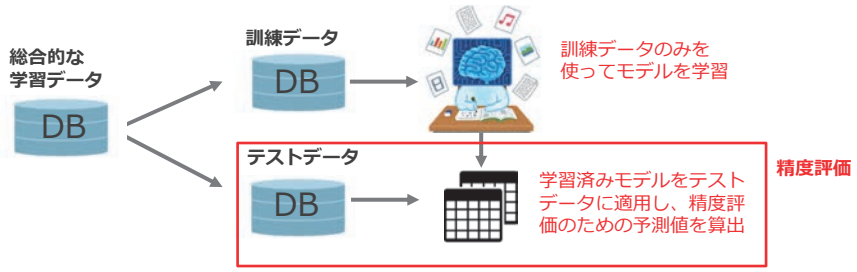
学習済みモデルの精度評価

精度検証とは

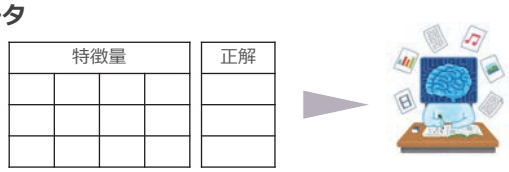
機械学習の目的は、学習データから汎用的なパターンを見出すように学習を行い、未知データに対しても精度よく予測できるようになること

- 未知データに対して高精度に予測する能力は**汎化性能**という
- 学習済みモデルを本番環境で使う前に、その汎化性能を証明することが重要
- 精度を検証する際にまず学習データを２部分に分ける
  - **訓練データ**（学習データと呼ぶこともある）：モデルを学習するため
  - **テストデータ**：汎化性能を評価するため



精度検証の概念図

■ 訓練データ

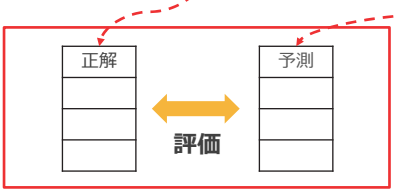


訓練データのみを
使ってモデルを学習

■ テストデータ

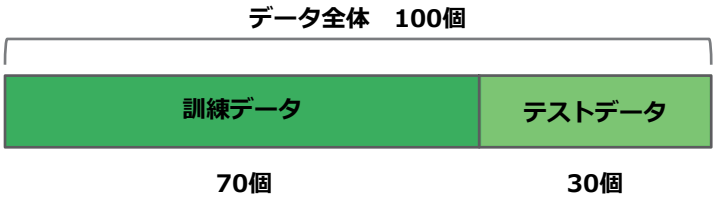
学習済みモデルにテストデータを入力し、
精度評価のための予測値を算出

■ 精度検証



データ分割の手法

- データを分割する方法の中で一番基本なのは **ホールドアウト法**
  - **固定の割合**をもって、訓練データとテストデータ２つの部分に分割する
  - 訓練：テスト = 7:3　か　8:2 の場合が多い



- 単純な手法なので、**短時間で精度検証が済む**（データ量が大きい時に使いたい）
- 一方で、分割後のデータに**ばらつきや偏りが生じると、検証の信頼性が落ちる**
  （特にデータ量が少ないときに注意）
  例：スパム検出モデルの作成において、データ分割後、スパムデータがほぼ全てテストデータの方に入ってしまったら、高精度なモデルを作れない

【必要なライブラリを読み込む】

```
#必要なパッケージを呼び出す
%matplotlib inline

from sklearn.tree import DecisionTreeClassifier as dtc
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

分析作業は、データをpandasの表オブジェクトDataFrameに格納するところから始まることが多い

【データを読み込む】

```
df = pd.read_csv("./data/cancer_cell.csv")
```

pandas の read_csv() 関数

書式：DataFrame名 = **pd.read_csv**(“ファイルパス”)

読み込んだデータの確認

```
print(df.shape)
df.head()
```

(569, 31)

関数 shape() を用いて、DataFrameの構造を (行数, 列数) で確認

関数 head() を用いて、DataFrameの冒頭の数行の実際の様子を確認

mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

```
df.columns
```

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

DataFrameのカラム名を一覧で見る

これで、データが正しく読み込まれたのか、を簡単に確認した

データを理解する（正解カラム; target）

モデルを構築する前に、各カテゴリのボリューム、欠損の有無、変数間の相関関係などを確認し、データと「知り合う」ことが重要

```
1  df.target.unique()
```

Targetカラムに含まれる一意な値を確認（0と1）

```
array([0, 1])
```

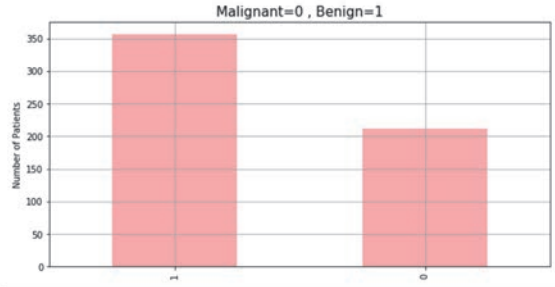
```
1  #良性、悪性のそれぞれのデータボリュームを見る
2  df['target'].value_counts()
```

```
1  357
0  212
Name: target, dtype: int64
```

Targetには２つのカテゴリがある
それぞれのデータボリュームを確認

0: 悪性 Malignant
1: 良性 Benign

```
1  df.target.value_counts().plot(kind='bar', alpha = 0.3, facecolor = 'r', figsize=(10,5))
2  plt.title("Malignant=0 , Benign=1", fontsize = '15')
3  plt.ylabel("Number of Patients")
4  plt.grid(b=True)
```



悪性と良性のボリュームを可視化

離散的なカテゴリデータなので
棒グラフを使用

データを理解する（欠損値の有無）

```
1  df.isnull().sum()
```

```
mean radius              0
mean texture             0
mean perimeter           0
mean area                0
mean smoothness          0
mean compactness         0
mean concavity           0
mean concave points      0
mean symmetry            0
mean fractal dimension   0
radius error             0
texture error            0
perimeter error          0
area error               0
smoothness error         0
compactness error        0
concavity error          0
concave points error     0
symmetry error           0
fractal dimension error  0
worst radius             0
worst texture            0
worst perimeter          0
worst area               0
worst smoothness         0
worst compactness        0
worst concavity          0
worst concave points     0
worst symmetry           0
worst fractal dimension  0
target                   0
dtype: int64
```

Pandasのisnull().sum() メソッドを用いると、DataFrameの各列に欠損値が何個あるかを見ることができる

➡ 欠損値がないことが分かった

- 今回はscikit-learnで提供されている「勉強用」のデータを使用
- 一方で、実務で使う実世界の中で蓄積されたデータには異常値や外れ値や欠損値があることが多い