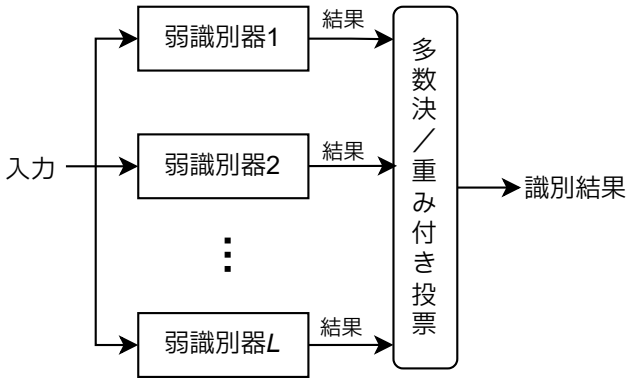
10. アンサンブル学習

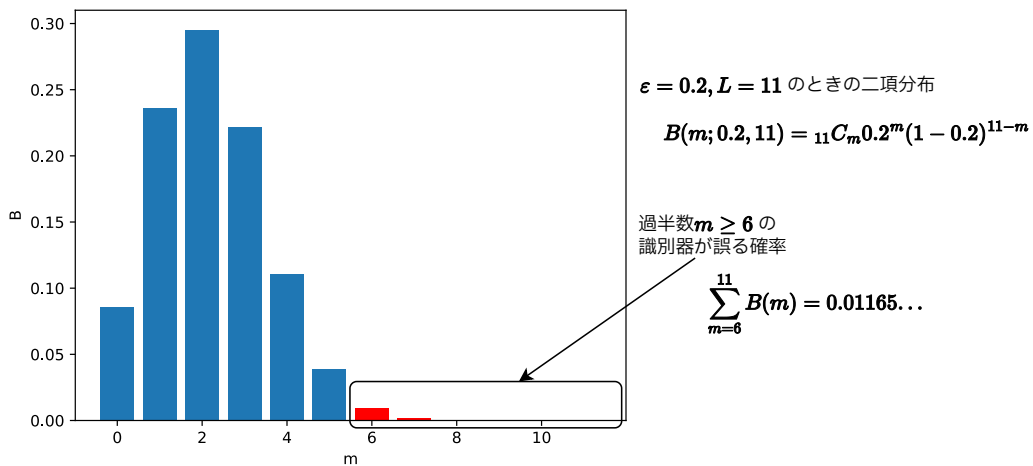


- 10.1 なぜ性能が向上するのか
- 10.2 バギング
- 10.3 ランダムフォレスト
- 10.4 ブースティング
- 10.5 勾配ブースティング

- 荒木雅弘：『Pythonではじめる機械学習』（森北出版，2025年）
- スライドとコード

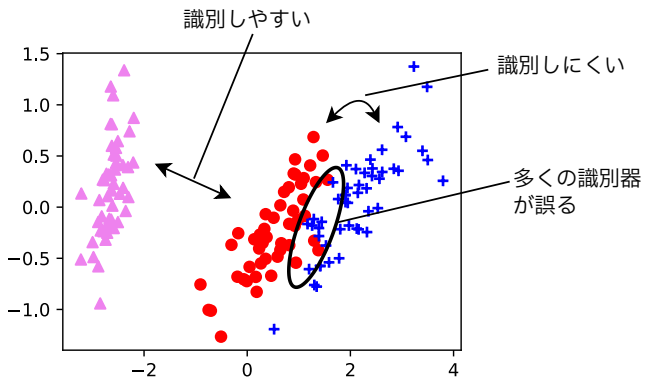
10.1 なぜ性能が向上するのか (1/2)

- アンサンブル学習の基本的なアイディア
  - 複数の識別器の結果を多数決で統合することで，より強力な識別器を作成する
    - 学習データから L 個の弱識別器（誤り率 ϵ, 誤りは独立）を作成できると仮定
    - m 個の弱識別器が誤る確率は二項分布 $B(m;\epsilon,L) = {}_LC_m\epsilon^m(1-\epsilon)^{L-m}$ に従う
    - $\epsilon < 0.5$ のとき，$m > L/2$ となる確率は小さい値になる



10.1 なぜ性能が向上するのか (2/2)

- このアイディア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」
  - 現実には，弱識別器間で誤りの相関がある



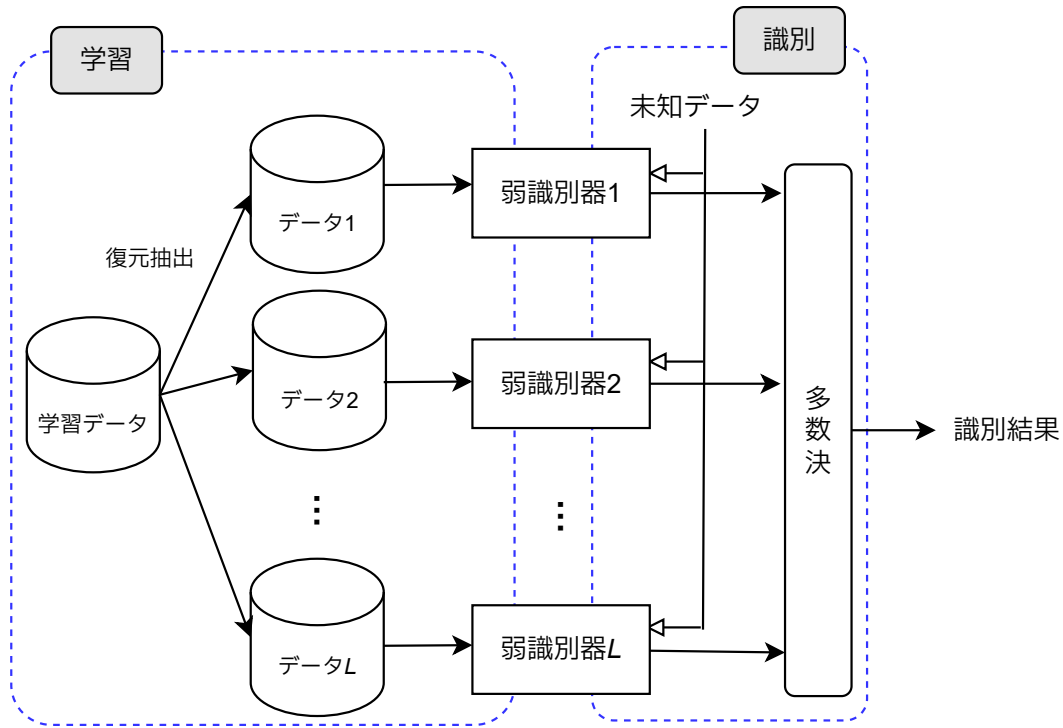
- アンサンブル学習の目標
  - 「弱識別器の誤りが独立」に近づけるために，なるべく異なる振る舞いをする弱識別器を作成する

10.2 バギング (1/3)

- アイディア
  - 異なる学習データから作成された弱識別器は，異なる振る舞いをする可能性が高い
- Bagging (bootstrap aggregating) とは
  - 元の学習データから復元抽出によって異なるデータ集合を複数作成
  - それぞれに対して弱識別器を作成
  - 結果を多数決で統合

10.2 バギング (2/3)

- バギングの構成



10.2 バギング (3/3)

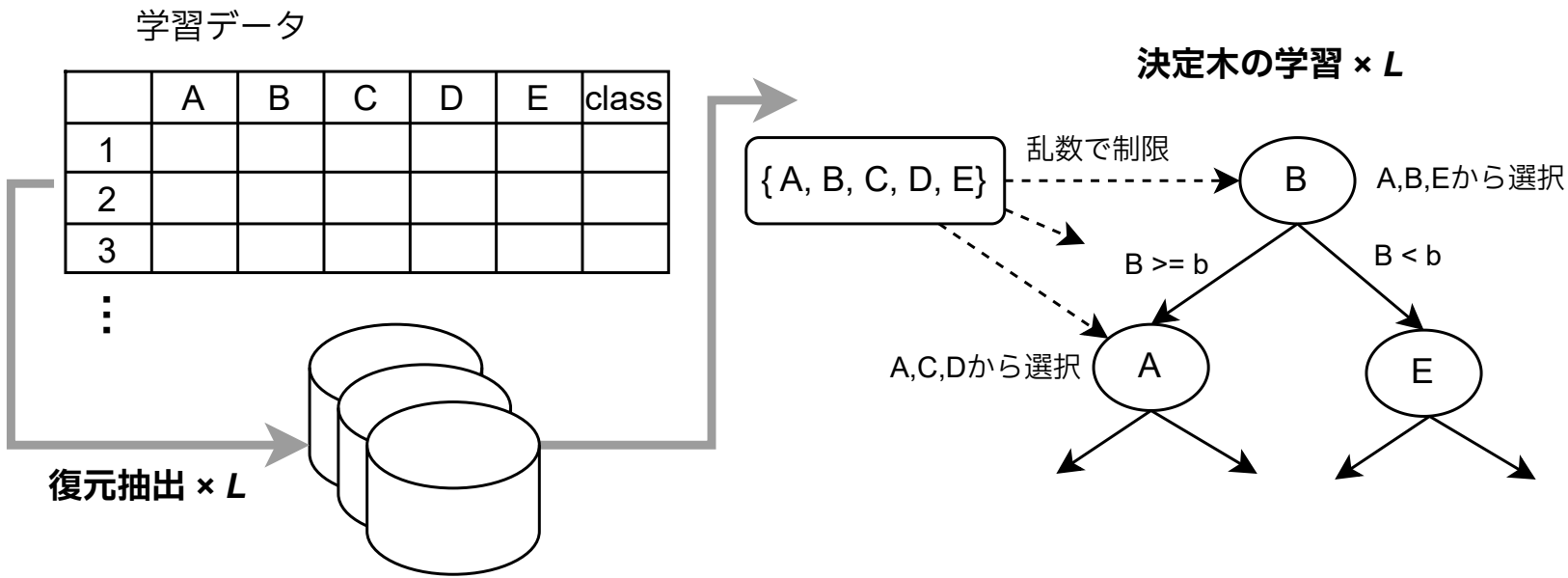
- 学習
  - 学習データから復元抽出を行って，元データと同サイズのデータ集合を複数作成する
    - N回抽出を行って，特定のデータが抽出されない確率: $(1 - \frac{1}{N})^N$
      - $N \to \infty$ で $\frac{1}{e} \sim 0.368$
  - 各々のデータ集合に対して同じアルゴリズムで弱識別器を作成する
  - アルゴリズムは不安定（学習データの違いに敏感）なほうがよい
    - 例) 枝刈りをしない決定木
- 識別
  - 弱識別器に差を付ける根拠がないので，結果の統合は多数決
- バギングの欠点
  - 現実には，作成される弱識別器はそれほど大きく異ならない

10.3 ランダムフォレスト (1/2)

- アイディア
  - バギングにおける弱識別器作成アルゴリズムにランダム性を入れることで，弱識別器間の異なりを大きくする
- ランダムフォレストとは
  - ランダム：学習の際に用いる特徴を乱数によって制限する
  - フォレスト：森 = 複数の決定木
- 利点
  - 各抽出データ毎に比較的大きく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

10.3 ランダムフォレスト (2/2)

- ランダムフォレストの学習

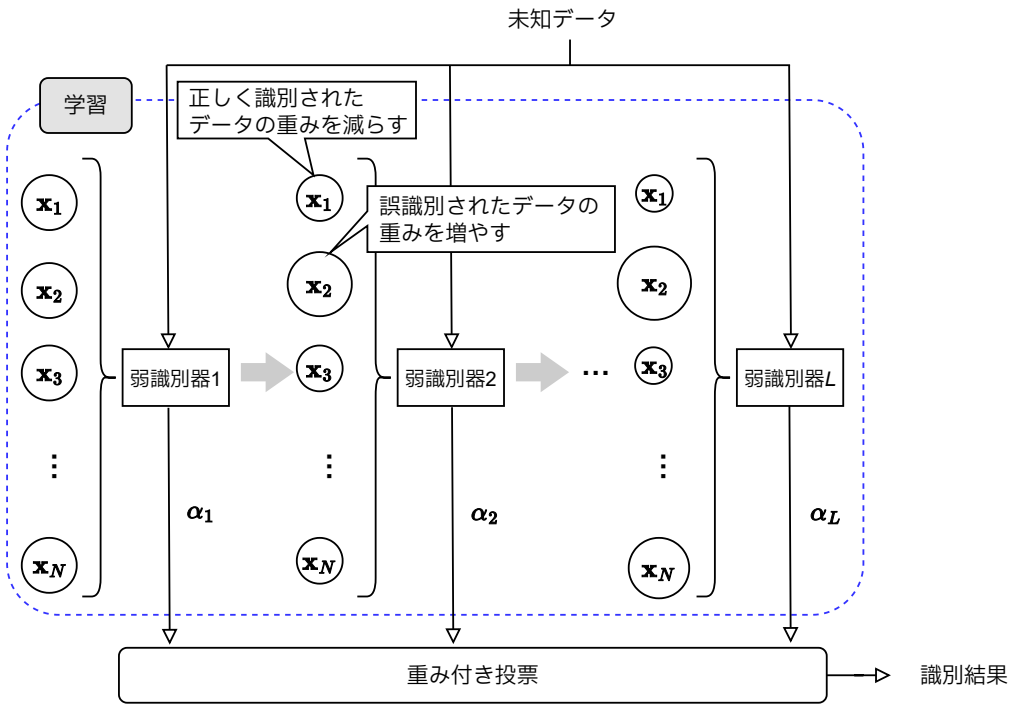


10.4 ブースティング (1/3)

- アイディア
  - 逐次的に能力を向上させる（boost させる）弱識別器を追加してゆく
  - ステップ $t-1$ に作成した弱識別器が誤識別したデータの重みを大きくし，ステップ t では，そのデータを正しく識別しやすい弱識別器を作成
- 構成
  - 学習アルゴリズムは重み付きデータに対応しているものが望ましい
    - 重みに対応していない場合は，重みに比例してデータを復元抽出
  - 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
  - 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 (2/3)

- ブースティングの構成



10.4 ブースティング (3/3)

- AdaBoostアルゴリズム(2値分類の場合)
  1. 開始時は全学習データに同じ重み $w_0^{(i)} = \frac{1}{N}$ を付与
  2. $t = 1$ から L まで以下を繰り返し
     1. t 番目の弱識別器を作成．それを用いて誤り率 r_t (< 0.5) を算出

$$r_t = \frac{\sum_{i=1}^{N} w_t^{(i)} [y_i \neq h_t(\boldsymbol{x}_i)]}{\sum_{i=1}^{N} w_t^{(i)}}$$

     2. 弱識別器の重み $\alpha_t = \eta \ln \frac{1-r_t}{r_t}$ を計算（η は学習率）
     3. 誤識別されたデータの重みを $w_{t+1}^{(i)} = w_t^{(i)} \exp(\alpha_t)$ に修正
     4. 全体の重みを正規化

$$w_{t+1}^{(i)} = \frac{w_{t+1}^{(i)}}{\sum_{j=1}^{N} w_{t+1}^{(j)}}$$

10.5 勾配ブースティング (1/8)

- 勾配ブースティングとは
  - ブースティングにおける各ステップで，これまでで作成したアンサンブル識別器に追加したときの損失関数の値が最小となる弱識別器を加える
- ブースティングの一般化
  - 全体の識別器の出力は，$\boldsymbol{F}(\boldsymbol{x})$ は，M 個の識別器 $h(\boldsymbol{x};\gamma_m)$ の重み付き和で得られる
    - γ_m は識別器のパラメータ

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

10.5 勾配ブースティング (2/8)

- 学習の手順
  - 識別器の学習を逐次加法的な学習プロセスに分解

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

  - 第 m ステップで加える弱識別器 $h_m(\boldsymbol{x})$ を選択するための損失関数の導入
    - ステップ m の時点でのアンサンブル識別器の損失を L_m とする
    - 損失関数 L_m の例
      - 回帰: 2乗誤差，Huber損失
      - 識別: log損失，指数損失

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

10.5 勾配ブースティング (3/8)

- 勾配ブースティングのアイディア
  - 損失関数の値が最小になるものを求め，新しい識別器を構成する

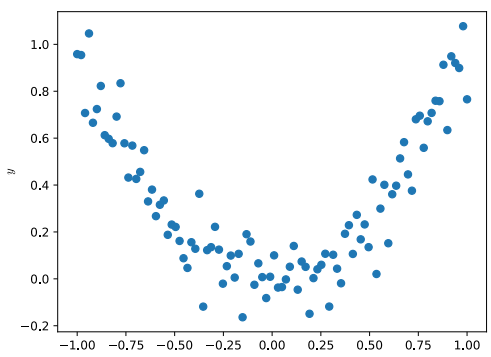
$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \alpha_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(\boldsymbol{x}_i))$$

    - 損失関数を差分とみなし，これを最小化する弱識別器を追加していると解釈できる

10.5 勾配ブースティング (4/8)

- 弱回帰器の例
  - 回帰対象のデータを作成

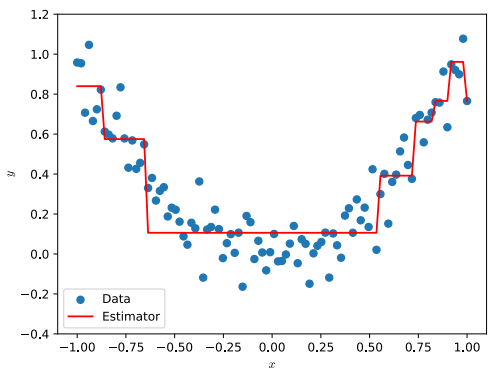
```
np.random.seed(2)
X = np.linspace(-1, 1, 100)
y = X**2 + np.random.normal(0, 0.1, 100)
plt.scatter(X, y)
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")
plt.show()
```



10.5 勾配ブースティング (5/8)

- 最初の弱回帰器
  - 深さ3の回帰木を用いて回帰を行う
  - 関数 `reg_plot()` の定義は，サポートページのコード chap10.ipynb を参照

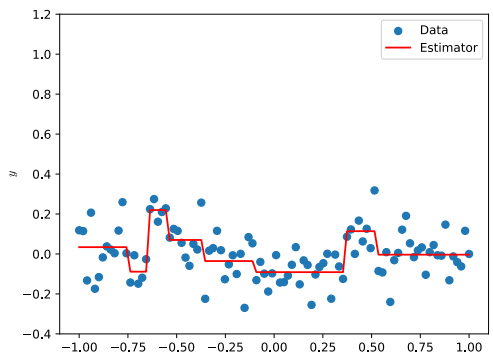
```
tree_reg1 = DecisionTreeRegressor(max_depth=3, random_state=2)
tree_reg1.fit(X.reshape(-1, 1), y)
reg_plot(X, y, tree_reg1)
```



10.5 勾配ブースティング (6/8)

- 差分をターゲットとして，次の弱回帰器を作成

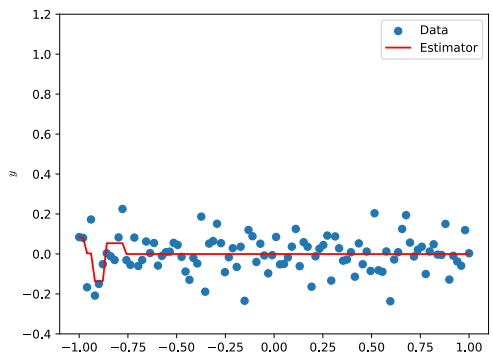
```
y2 = y - tree_reg1.predict(X.reshape(-1, 1))
tree_reg2 = DecisionTreeRegressor(max_depth=3, random_state=2)
tree_reg2.fit(X.reshape(-1, 1), y2)
reg_plot(X, y2, tree_reg2)
```



10.5 勾配ブースティング (7/8)

- 同様の手順で次の回帰木を作成

```
y3 = y2 - tree_reg2.predict(X.reshape(-1, 1))
tree_reg3 = DecisionTreeRegressor(max_depth=3, random_state=2)
tree_reg3.fit(X.reshape(-1, 1), y3)
reg_plot(X, y3, tree_reg3)
```

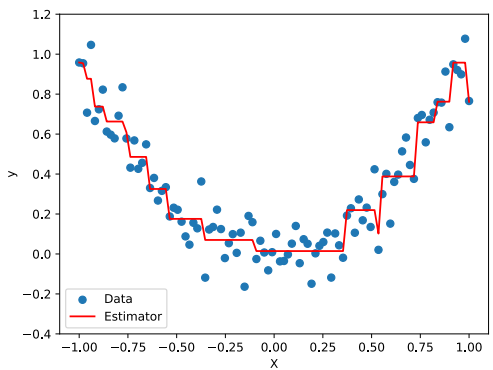


10.5 勾配ブースティング (8/8)

- 勾配ブースティングで作成したアンサンブル回帰木

```
class FinalRegressor:
    def predict(x):
        return tree_reg1.predict(x) + tree_reg2.predict(x) + tree_reg3.predict(x)

reg_plot(X, y, FinalRegressor)
```



まとめ

- アンサンブル学習とは
  - 弱識別器を複数組み合わせ，それらの結果を統合
  - 異なる振る舞いをする弱識別器を作る
- バギング
  - 復元抽出で異なる学習データを作成
  - ランダムフォレストは，バギング＋決定木学習において乱数で特徴を制限
- ブースティング
  - 以前の識別器の誤りを補完する識別器を順次作成
  - 勾配ブースティングは，損失関数の勾配に基づいて弱識別器を追加