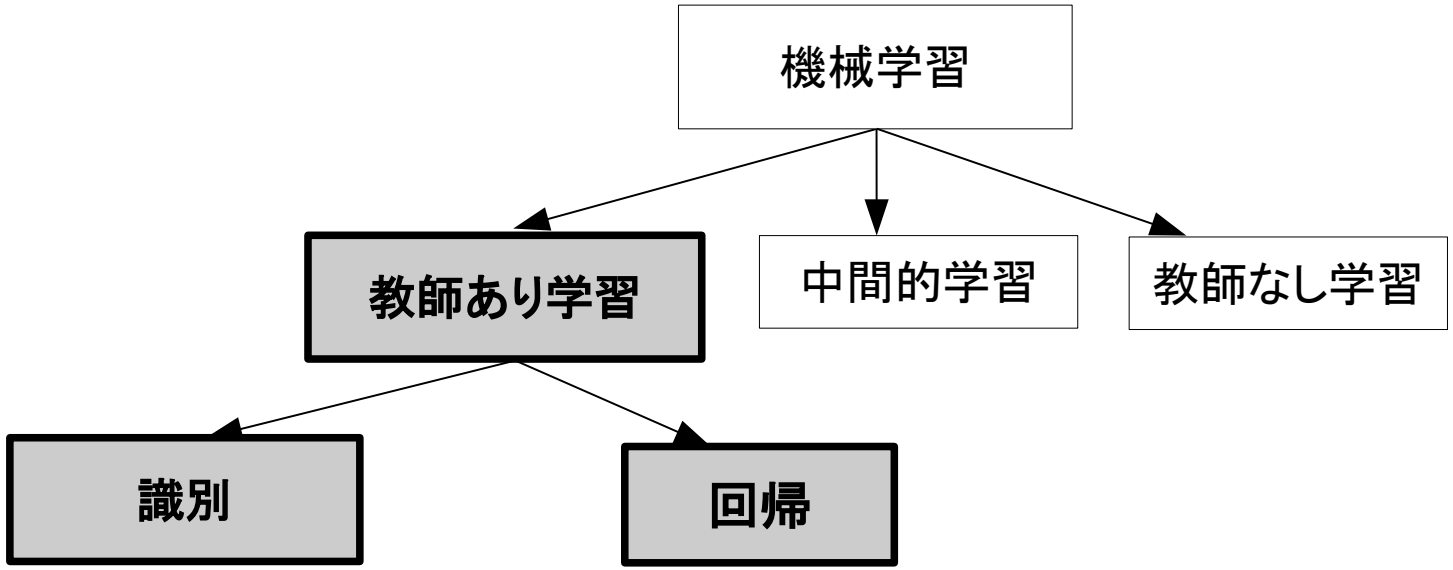
10. アンサンブル学習



- 数値特徴

気温 = 20
湿度 = 60
降水確率 = 0

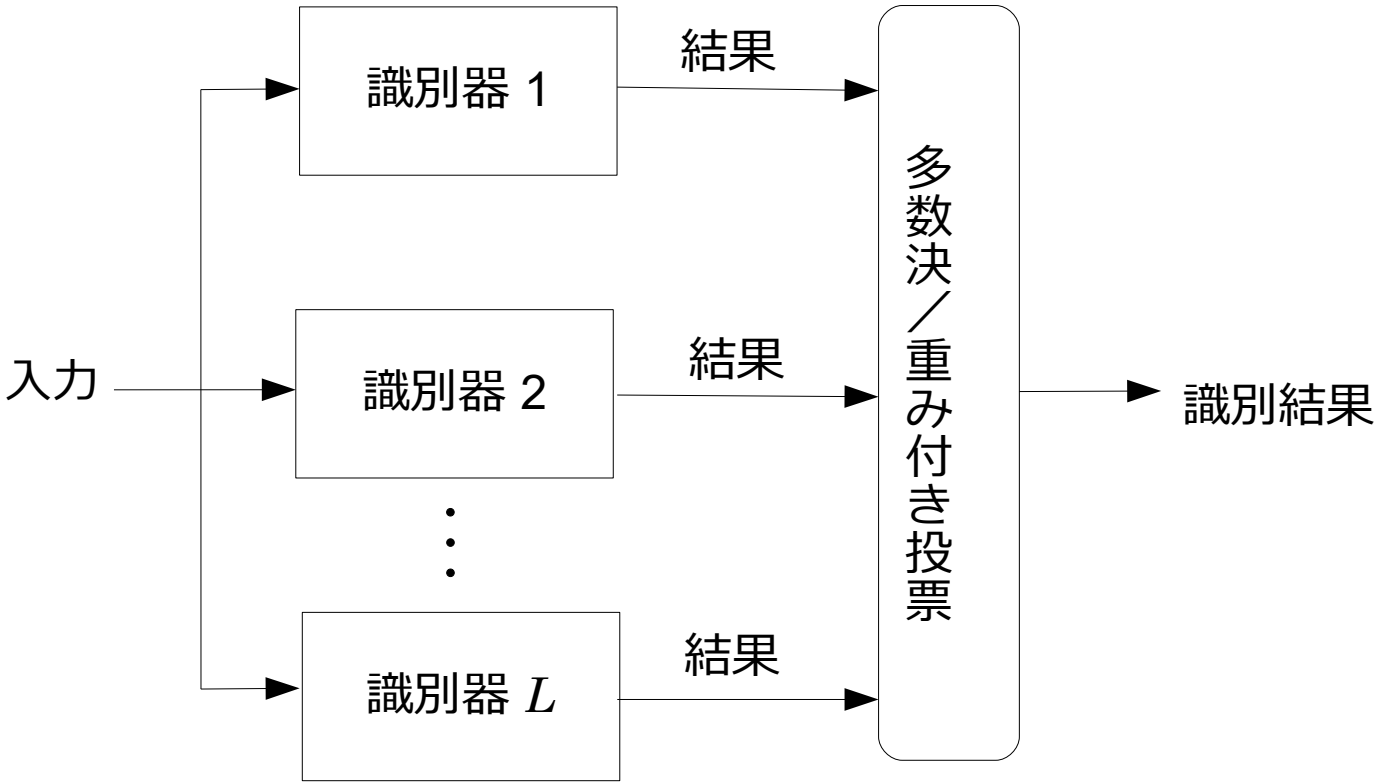
→ Play = Yes

主記憶 =256
キャッシュ =64
チャネル =16

→ 性能 = 360.5

10. アンサンブル学習

- アンサンブル学習とは
  - 識別器を複数組み合わせ、それらの結果を統合することで個々の識別器よりも性能を向上させる方法
  - 個々の識別器を弱識別器とよぶ



10.1 なぜ性能が向上するのか

- アイディア
  - 学習データから全く独立に L 個の識別器 (誤り率 ε, 誤りは独立) を作成

    $\rightarrow$ m 個の識別器が誤る確率は二項分布 $B(m;\varepsilon,L)$

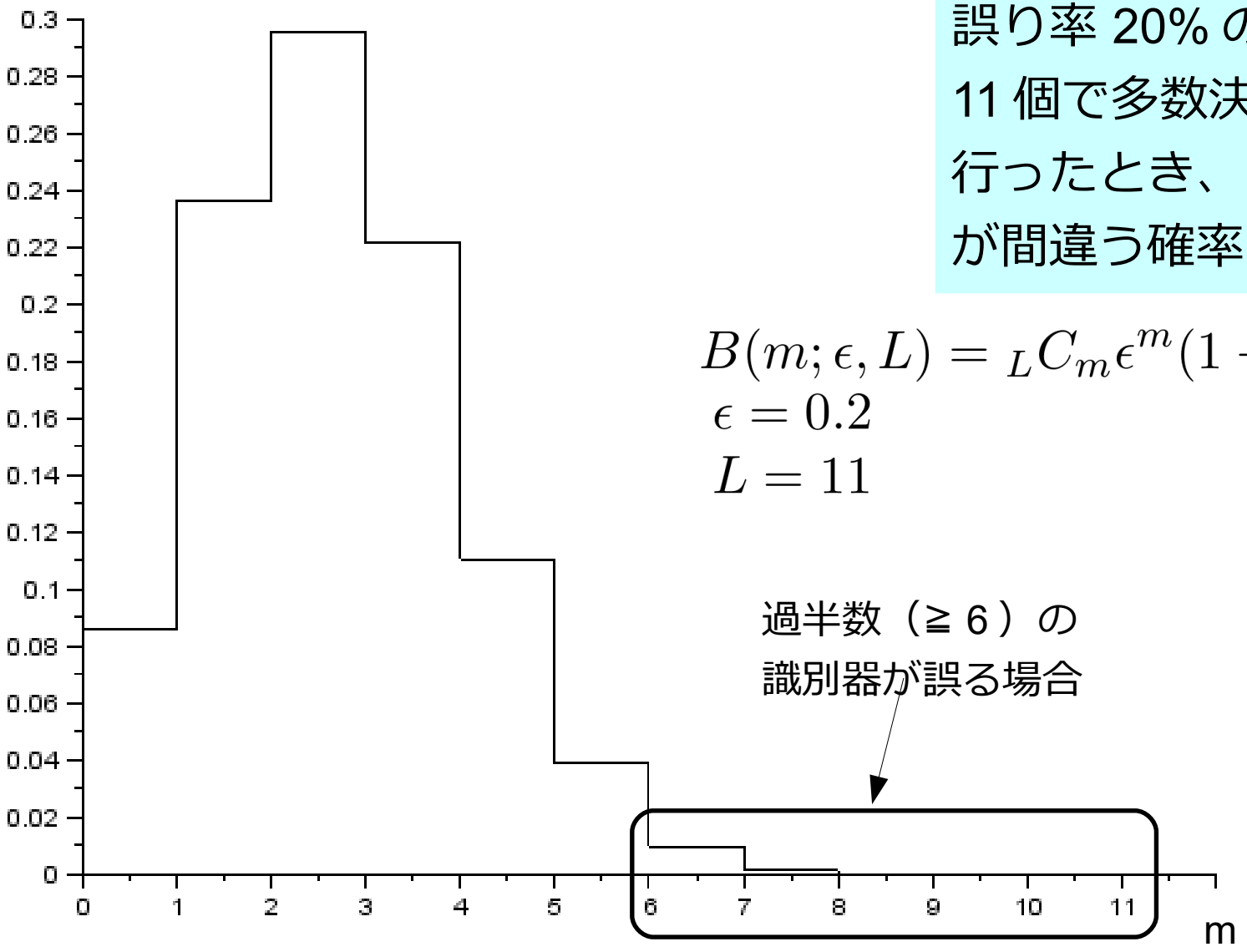
    $$B(m;\epsilon,L) = {}_{L}C_{m}\epsilon^{m}(1-\epsilon)^{L-m}$$

    $\rightarrow \varepsilon<0.5$ のとき、 $m>L/2$ となる B は小さい値

10.1 なぜ性能が向上するのか

誤り率 20% の弱識別器 11 個で多数決統合を行ったとき、 6 個以上が間違う確率は 1.2%

$$B(m;\epsilon,L) = {}_LC_m\epsilon^m(1-\epsilon)^{L-m}$$
$$\epsilon = 0.2$$
$$L = 11$$

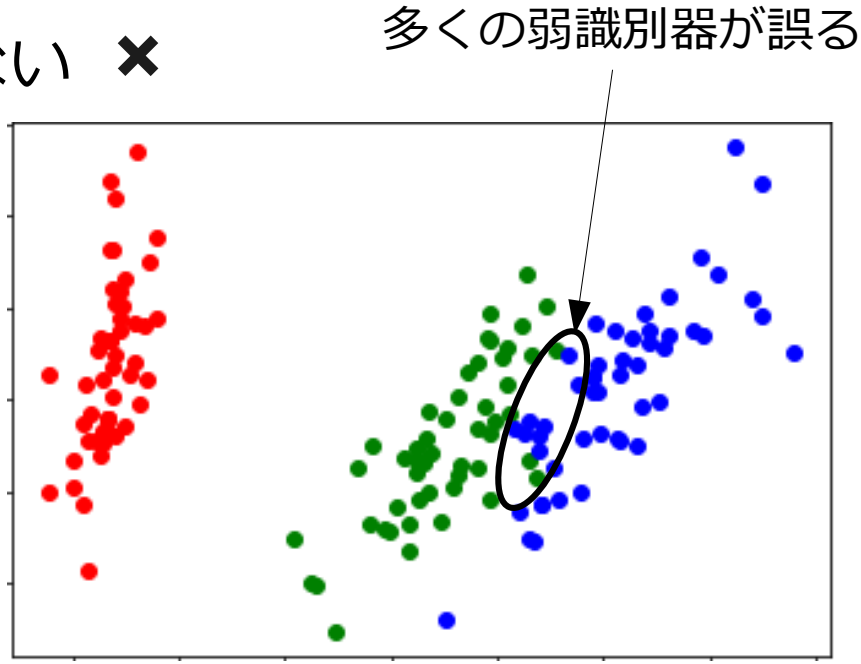


10.1 なぜ性能が向上するのか

- ここまでの議論の非現実的なところ
  - 「それぞれの弱識別器の誤りが独立」

    ➡ データの誤りやすさに差はない ✖

    識別面付近のデータなど、
    普通は成立しない



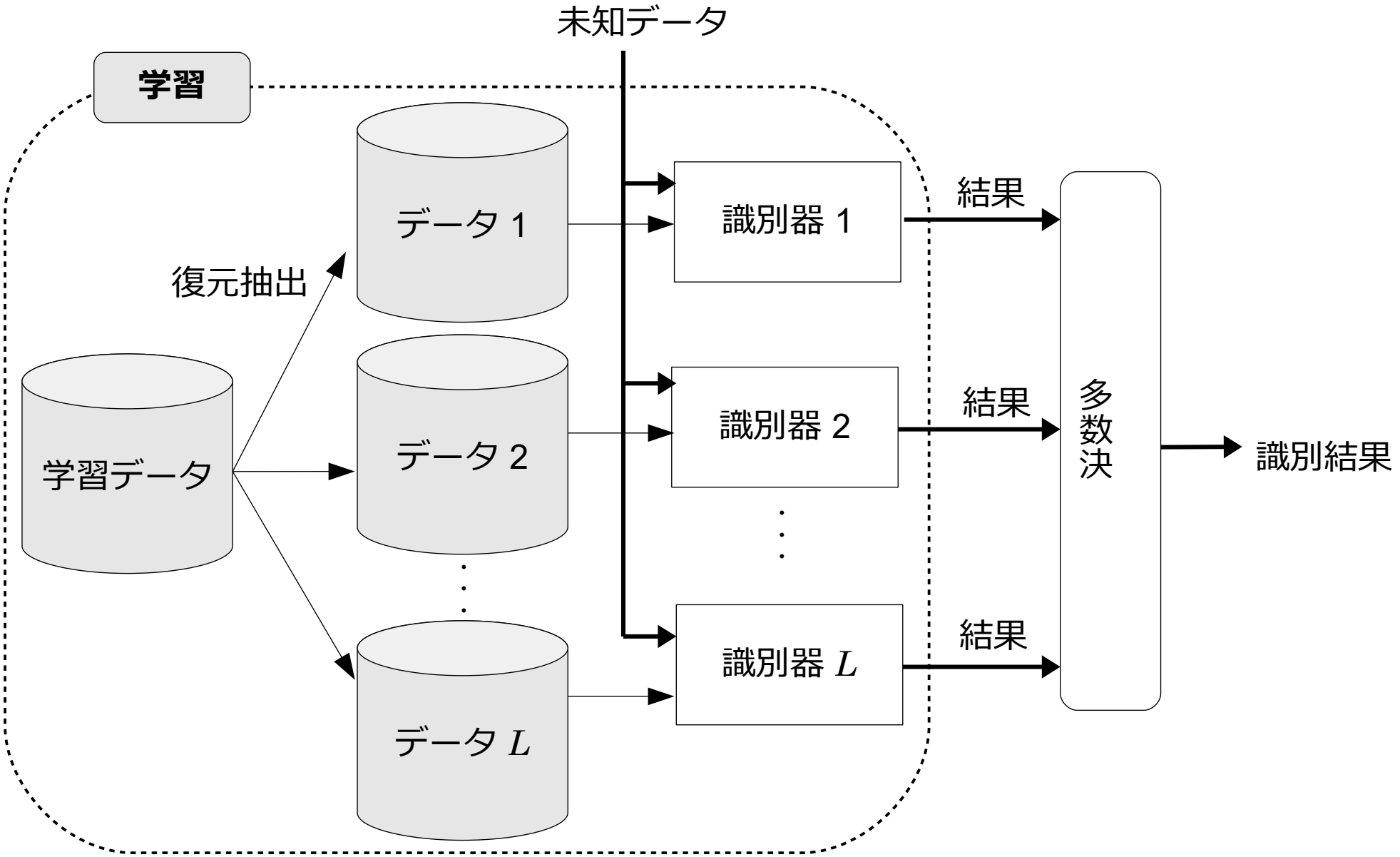
- アンサンブル学習の目標
  - なるべく異なる振る舞いをする弱識別器を作成する

10.2 バギング

- アイディア
  - 異なる学習データから作成された識別器は異なる
- Bagging とは
  - **b**ootstrap **agg**regat**ing**
  - 元の学習データから異なる部分集合を作成
  - それぞれに対して識別器を作成
  - 結果を統合

10.2 バギング

- バギングの構成



10.2 バギング

- 特徴
  - 学習データから復元抽出することで、元のデータと同じサイズの独立なデータ集合を作成する。

    n 回行って、あるデータが抽出されない確率：$(1-\frac{1}{n})^n$

    n→∞ で
    約 0.368

  - 各々のデータ対して同じアルゴリズムで弱識別器を作成する
    - アルゴリズムは不安定 (学習データの違いに敏感) な方がよい

      例) 枝刈りをしない決定木
  - 結果の統合は多数決

10.3 ランダムフォレスト

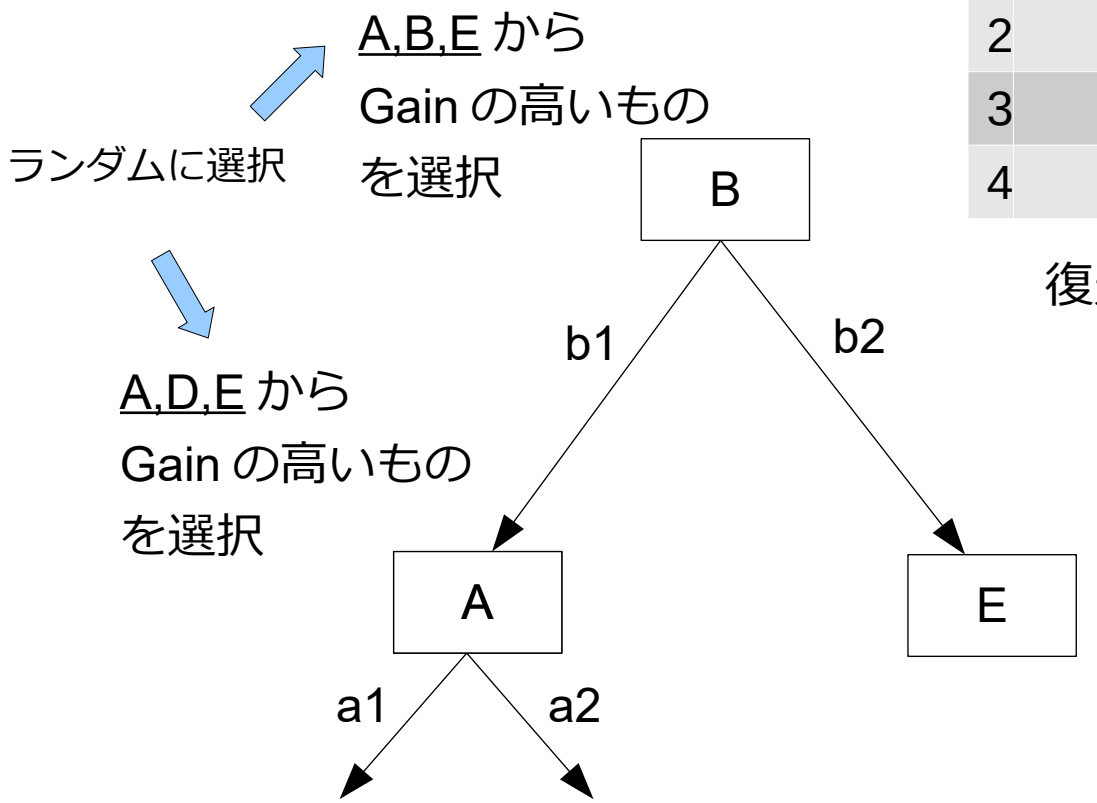
- アイディア
  - 識別器を作成する毎に異なる特徴を用いることで、異なった弱識別器を複数作成する
- ランダムフォレストとは
  - ランダム：乱数によって用いる特徴を制限する
  - フォレスト：森　＝　複数の決定木
  - バギングのアイディアも取り入れて、学習毎にデータセットは復元抽出

10.3 ランダムフォレスト

- ランダムフォレストの学習

	A	B	C	D	E	class
1						
2						
3						
4						

復元抽出された学習データ



葉が単一クラスになるまで（過）学習

これを複数回繰り返す

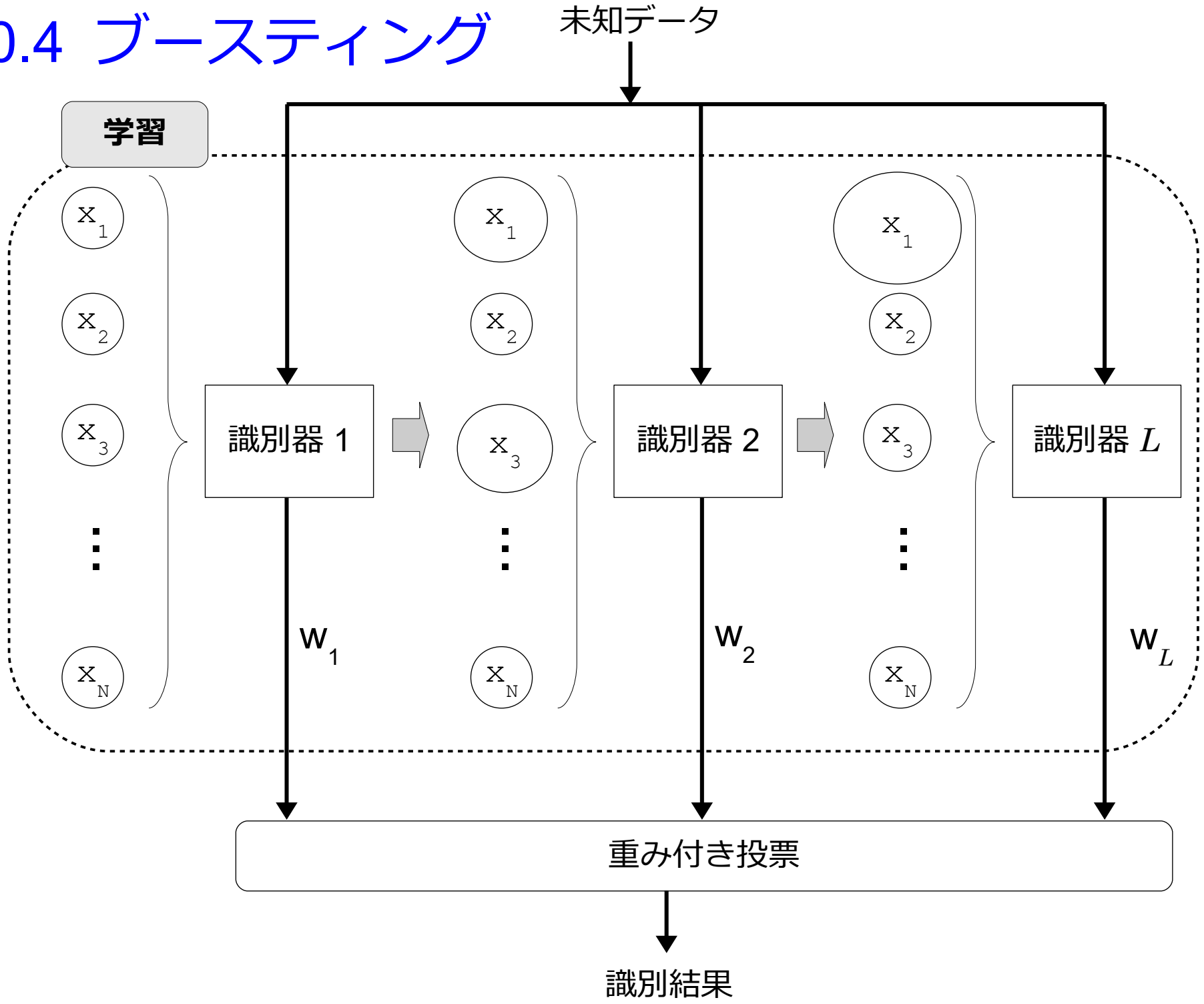
10.3 ランダムフォレスト

- 特徴
  - 識別器作成に使用できる特徴をランダムに制限することで各抽出データ毎に異なった弱識別器ができる
  - 森のサイズ（＝決定木の数）を大きくしても、過学習が起きにくいことが知られている

10.4 ブースティング

- アイディア
  - 現在の識別器が誤識別を起こすデータを正しく識別する弱識別器を逐次的に追加
  - 過学習とならないように、弱識別器として浅い決定木を用いることが多い
- Boosting とは
  - boost: （能力などを）上昇（向上）させる

10.4 ブースティング



10.4 ブースティング

- 特徴
  - 逐次的に相補的な弱識別器を作成
  - 以前の弱識別器が誤った事例の重みを大きくして次の弱識別器を学習
  - 学習アルゴリズムが重みに対応していない場合は、重みに比例した数を復元抽出
  - 結果は弱識別器の性能に基づく重み付き投票

10.4 ブースティング

- 重みの計算

  1.開始時は全学習データに同じ重みを付与

  2.最初の識別器を作成。誤り率 ε_t (<0.5) を算出

  - 誤識別されたデータの重みを $1/(2\varepsilon_t)$ 倍
  - 正しく識別されたデータの重みを $1/\{2(1-\varepsilon_t)\}$ 倍

  3. t 回目の学習においても上記の更新を適用

両方の重みの和を等しくしている

10.5 勾配ブースティング

- ブースティングの一般化
  - 全体の識別器は、 M 個の弱識別器の重み付き和

$$F(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m h(\boldsymbol{x}; \gamma_m)$$

  - 逐次加法的な学習プロセスの表現

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \alpha_m h(\boldsymbol{x}; \gamma_m)$$

  - 各ステップで h_m を選択するための損失関数 L の導入

$$(\alpha_m, \gamma_m) = \arg\min_{\alpha,\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \alpha h(\boldsymbol{x}_i; \gamma))$$

10.5 勾配ブースティング

- 勾配ブースティングの学習
  - L の勾配を求めることで最小化問題を置き換える

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \alpha_m \sum_{i=1}^{N} \nabla_F L(y_i, F_{m-1}(\boldsymbol{x}_i))$$

損失関数の値を最も減らす木を求める

- 損失関数
  - 二乗誤差、誤差の絶対値、フーバー損失（ $|x| \leq \epsilon$ の範囲で 2 次関数、その外側の範囲で線形に増加）などが用いられる

まとめ

- Weka デモ
  - diabetes データ
  - Bagging, RandomForest, AdaBoostM1
- アンサンブル学習
  - 異なる振る舞いをする識別器を多数学習する
    - バギング：異なる学習データを用いる
    - ランダムフォレスト：乱数で使用する特徴を制限
    - ブースティング：誤ったデータに注目して逐次的に識別器を作成