機械学習
さまざまな予測性能評価

Pythonによる機械学習入門

第１３章

CONTENTS	
１３．１	回帰の予測性能評価
１３．２	分類の予測性能評価
１３．３	K分割交差検証
１３．４	第１３章のめとめ
１３．５	練習問題
１３．６	練習問題の解答

１３．１　回帰の予測性能評価　P458～P463

１３．１．１　平均絶対誤差（MAE）の復習　P458～P458

回帰の評価指標
- 決定係数
- 平均絶対誤差
- 平均２乗誤差

	１件目	２件目	３件目
１．予測の値	1	3	-1
２．実際の値	2	0	1
３．（予測結果）-（実際）の値	-1	3	-2
４．３の結果の絶対値	1	3	2

平均絶対誤差
MAE

（１＋３＋２）／３＝２

１３．１．２　平均２乗誤差（MSE）　P459～P459

回帰の評価指標
- 決定係数
- 平均絶対誤差
- 平均２乗誤差

	１件目	２件目	３件目
１．予測の値	1	3	-1
２．実際の値	2	0	1
３．（予測結果）-（実際）の値	-1	3	-2
４．３の２乗値	1	9	4

平均２乗誤差
MSE

（１＋9＋４）／３＝
１４／３

平均絶対誤差　プラスかマイナスかの場合分けが必要

平均２乗誤差　２乗するだけなので計算が楽

誤差を２乗するので
外れ値が評価に与える影響が大きくなる　ただし

コード13-1 データとモデルの準備

```
# 欠損値があるままでは学習できないので欠損値処理だけ行う
import pandas as pd

# cinema.csvを読み込む
df = pd.read_csv('cinema.csv')
# 欠損値を平均値で穴埋め
df = df.fillna(df.mean())
# 特徴量
x = df.loc[:, 'SNS1':'original']
# 正解データ
t = df['sales']
# 線形回帰をインポート
from sklearn.linear_model import LinearRegression
# 線形回帰モデルを作成しインスタンスを生成
model = LinearRegression()
# 学習
model.fit(x, t)
```

cinema.csv

cinema_id	SNS1	SNS2	actor	original	sales
1375	291	1044	8808.994029	0	9731
1000	363	568	10290.70937	1	10210
1390	158	431	6340.388534	1	8227
1499	261	578	8250.485081	0	9658
1164	209	683	10908.53955	0	9286
1009		866	9427.21452	0	9574
1417	153	362	7237.639848	1	7869
1688	473	856		1	9804
1503	117	114	8843.854509	1	9023

特徴量

正解データ

データの各列の内容

列名	意味
cinema_id	映画作品のID
SNS1	公開後１０日以内にSNS1でつぶやかれた数
SNS2	公開後１０日以内にSNS２でつぶやかれた数
actor	主演俳優の昨年のメディア露出度。actorの値が大きいほどりしゅつしている
original	原作があるかどうか（あるなら１、ないなら０）
sales	最終的な興行収入（単位：万円）

コード13-2 平均２乗誤差を計算する

```
from sklearn.metrics import mean_squared_error
# 訓練データでのMSE値
# モデルに予測させる
pred = model.predict(x)
# 予測値と実際値でMSEを計算
mse = mean_squared_error(pred, t)
mse
```

実行結果

```
151986.03957624524
```

平均２乗誤差の計算

```
mean_squared_error( 予測結果, 正解データ )
```

※ 事前に from sklearn.metrics import mean_squared_error
でインポートする必要がある。

１３．１．３　２乗平均平方根誤差（RMSE）　P461～P463

２乗平均平方根誤差

$$\mathrm{RMSE} = \sqrt{MSE}$$

コード13-3 RMSEの計算

```
import math
math.sqrt(mse) # RMSEの計算
```

実行結果

```
389.8538695155471
```

- ２乗平均平方根誤差（RMSE）：外れ値を敏感に検知する
- 平均絶対誤差（MAE）：外れ値があってもそれほど変化しない

コード13-4 予測結果と実際の誤差を検証する

```
from sklearn.metrics import mean_absolute_error
yosoku = [2, 3, 5, 7, 11, 13] # 予測結果をリストで作成
target = [3, 5, 8, 11, 16, 19] # 実際の結果をリストで作成

mse = mean_squared_error(yosoku, target)
print('rmse:{}'.format(math.sqrt(mse)))
print('mae:{}'.format(mean_absolute_error(yosoku, target)))

print('外れ値の混入')
yosoku = [2, 3, 5, 7, 11, 13, 46] # 実際には23だけど46と予測
target = [3, 5, 8, 11, 16, 19, 23]
mse = mean_squared_error(yosoku, target)
print('rmse:{}'.format(math.sqrt(mse)))
print('mae:{}'.format(mean_absolute_error(yosoku, target)))
```

実行結果

```
rmse:3.8944404818493075
mae:3.5
外れ値の混入
rmse:9.411239481143202
mae:6.285714285714286
```

ほぼ同じ

外れ値の影響を受けて乖離している

１３．２ 分類の予測性能評価 P464～P472

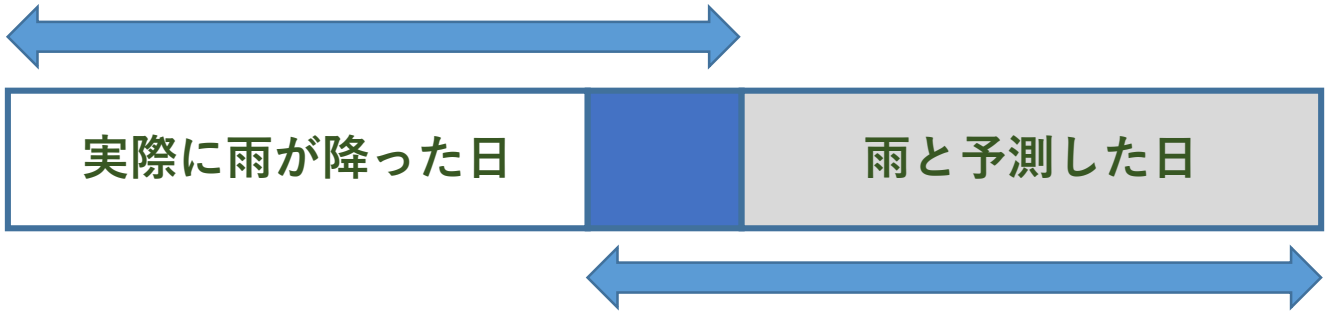
１３．２．１ 適合率と再現率 P464～P471

分類における性能評価の指標
- 適合率
- 再現率

リスクとコストのどちらかを重視する予測モデルの性能の指標

降水予測
- コスト：傘を持ち歩く
- リスク：雨に降られる

■適合率 雨が降ると予測した件数のうち、実際に雨が降った件数の比率



＜浅木さんの場合＞

		予測	
		降らない	降る
実際	降らなかった	40	30
	降った	2	28

雨の適合率＝28/(28+30)＝約0.48

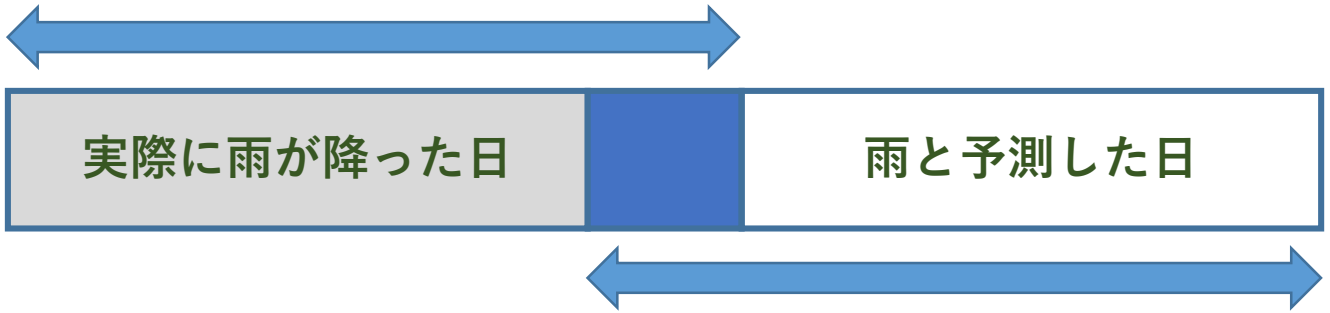
＜松田くんの場合＞

		予測	
		降らない	降る
実際	降らなかった	69	1
	降った	20	10

雨の適合率＝10/(1+10)＝約0.90

コストを抑えたい

■再現率　実際に雨が降った件数のうち、雨が降ると予測した件数の比率



＜浅木さんの場合＞

		予測	
		降らない	降る
実際	降らなかった	40	30
	降った	2	28

雨の再現率=28/(2+28)=約0.93

＜松田くんの場合＞

		予測	
		降らない	降る
実際	降らなかった	69	1
	降った	20	10

雨の再現率=10/(20+10)=約0.33

リスクを押さえたい

自分が作りたい予測モデルが実際に運用されているところをイメージして、適合率を重視するべきなのか、
再現率を重視すべきなのかを判断する必要がある

コード13-5 データの準備

```
# データの準備
# Survived.csvの読み込み
df = pd.read_csv('Survived.csv')
# 欠損値を平均値で穴埋め
df = df.fillna(df.mean())
# 特徴量
x = df[['Pclass', 'Age']]
# 正解データ
t = df['Survived']
```

コード13-6 モデルの準備

```
# モデルの準備
# 決定木をインポート
from sklearn import tree
# 決定木モデルを生成
model = tree.DecisionTreeClassifier(
            max_depth = 2,  random_state = 0)
# 学習
model.fit(x, t)
```

Survived.csv

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	male	22	1	0	A/5 21171	7.25		S
2	1	1	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	female	35	1	0	113803	53.1	C123	S
5	0	3	male	35	0	0	373450	8.05		S
6	0	3	male		0	0	330877	8.4583		Q
7	0	1	male	54	0	0	17463	51.8625	E46	S

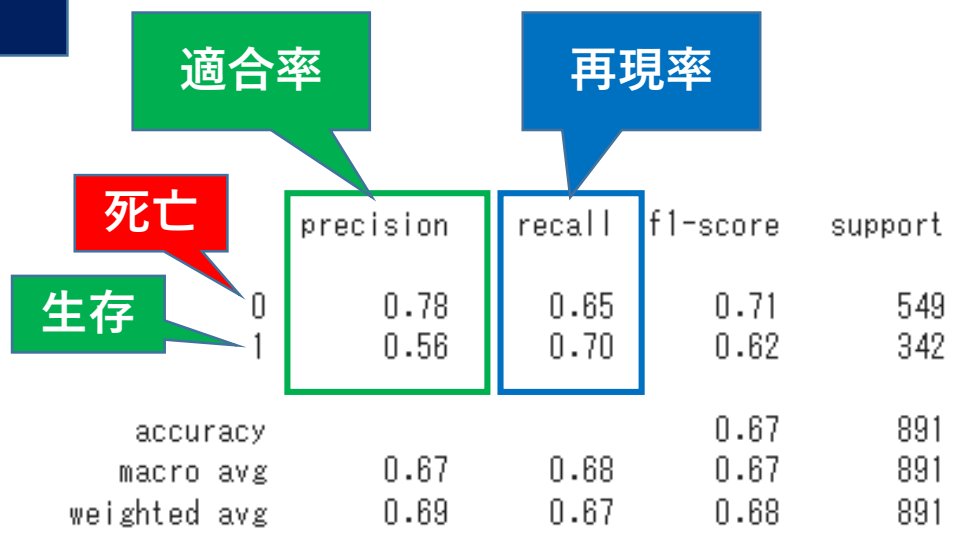
各列の意味

列名	意味
PassengerId	乗客ID
Pclass	チケットクラス（１等、２等、３等）
Age	年齢
Parch	同乗した、自身の親と子供の総数
Fare	運賃
Emberked	搭乗港
Survived	１：生存、０：死亡
Sex	性別
SibSp	同乗した兄弟や配偶者の総数
Ticket	チケットID
Cabin	部屋番号

コード13-7 再現率と適合率を一括で計算

```
# classification_report関数のインポート
from sklearn.metrics import classification_report
# 予測
pred = model.predict(x)
# 再現率と適合率を求める
out_put = classification_report(y_pred = pred, y_true = t)
# 表示
print(out_put)
```

実行結果



	precision	recall	f1-score	support
0	0.78	0.65	0.71	549
1	0.56	0.70	0.62	342
accuracy			0.67	891
macro avg	0.67	0.68	0.67	891
weighted avg	0.69	0.67	0.68	891

コード13-8 classification_report関数にパラメータ引数を指定

```
# パラメータ引数を指定
out_put = classification_report(y_pred = pred,
                  y_true = t, output_dict = True)
# out_putをデータフレームに変換
pd.DataFrame(out_put)
```

戻り値をディクショナリ型で出力

実行結果

死亡

生存

適合率

再現率

	0	1	accuracy	macro avg	weighted avg
precision	0.778742	0.558140	0.672278	0.668441	0.694066
recall	0.653916	0.701754	0.672278	0.677835	0.672278
f1-score	0.710891	0.621762	0.672278	0.666326	0.676680
support	549.000000	342.000000	0.672278	891.000000	891.000000

f1-score ＝ F値 適合率と再現率の平均と解釈できる指標

適合率と再現率はトレードオフの関係

一方が大きくなるようにチューニングするともう一方は低下する傾向が強い

２つの指標を同時にモデルチューニング時の指標にするのは難しい

同時に考えたいときは、f1-scoreを利用する

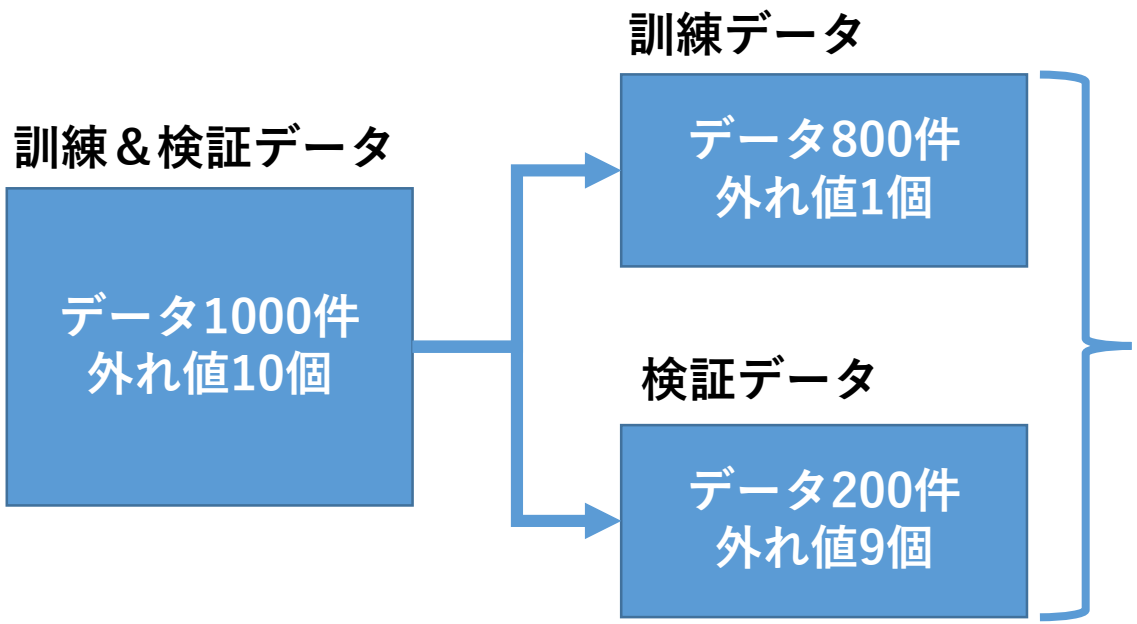
f1-scoreは必ず０以上１以下となり、１に近いほど予測精度が高い

１３．３ K分割交差検証 P473～P484

１３．３．１ ホールドアウト法の問題点 P473～P475

ホールドアウト法 学習に利用するデータと予測性能をテストするデータに分割すること

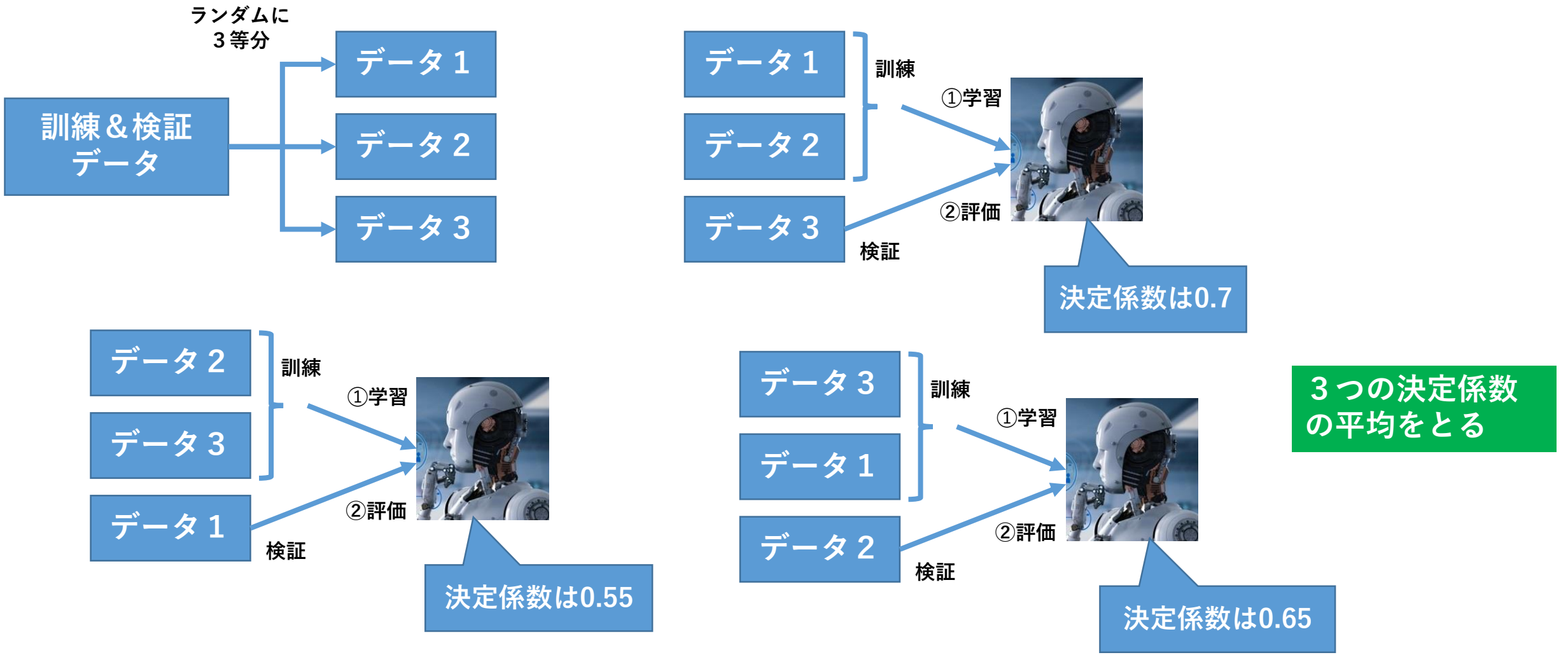
ホールドアウト法の問題点 外れ値が訓練データに混ざっているとき



適切なチューニングができていても、
検証データではいい結果が得られない。

モデルの性能の悪い原因が、
本質的なチューニングなのか、
分割時のデータの型よりなのか、
がわからない。

K分割交差検証の「K」には３以上の整数が入る

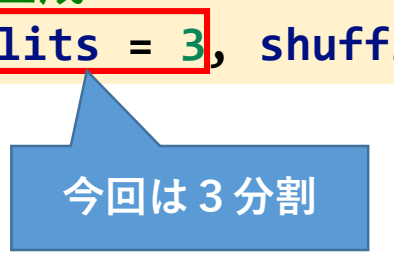


コード13-9 K分割交差検証のためのデータ準備

```
# cinema.csvの読み込み
df = pd.read_csv('cinema.csv')
# 学習できないので欠損値処理だけ行う
# 欠損値を平均値で穴埋め
df = df.fillna(df.mean())
# 特徴量
x = df.loc[:, 'SNS1':'original']
# 正解データ
t = df['sales']
```

コード13-10 KFoldの処理で分割時の条件を指定

```
# KFoldをインポート
from sklearn.model_selection import KFold
# インスタンスを生成
kf = KFold(n_splits = 3, shuffle = True, random_state = 0)
```



コード13-11 cross_validate関数で交差検証を行う

```
# 線形回帰モデルをインポート
from sklearn.linear_model import LinearRegression
# 線形回帰モデルを生成
model = LinearRegression()
# cross_validate関数をインポート
from sklearn.model_selection import cross_validate
# 交差検証を行う
result = cross_validate(model, x, t, cv = kf, scoring = 'r2',
                        return_train_score = True)

print(result)
```

分割条件

評価指標の指定
（今回は決定係数）

実行結果

```
{'fit_time': array([0.01177454, 0.01562929, 0. ]),
 'score_time': array([0., 0., 0.]),
 'test_score': array([0.72465051, 0.71740834,0.75975591]),
 'train_score':array([0.76928501, 0.76368104, 0.75780074])}
```

検証データでの
３回の決定係数値

訓練データでの３回の決定係数値（過学習確認用）

コード13-12 平均値を計算する

```
# 平均値を計算する
sum(result['test_score']) / len(result['test_score'])
```

実行結果

```
0.733938254177434
```

K分割検証

・分割条件の指定

```
変数 = KFold( n_splits = 分割数, shuffle = True, randam_state = 整数)
```

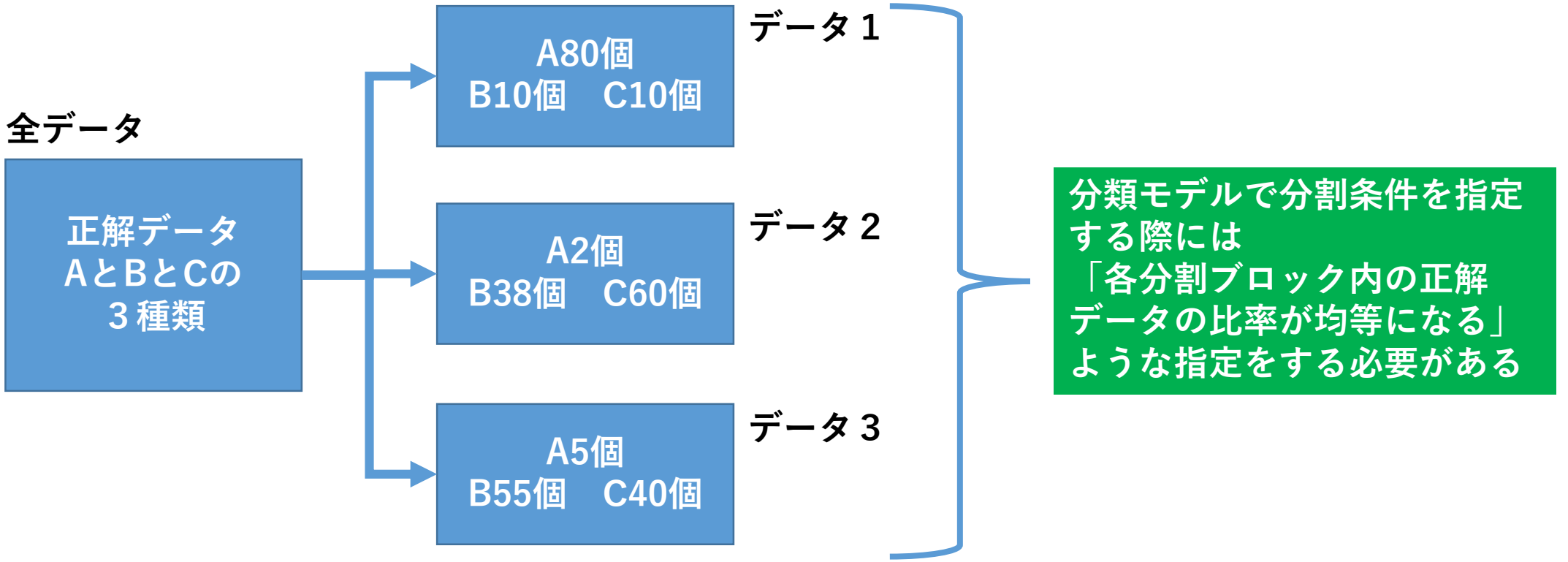
※ from sklearn.model_selection import KFold を事前に行っている
※ shuffle = False を指定するとランダムに分割されない
※ random_state はランダム分割時の乱数固定

・K分割交差検証

```
cross_validate(モデル変数, 特徴量データ, 正解データ, cv=分割条件,
        scoring='評価仕様', return_train_score=True)
```

※ from sklearn.model_selection import cross_validate を事前にしている
※ モデル変数は事前に作成済みの学習モデル（学習する前の状態）
※ 評価指標は、”r2” や “accuracy” などを指定できる
※ 評価指標はリスト形式で指定すると、複数個を同時に検証できる
※ return_train_score = False を指定すると、訓練データの予測性能は計算しない

分割したデータの中で、正解データの偏りが発生しやすくなって、結局いいモデルが作れなくなってしまう



コード13-13 StratifieldFoldのインポート

```
# StratifieldKFoldのインポート
from sklearn.model_selection import StratifiedKFold
# StratifieldKFoldインスタンスを生成（引数はKFoldと同じ）
skf = StratifiedKFold(n_splits = 3, shuffle = True,random_state = 0)
```