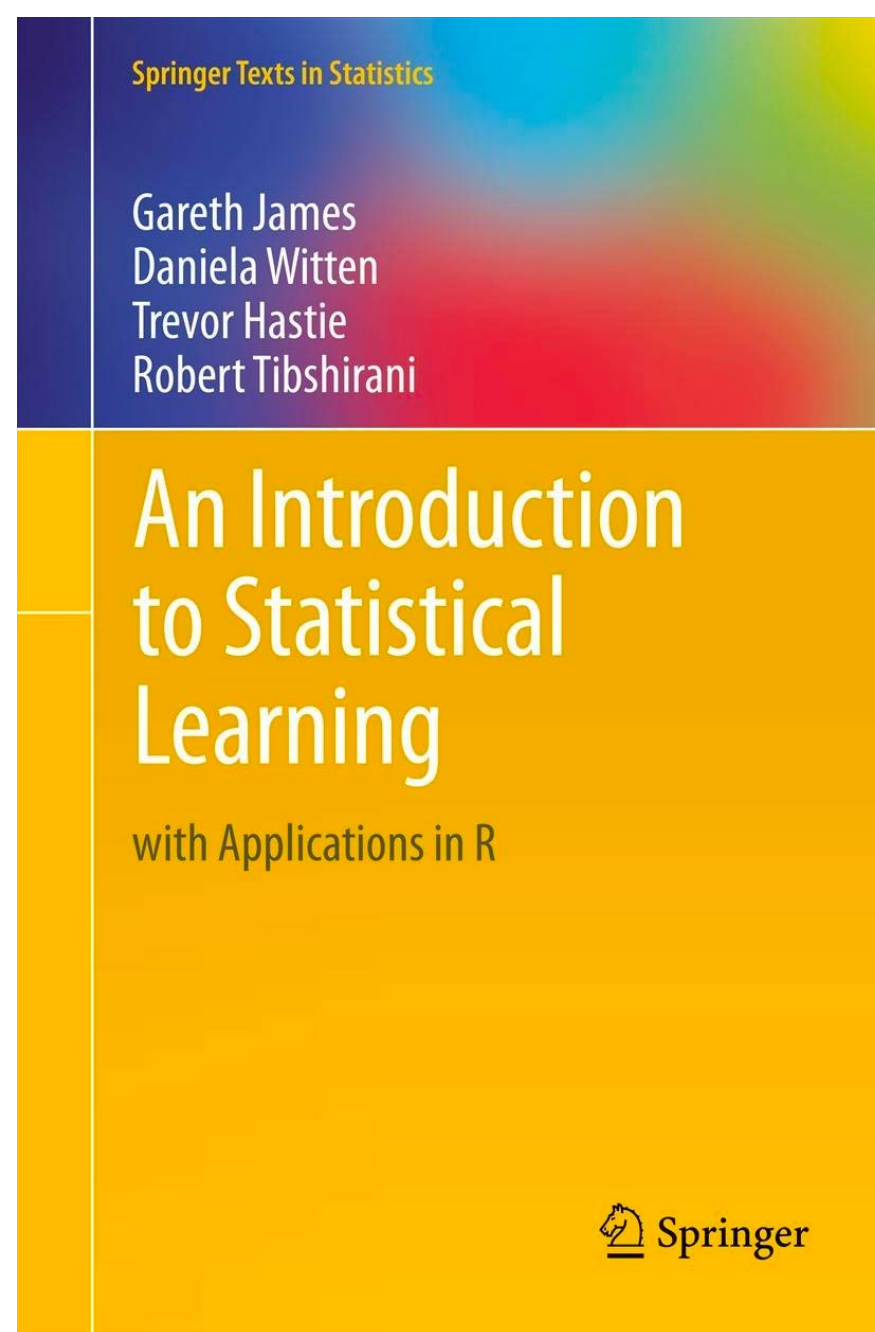


Tree-based methods

正田 備也

masada@rikkyo.ac.jp

この本の第8章を使って、
授業内容を組み立てました。
(図も同章から取っています。)

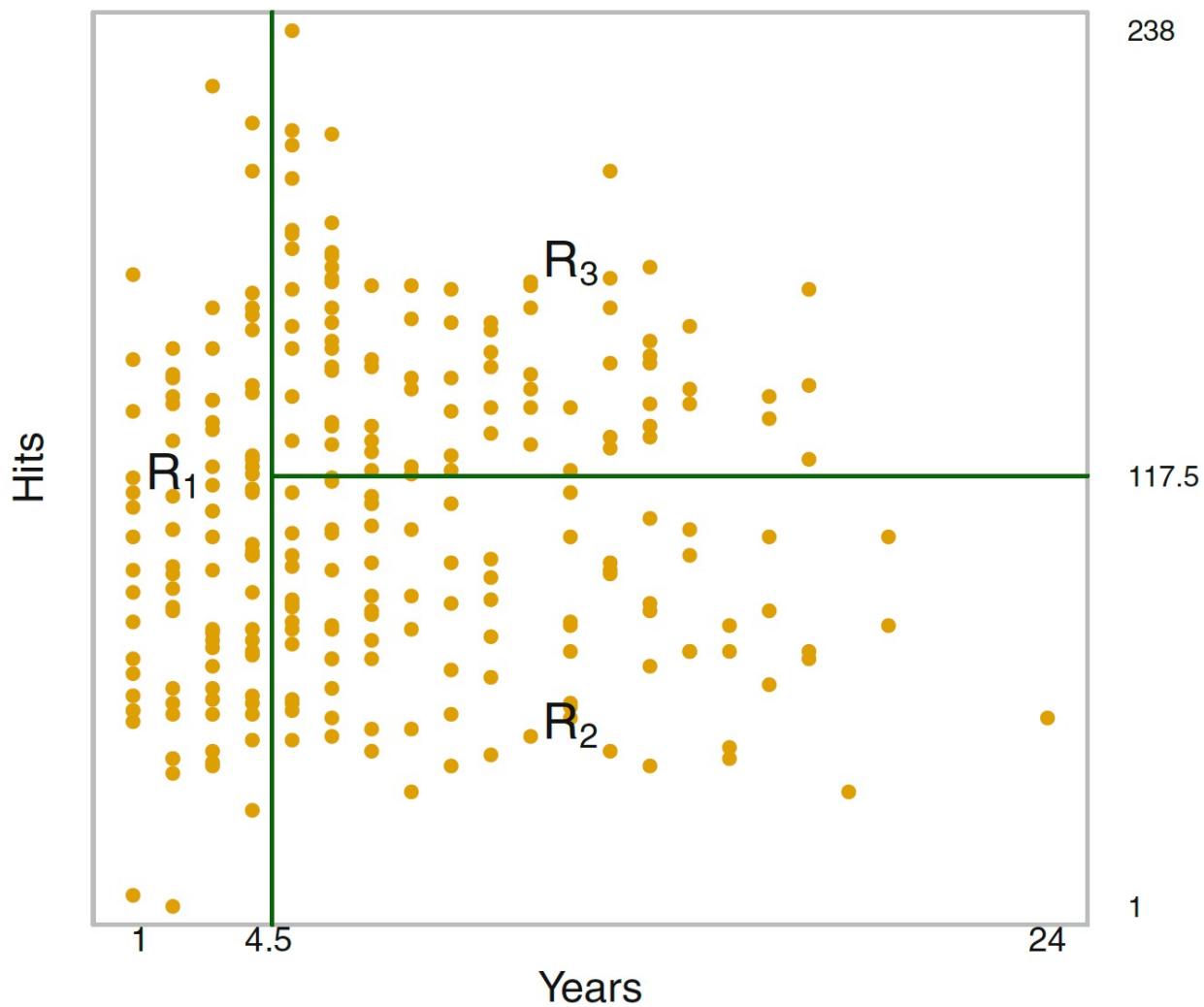


Tree-based methodsの概略

- 入力ベクトルが属する空間を、単純な領域へ分割
 - 分割していく一連のルールを、木構造で表現する
 - そのため、tree-based methodsと呼ばれる
- 未知の入力ベクトルについては、以下のように予測
 - その未知ベクトルが属する領域を、つきとめる
 - その領域に属する訓練データについて、対応する目的変数の
平均値（回帰）や最頻値（分類）を求め、これを予測値とする
 - ちなみに「決定木」は、分類のためのtree-based methodsのこと。



空間を分割していく一連のルールを
木構造として要約したもの
(下端の数値は野球選手の年俸)



空間が分割されている様子の図示

分割の方針：RSS（残差平方和）の最小化

- 下記で定義されるRSSができるだけ小さくなるように、入力ベクトルが属する空間の分割 R_1, \dots, R_J を求める

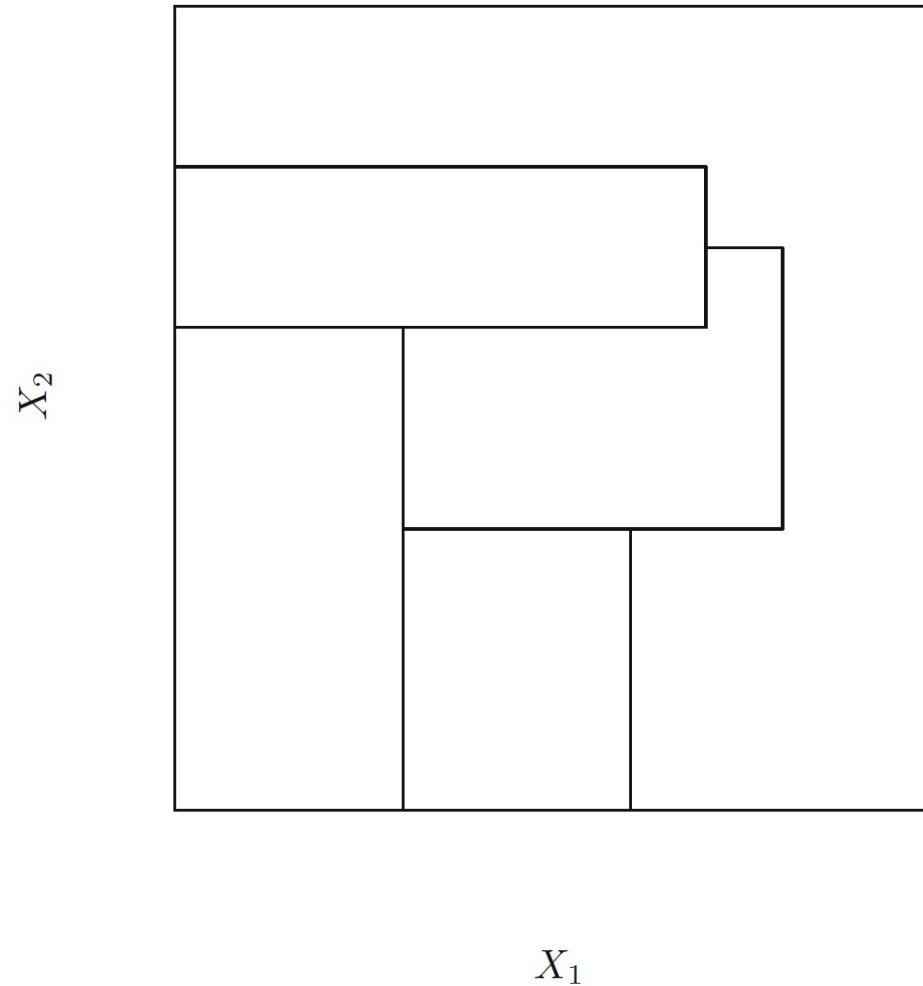
$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

- R_j は、第 j 番目の領域に含まれる訓練データの添字の集合
- \hat{y}_{R_j} は、第 j 番目の領域に含まれる訓練データの目的変数の値の平均

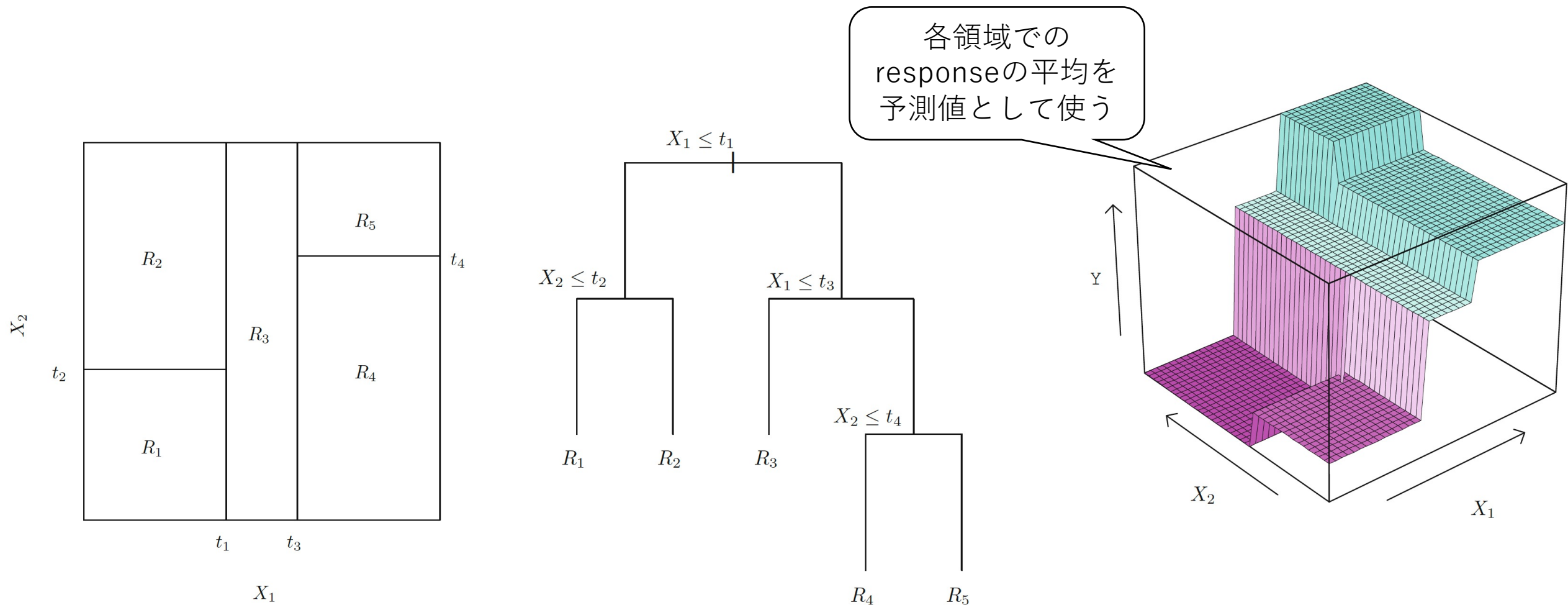
学習アルゴリズム：Recursive Binary Splitting

- 入力ベクトルが属する空間を、貪欲に2分割していく方法
 - 貪欲=先のことを考えず、毎ステップで最善の答えを選択する
- 1. 説明変数のうち一つとその切断点 s とを、下記の基準で決定
 - 当該の説明変数の値が s 未満のものと、 s 以上のものとへ訓練データを分割した場合に、RSSを最も大きく減らすことができる。
- 2. 前もって定めた終了条件が満たされていないなら1.へ戻り、満たされていれば終了
 - 終了条件の例：「どの領域にも5個以下しか訓練データが含まれない」

recursive binary splittingでは実現できない分割の例



recursive binary splittingの実行例



Tree Pruning

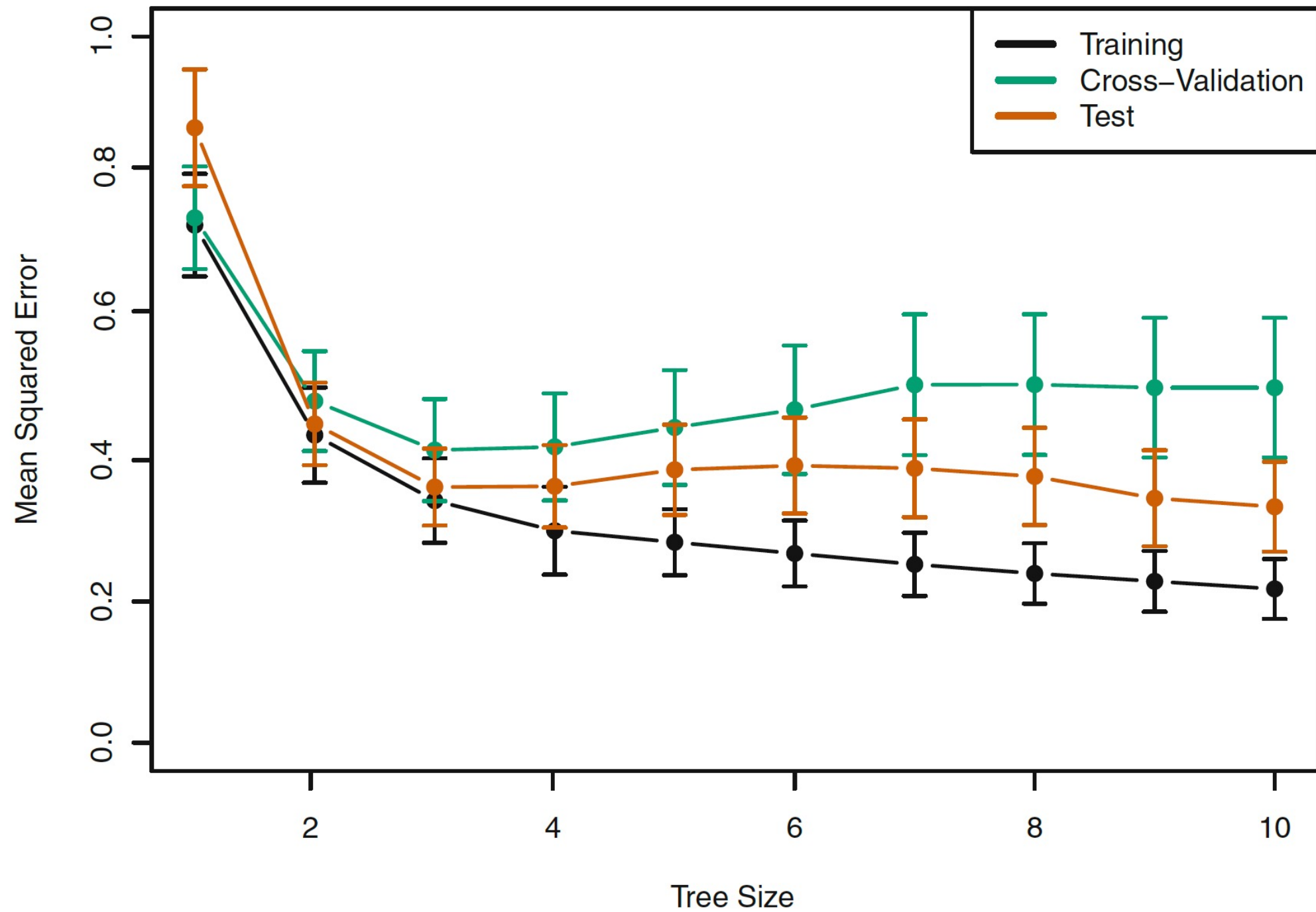
- 上記の手法で得た分割は、訓練データに合いすぎていて、訓練データ以外のデータでうまくいかないことが多い
 - いわゆる過学習。
- そこで、枝刈りをする
- 枝刈りの単純な方法
 - Recursive binary splittingで木を求める。この木を T_0 と呼ぶ。
 - T_0 の全ての部分木 T を交差検証で評価し、最も良い部分木を選ぶ。
 - しかし、この方法は時間がかかりすぎる。

木の複雑さをコントロールする

- RSSに、木の複雑さを表す項を追加する

$$\sum_{j=1}^{|T|} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|$$

- 上記の関数を最小化する α を、交差検証で決定する
 - $|T|$ は木 T の葉ノードの個数。つまり分割によって得られる領域の個数。
 - $\alpha |T|$ は、木が複雑になることによるペナルティを表す項。



分類にtree-based methodsを使う場合

- 予測：各領域で最多数派を予測に使う
- 学習：領域を分割していくとき、何を最小化するか？
 - 領域を表す添字を j 、クラスを表す添字を k とする。
- 1. 最多数派以外のデータ点の割合： $1 - \max_k(\hat{p}_{jk})$
 - これではうまくいかないらしい。
- 2. Gini係数（分散に相当）： $\sum_{k=1}^K \hat{p}_{jk}(1 - \hat{p}_{jk}) = 1 - \sum_{k=1}^K \hat{p}_{jk}^2$
- 3. エントロピー： $-\sum_{k=1}^K \hat{p}_{jk} \log \hat{p}_{jk}$

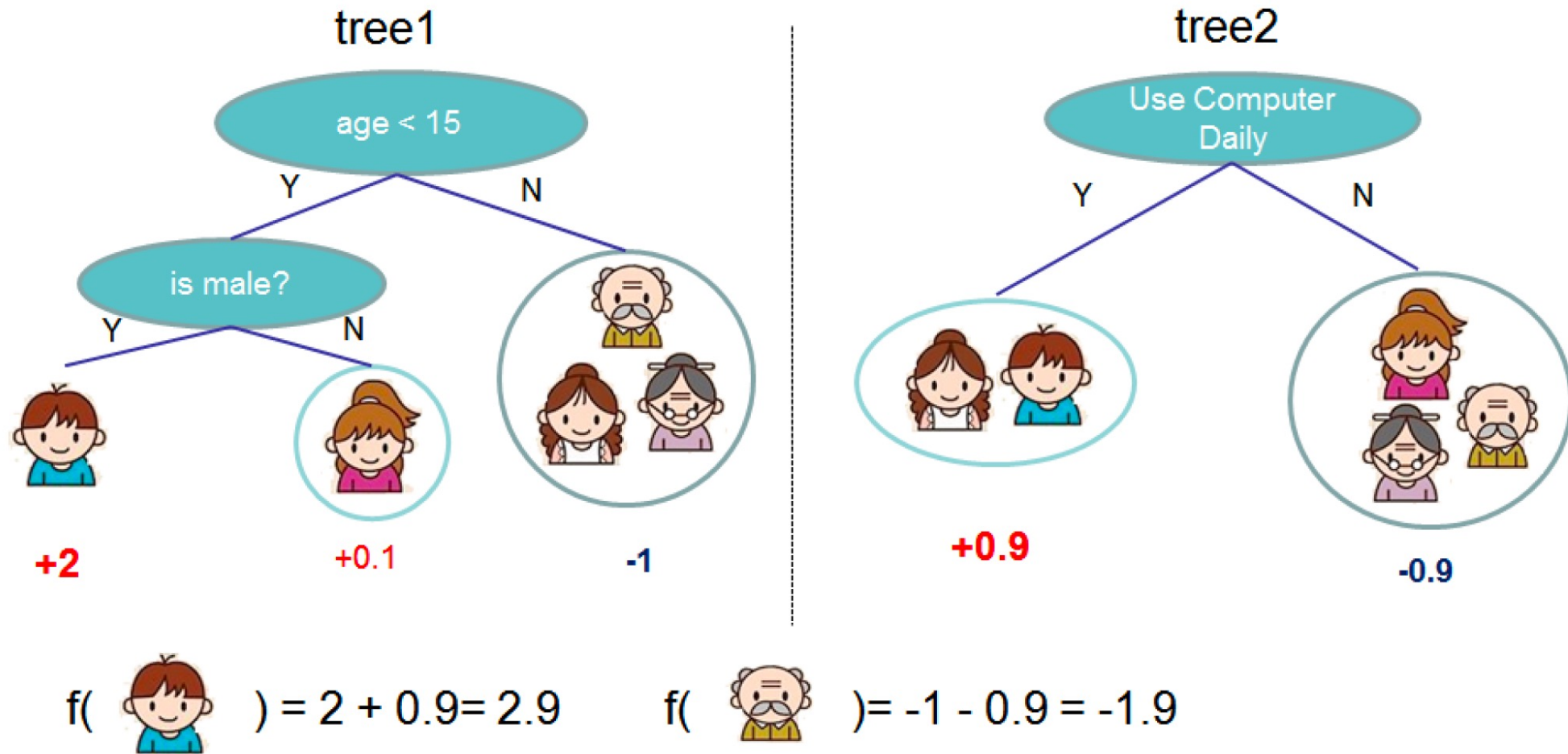
Tree-based methodsの長所と短所

- 人に説明しやすい
- 人間のdecision-makingに近い（？）
- 可視化しやすい（解釈が容易）
- カテゴリカル変数の扱いも簡単（ダミー変数不要）
- 他の回帰手法や分類手法に比べて性能が悪いことがある
- データ集合の少しの変更が木の構造を大きく変えることがある
 - つまり過学習を起こしやすい

短所を克服する3つの方法

1. バギング (bagging)
 2. ランダム・フォレスト (random forests)
 3. ブースティング (boosting)
- いずれも複数の木を使う (アンサンブル学習)

Tree ensemble model



バギング (bagging)

- 例えば、訓練データをランダムに等分してそれぞれで木を作ると、かなり違った木になることが多い
 - こういう現象をhigh varianceと呼ぶ。
- そこで、訓練データからランダムに部分集合をいくつも採って、それぞれで木を作り、これらの出力の平均を予測に使う
 - 部分集合をランダムに選ぶときは、復元抽出をおこなう。
 - 分類の場合は、複数の木の出力で多数決をとり、予測に使う。
 - 元はhigh varianceでも、平均をとるとvarianceを下げられる。
- 補足：バギングは他の分類手法でも使える

ランダム・フォレスト

- バギングとほとんど同じ
- 違うのは、領域を分割するときに、 p 個ある説明変数すべてではなく、ランダムに選んだ m 個の説明変数だけを考慮する点
 - バギングでは、すべての説明変数を考慮する。
 - つまり、 p 通りの分割の可能性のすべてを考える。
 - いくつの説明変数を考慮するかについては、例えば $m \approx \sqrt{p}$ と設定する。
- 一部の説明変数だけを考慮することで、たくさん作る木が互いに似てきてしまうことを防ぐ

ブースティング

- 訓練データに変更を加えつつ、順番にたくさんの木を作る
- 最初は、元の訓練データそのものを使って、木を作る
- 次は、いま作った木による予測の、訓練データからのズレ（残差）を、新たな訓練データと思って、二つ目の木を作る
- その次は、いま作った木による予測の、直前の訓練データからのズレを、あらなた訓練データと思って、三つ目の木を作る
- 以下、この繰り返し

• 入力：訓練データ (\mathbf{X}, \mathbf{y}) ただし $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ 、 $\mathbf{y} = \{y_1, \dots, y_N\}$

1. モデルを $\hat{f}(\mathbf{x}) = 0$ と初期化、また、全ての \mathbf{x}_i について $r_i = y_i$ とする

2. for b in $[1, \dots, B]$:

a. 木 \hat{f}^b に、深さが d になるまで、訓練データ (\mathbf{X}, \mathbf{r}) で学習させる

b. 予測に使うモデル \hat{f} を、以下のように更新 (λ は 0.01 や 0.001 などの値を使う)

$$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda \hat{f}^b(\mathbf{x})$$

c. 残差を、以下のように更新

$$r_i \leftarrow r_i - \lambda \hat{f}^b(\mathbf{x}_i)$$

3. 以下のモデルを出力する

$$\hat{f}(\mathbf{x}) = \sum_{b=1}^B \lambda \hat{f}^b(\mathbf{x})$$

今日の課題

- 有名なデータセット California housing datasetでmedian house valueを予測しよう
- RMSEによって評価される予測性能を、良くして下さい
- test setとそれ以外の部分の分割は、変えないでください
- test set以外の部分をどう使うかは、自由です
 - 交差検証の方法は何でもよい。
- tree-basedな手法なら、何を使ってもいいです
- test setでのRMSEによる評価は最後に一回おこなうだけです