

機械学習 アンサンブル学習

管理工学科
篠沢佳久

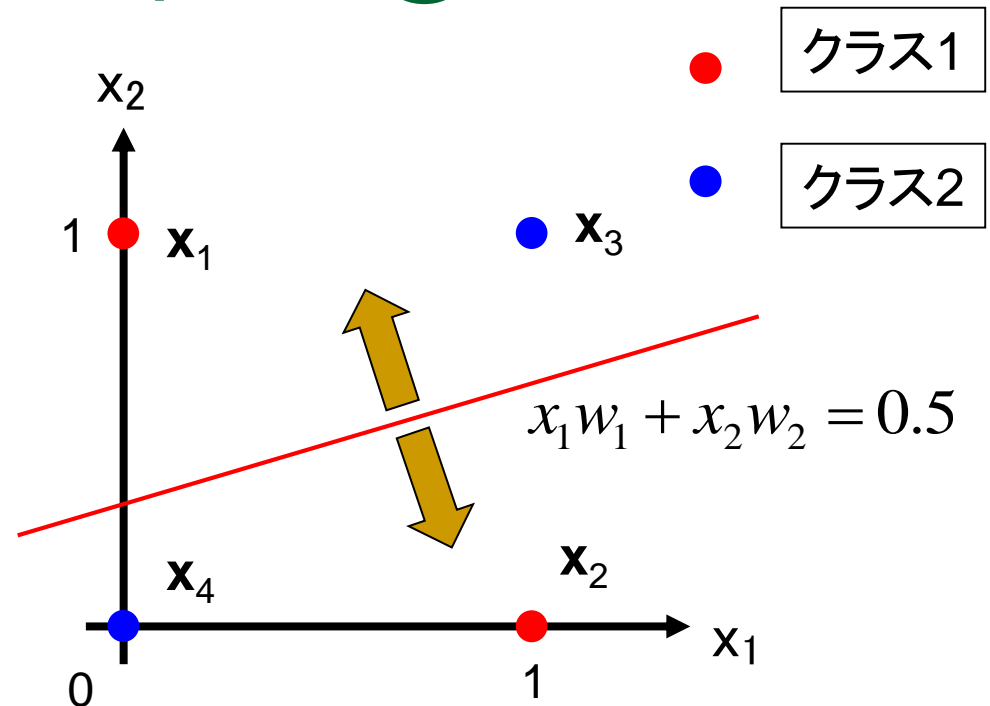
資料の内容

- アンサンブル学習
 - バギング, ブートストラップ
 - ランダムフォレスト
- 実習
 - バギング (Breast Cancer Dataset)
 - ランダムフォレスト
 - クラス分類 (Breast Cancer Dataset)
 - 回帰 (Boston Dataset)

アンサンブル学習

線形分離不可能な問題①

	x_1	x_2	
x_1	0	1	クラス1
x_2	1	0	クラス1
x_3	1	1	クラス2
x_4	0	0	クラス2

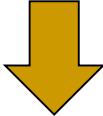


クラス1とクラス2を識別できる重みは存在しない

→ 線形分離不可能

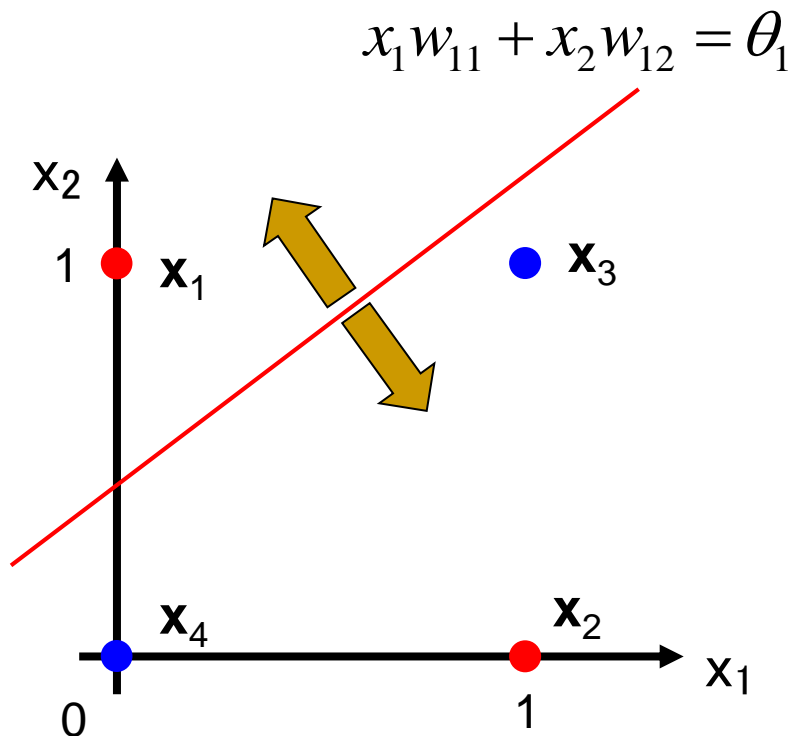
→ パーセプトロンでは解けない問題

線形分離不可能な問題②

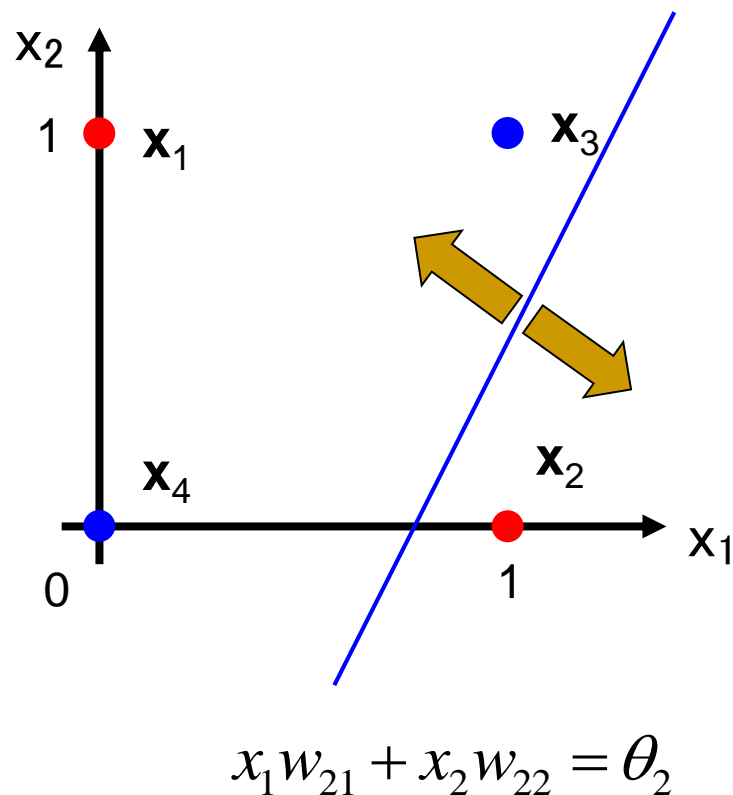
- 機械学習において対象となる問題は、線形分離不可能な問題が占める
 - パーセプトロンは役に立たない手法?
 - Marvin Minsky (1969)
- 
- 線形識別関数を組み合わせることによって解決が可能

線形識別関数を組み合わせた解法①

識別関数1



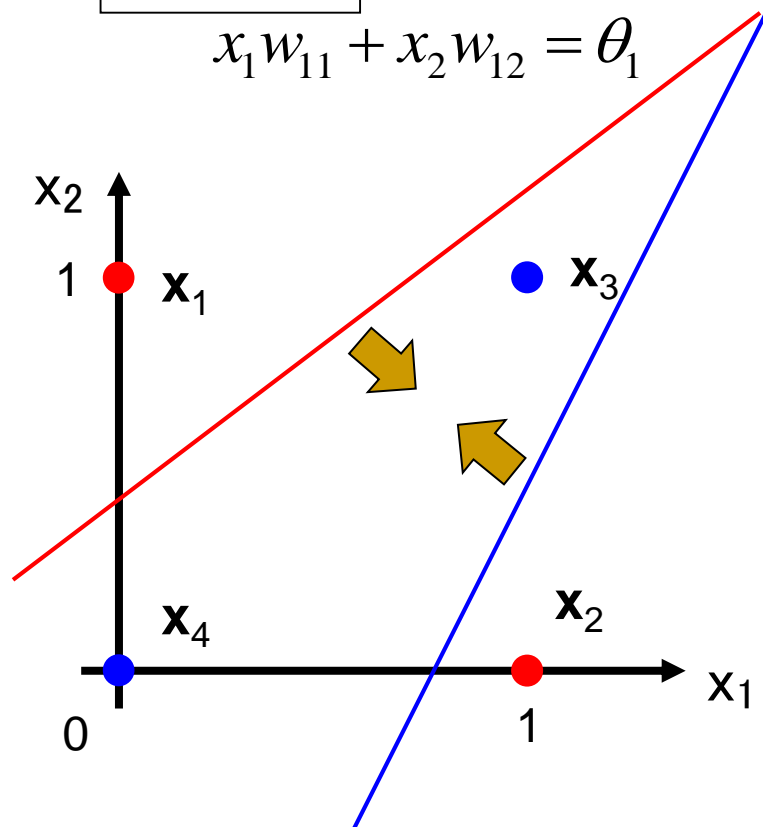
識別関数2



線形識別関数を組み合わせた解法②

識別関数1

$$x_1 w_{11} + x_2 w_{12} = \theta_1$$



識別関数2

$$x_1 w_{21} + x_2 w_{22} = \theta_2$$

新しい識別関数

$$F(\mathbf{x}) = \alpha_1 f_1(\mathbf{x}) + \alpha_2 f_2(\mathbf{x})$$

識別関数1

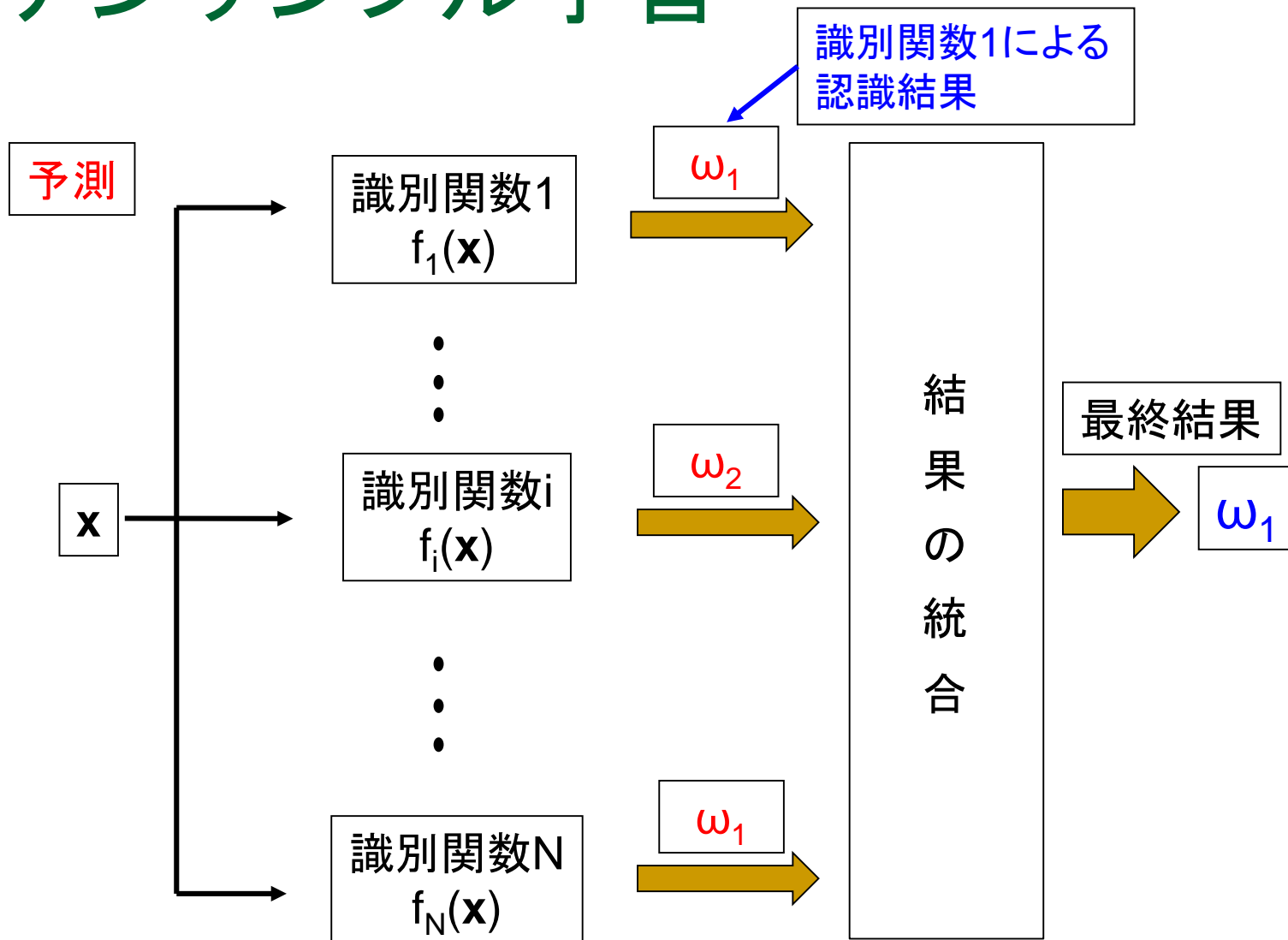
識別関数2

線形識別関数を組み合わせること
によって新しい識別関数を構築

識別関数を組み合わせた解法

- アンサンブル学習(集合学習)
 - 識別関数(次頁)を複数個組み合わせ, 複数個の予測結果を統合し, 最終的な予測結果を求める
- ニューラルネットワーク(人工的神経回路網)
 - パーセプトロンを拡張
 - 誤差逆伝播則(一般化デルタルール)

アンサンブル学習



*クラス分類, 回帰ともにできます

アンサンブル学習における識別関数

- 弱分類器, 仮説と呼ばれる
- どのような手法を利用しても良い*
 - ロジスティック回帰
 - 最近傍法
 - ベイズ決定則
 - 決定木
 - 線形識別関数
 - サポートベクターマシン

*当然ですが, ニクラス分類問題では正解率が50%以上の識別関数を用意する必要があります

アンサンブル学習の例①

クラス1とクラス2を分ける識別関数を求める

x_1	x_2	x_3	x_4	正解
1	0	1	1	クラス1
1	0	1	1	クラス1
1	1	1	1	クラス1
1	1	1	0	クラス2
1	0	1	0	クラス2
1	1	0	1	クラス2
1	0	0	1	クラス2
1	1	0	1	クラス2

アンサンブル学習の例②

識別関数1

$$f_1(\mathbf{x}) = \begin{cases} 1 & \text{if } x_3 = 1 \\ 0 & \text{if } x_3 = 0 \end{cases}$$

x_1	x_2	x_3	x_4	正解	識別関数1
1	0	1	1	クラス1	1
1	0	1	1	クラス1	1
1	1	1	1	クラス1	1
1	1	1	0	クラス2	1
1	0	1	0	クラス2	1
1	1	0	1	クラス2	0
1	0	0	1	クラス2	0
1	1	0	1	クラス2	0

アンサンブル学習の例③

識別関数2

$$f_2(\mathbf{x}) = \begin{cases} 1 & \text{if } x_4 = 1 \\ 0 & \text{if } x_4 = 0 \end{cases}$$

x_1	x_2	x_3	x_4	正解	識別関数1	識別関数2
1	0	1	1	クラス1	1	1
1	0	1	1	クラス1	1	1
1	1	1	1	クラス1	1	1
1	1	1	0	クラス2	1	0
1	0	1	0	クラス2	1	0
1	1	0	1	クラス2	0	1
1	0	0	1	クラス2	0	1
1	1	0	1	クラス2	0	1

アンサンブル学習の例④

識別関数

$$F(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x}) \Rightarrow \begin{cases} \mathbf{x} \in \omega_1 & \text{if } F(\mathbf{x}) = 2 \\ \mathbf{x} \in \omega_2 & \text{if } F(\mathbf{x}) \neq 2 \end{cases}$$

x_1	x_2	x_3	x_4	正解	識別関数1	識別関数2	最終結果
1	0	1	1	クラス1	1	1	クラス1
1	0	1	1	クラス1	1	1	クラス1
1	1	1	1	クラス1	1	1	クラス1
1	1	1	0	クラス2	1	0	クラス2
1	0	1	0	クラス2	1	0	クラス2
1	1	0	1	クラス2	0	1	クラス2
1	0	0	1	クラス2	0	1	クラス2
1	1	0	1	クラス2	0	1	クラス2

統合方法

- 多数決による統合方法

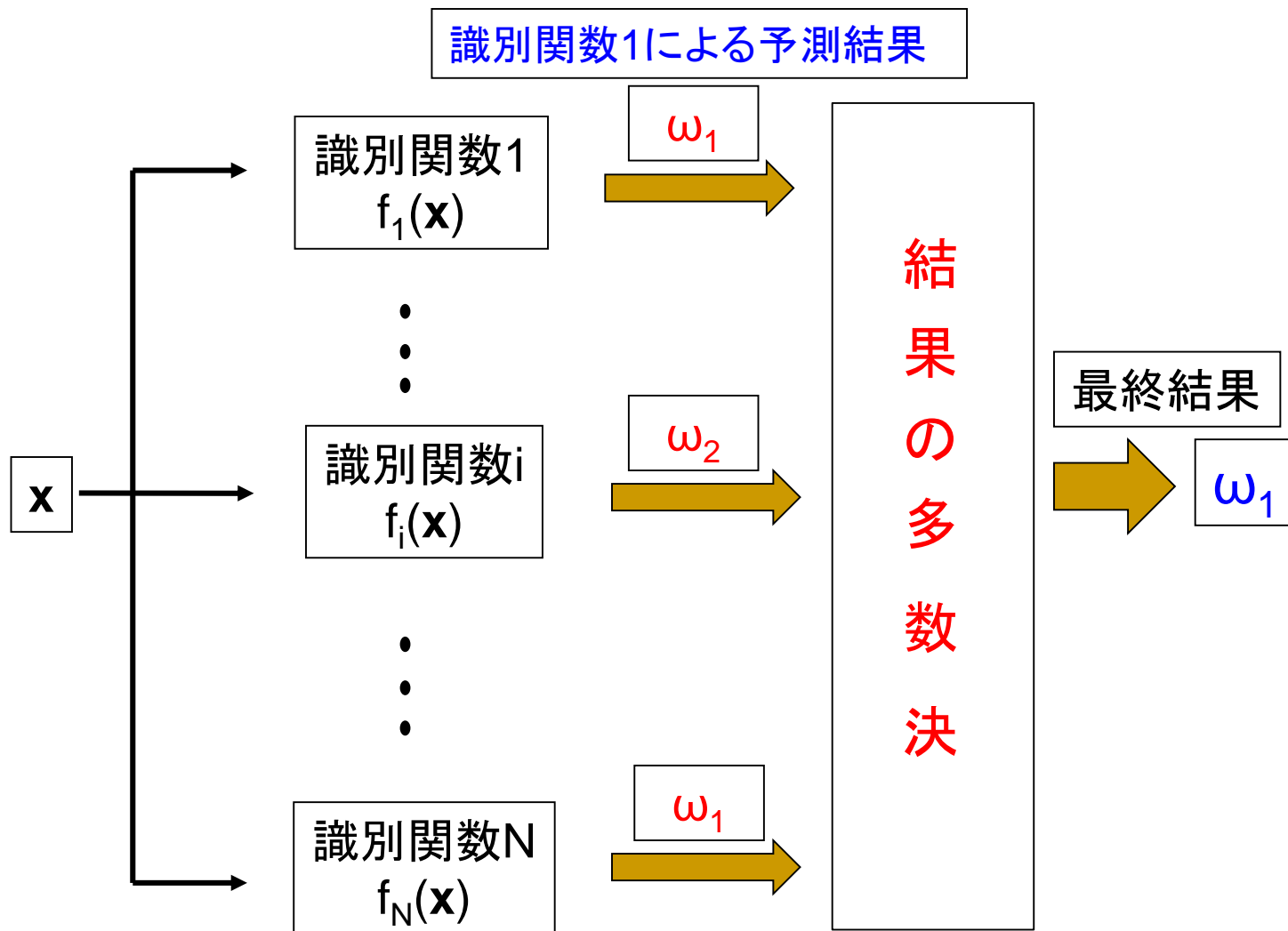
- 平均値による統合方法

$$F(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^N f_i(\mathbf{x})$$

- 重みづけによる統合方法

$$F(\mathbf{x}) = \sum_{i=1}^N \alpha_i f_i(\mathbf{x})$$

多数決による統合方法①



多数決による統合方法②

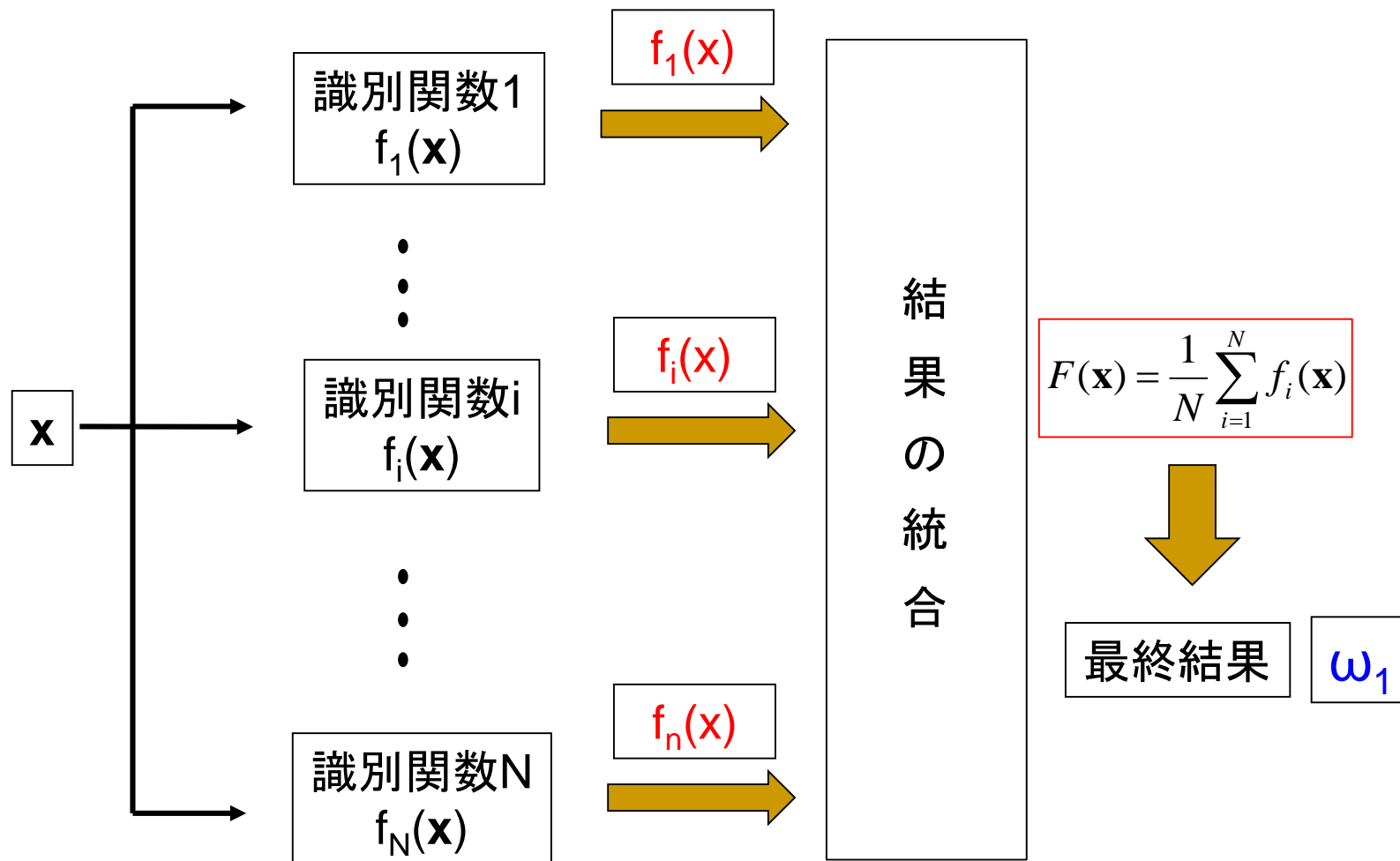
三個の識別関数による予測結果
を用いて多数決を行なう

識別関数の出力値

1→クラス1
0→クラス2

識別関数1	識別関数2	識別関数3	最終結果
1	1	1	クラス1
1	1	0	クラス1
1	0	1	クラス1
0	1	1	クラス1
0	1	0	クラス2
1	0	0	クラス2
0	0	1	クラス2
0	0	0	クラス2

平均値による統合方法①*



* $f_i(\mathbf{x})$ は確率(尤度)を考えた場合, $\{0,1\}$ の離散値でない場合もあります

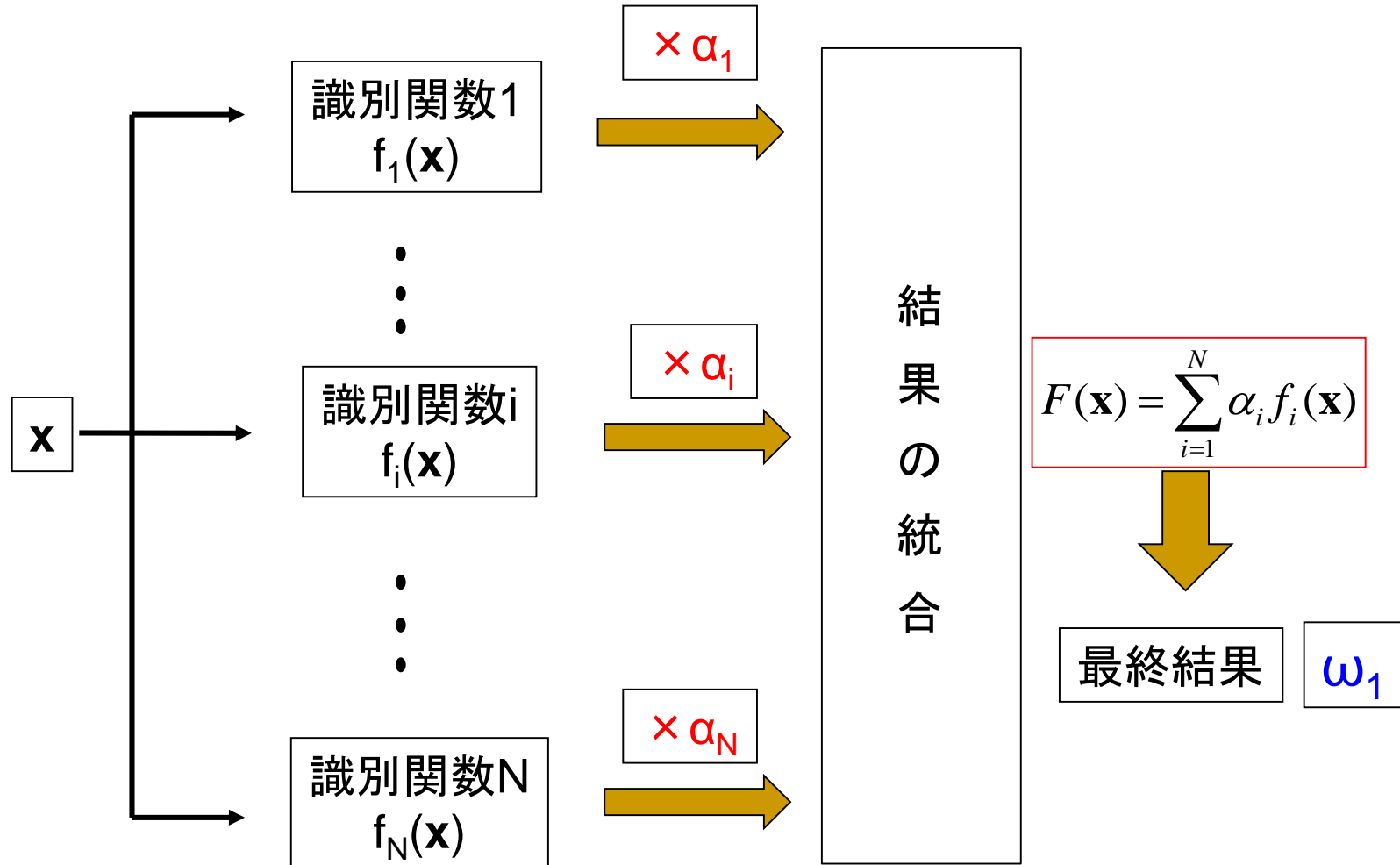
平均値による統合方法②

識別関数の出力値

$$F(\mathbf{x}) = \begin{cases} \mathbf{x} \in \omega_1 & \text{if } \mathbf{x} > 0.5 \\ \mathbf{x} \in \omega_2 & \text{if } \mathbf{x} < 0.5 \end{cases}$$

識別関数1	識別関数2	識別関数3	平均	最終結果
0.6	0.7	0.8	0.7	クラス1
0.8	0.7	0.2	0.566	クラス1
0.7	0.3	0.9	0.633	クラス1
0.2	1.0	0.6	0.6	クラス1
0.1	0.8	0.2	0.366	クラス2
0.9	0.3	0.4	0.533	クラス1
0.2	0.1	0.7	0.333	クラス2
0.1	0.2	0.2	0.166	クラス2

重みづけによる統合方法



アンサンブル学習の種類①

■ 方法①

バギング

- 多数の識別関数を独立に学習
- 学習した識別関数全てを用いて予測し、多数決などにより結果を判定

■ 方法②

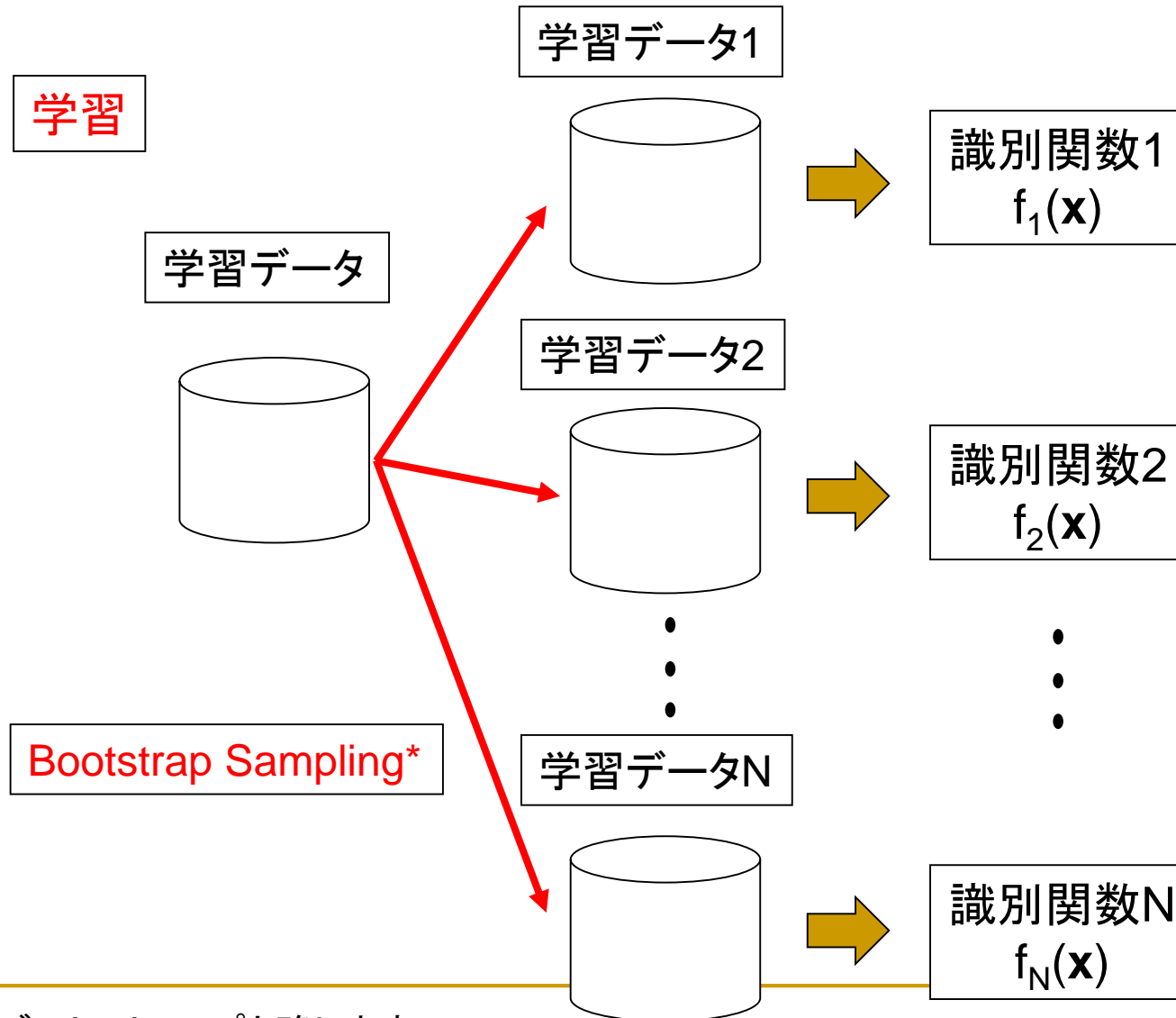
ブースティング

- 識別関数を学習
- その識別関数で誤分類したデータを正しく分類できるように、新しい識別関数を学習
- これを何度も繰り返す

アンサンブル学習の種類②

- バギング (Bagging)
 - 学習データを複数組作成 (Bootstrap Sampling)
 - 作成した学習データごとに識別関数 (弱分類器) を学習
 - 識別関数ごとにテストデータを予測, 結果を統合
- ブースティング (Boosting)
 - 学習データを用いて識別関数を学習
 - その識別関数によって誤分類したデータを対象に, 正しく分類できる識別関数を別途学習
 - 複数回繰り返した後, 重み付けにより結果を統合

バギング①



*ブートストラップと略します

バギング②

予測

未知データ

x

識別関数1
 $f_1(x)$

⋮

識別関数*i*
 $f_i(x)$

⋮

識別関数*N*
 $f_N(x)$

ω_1

ω_2

ω_1

識別関数1による
予測結果

結果
の
統
合

最終結果

ω_1

ブースティング①

□ はデータに対する重み (誤認識の場合は大きい)

正しく学習できなかった場合、次の識別関数で訂正

学習データ

x_1

x_2

x_3

⋮

x_P

識別関数1
 $f_1(x)$
 α_1

x_1

x_2

x_3

⋮

x_P

識別関数2
 $f_2(x)$
 α_2

x_1

x_2

x_3

⋮

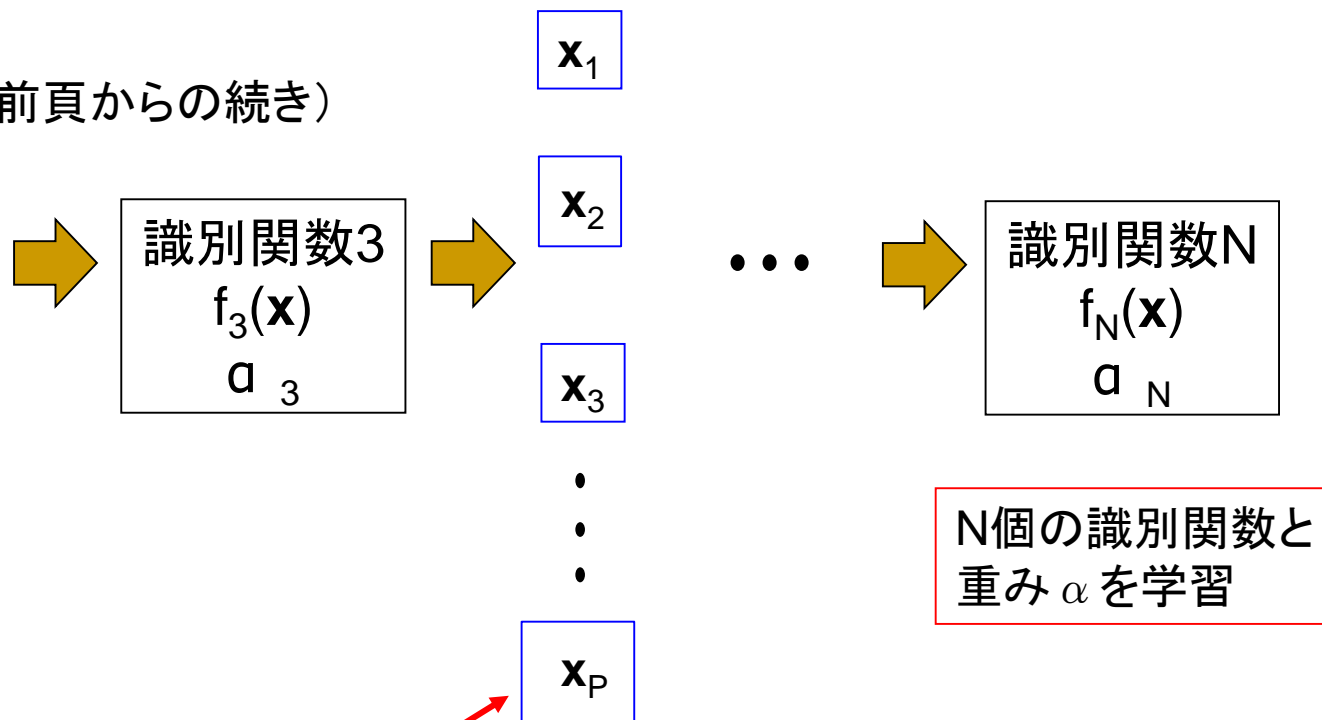
x_P

x_1 を識別関数1では正しく学習できなかった
→ x_1 に対する重みを大きくし、次の識別関数の学習の際、学習できるようにする

x_2 を識別関数2では正しく学習できなかった
→ x_2 に対する重みを大きくし、次の識別関数の学習の際、学習できるようにする

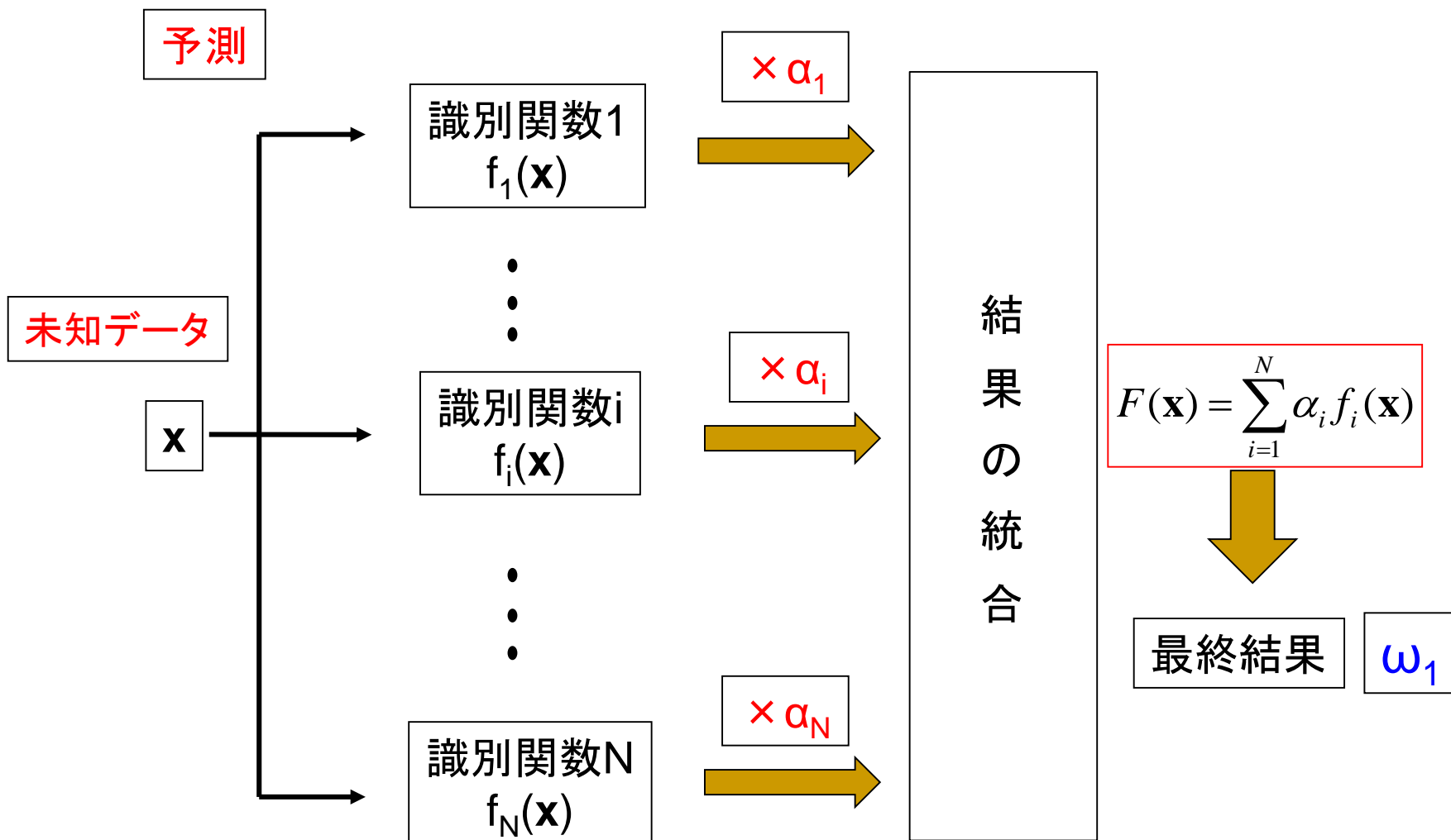
ブースティング②

(前頁からの続き)



x_P を識別関数3では正しく学習できなかった
→ x_P に対する重みを大きくし, 次の識別関数の学習の際, 学習できるようにする

ブースティング③



バギング

ランダムフォレスト

バギング

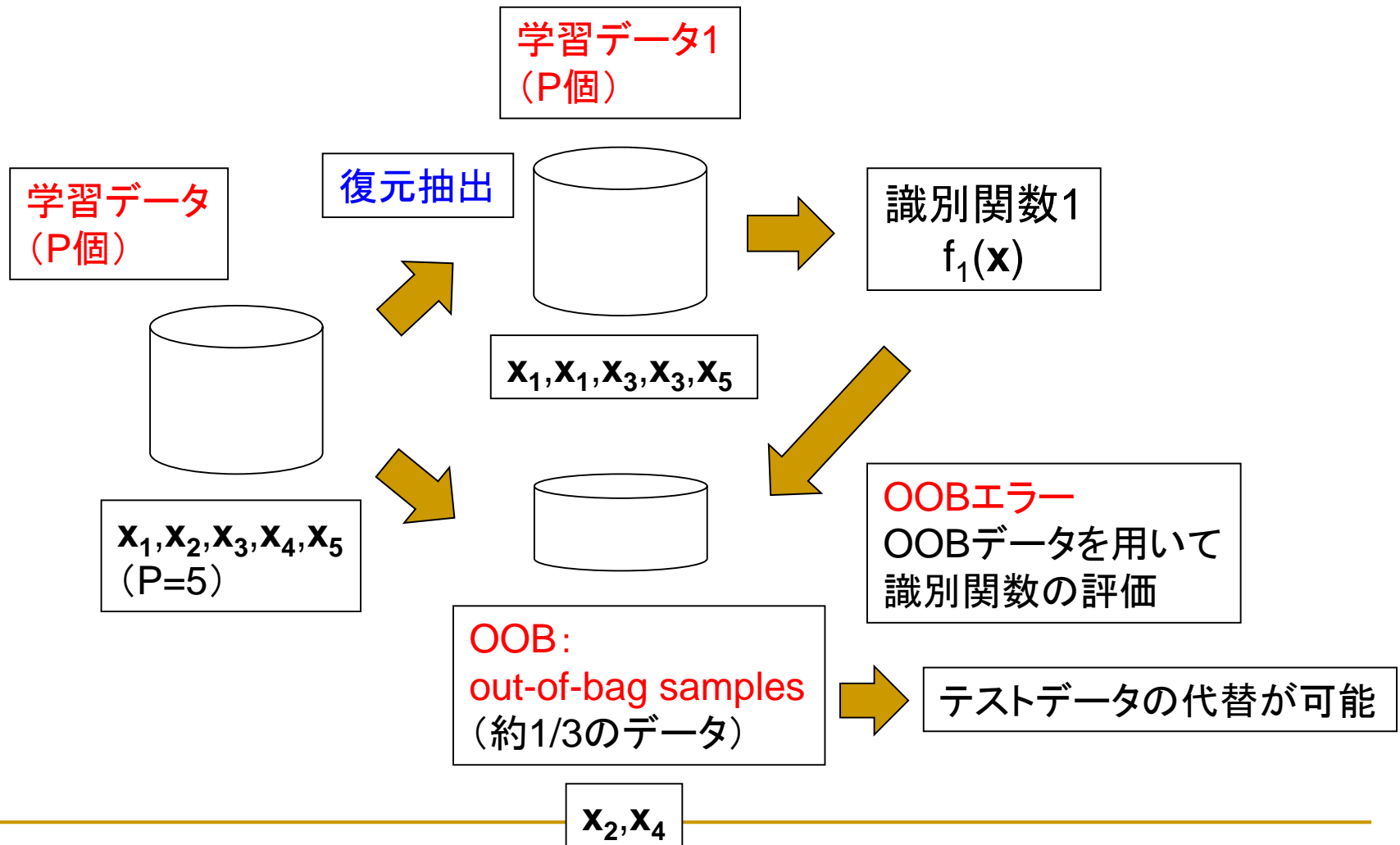
Bagging (Bootstrap Aggregating)

- ブートストラップ (Bootstrap Sampling)
 - P個の学習データから、重複を許し、P個のデータ*をサンプリング (復元抽出)
- out-of-bag samples (OOB)
 - ブートストラップにより選ばれなかったデータ
 - サンプリング後の任意のデータが元の学習データに含まれない割合は、 $((P-1)/P)^P$
 - $P \rightarrow \infty$ の場合、その割合は約0.368

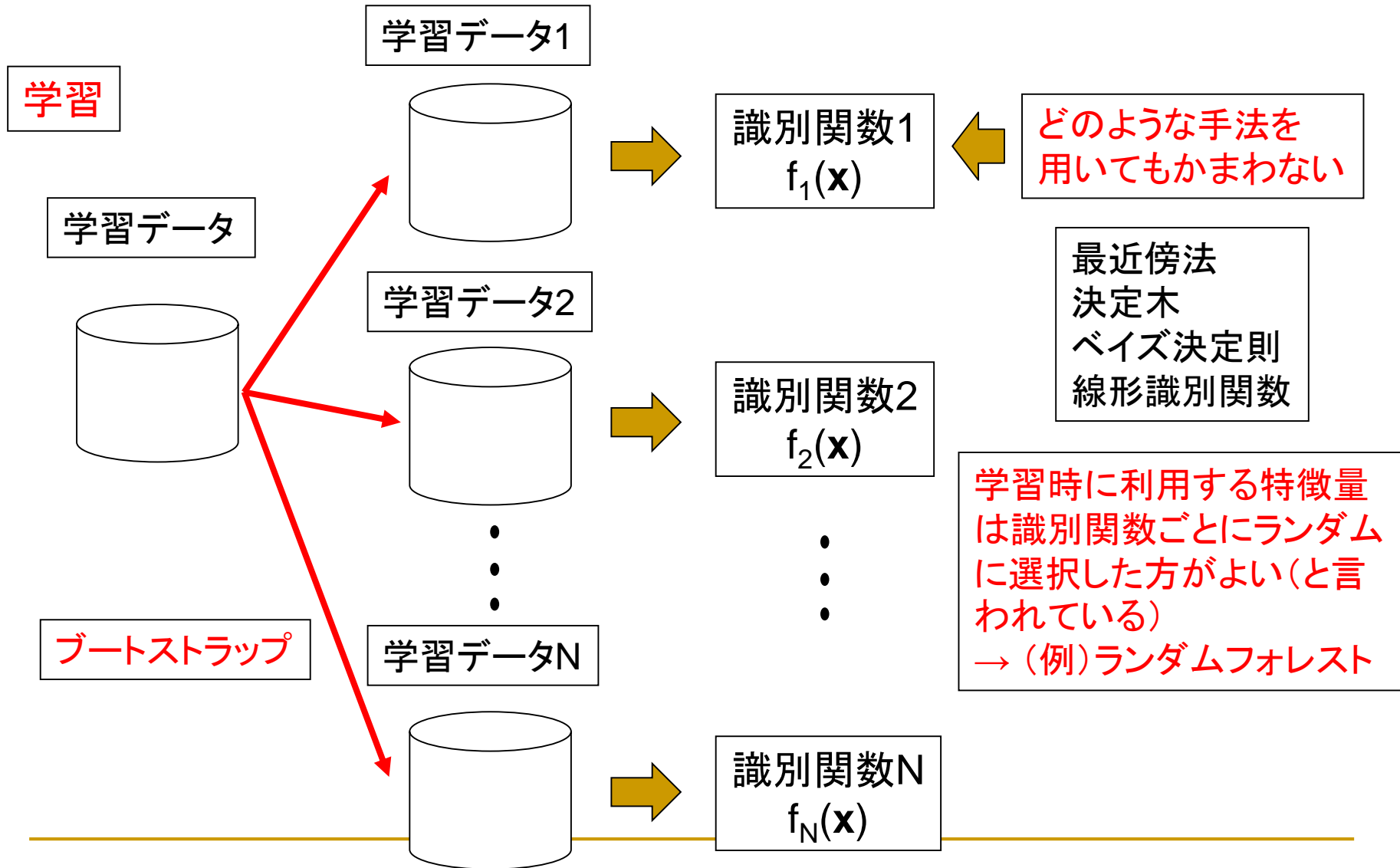
復元抽出の場合、1/3のデータが学習に利用されない

*P個のデータをサンプリングしない場合もあります

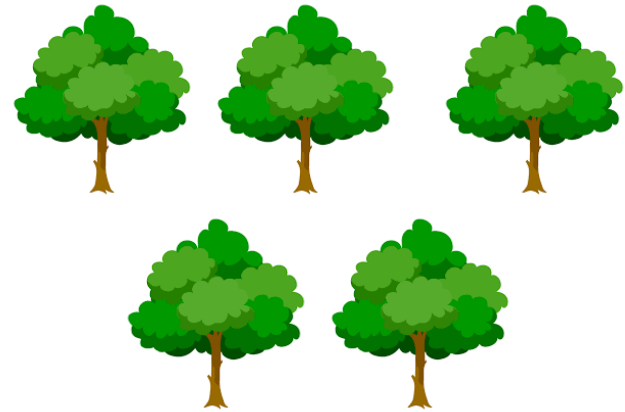
ブートストラップ



バギング



ランダムフォレスト①

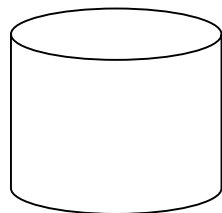


- 識別関数(弱分類器)
 - 決定木
- N回ブートストラップを行ない, N個の学習データを作成
→ 学習データごとにN個の決定木を学習
- 分類問題の場合
 - N個の決定木の結果を多数決
- 回帰問題の場合
 - N個の回帰木の結果の平均値

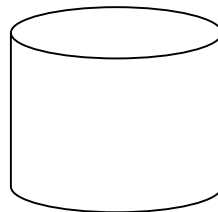
ランダムフォレスト②

学習

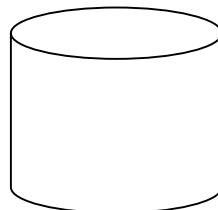
学習データ



学習データ1

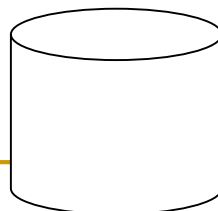


学習データ2



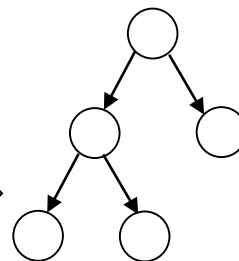
⋮

学習データN

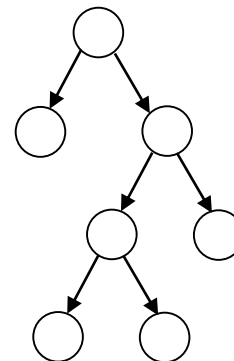


ブートストラップ

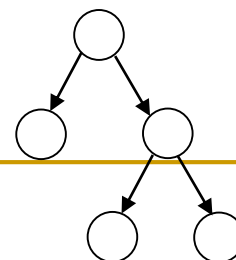
決定木1



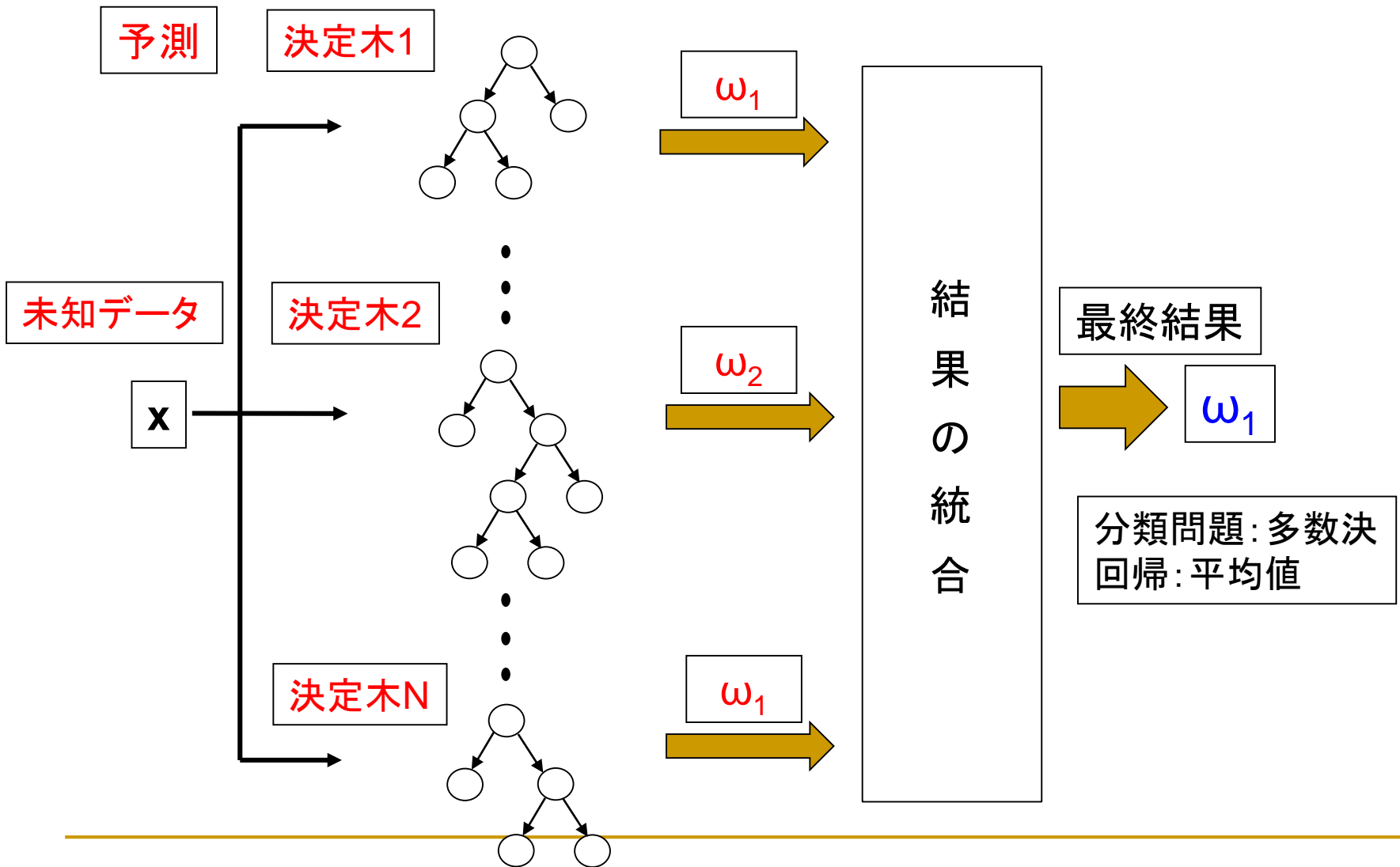
決定木2



決定木N



ランダムフォレスト③



学習アルゴリズム

for i in range(N):

$S_i \leftarrow \text{ブートストラップ}(S)$

決定木 T_i の学習(S_i)

S: 学習データ

N: 決定木の個数

決定木の学習(S_i):

if 停止条件を満たしている:

停止

通常の決定木の学習アルゴリズムとは異なる点

D個の特徴量中, d個をランダムに選ぶ

ゲインが最大となる特徴量

Sを分類するため, d個の中から最適な特徴量を選択

$S_{iL}, S_{iR} \leftarrow \text{分類結果}$

決定木の学習(S_{iL})

S_{iL} : 左ノードで対象となるデータ

決定木の学習(S_{iR})

S_{iR} : 右ノードで対象となるデータ

特徴の選択方法(復習)

- エントロピー(もしくはジニ係数)が最も小さくなる特徴を選択
- ゲイン

$$Gain(D, a) = E(D) - \sum_{c \in Value(a)} \frac{|D_c|}{|D|} E(D_c)$$

現在のエントロピー

分類後のエントロピー

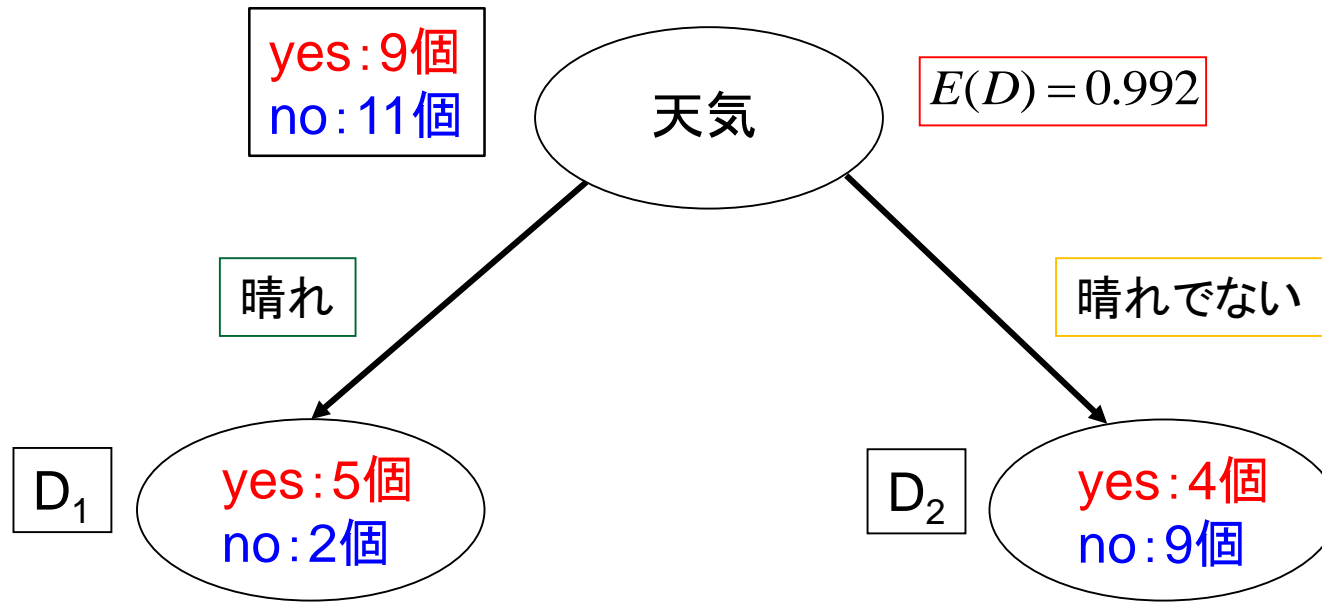
a: 特徴量

Value(a): 特徴量aの値

D_c : 特徴量aの値を持つデータの集合

- ゲインが最大となる特徴を選択

天気で分類した場合（復習）



晴れ

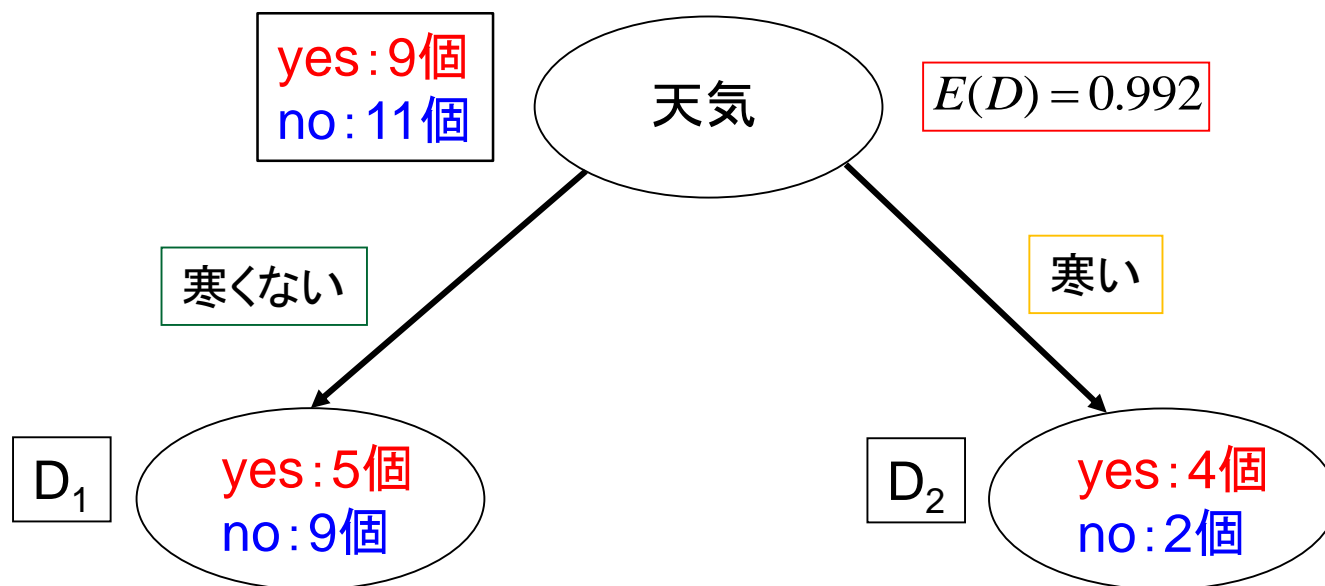
晴れでない

$$E(D_1) = -\frac{5}{7} \log \frac{5}{7} - \frac{2}{7} \log \frac{2}{7} = 0.863$$

$$E(D_2) = -\frac{4}{13} \log \frac{4}{13} - \frac{9}{13} \log \frac{9}{13} = 0.890$$

$$\begin{aligned} \text{Gain}(D) &= E(D) - \frac{7}{20} E(D_1) - \frac{13}{20} E(D_2) \\ &= 0.992 - \frac{7}{20} \times 0.863 - \frac{13}{20} \times 0.890 = 0.111 \end{aligned}$$

気温で分類した場合（復習）

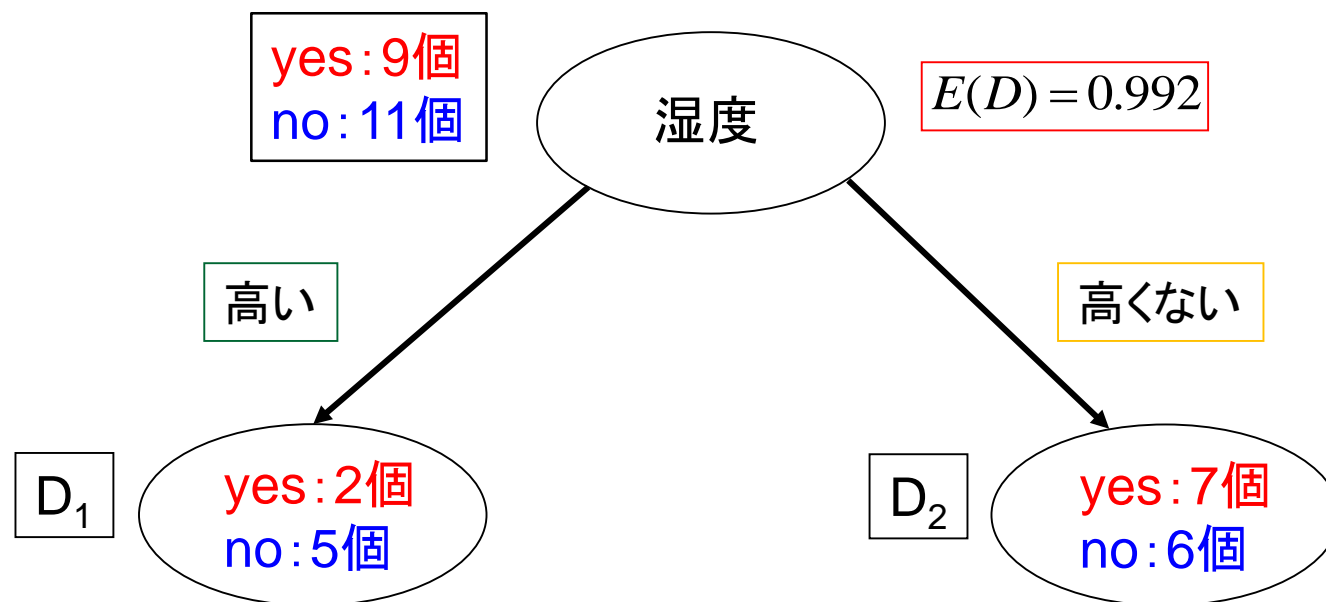


$$E(D_1) = -\frac{5}{14} \log \frac{5}{14} - \frac{9}{14} \log \frac{9}{14} = 0.940$$

$$E(D_2) = -\frac{4}{6} \log \frac{4}{6} - \frac{2}{6} \log \frac{2}{6} = 0.918$$

$$\begin{aligned} \text{Gain}(D) &= E(D) - \frac{14}{20} E(D_1) - \frac{6}{20} E(D_2) \\ &= 0.992 - \frac{14}{20} \times 0.979 - \frac{6}{20} \times 0.811 = 0.059 \end{aligned}$$

湿度で分類した場合（復習）



高い

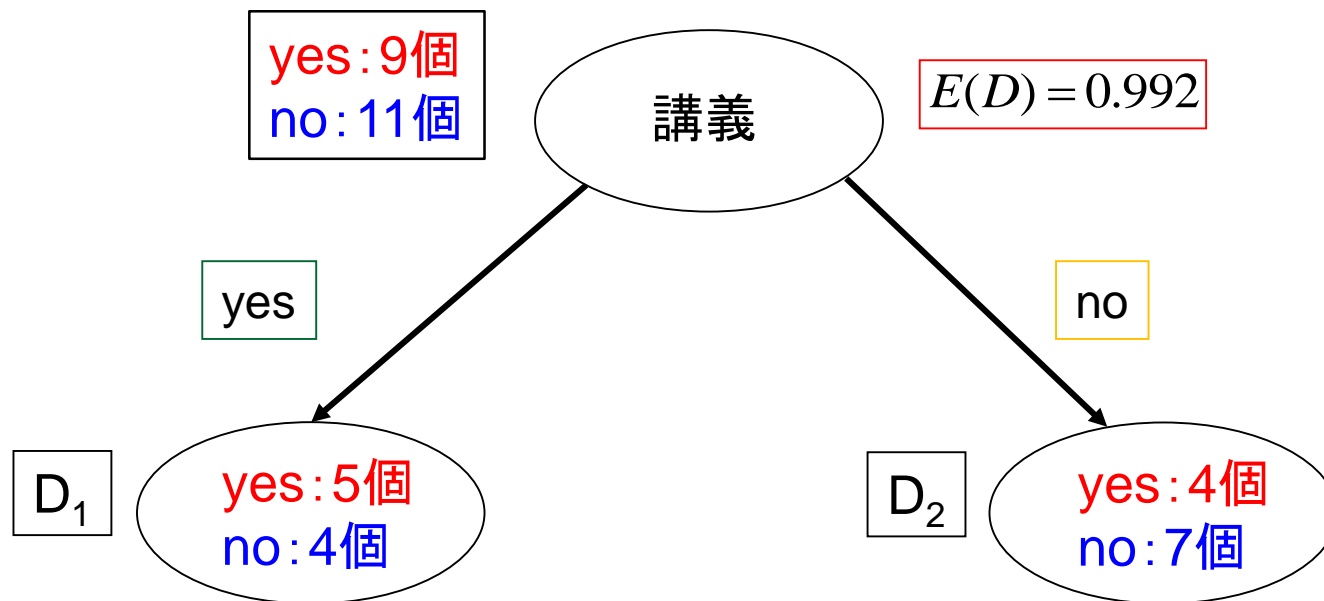
$$E(D_1) = -\frac{2}{7} \log \frac{2}{7} - \frac{5}{7} \log \frac{5}{7} = 0.863$$

高くない

$$E(D_2) = -\frac{7}{13} \log \frac{7}{13} - \frac{6}{13} \log \frac{6}{13} = 0.995$$

$$\begin{aligned} \text{Gain}(D) &= E(D) - \frac{7}{20} E(D_1) - \frac{13}{20} E(D_2) \\ &= 0.992 - \frac{7}{20} \times 0.863 - \frac{13}{20} \times 0.995 = 0.043 \end{aligned}$$

講義で分類した場合（復習）



講義=yes

講義=no

$$E(D_1) = -\frac{5}{9} \log \frac{5}{9} - \frac{4}{9} \log \frac{4}{9} = 0.991$$

$$E(D_2) = -\frac{4}{11} \log \frac{4}{11} - \frac{7}{11} \log \frac{7}{11} = 0.945$$

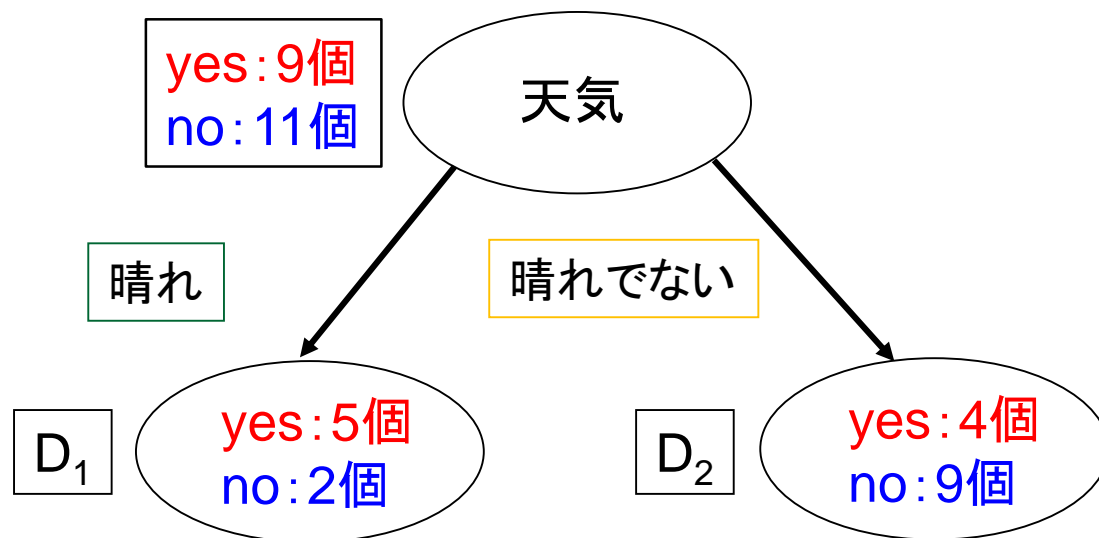
$$\begin{aligned} \text{Gain}(D) &= E(D) - \frac{9}{20} E(D_1) - \frac{11}{20} E(D_2) \\ &= 0.992 - \frac{9}{20} \times 0.991 - \frac{11}{20} \times 0.945 = 0.026 \end{aligned}$$

分類するための特徴選択(復習)

- 天気で分類した場合 → ゲイン = 0.111 最大
- 気温で分類した場合 → ゲイン = 0.059
- 湿度で分類した場合 → ゲイン = 0.043
- 講義で分類した場合 → ゲイン = 0.026



- 天気で分類



特徴選択の改良①

■ 特徴量

- 同一の特徴量を用いた場合, 似た構造の決定木となる
- 特徴選択の際, D 個の特徴量中, ランダムに d 個の特徴量を選ぶ
- 選ばれた d 個の特徴量のゲインを計算

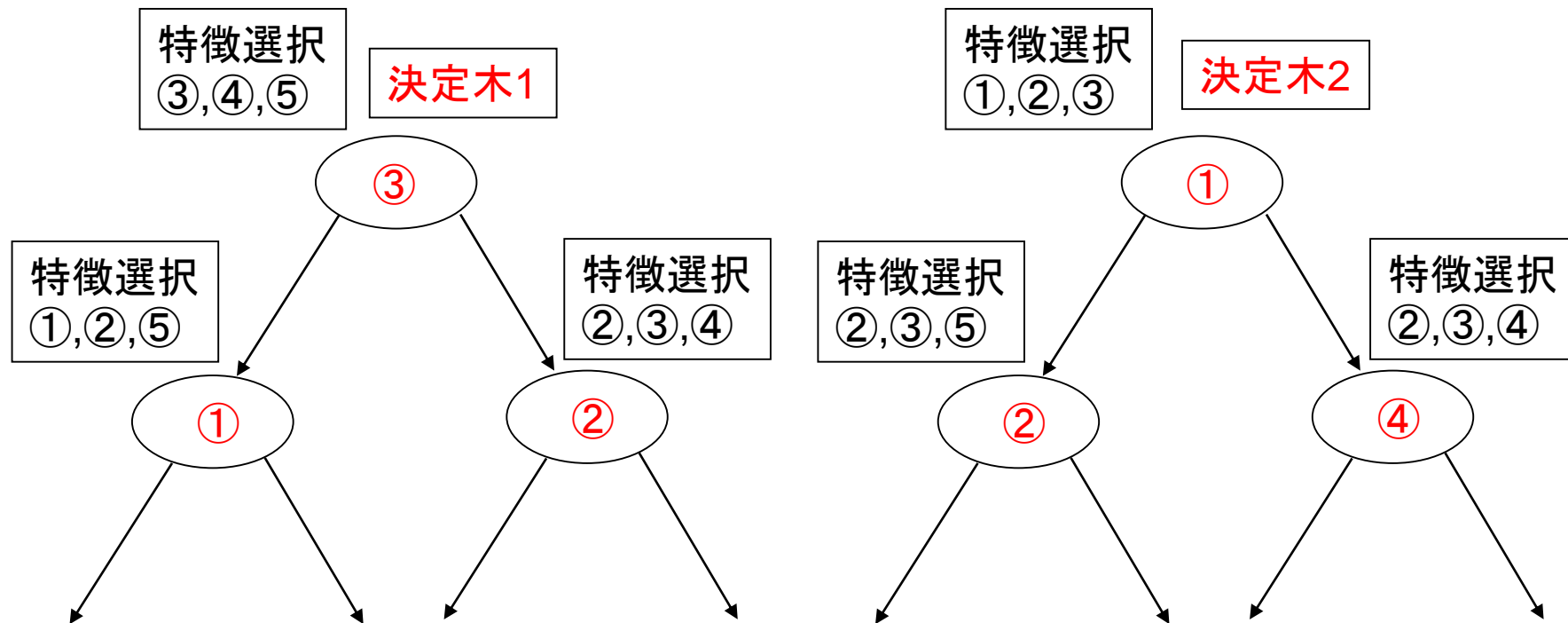
$$d = \sqrt{D} \text{ もしくは } d = \log_2 D$$

- ゲインが最大の特徴量を用いて分類

特徴選択の改良②

■ 特徴量

□ $D=5$ (特徴量①,②,③,④,⑤), $d=3$



特徴の重要度

- N : 決定木の個数
- A_i : OOBデータを予測対象とした場合, 決定木 i による正解率
- A_{it} : OOBデータ中, 特徴量 t のみをランダムに並び替える
→ 並び替えたデータを決定木 i によって予測した場合の正解率
- I_t : 特徴量 t の重要度

$$I_t = \frac{1}{N} \sum_{i=1}^N (A_i - A_{it})$$

I_t が0に近い場合

- 特徴量 t を入れ替えても影響はない
- 特徴量 t は重要ではない

I_t の値が大きい

- 特徴量 t を入れ替えると影響が生じる
- 特徴量 t は重要

バギングによるアンサンブル学習

バギング
ランダムフォレスト

バギング (Cancer_Bagging.py)

- クラス分類
- データセット
 - breast cancer
- 利用する弱分類器
 - ロジスティック回帰
 - k近傍法
 - 決定木

用途	クラス分類
データ数	569
特徴量	30
目的変数	2
正例	212
負例	357

プログラムはscikit-learnのバージョンが0.19.2用です. OOBの処理がバージョンによって異なります.

- 弱分類器の個数: 10
- 弱分類器ごとで学習に使用する特徴量の割合: 0.5

```
import sys
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

識別関数(弱分類器)のimport

```
from sklearn.linear_model import LogisticRegression
```

ロジスティック回帰

```
from sklearn.neighbors import KNeighborsClassifier
```

k近傍法

```
from sklearn import tree
```

決定木

バギングのimport

```
from sklearn.ensemble import BaggingClassifier
```

バギングを行なうために必要

データのロード

```
cancer = datasets.load_breast_cancer()
```

特徴量(30次元)

```
feature_names=cancer.feature_names
```

```
data = cancer.data
```

データの大きさ
(569, 30)

```
# 目的変数( malignant, benign )
```

```
name = cancer.target_names
```

```
label = cancer.target
```

データの大きさ
569次元

ホールドアウト法

```
# 学習データ, テストデータ
```

```
train_data, test_data, train_label, test_label = train_test_split(data, label,  
test_size=0.5, random_state=None)
```

```
print( '\n [ 弱分類器の選択 ] ' )
```

```
print( ' ロジスティック回帰 -> 1 ' )
```

```
print( ' k近傍法 -> 2 ' )
```

```
print( ' 決定木 -> 3 ' )
```

```
ans = int( input( ' > ' ) )
```

弱分類器の選択

```
# ロジスティック回帰
```

```
if ans == 1:
```

```
    wc = LogisticRegression()
```

弱分類器①
ロジスティック回帰

k近傍法

```
elif ans == 2:
```

```
    wc = KNeighborsClassifier()
```

弱分類器②
k近傍法

決定木(ランダムフォレスト?)

```
elif ans == 3:
```

```
    wc = tree.DecisionTreeClassifier(max_depth=3)
```

弱分類器③
決定木

バギング

```
model = BaggingClassifier(base_estimator=wc, bootstrap=True,  
n_estimators=10, max_samples=1.0, max_features=0.5, oob_score=True)
```

base_estimator
弱分類器を指定

bootstrap
デフォルトはTrue

n_estimators
弱分類器の個数

max_samples
bootstrapする
データの割合

max_features
利用する特徴量の割合
($30 \times 0.5 = 15$ 次元)

oob_score
OOBエラーを求める場合
→ True (デフォルトはFalse)

学習

```
model.fit(train_data, train_label)
```

*回帰用のメソッドは、BaggingRegressorです。参考文献②で調べてみてください

```
print( "¥n [ ブートストラップ ]" )  
print( " [ 0番目の弱分類器でサンプリングしたデータ ]" )  
print( model.estimators_samples_[0] )
```

estimators_samples_
True: bootstrapで選ばれたデータ
False: 選ばれなかったデータ

```
print( "¥n [ OOBデータの割合 ]" )
```

estimators_samples_[0]
0番目の弱分類器で選ばれたデータ*

```
for i in range( model.n_estimators ):  
    print( i , ":" , 1 - np.count_nonzero( model.estimators_samples_[i] ) /  
        len( model.estimators_samples_[i] ) )
```

Trueの個数

OOBデータの割合の計算

使用した特徴

```
print( "¥n [ 選択された特徴量 ]" )
```

```
for i in range( model.n_estimators ):  
    print( i , ":" , model.estimators_features_[i] )
```

estimators_features_
弱識別器で使用了た特徴量

予測

```
predict = model.predict(test_data)
```

*scikit-learnのバージョンで動作が異なります(資料は0.19.2で作成しました)

```
print( "¥n [ OOB score ]" )
```

```
print( model.oob_score_ )
```

OOBデータに対する正解率

```
print( "¥n [ 予測結果 ]" )
```

```
print( classification_report(test_label, predict) )
```

結果の表示

```
print( "¥n [ 正解率 ]" )
```

```
print( accuracy_score(test_label, predict) )
```

```
print( "¥n [ 混同行列 ]" )
```

```
print( confusion_matrix(test_label, predict) )
```

BaggingClassifier

bootstrapを行なう場合はTrue
(デフォルトはTrue)

```
from sklearn.ensemble import BaggingClassifier
```

```
BaggingClassifier(base_estimator=弱分類器, bootstrap=True,  
n_estimators=弱分類器の個数, max_samples=ブートストラップに用いる割合,  
max_features=用いる特徴の割合, oob_score=True)
```

OOBエラーを求める場合はTrue
(デフォルトはFalse)

```
# 弱分類器にロジスティック回帰
```

```
wc = LogisticRegression()
```

```
model = BaggingClassifier(base_estimator=wc, bootstrap=True,  
n_estimators=10, max_samples=1.0, max_features=0.5, oob_score=True)
```

実行結果①

[illegible]

弱分類器

ロジスティック回帰を選択

ブートストラップ

学習データ: 285個
True: 利用したデータ
False: 利用しなかったデータ (OOB)

scikit-learnのバージョンで動作が異なります
資料は0.19.2で作成しました

実行結果(OOBについての注意)

```
コマンド プロンプト

[ ブートストラップ ]
[ 0番目の弱分類器でサンプリングしたデータ ]
[ 91 217 125 83 254 234 192 236 241 124 71 147 207 203 173 146 151 32
57 156 228 126 104 276 259 25 111 8 100 72 74 181 11 233 223 47
153 228 196 157 276 41 245 105 191 38 140 3 224 275 187 256 115 195
181 4 271 263 129 195 100 230 4 73 229 230 226 68 131 33 2 193
208 209 173 186 125 61 141 69 8 159 5 184 231 121 14 271 171 172
98 141 90 163 145 165 95 32 283 92 117 57 209 0 41 106 280 83
225 60 157 186 224 104 95 71 113 161 163 8 30 104 248 155 34 14
51 86 159 80 273 250 220 121 197 122 157 91 267 160 129 152 138 122
100 237 144 1 78 24 102 135 35 165 86 198 238 241 154 79 245 47
281 223 253 84 278 196 20 198 15 59 194 154 166 54 191 194 116 212
182 160 115 166 158 143 52 142 233 9 282 199 49 242 181 280 195 278
251 103 2 169 57 123 214 65 11 225 38 15 139 5 116 256 185 19
238 61 226 249 248 190 1 259 236 85 45 21 199 173 127 65 0 168
3 146 228 210 259 30 258 265 120 171 8 62 111 67 58 70 52 5
0 199 183 119 205 258 131 68 169 0 141 196 162 195 187 125 147 84
247 144 119 208 21 16 78 272 15 162 248 126 127 166]

[ OOBデータの割合 ]
0 : 0.014084507042253502
1 : 0.0035211267605633756
2 : 0.0
3 : 0.0035211267605633756
4 : 0.010563380281690127
5 : 0.0035211267605633756
6 : 0.0
7 : 0.0
8 : 0.0
```

ブートストラップ

学習データ:285個
最近のscikit-learnのバージョンの場合
選択されたデータの番号が表示されます

実行結果②

```
選択 C:\Windows\system32\cmd.exe

[ OOBデータの割合 ]
0 : 0.397887323943662
1 : 0.34507042253521125
2 : 0.37676056338028174
3 : 0.3556338028169014
4 : 0.34507042253521125
5 : 0.35915492957746475
6 : 0.3274647887323944
7 : 0.37676056338028174
8 : 0.34507042253521125
9 : 0.34507042253521125

[ 選択された特徴量 ]
0 : [ 23 9 25 19 6 1 4 0 5 22 7 24 27 26 13 ]
1 : [ 1 10 12 25 20 16 29 11 2 26 28 8 22 5 13 ]
2 : [ 22 23 6 14 25 1 8 17 28 29 9 5 2 20 24 ]
3 : [ 20 3 29 25 11 27 0 28 10 7 6 18 4 23 8 ]
4 : [ 23 11 18 7 24 19 21 28 6 26 9 4 5 17 10 ]
5 : [ 25 1 10 28 15 14 8 29 5 12 22 26 11 2 13 ]
6 : [ 7 14 12 27 18 13 20 23 5 11 22 19 21 17 0 ]
7 : [ 22 1 19 8 15 6 5 29 28 24 9 17 23 0 16 ]
8 : [ 23 2 9 26 10 22 21 4 13 16 27 28 14 17 8 ]
9 : [ 4 13 22 3 21 28 27 14 9 18 26 20 6 7 16 ]

[ OOB score ]
0.9401408450704225
```

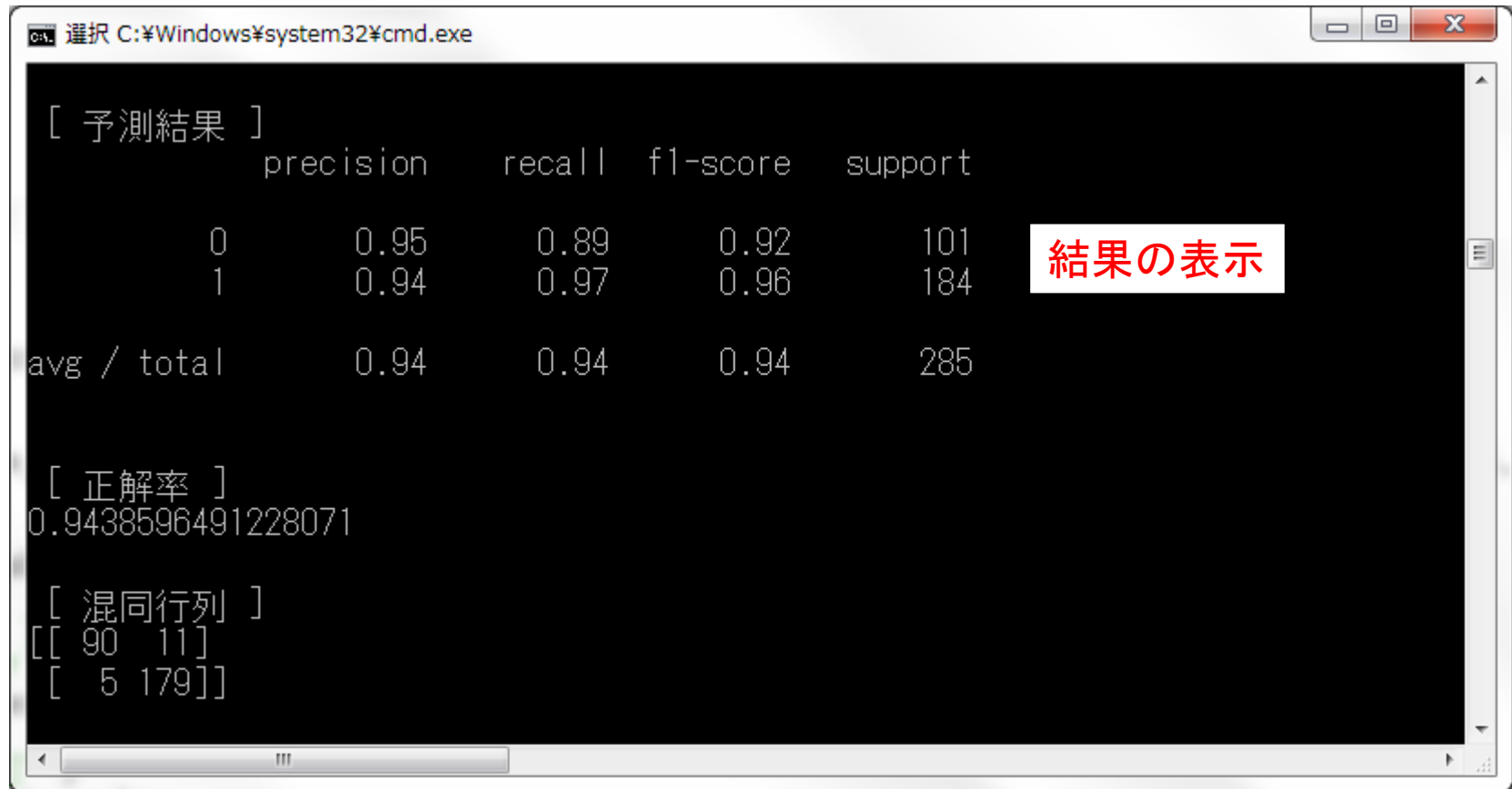
OOBデータの割合
→ 36%のデータが選択されない

特徴量: 30個

弱分類器ごとに選択
された特徴量(15個)

OOBデータに対する正解率

実行結果③



```
選択 C:\Windows\system32\cmd.exe

[ 予測結果 ]
      precision    recall  f1-score   support

     0       0.95      0.89      0.92       101
     1       0.94      0.97      0.96       184

 avg / total       0.94      0.94      0.94       285

[ 正解率 ]
0.9438596491228071

[ 混同行列 ]
[[ 90  11]
 [  5 179]]
```

結果の表示

ランダムフォレストのプログラム

- 分類木
 - breast cancerデータセット
 - Cancer_RF.py

用途	クラス分類
データ数	569
特徴量	30
目的変数	2
正例	212
負例	357

- 回帰木
 - Bostonデータセット
 - Boston_RF.py

用途	回帰
データ数	506
特徴量	13
目的変数	1

分類木の場合 (Cancer_RF.py)

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

パッケージのimport

Cancer_RF.py

ランダムフォレスト(クラス分類)を行なうために必要

データのロード

```
cancer = datasets.load_breast_cancer()
```

特徴量

```
feature_names=cancer.feature_names
```

```
data = cancer.data
```

データの大きさ
(569, 30)

目的変数(malignant, benign)

```
name = cancer.target_names
```

```
label = cancer.target
```

データの大きさ
569次元

学習データ, テストデータ

ホールドアウト法

```
train_data, test_data, train_label, test_label = train_test_split(data, label,  
test_size=0.5, random_state=None)
```

n_estimators
決定木の個数

criterion
分類指標 (デフォルトはジニ係数)
エントロピーの場合, entropy

```
model = RandomForestClassifier(n_estimators=10, criterion="gini",  
max_features="sqrt", bootstrap=True, oob_score=True)
```

max_features
使用される特徴数
sqrt → 全特徴数の平方根
log2 → \log_2 全特徴数

bootstrap
デフォルトはTrue

oob_score
OOBエラーを求める場合
→ True (デフォルトはFalse)

学習

```
model.fit(train_data, train_label)
```

予測

```
predict = model.predict(test_data)
```

```
print( "¥n [ OOB score ] " )
```

OOBデータに対する正解率

```
print( model.oob_score_ )
```

```
print( "¥n [ 特徴の重要度 ]" )  
for i in range(len(feature_names)):  
    print( " {0:25s} : {1:7.5f}".format( feature_names[i] ,  
        model.feature_importances_[i] ) )
```

feature_importances_
特徴の重要度

```
print( "¥n [ 予測結果 ]" )  
print( classification_report(test_label, predict) )
```

```
print( "¥n [ 正解率 ]" )  
print( accuracy_score(test_label, predict) )
```

結果の表示

```
print( "¥n [ 混同行列 ]" )  
print( confusion_matrix(test_label, predict) )
```

RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
RandomForestClassifier(n_estimators=決定木の個数,  
criterion=分類指標, max_features=使用される特徴数,  
bootstrap=True, oob_score=True)
```

bootstrapを行なう場合はTrue
(デフォルトはTrue)

OOBエラーを求める場合はTrue
(デフォルトはFalse)

criterion
分類指標(デフォルトはジニ係数)
エントロピーの場合, entropy

```
model = RandomForestClassifier(n_estimators=10, criterion="gini",  
max_features="sqrt", bootstrap=True, oob_score=True)
```

max_features
使用される特徴数
sqrt → 全特徴数の平方根
log2 → \log_2 全特徴数

実行結果①



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays a list of feature importance values. A red rectangular box is overlaid on the right side of the list, containing the text "特徴の重要度" in red. The list of features and their values is as follows:

[特徴の重要度]	
mean radius	: 0.00976
mean texture	: 0.01209
mean perimeter	: 0.07262
mean area	: 0.06496
mean smoothness	: 0.00422
mean compactness	: 0.00000
mean concavity	: 0.04416
mean concave points	: 0.01760
mean symmetry	: 0.00000
mean fractal dimension	: 0.00654
radius error	: 0.00430
texture error	: 0.00490
perimeter error	: 0.00426
area error	: 0.07903
smoothness error	: 0.00629
compactness error	: 0.00976
concavity error	: 0.00932
concave points error	: 0.00000
symmetry error	: 0.00117
fractal dimension error	: 0.00888
worst radius	: 0.08106
worst texture	: 0.01501
worst perimeter	: 0.15702
worst area	: 0.10663
worst smoothness	: 0.01089
worst compactness	: 0.01090
worst concavity	: 0.02569
worst concave points	: 0.18794
worst symmetry	: 0.01940
worst fractal dimension	: 0.02560

実行結果②

```
C:\Windows\system32\cmd.exe

[ OOB score ]
0.9330985915492958

[ 予測結果 ]
precision    recall  f1-score   support

     0       0.94      0.94      0.94       109
     1       0.97      0.96      0.96       176

avg / total       0.95      0.95      0.95       285

[ 正解率 ]
0.9543859649122807

[ 混同行列 ]
[[103   6]
 [  7 169]]
```

OOBデータに対する正解率

結果の表示

回帰木の場合 (Boston_RF.py)

```
import numpy as np
```

```
from sklearn import datasets
```

パッケージのimport

Boston_RF.py

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report, accuracy_score,  
confusion_matrix
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
import matplotlib.pyplot as plt
```

ランダムフォレスト(回帰)を行なうために必要

```
# データのロード
```

```
boston = datasets.load_boston()
```

```
# 特徴量(13次元)
```

```
feature_names=boston.feature_names
```

```
data = boston.data
```

データの大きさ
(506, 13)

価格

```
price = boston.target
```

データの大きさ
506次元

学習データ, テストデータ

ホールドアウト法

```
train_data, test_data, train_price, test_price = train_test_split(data, price,  
test_size=0.5, random_state=None)
```

n_estimators
決定木の個数

criterion
分類指標(デフォルトはmse)

```
model = RandomForestRegressor(n_estimators=20, criterion="mse",  
max_features="sqrt", bootstrap=True, oob_score=True)
```

max_features
選択される特徴数
sqrt → 全特徴数の平方根
log2 → \log_2 全特徴数

bootstrap
デフォルトはTrue

oob_score
OOBエラーを求める場合
→ True(デフォルトはFalse)

学習

```
model.fit(train_data, train_price)
```

予測

```
predict = model.predict(test_data)
```

```
print( "¥n [ OOB score ]" )
```

OOBデータに対する R^2

```
print( model.oob_score_ )
```

```
print( "¥n [ 特徴の重要度 ]" )
```

```
for i in range(len(feature_names)):
```

```
    print( " {0:25s} : {1:7.5f}".format( feature_names[i] ,
```

```
model.feature_importances_[i] ) )
```

feature_importances_
特徴の重要度

R^2 を求める

```
train_score = model.score(train_data, train_price)
```

```
test_score = model.score(test_data, test_price)
```

R^2 を求める

```
print( "¥n [ R2 ]" )
```

```
print( " 学習データ : {0:7.5f}".format( train_score ) )
```

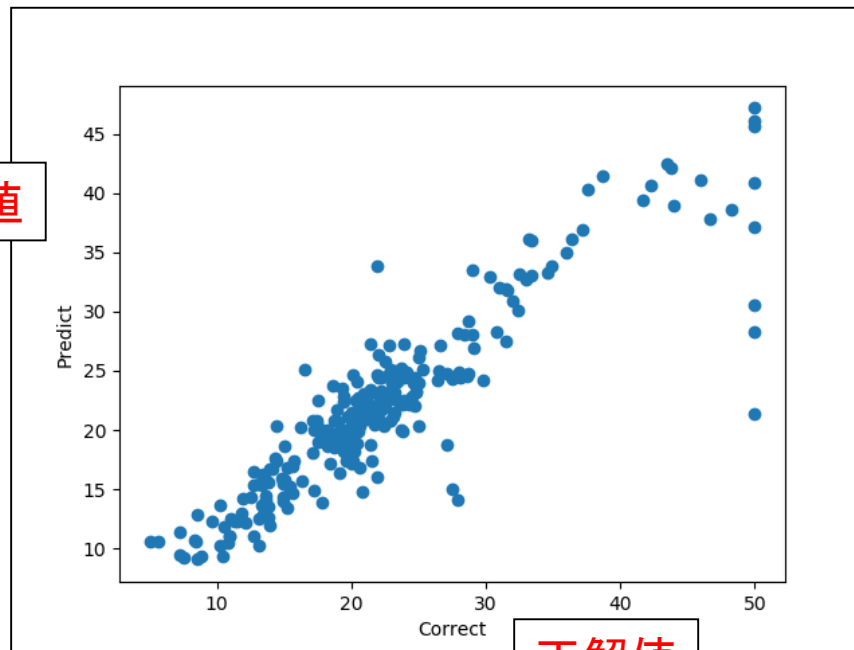
```
print( " テストデータ: {0:7.5f}".format( test_score ) )
```

散布図の描画

```
fig = plt.figure()  
plt.scatter( test_price , predict )  
plt.xlabel("Correct")  
plt.ylabel("Predict")  
fig.savefig("result.png")
```

散布図の表示

予測値



正解値

RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
RandomForestRegressor(n_estimators=回帰木の個数,  
criterion=分類指標, max_features=使用される特徴数,  
bootstrap=True, oob_score=True)
```

bootstrapを行なう場合はTrue
(デフォルトはTrue)

OOBエラーを求める場合はTrue
(デフォルトはFalse)

criterion
分類指標(デフォルトはmse)

```
model = RandomForestRegressor(n_estimators=20, criterion="mse",  
max_features="sqrt", bootstrap=True, oob_score=True)
```

max_features
使用される特徴数
sqrt → 全特徴数の平方根
log2 → \log_2 全特徴数

実行結果

```
C:\Windows\system32\cmd.exe
C:\home\shino\ML-2019\ランダムフォレスト\program>python

[ OOB score ]
0.8202524892885106

[ 特徴の重要度 ]
CRIM      : 0.07933
ZN        : 0.02293
INDUS     : 0.08121
CHAS      : 0.00457
NOX       : 0.05765
RM        : 0.21936
AGE       : 0.02781
DIS       : 0.06313
RAD       : 0.01130
TAX       : 0.07790
PTRATIO   : 0.08833
B         : 0.03841
LSTAT     : 0.22805

[ R2 ]
学習データ : 0.97876
テストデータ: 0.80710
```

OOBデータに対する R^2

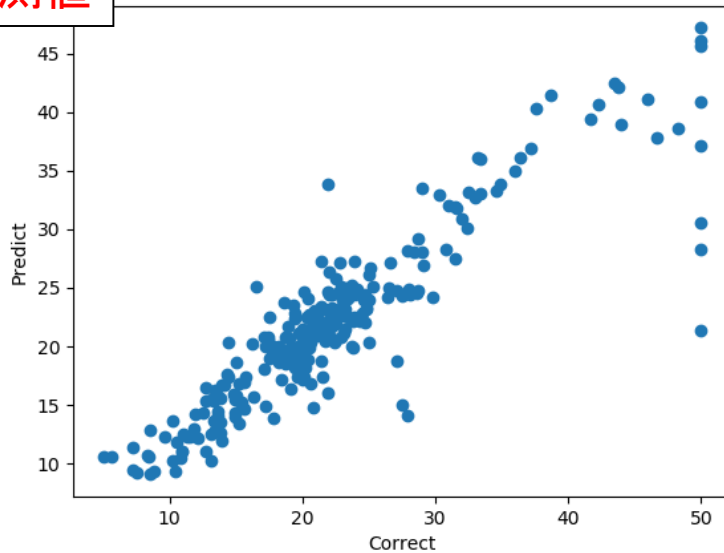
特徴の重要度

R^2 の表示

ランダムフォレスト vs 決定木

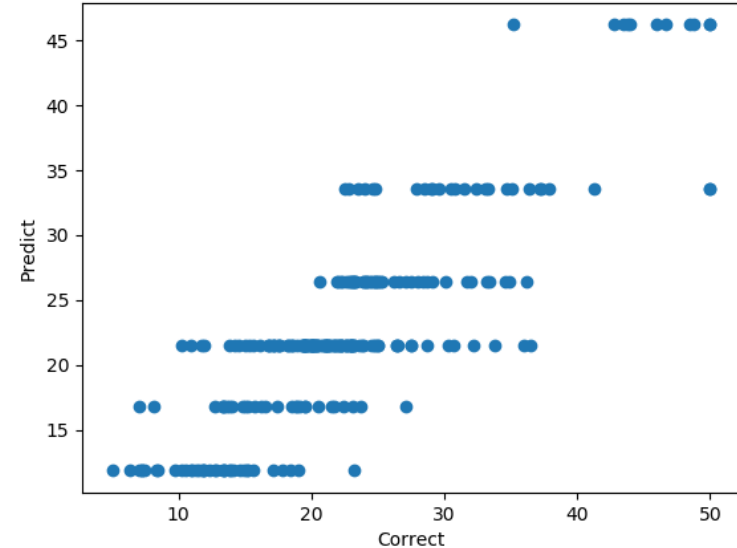
ランダムフォレスト

予測値



正解値

決定木

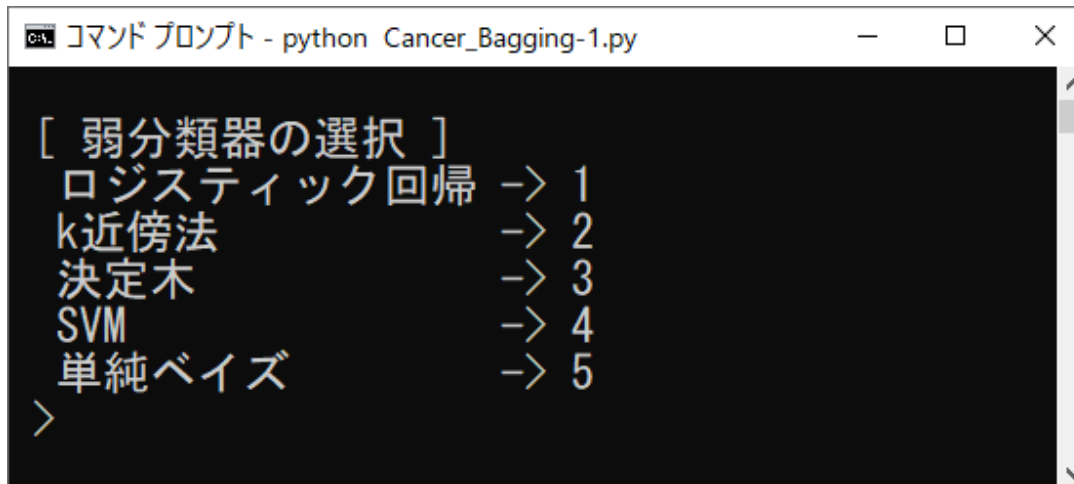


ランダムフォレスト

予測値を、複数個の決定木による平均値とするため、正解値に近づく

練習問題

- バギングのプログラム(Cancer-Bagging.py)の弱分類器にサポートベクターマシンとベイズ決定則(GaussianNB)を追加しなさい。



```
コマンド プロンプト - python Cancer_Bagging-1.py

[ 弱分類器の選択 ]
ロジスティック回帰 -> 1
k近傍法             -> 2
決定木              -> 3
SVM                  -> 4
単純ベイズ          -> 5
>
```

*宿題とします。1/6(月)13時までにプログラム(python)とワープロに実行結果を貼り付けて提出して下さい。

練習問題

SVC (RBFカーネルの場合)

```
コマンドプロンプト

[ 予測結果 ]
      precision    recall  f1-score   support

         0       0.94      0.59      0.73       101
         1       0.81      0.98      0.89       184

   accuracy          0.84       285
  macro avg       0.88      0.79      0.81       285
weighted avg       0.86      0.84      0.83       285

[ 正解率 ]
0.8421052631578947

[ 混同行列 ]
[[ 60  41]
 [  4 180]]
```

ベイズ決定則 (GaussianNB)

```
コマンドプロンプト

[ 予測結果 ]
      precision    recall  f1-score   support

         0       0.99      0.87      0.92       104
         1       0.93      0.99      0.96       181

   accuracy          0.95       285
  macro avg       0.96      0.93      0.94       285
weighted avg       0.95      0.95      0.95       285

[ 正解率 ]
0.9473684210526315

[ 混同行列 ]
[[ 90  14]
 [  1 180]]
```


参考文献①

- 加藤直樹他：データマイニングとその応用，朝倉書店，2009
- 平井有三：はじめてのパターン認識，森北出版株式会社，2012
- 後藤正幸他：入門 パターン認識と機械学習，コロナ社，2014
- 株式会社システム計画研究所編：Pythonによる機械学習入門，オーム社，2016
- 竹村彰通他：機械学習，朝倉書店，2017
- 荒木雅弘：機械学習入門，森北出版株式会社，2018

参考文献②

- BaggingClassifier
 - バギング(クラス分類)
 - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html>
- BaggingRegressor
 - バギング(回帰)
 - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html>

参考文献③

- RandomForestClassifier
 - ランダムフォレスト(クラス分類)
 - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- RandomForestRegressor
 - ランダムフォレスト(回帰)
 - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>