linear relationship giữa các biến, thực tế hiếm khi điều đó đủ, đúng.

Multiple imputation



Imputation:

Tính các Missing values của tập dữ liệu chưa hoàn chỉnh m lần (m = 3 trong hình). Lưu ý rằng các giá trị được quy định được rút ra từ một phân phối. Sự mô phỏng về việc lấy mẫu ngẫu nhiên không bao gồm sự không chắc chắn (uncertainty) trong các model params. Cách tiếp cận tốt hơn là sử dụng Markov Chain Monte Carlo (MCMC) simulation.

Bước này dẫn đến m bộ dữ liệu hoàn chỉnh.

Analysis:

Phân tích từng bộ dữ liệu trong số m đã hoàn thành với statistical method of interest

Pooling:

Tích hợp kết quả phân tích về các model params từ m mẫu để đưa ra suy diễn thống kế (inference)

4.3.2 General problem

Imputation of Categorical Variables Mode imputation là một phương pháp nhưng nó chắc chắn sẽ gây bias

Missing values có thể được coi như một danh mục riêng biệt. Có thể tạo một danh mục khác cho các Missing values và sử dụng chúng như một công cụ khác. Đây là phương pháp đơn giản nhất.

Prediction models: chia dataset của mình thành hai tập: Một tập không có missing values cho huấn luyện và một tập khác có giá trị bị thiếu để test

4.3.3 Using KNN (K Nearest Neighbors)

KNN có thể dự đoán cả thuộc tính rời rạc (giá trị thường xuyên nhất trong số k láng giềng gần nhất) và thuộc tính liên tục (giá trị trung bình trong số k láng giềng gần nhất)

1. Continuous data: Các độ đo khoảng cách thường được sử dụng cho dữ liệu liên tục là Euclidean, Manhattan và Cosine
2. Categorical data: Khoảng cách Hamming thường được sử dụng trong trường hợp này. Nó đặt ra 2 mẫu có số thuộc tính bằng nhau, đếm các giá trị không giống nhau giữa hai mẫu => Khoảng cách Hamming bằng với số thuộc tính mà giá trị khác nhau. KNN đơn giản để hiểu và dễ thực hiện Một trong những hạn chế rõ ràng của thuật toán KNN là nó trở nên tốn thời gian khi phân tích các tập dữ liệu lớn vì nó tìm kiếm các trường hợp tương tự thông qua toàn bộ tập dữ liệu. Hơn nữa, độ chính xác của KNN có thể bị suy giảm nghiêm trọng với dữ liệu chiều cao vì có rất ít sự khác biệt giữa neighbor gần nhất và xa nhất.

Trong số tất cả các phương pháp được thảo luận ở trên, multiple imputation và KNN được sử dụng rộng rãi và multiple imputation đơn giản hơn thường được ưu tiên