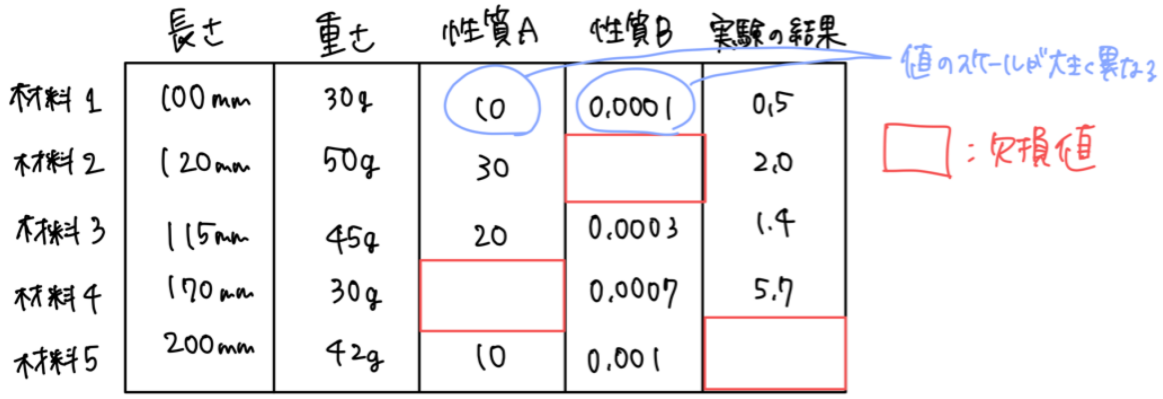
2章 データの前処理

機械学習でデータを解析する際には「データの前処理」が必要です。

なぜ？前処理をしないとどうなるの？
実験を重ねて用意した材料データをそのまま機械学習で解析しても、ほとんどの場合はうまくいきません。データの中には欠損している部分やエラーの結果が含まれていたり、変数ごとに数値のスケールが大きく異なっていたりして、正しい解析が妨げられてしまいます。 データの前処理とは、機械学習の前にデータの欠損値を補完したり、数値のスケールを揃えたりすることを指します。



2.1 欠損値に対して

代表的な3つの方法を紹介します。

1. リストワイズ：欠損値がある行を消す
2. 平均値代入法：欠損値を補完（平均値）
3. 回帰代入法：欠損値を補完（周りのデータから回帰学習した値で埋める）

コードを組んで実際に動かしていきましょう。

今回はリストワイズと平均値代入法について紹介します。

はじめに、練習に使うデータセットを作ります。

```
import numpy as np
import pandas as pd

df = pd.DataFrame(
    {
        'A':[1, np.nan, 3, 4, 5],
        'B':[2, 4, 6, np.nan, 10],
        'C':[10, 20, 30, 40, 50]
    }
)
df
```

```
# DataFrameの各要素が欠損値か確かめる
df.isnull()
```

リストワイズ

```
# 欠損値削除（DataFrameのdropnaメソッド）

df_drop = df.dropna()
df_drop
```

平均値代入法

```
# 欠損値補完1(Dataframeのfillnaメソッド)

df_fillna = df.fillna(df.mean())
df_fillna
```

```
# 欠損値補完2（scikit-learnのimputeモジュールのSimpleImputerクラス）

from sklearn.impute import SimpleImputer

# 平均値で欠損値を補完するためのインスタンスを作成する
imp = SimpleImputer(strategy = 'mean')
# 欠損値を補完
imp.fit(df)
imp.transform(df)
```

- 一般的には、欠損値の割合が10%以下であればリストワイズでも良いとされています。
- 欠損値の分布がランダム（MAR）か偏っている（MNAR）かに応じて対処も必要です。

  MCAR：欠損値が完全にランダムに発生している

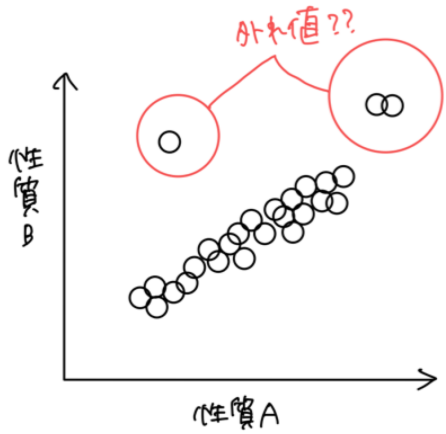
  MAR：欠損値の発生が完全にランダムではないものの、欠損しているか否かが欠損値そのものに依存していない
  （例）多くの女性が体重を自己申告しなかった場合。欠損値は体重が重い・軽いに関係なく発生しています。すなわち体重の値そのものには依存していません。

  MNAR：欠損値の発生が欠損値そのものに依存している
  （例）肥満者の多くが体重を自己申告しなかった場合。欠損値の発生は体重が重い場合に起こりやすくなるので、体重の値そのものに依存していることになります。

  MNARに対して統計的に対処することは難しく、そのような欠損値を多く含む変数は分析データから除外することを検討しましょう。
  ただしMARとMNARを識別するのは困難です。
  このような場合、多重代入法を用いることが推奨されます。

  多重代入法では、欠損値を代入したデータセットを複数作成し、データセットごとに推計値を求め、その結果を統合します。

2.2 ハズレ値の対応



- 人間が取り除く
- クラスタリングで数の少ないクラスタを消去
- 正規分布を仮定したモデルを構築し、検定から外れ値を検出する方法（統計的検定アプローチ）
- ハズレ値かもしれないデータをとりあえず排除してみたとき分散がどのように変化するかという観点から外れ値を検出する手法（偏差アプローチ）

2.3 データの正規化・標準化

	性質A	性質B	性質C
材料1	10	0.1	0.0000002
材料2	20	0.3	0.00000001
材料3	50	0.2	0.000005

値の大きさが全然違う!

データ間で大きくスケールの異なるデータを使用して機械学習で解析しても、ほとんどの場合うまくいきません。

なぜなら、機械学習は大きなスケールのデータの影響を過度に受けてしまうからです。

データのスケールを揃えるには、以下の2つの方法があります。

1. 正規化 ：データの値を一定の範囲に収める（0〜1か、−1〜1の範囲にすることが多い）
2. 標準化 ：データの平均を0、分散を1にする

実際にコードを動かしてみましょう。

今回は5×2行列のデータセットを作ります。

```
# DataFrameを作成する
df = pd.DataFrame(
    {
        'A':[1, 2, 3, 4, 5],
        'B':[100, 200, 400, 550, 800]
    }
)
df
```

正規化のコードです。

```
# 最小最大正規化（=正規化）
from sklearn.preprocessing import MinMaxScaler

# 最小最大正規化のインスタンスを作成
mmsc = MinMaxScaler()

# 最小最大正規化を実行
mmsc.fit(df)
mmsc.transform(df)
```

標準化のコードです。

```
# 分散正規化（=標準化）
from sklearn.preprocessing import StandardScaler

# 分散正規化のインスタンスを作成
stdsc = StandardScaler()

# 分散正規化を実行
stdsc.fit(df)
stdsc.transform(df)
```

- 正規化と標準化の使い分けについて
  MIの分野では 標準化 を使うことが多いです。
  なぜなら、正規化は外れ値の影響を大きく受けてしまうからです。

  （厳密には、機械学習のアルゴリズムによって使い分けますが、最初のうちは気にしなくて大丈夫です。）