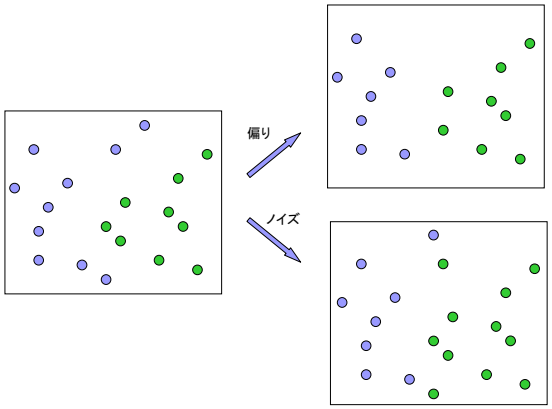
知的情報処理
7. 過学習: すべてを鵜呑みにしてはいけない

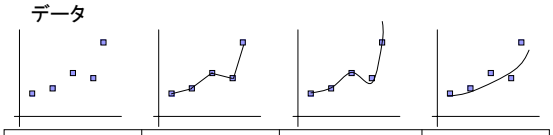
櫻井彰人
慶應義塾大学理工学部

過学習

- over-learning とか over-training と呼ばれる
- 学習すべきでないものまで、学習してしまう
- 学習すべきでないもの
  - 学習データに含まれる偏り
    - 無限集合（真の概念が含む事例は無限個ある）の有限部分集合であるため、かならず、偏りがある。
  - 学習データに含まれる誤り
    - 現実データにはノイズがある。分類クラスにも属性値にもノイズは存在する。
- 学習してしまう
  - 学習能力が高いから
    - 調節可能なパラメータ数が多い



過学習の例（関数近似とノイズ）

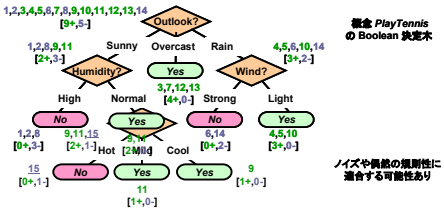


	区分線形	全点を通る4次多項式	2次多項式
パラメータ数	2×4+3=11	5	3+ノイズ

本格的な(?) 関数近似のデモ:
http://www.mste.uiuc.edu/users/exner/java.f/leastsquares/

決定木における過学習: 例

- 既出例: 帰納した木



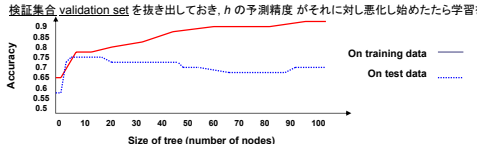
概念 *PlayTennis* の Boolean 決定木

ノイズや偶然の規則性に適合する可能性あり

- 訓練事例にノイズがあると
  - 事例 15: <*Sunny, Hot, Normal, Strong,* ->
    - この例は実は noisy である. すなわち、正しいラベルは +
    - 以前に作成した木は、これを、誤分類する
  - 決定木はどのように更新されるべきか (incremental learning を考える)?
  - 新しい仮説 $h' = T'$ の性能は $h = T$ より*悪く* なると予想される（ノイズに騙されているから！）

決定木学習: 過学習の予防と回避

- 過学習にどう立ち向かうか?
  - 予防策
    - *重要な* 属性を選択 (i.e., 決定木では有用)
    - *重要性* の予測: 属性を filter する, または 部分集合選択
  - 回避策
    - 検証集合 validation set を抜き出しておき, h の予測精度 がそれに対し悪化し始めたたら学習を停止

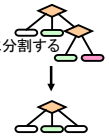


- "最良の" モデル (決定木) の選び方
  - 上述: 性能を測定するにあたって、訓練データとそれとは別の検証データを用いる
  - 別法: 最小記述長 Minimum Description Length (MDL):
    最小化せよ: *size*($h \equiv T$) + *size* (*誤分類 misclassifications* ($h \equiv T$))

決定木学習: 過学習の予防と回避

- 基本的なアプローチが2つある
  - Pre-pruning (回避): *木を作成する途中で木の生長を止める.* 信頼性ある選択をするにたる十分なデータはないと判断されたとき
  - Post-pruning (回復): *木を一杯まで構築し節を削除する.* 削除するのは, 十分な証拠がないとみなされるもの
- 枝刈りすべき部分木を評価する方法
  - Cross-validation: 仮説の有用性を評価するために、予めデータをとりおく (Mitchell 第4章)
  - 統計的検定: 観測された規則性が偶然起こったものとして捨ててよいかどうかをテストする (Mitchell 第5章)
  - 最小記述長 Minimum Description Length (MDL)
    - 仮説 T の複雑度の増加分は, 単に(説明しようとしているデータの)*例外* を記憶するに必要な記述量より大きいか/小さいか?
    - Tradeoff: *モデル* を記述する versus *残余誤差* を記述する

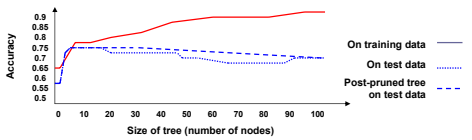
Reduced-Error Pruning



- Post-Pruning, Cross-Validation Approach
- 所与のデータを 訓練データ training set と 検証データ Validation set に分割する
- 関数 *Prune*(*T, node*)
  - 引数 *node* を根節とする部分木を除去
  - 引数 *node* を葉節とする (そこにある事例には多数派のラベルを付与)
- アルゴリズム *Reduced-Error-Pruning* (*D*)
  - *D* を分割する. D_{train} (訓練 training / "growing"), $D_{validation}$ (検証 validation / "pruning")
  - D_{train} に *ID3* を適用して, 完全な木 *T* を作る
  - UNTIL $D_{validation}$ で計測した精度が悪化する DO
    FOR *T* 中のそれぞれの内節 *candidate*
    ***Temp*[*candidate*] ← *Prune* (*T, candidate*)**
    ***Accuracy*[*candidate*] ← *Test* (*Temp*[*candidate*], $D_{validation}$)**
    T ← *T'* ∈ *Temp* 中で *Accuracy* が最良のもの
  - RETURN (pruneしおえた) *T*

Reduced-Error Pruning の効果

- Reduced-Error Pruning によるテスト誤差の減少



  - 節を刈ることによってテスト誤差が減少する
  - *注*: $D_{validation}$ は D_{train} と D_{test} のどちらとも異なる
- 賛成論 と 批判論
  - 賛成: 最も正確な *T'* (*T* の部分木) のうちで最小のものが生成できる
  - 批判: *T* を作るのにわざわざデータ量を減らしている
    - $D_{validation}$ をとりおくだけの余裕があるか?
    - データ量が十分でなければ, 誤差をなおさら大きくする (D_{train} が不十分)

Rule Post-Pruning

- しばしば用いられる方法
  - これもよく知られたoverfitting 対応策
  - *C4.5* でその亜種が用いられた. *C4.5* は *ID3* の派生・後継.
- アルゴリズム *Rule-Post-Pruning* (*D*)
  - *D* から *T* を生成 (*ID3* を使用) – 可能な限り *D* に適合するまで成長させる (過学習も許す)
  - *T* を等価な規則集合に変換 (根節から葉節へ道一つにつき1規則)
  - それぞれの規則を, *独立に*, 条件をどれでも, 推定精度が改善する限り, 除去することにより刈り込む(一般化する)
  - 刈り込んだ規則をソートする
    - 推定精度に従ってソートする
    - 列に並べて, D_{test} に適用する