情報意味論(3) 決定木と過学習

櫻井彰人
慶應義塾大学理工学部

決定木

- 多くの方には復習ですね。ご容赦を。

applet によるデモ

- applet によるデモがあります

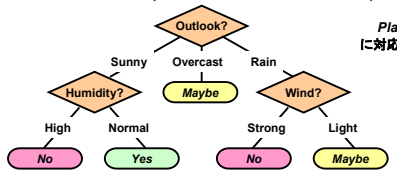
http://www.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html

http://www.cs.ubc.ca/labs/lci/CIspace/Version4/dTree/

- 後者の簡単な説明を最後に付加しておきます

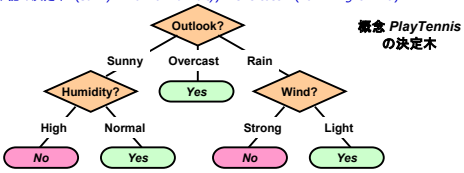
決定木 Decision Trees

- 分類器 Classifiers である
  - 事例 (ラベルのついていないもの): 属性 attribute (または特徴 feature) のベクトル
- 内節 Internal Nodes: 属性値のテストを行う
  - 典型的: 等しいかどうかのテスト (e.g., "Wind = ?")
  - その他 不等式や様々なテストが可能
- 枝 Branches: 枝を選ぶ条件である属性値 (テストが等式以外のときはテストの結果)
  - 一対一対応 (e.g., "Wind = Strong", "Wind = Light")
- 葉 Leaves: 割当てた分類結果 (分類クラスのラベル Class Labels)



決定木はブール関数

- 決定木はブール関数
  - 表現力: 任意のブール関数 (リテラルは属性変数のテスト)が表現可能
  - なぜ?
    - 決定木のあらわす論理関数は、Disjunctive Normal Form (DNF) でかける
    - 下記の決定木: (*Sunny* ∧ *Normal-Humidity*) ∨ *Overcast* ∨ (*Rain* ∧ *Light-Wind*)



- 概念を表現するブール関数の例
  - ∧, ∨, ⊕ (XOR)
  - (*A* ∧ *B*) ∨ (*C* ∧ ¬*D* ∧ *E*)
  - *m*-of-*n*

他の例: 帝王切開のリスク予測

- 1000 人の女性患者記録から学習
- 負例が帝王切開に対応
  - Prior distribution: [833+, 167-] 0.83+, 0.17-
  - *Fetal-Presentation* = 1: [822+, 167-] 0.88+, 0.12-
    - *Previous-C-Section* = 0: [767+, 81-] 0.90+, 0.10-
      - *Primiparous* = 0: [399+, 13-] 0.97+, 0.03-
      - *Primiparous* = 1: [368+, 68-] 0.84+, 0.16-
        - *Fetal-Distress* = 0: [334+, 47-] 0.88+, 0.12-
          - ***Birth-Weight*** **< 3349** **0.95+, 0.05-**
          - ***Birth-Weight*** **≥ 3347** **0.78+, 0.22-**
        - *Fetal-Distress* = 1: [34+, 21-] 0.62+, 0.38-
    - *Previous-C-Section* = 1: [55+, 35-] 0.61+, 0.39-
  - *Fetal-Presentation* = 2: [3+, 29-] 0.11+, 0.89-
  - *Fetal-Presentation* = 3: [8+, 22-] 0.27+, 0.73-

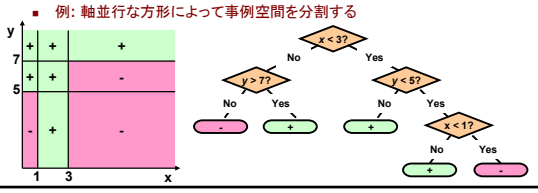
C-section: 帝王切開, Fetal_Presentation: 胎児の胎位,
Primiparous: 初産

どんな時、決定木を用いるか

- 事例が属性 - 属性値ペアで表現される
- 目標関数が離散値をとる（分類問題）
- 選言を含む仮説が必要
  - 属性値に関する連言であれば、「概念学習」で可能
- ノイズが入っている可能性がある
- 例（実際に Mitchell が適用した）
  - 機器故障診断、病名診断
  - 与信リスクの分析
    - クレジットカード, ローン
    - 保険
    - 消費者による不正行為
    - 従業員の不正行為

決定木と判別境界

- 事例は, 多くの場合, 離散属性値で表現される
  - 勿論、連続値も扱う拡張がある
  - 典型的な型
    - 名義・名辞 nominal ({red, yellow, green})
    - 離散化・量子化 quantized ({low, medium, high})
  - 数値の取り扱い
    - 離散化 discretization, ベクトル量子化 vector quantization: 閾値を用いて分割する
      ex. U. M. Fayyad and K. B. Irani, Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, Proc. 13th IJCAI (1993).
- 例: 軸並行な方形によって事例空間を分割する

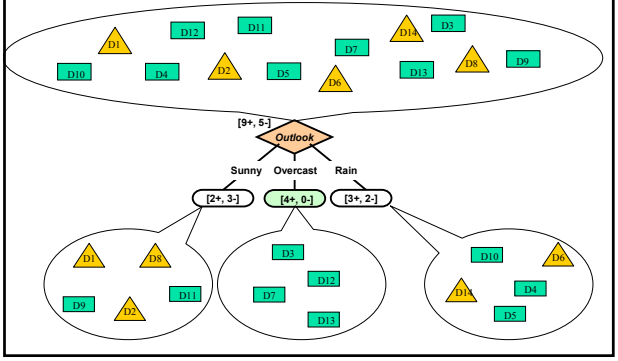


決定木の学習: トップダウン帰納 (*ID3*)

- アルゴリズム *Build-DT* (*Examples*, *Attributes*) //部分木に再帰的に適用される
  - Examples: 事例の部分集合、Attributes: 属性の部分集合

  IF *Examples* の *label* が同一 THEN RETURN (その *label* を付した葉節)
  ELSE
  IF *Attributes* が空集合 THEN RETURN (*多数派 label* を付した葉節)
  ELSE
  最良属性 *A* を根節として選ぶ. 以下で作る木を子とする木を作り、値とする.
  FOR *A* のそれぞれの値 *v*
  条件 *A* = *v* に対応した, 根節からの枝を作成する
  IF {*x* ∈ *Examples* | *x.A* = *v* } = Ø THEN *多数派 label* を付した葉節 を作成
  ELSE *Build-DT* ({*x* ∈ *Examples* | *x.A* = *v* }, Attributes – {A})
- どの属性が最良か?

決定木の学習: 例

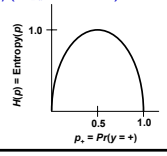


“最良”の属性の選択

- 目的
  - できるだけ小さい決定木を作る (オッカムの剃刀)
  - 条件: 訓練データの分類・ラベルにできるだけ適合する（consistentである）べし
    - 枝分かれ後、分かれた事例が一個の分類・ラベルになるのがベスト
- 障害
  - *最小の* consistent な仮説 (i.e., 決定木) を見出すことは NP-hard
  - 先ほどの *Build-DT* (再帰的なアルゴリズム)は
    - 単純な木を作る グリーディ（greedy）な発見的な（heuristic）探索 であって、
    - 最適性の保証はできない
- そこで:
  - 要請: できるだけ同一のラベルをもつ集合に、事例を分割するような属性を選べ
  - その結果: 葉節(ラベルが同一)に近くなる（はず）
  - これは、もっともよく使われるヒューリスティックである
    - もともと J. R. Quinlan が提案したものであり、
    - 情報増分 information gain に基づき、
    - *ID3* アルゴリズムで使用されている

エントロピー: 直感的説明

- 不確実さ・不明瞭さの尺度; 不確実なほど大きくなる値
  - 計る対象
    - 純粋さ purity: 事例集合が、ただ一つのラベルをもつ状態に、どれだけ近いか
    - 不純さ impurity (乱雑さ disorder): ラベルがまったく分からない状態にどれだけ近いか
  - 尺度: エントロピー
    - 比例する対象: 不純さ impurity, 不確実さ uncertainty, 不規則さ irregularity, 驚き surprise
    - 反比例する対象: 純粋さ purity, 確かさ certainty, 規則性 regularity, 冗長さ redundancy
- 例
  - 簡単のため, H = {0, 1}, ある分布 *Pr*(*y*) に従うと仮定
    - (2個より多い)離散的なクラスラベルでも同様
    - さらに連続確率変数でもよい: 微分エントロピー differential entropy (和を積分にしただけ)
  - *y* に関して最も純粋: 次のいずれかの場合
    - *Pr*(*y* = 0) = 1, *Pr*(*y* = 1) = 0
    - *Pr*(*y* = 1) = 1, *Pr*(*y* = 0) = 0
  - 純粋さが最も少ない確率分布は?
    - *Pr*(*y* = 0) = 0.5, *Pr*(*y* = 1) = 0.5
    - 最大: 不純さ/不確実さ/不規則性/驚き
    - エントロピーの性質: 凹関数 (“上向きに凸”)



エントロピー: 情報理論的定義

- 考察に関わる要素
  - D: 事例の集合 $\{<x_1, c(x_1)>, <x_2, c(x_2)>, \ldots, <x_m, c(x_m)>\}$
  - $p_+ = Pr(c(x) = +)$, $p_- = Pr(c(x) = -)$
- 定義
  - H は確率密度関数 p 上で定義する
  - D の事例に対して, その + と – ラベルの頻度を p_+ と p_- で表す
  - D の c に対するエントロピー は:

$$H(D) \equiv -p_+ \log_b(p_+) - p_- \log_b(p_-)$$

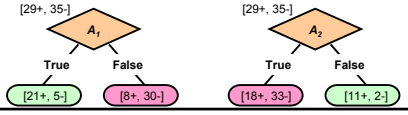
- 単位は?
  - 対数の底による ($b = 2$ なら bits, $b = e$ なら nats, 等.)
  - ビットは、最悪の場合 ($p_+ = 0.5$) の一事例を符号化するのに必要とされる
  - 不確実さが小さければ (e.g., $p_+ = 0.8$), 1ビットより小さくて十分

情報量増分: 情報理論的定義

- 属性値に基づく分割
  - 復習: D の分割 partition は, 和集合が D となるような排他的部分集合の集合
  - 目標: 属性 A の属性値に基づく分割により削減される不確実性・不純性を計る
- 定義
  - 属性 A に関する D の情報量増分 は, A を用いた分割によるエントロピー減少分の期待値:

$$Gain(D, A) \equiv H(D) - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \cdot H(D_v) \right] = \frac{1}{|D|} \left(|D| \cdot H(D) - \sum_{v \in values(A)} |D_v| \cdot H(D_v) \right)$$

    但し D_v は $\{x \in D \mid x.A = v\}$, すなわち, D 中の事例で属性 A の値が v であるものの集合
  - 補足: A による分割によって生じる部分集合 D_v の大きさに従ってエントロピーの大きさを調整
    - エントロピー値は、「集合の要素一個あたり」の情報量となっているため
- どちらの属性を使うのがいい?



例

- 概念 *PlayTennis* 用の訓練事例

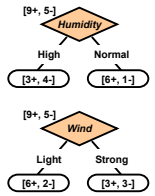
Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

- *ID3* ≡ *Build-DT* 但し *Gain*(•) を使用
- *ID3* の動きを追ってみよう

ID3 による *PlayTennis* 決定木作成 [1]

- 根節の属性を選ぶ

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No



- 事前 (無条件) 分布: 9+, 5-
  - $H(D)$ = -(9/14) log (9/14) - (5/14) log (5/14) *bits* = 0.94 bits
  - $H(D, Humidity$ = High) = -(3/7) log (3/7) - (4/7) log (4/7) = 0.985 bits
  - $H(D, Humidity$ = Normal) = -(6/7) log (6/7) - (1/7) log (1/7) = 0.592 bits
  - $Gain(D, Humidity)$ = 0.94 – ((7/14) * 0.985 + (7/14) * 0.592) = 0.151 bits
  - 同様に, $Gain$ (D, *Wind*) = 0.94 – ((8/14) * 0.811 + (6/14) * 1.0) = 0.048 bits

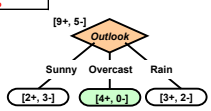
$$Gain(D, A) \equiv H(D) - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \bullet H(D_v) \right]$$

ID3 による *PlayTennis* 決定木作成 [2]

- 根節の属性を選ぶ

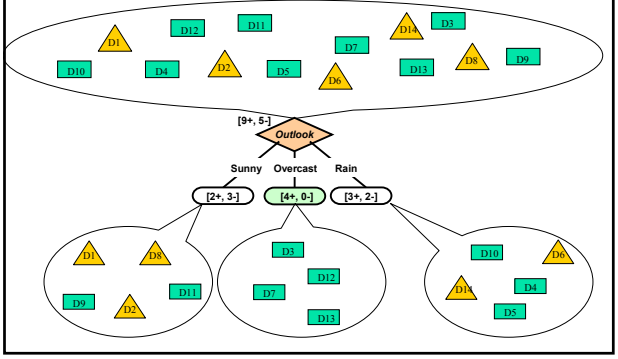
Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

  - $Gain(D, Humidity)$ = 0.151 bits
  - $Gain(D, Wind)$ = 0.048 bits
  - $Gain(D, Temperature)$ = 0.029 bits
  - $Gain(D, Outlook)$ = 0.246 bits



- 次の属性を選ぶ (部分木の根節)
  - (葉への道の上で)属性を使いきるか葉の純粋度 = 100% になるまで続ける
  - 純粋度 = 100% は、一つのラベルしかないということ
  - ところで $Gain(D, A) < 0$ となりうるか?

再掲: 決定木の学習: 例



ID3 による *PlayTennis* 決定木作成 [3]

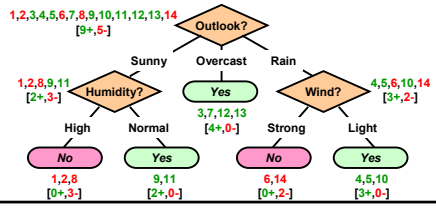
- 次の属性の選択 (部分木の根節)

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

  - 約束: 0 log (0/a) = 0
  - *Gain*(D_{Sunny}, *Humidity*) = 0.97 - (3/5) * 0 - (2/5) * 0 = 0.97 bits
  - *Gain*(D_{Sunny}, *Wind*) = 0.97 - (2/5) * 1 - (3/5) * 0.92 = 0.02 bits
  - *Gain*(D_{Sunny}, *Temperature*) = 0.57 bits
- トップダウン再帰
  - 離散値属性しかないなら, O(*n*) 回分割をすれば終了 (*n* は属性数)
  - 木のレベルそれぞれで、訓練データを一回スキャン (なぜ?)

ID3 による *PlayTennis* 決定木作成 [4]

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

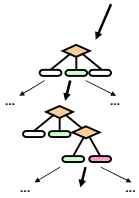


適用範囲を広げるには

- これまでのアルゴリズムでの仮定. その克服
  - 離散 *出力*
    - 実数値出力も可能
    - Regression trees [Breiman *et al*, 1984]
  - 離散 *入力*
    - 量子化の方法あり
    - 内節の等式テストの変わりに*不等式* を使用する (以前の方形の例)
- 規模の拡大
  - 大規模データベース (VLDB) からの知識発見やデータマイニング (KDD) では重要
  - 長所: 多くの *事例* を対象とするよいアルゴリズムあり
  - 弱点: あまりに多い*属性* を扱うのは難しい
- あると助かる他の耐性
  - *ノイズ*のあるデータ (分類*ノイズ* classification noise = ラベルの間違い; 属性*ノイズ* attribute noise = 不正確または低精度のデータ) への耐性
  - 欠測値への耐性

ID3 による仮説空間探索

- 探索問題
  - 探索の対象は *決定木全部の空間*, すなわちブール関数をすべて表現可能な空間
    - Pros: 表現力; 柔軟性
    - Cons: 計算量; 巨大, 意味の分からない木も含む
  - 目的: もっともよい決定木を見出す (最小な consistent な木)
  - 障害: この木を見出す問題は NP-hard
  - Tradeoff
    - heuristics の使用(探索の案内役としての目の子)
    - 貪欲 greedy アルゴリズムの使用
    - すなわち、バックトラックなしの山登り hill-climbing (gradient "descent")
- 統計的学習
  - 事例の部分集合 D_t の統計的な量 p_+, p_- に基づく決定
  - ID3 では, 全てのデータを使用
  - ノイズのあるデータに対してロバスト

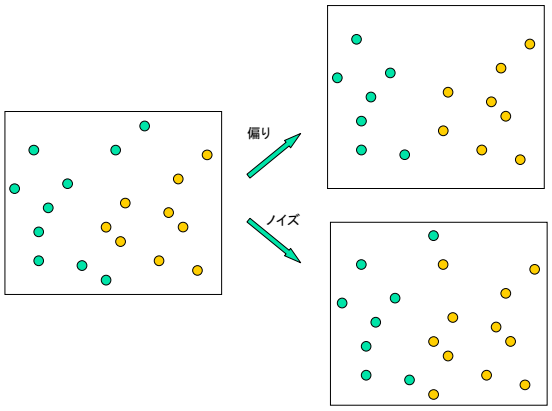


ID3 の帰納バイアス

- 探索におけるヒューリスティックは帰納バイアスである
  - *H* は *X* の冪集合 (全部分集合の集合)
  - ⇒ 帰納バイアスなしと言ってよいか? いや、そうではない…
    - 短い木への選好 (終了条件から) がある
    - 情報量増分が高い属性を根節に近いところにおくという選好がある
    - *Gain*(•): *ID3* の帰納バイアスを体現するヒューリスティック関数
  - *ID3* の帰納バイアス
    - ある仮説への選好をヒューリスティック関数に表現している
    - 比較してみる: 仮説空間 *H* を制限すること(命題論理の正規形に基づく制限: *k*-CNF, etc.)
- 短い木を好むこと
  - データに適合する木の中で最短のものを選ぶ
  - オッカムの剃刀バイアス: 観測を説明する最短の仮説

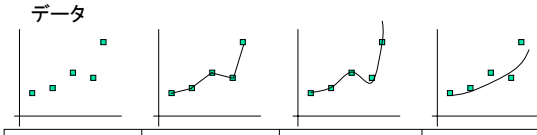
過学習

- 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



再掲: 関数近似の例(ノイズ)

データ



	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

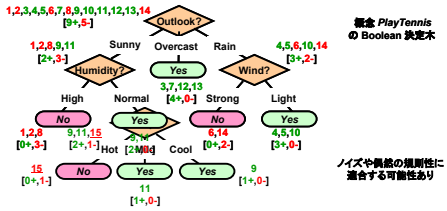
関数近似の applet

- 分かりきったことかもしれませんが、デモプログラムを用いて実験してみると、よく分かります。

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



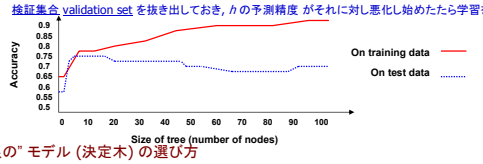
- 訓練事例にノイズがあると
  - 事例 15: *<Sunny, Hot, Normal, Strong, ->*
    - この例は実は noisy である. すなわち、正しいラベルは +
    - 以前に作成した木は、これを、誤分類する
  - 決定木はどのように更新されるべきか (incremental learning を考える)?
  - 新しい仮説 *h' = T'* の性能は *h = T* より*悪く* なると予想される(ノイズに騙されているから!)

帰納学習における過学習

- 定義
  - 仮説 *h* が訓練データ集合 *D* を過学習する (~にoverfits する) というのは、もし他の仮説 *h'* で $error_D(h) < error_D(h')$ であるが $error_{test}(h) > error_{test}(h')$ となるものがあること
  - 原因: 訓練事例が少なすぎる (あまりにも少ないデータに基づく決断); ノイズ; 単なる偶然
- 過学習に対応するには?
  - 予防策
    - 過学習が発生する前に対応する
    - *重要な relevant* 属性(i.e., モデルにとって有用そうなもの)のみを用いる
      - *注意*: 鶏と卵の問題; 重要性 relevance を予測する尺度が必要
  - 回避策
    - 問題が起こりそうなときに、脇をすりぬける
    - テスト集合を確保しておき, 仮説 *h* がその上で悪くなりそうなときに、学習を停止する
  - 泳がせ策
    - 問題は発生するにまかせ, 発生を検出し, その後回復する
    - モデルを作ってみて, 過学習に寄与する要素を発見・除去する (刈る prune)

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

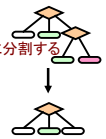


- "最良の" モデル (決定木) の選び方
  - 上述: 性能を測定するにあたって、訓練データとそれとは別の検証データを用いる
  - 別法: 最小記述長 Minimum Description Length (MDL):
    最小化せよ: *size*(*h* = *T*) + *size* (*誤分類 misclassifications* (*h* = *T*))

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

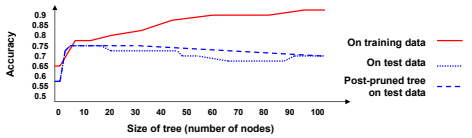
Reduced-Error Pruning



- 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



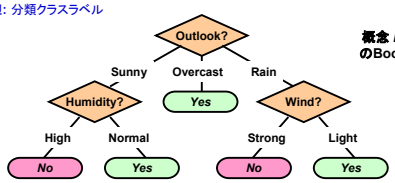
  - 節を刈ることによってテスト誤差が減少する
  - *注*: $D_{validation}$ は D_{train} と D_{test} のどちらとも異なる
- 賛成論 と 批判論
  - 賛成: 最も正確な *T* (*T* の部分木) のうちで最小のものが生成できる
  - 批判: *T* を作るのにわざわざデータ量を減らしている
    - $D_{validation}$ をとりおくだけの余裕があるか?
    - データ量が十分でなければ, 誤差をなおさら大きくする (D_{train} が不十分)

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

決定木を規則に変換する

- 規則の構文
  - 左辺: 条件 (属性の等式テスト上の連言標準形 conjunctive formula)
  - 右辺: 分類クラスラベル

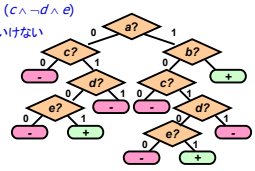


概念 *PlayTennis* のBoolean決定木

- 例
  - IF (*Outlook* = *Sunny*) ∧ (*Humidity* = *High*) THEN *PlayTennis* = *No*
  - IF (*Outlook* = *Sunny*) ∧ (*Humidity* = *Normal*) THEN *PlayTennis* = *Yes*
  - …

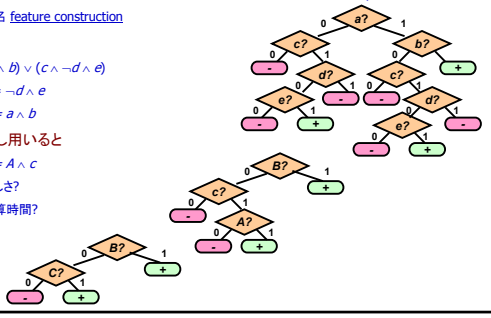
決定木における重複

- 決定木: 表現上の短所
  - 決定木は, 一番簡単な表現というわけではない
  - ポイント: 属性を重複 replication させる必要がある場合がある
- 属性重複の例
  - e.g., Disjunctive Normal Form (DNF): $(a \wedge b) \vee (c \wedge \neg d \wedge e)$
  - (どちらかの) 連言は部分木として重複させないといけない
- 部分解
  - *新しい属性* を作る
  - 別名 constructive induction (CI)
  - Mitchell の第10章参照



少しだけ: 決定木の構成的帰納法

- 新しい属性の合成
  - 一つの "+ 節" に到る直前の二つの属性の連言から新しい属性を合成 synthesize する
  - 別名 feature construction
- 例
  - $(a \wedge b) \vee (c \wedge \neg d \wedge e)$
  - $A = \neg d \wedge e$
  - $B = a \wedge b$
- 繰り返し用いると
  - $C = A \wedge c$
  - 正しさ?
  - 計算時間?



決定木: 他の話題

- 他の機械学習に共通する課題

連続値属性

- 連続値属性を扱う2つの方法
  - 離散化
    - 実数値属性を, 予め, いくつかの範囲に分ける
    - e.g., {*high* = *Temp* > 35° C, *med* = 10° C < *Temp* ≤ 35° C, *low* = *Temp* ≤ 10° C}
  - 内節を分けるのに, 閾値を用いる
    - e.g., $A \le a$ によって二つの部分集合 $A \le a$ と $A > a$ ができる
    - この離散化に際して, 情報増分が同様に計算される
- 情報増分を最大にする分割はどうやって得るか?
  - FOR 連続値属性 A のそれぞれ
    事例 $\{x \in D\}$ を $x.A$ に従って, 分割する
    FOR 異なったラベルを持つ A の値の順序対 (l, u) それぞれ
    閾値の候補として, 中点 mid-point の情報量増分を評価, i.e., $D_{A \le (l+u)/2}$, $D_{A > (l+u)/2}$
  - 例

A = *Length*:	10	15	21	28	32	40	50
Class:	-	+	+	-	+	+	-
閾値のチェック: *Length* ≤ 12.5?	≤ 24.5?	≤ 30?	≤ 45?				

多値属性に伴う問題

- 問題
  - もしある属性が多値であると, *Gain*(•) はそれを選びやすい (なぜ?)
  - 例えば、日付 (2007/11/01等) を属性として用いることを想像してみればわかる!
- 一つのアプローチ: *GainRatio* を *Gain* の代わりに使用

$$Gain(D,A) \equiv H(D) - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \bullet H(D_v) \right]$$

$$GainRatio(D,A) \equiv \frac{Gain(D,A)}{SplitInformation(D,A)}$$

$$SplitInformation(D,A) \equiv - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \log \frac{|D_v|}{|D|} \right]$$

  - *SplitInformation*: $c = |\ values(A)\ |$ に, ほぼ, 比例
  - i.e., 多くの値をもつ属性にハンディを負わせる
    - e.g., 仮定: $c_1 = c_{Date} = n$ そして $c_2 = 2$
    - *SplitInformation* $(A_1) = \log(n)$, *SplitInformation* $(A_2) = 1$
    - もし *Gain*(D, A_1) = *Gain*(D, A_2) とすると, *GainRatio* (D, A_1) << *GainRatio* (D, A_2)
  - すなわち, *GainRatio*(•) を用いれば, (分岐数が少ない方への)選択バイアスが 表現できる

補足: Gini index

- もう一つの分割の指標
  - n は分類･クラスの個数
  - Gini(D) は D 内の分布が偏れば偏るほど、すなわち、pure になるほど小さくなる

$$Gini(D) = \sum_{i \neq j} p_i p_j = 1 - \sum_{i=1}^{n} p_i^2$$

$$GiniGain(D,A) = Gini(D) - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \cdot Gini(D_v) \right]$$

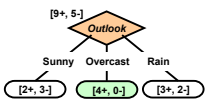
コスト付き属性

- 応用分野毎
  - 医療: *体温検査* のコストは 1000円; *血液検査* 1500円; *生検* 50000円
    - また検査の侵襲性･無侵襲性も考慮する必要あり
    - 患者へのリスクも (e.g.,羊水検査)
  - 他のコスト
    - サンプリング時間: e.g., ロボットのソナー (レンジファインダー, etc.)
    - 人工物, 生体へのリスク (どんな情報を収集するか)
    - 関連する分野 (e.g., 断層装置): 非破壊検査
- 低い期待コストでいかに consistent な木を作るか?
  - 一つのアプローチ: 情報増分 gain を コスト正規化増分 *Cost-Normalized-Gain* で置き換える
  - 正規化関数の例
    - [Nunez, 1988]:
      $$Cost\text{-}Normalized\text{-}Gain(D,A) \equiv \frac{Gain^2(D,A)}{Cost(D,A)}$$
    - [Tan and Schlimmer, 1990]:
      $$Cost\text{-}Normalized\text{-}Gain(D,A) \equiv \frac{2^{Gain(D,A)} - 1}{(Cost(D,A)+1)^w} \quad w \in [0,1]$$
      但し w はコストの重要性を定める

欠測値: 属性値が不明

- 問題: 属性 *A* の値がない事例があるとどうなるか?
  - しばしば, 訓練時やテスト時に, 必ずしも全ての属性値が入手できるとは限らない
  - 例: 医療診断
    - <*Fever* = *true*, *Blood-Pressure* = *normal*, …, *Blood-Test* = *?*, …>
    - 値は、本当になかったり, またあっても信頼度が低かったりする
  - 欠測値: 訓練時 versus 分類時
    - 訓練時:ある $x \in D$ について *A* の値が与えられていないとき *Gain* (*D*, *A*) を評価する
    - 分類時: *A* の値を知らずに, 新しい事例 *x* を分類する
- 解: *Gain*(*D*, *A*) の計算の中に推測を入れる

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No



欠測値: 対応策

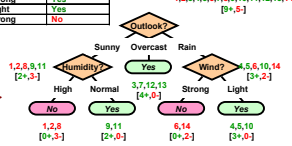
- 訓練事例はとにかく使用する. 木を(根節から)辿りつつ作っていくとき
  - 考慮すべき属性のどれについても, 事例中でもし値が知られていないなら、それを推測する
  - その推測は、今いる節に割当てられた事例の知られている値に基づく
- *x*.*A* の最もありそうな値を推測する
  - 第一案: 節 *n* で属性 *A* をテストするなら, *n* を通る事例の *A* の値でもっとも多いものを用いる
  - 第二案 [Mingers, 1989]:節 *n* で属性 *A* をテストするなら, *n* を通る事例で *x* と同じクラスラベルをもつものの *A* の値でもっとも多いものを用いる
- 推測値を分散させる
  - 両賭け: 値の分布に従い,推測値を分散させる
  - *x*.*A* の可能な値 v_i の分布に比例して確率 p_i を割当てる[Quinlan, 1993]
    - 木の子孫に, *x* の内の p_i 分を割当てる. データ数に 3.7個などという値が出現する
    - これを用いて *Gain* (*D*, *A*) or *Cost-Normalized-Gain* (*D*, *A*) を計算する
- どのアプローチにおいても, 新事例も同様に分類する

欠測値: 例

- *x*.*A* の最もありそうな値を予測する
  - 第一案: *Humidity* = *Normal*
  - 第二案: *Humidity* = *High* (*No* 事例はすべて *High*)
  - (最も Gain の大きなものはどうだろうか? *High*: *Gain* = 0.97, *Normal*: *Gain* < 0.97)

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

- 確率で重み付けする
  - *0.5 High, 0.5 Normal*
  - *Gain* < 0.97
- テスト事例: <*?, Hot, Normal, Strong*>
  - 5/14 *Yes* + 4/14 *Yes* + 5/14 *No* = *Yes*

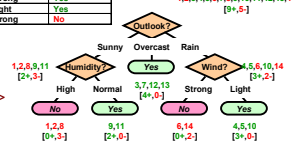


欠測値: 例

- *x*.*A* の最もありそうな値を予測する
  - 第一案: *Humidity* = *Normal*
  - 第二案: *Humidity* = *High* (*No* 事例はすべて *High*)
  - (最も Gain の大きなものはどうだろうか? *High*: *Gain* = 0.97, *Normal*: *Gain* < 0.97)

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis?*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

- 確率で重み付けする
  - *0.5 High, 0.5 Normal*
  - *Gain* < 0.97
- テスト事例: <*?, Hot, Normal, Strong*>
  - 1/3 *Yes* + 1/3 *Yes* + 1/3 *No* = *Yes*
  - 5/14 *Yes* + 4/14 *Yes* + 5/14 *No* = *Yes*



学習とバイアス

- バイアス: 仮説選択に順位があるとき、その順位
  - 同時に複数個の仮説をみたときの、選好順位
  - 一度に一個ずつ見るときの、探索順序
- データに適合する仮説は、一般に、多量にあるので、学習にはバイアスが伴う
  - 仮説を一個選択するのではなく、複数個の仮説を用いる場合でも、「データに適合する仮説をすべて用いる」のではない限り、バイアスが必要である。

オッカムの剃刀: ある選好バイアス

- 帰納バイアス2つ: 選好バイアス preference biases と言語バイアス language biases
  - 選好バイアス
    - *学習アルゴリズム*に(普通は暗黙的に)組み込まれている
    - 言い換えれば: 探索順序の規定
  - 言語バイアス
    - *知識 (仮説) の表現*に(普通は暗黙的に)組み込まれている
    - 言い換えれば: 探索空間の制限
    - *別名* 制限バイアス
- オッカムの剃刀 Occam's Razor: 賛成意見
  - 短い仮説の方が、長い仮説に比べ、個数が少ない
    - 例えば, ビット列で考えれば, 長さ *n* のものは $n+1$ のものに比べ半数, $n \geq 0$.
    - 短い仮説が、もしデータにぴったり合ったとしたら、偶然とは考え難い
      - 短い仮説は、個数が少ないので、説明できる現象の数が少ない
    - 長い仮説 (例: 200 個の節を持つ木,かつ $|D| = 100$) の場合には、偶然である可能性が高い
      - いずれかの木がデータにぴったり合う。どれに合うかは偶然であるが、どれかに合うこと自体は当然。
  - 得るものと捨てたもの
    - 他の条件が同一であれば, 複雑なモデルの汎化能力は単純なモデルほどではない
    - あとになってもっと柔軟な(微調整可能な)モデルが必要になることはないと仮定

オッカムの剃刀と決定木: 二つの問題

- オッカムの剃刀 Occam's Razor: 反対意見
  - 仮説空間 H に依存して $size(h)$ が決まる。同じ h でも H が異なると $size(h)$ が異なる。
  - 「小さき」を選好することへの疑問: "少ない" ことは正当化にならない
- オッカムの剃刀 Occam's Razor は Well-Defined か?
  - 内部の知識表現 knowledge representation によってどの h が "短い" かがきまる --- 恣意的?
    - 例えば, テスト "($Sunny \wedge Normal\text{-}Humidity$) $\vee$ $Overcast$ $\vee$ ($Rain \wedge Light\text{-}Wind$)" は一個?
  - 答: 表現言語を固定; 十分長いところでは、長い仮説は、内部表現によらず、やっぱり長い
    - 反論:答えになっていない。実際には「短い仮説」に関する議論が重要
- 「短い仮説」であって、どうして他の「小さい仮説空間」ではないのか?
  - 小さい仮説集合を定義する方法はいろいろとある.
  - 選好バイアスで用いる size が何であっても, 適当に基準 S を選べば $size(h)$ をその限界内に制限することができる (i.e., "S に合致する木のみ受理する")
    - e.g., 節の個数が素数であって, 文字 "Z" で始まる属性を用いている木
    - なぜ「小さな木であって, (例えば) A_1, A_1, …, A_{11} を順番にテストするもの」ではないのか?
    - $size(h)$ に基づいて小さな仮説集合を定義することに、特別の意味があるのか?
  - 参考: Chapter 6, Mitchell's Machine Learning

繰り返しですが、、、

Induction（帰納）

- OED (Oxford English Dictionary) によれば
  - the process of inferring a general law or printciple from the observations of particular instances
  - これは、inductive inference のこととする
  - inductive reasoning は: the process of reassigning a probability (or credibility) to a law or proposition from the observation of particular events

エピクロスの多説明原理

- ギリシャの哲学者 Epicurus
  - If more than one theory is consistent with the observations, keep all theories (Principle of Multiple Explanations).
- その一つの理由: 一つを他から選び出す理由がない

Occam の剃刀

- 人口に膾炙しているのは
  - Entities should not be multiplied beyond necessity.
- Bertrand Russell によれば
  - It is vain to do with more what can be done with fewer.
- 最も普通の解釈
  - Among the theories that are consistent with the observed phenomena, one should select the simplest theory.

Isaac Newton の言葉

- We are to admit no more causes of natural things than such as are both true and sufficient to explain the appearances. To this purpose the philosophers say that Nature does nothing in vain, and more is in vain when less will serve; for Nature is pleased with simplicity, and affects not the pomp of superfluous causes.

注目：残余誤差と複雑さの二律背反

- 観測値には測定誤差がある
- 残余誤差 0 となる理論は複雑過ぎる
  - 丸暗記（⇒役に立たない）に相当
- 簡単過ぎる理論は残余誤差が多い
  - 過剰な一般化：すぐに「皆が持っている」
- 理論の複雑さと残余誤差を両立させればよい
- でも、どうやって？

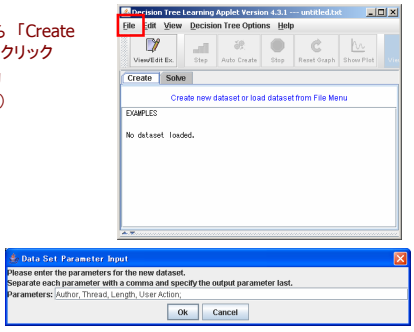
過学習とバイアス

- 見かけ上関係がない概念ではある。
- しかし、一般に
  - 仮説の複雑度が上がれば、過学習しやすい
  - 仮説の複雑度は、探索が先に延びるに従い、上がる
    （学習バイアスは、一般に、単純なものをまず調べるように書く。もっとも、計算可能な学習バイアスであれば、殆どの仮説については、単純なもの優先になる）
- すなわち、「あるバイアスに沿って、仮説を調べていき、行き過ぎないようにするか、行き過ぎたら戻る」という（普通の）手法が、この両者をつなげている

dTree の使い方（一部）

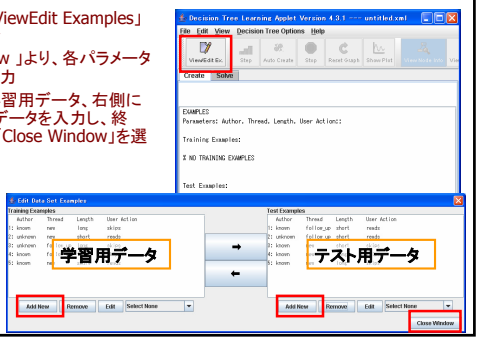
データセットの作り方①

- 左上の「File」から 「Create New Dataset」をクリック
- パラメータを入力
  （カンマで区切る）
- 「OK」をクリック



データセットの作り方②

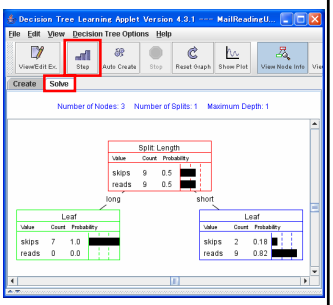
- 左上の「ViewEdit Examples」をクリック
- 「Add New 」より、各パラメータの値を入力
- 左側に学習用データ、右側にテスト用データを入力し、終わったら「Close Window」を選択



決定木の作成①

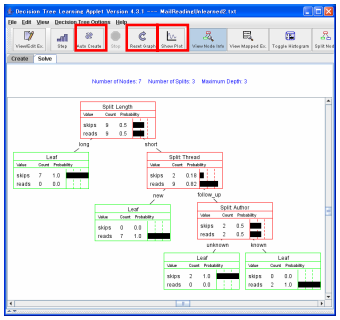
- 左上の「Solve」を選択
- 「Step」を選択すると、一段階ずつ木が作成される
- 各ノードの上で左クリックをすると、ノードの情報などを見ることができる

赤：根ノード
青：葉ノード（さらに分割可能）
緑：葉ノード（これ以上分割不可）



決定木の作成②

- 「Auto Create」を選択すると、最後まで木を生成する
- 「Reset Graph」を選択すると、木を作る前の状態に戻る
- 「Show Plot」を選択すると、error rate をグラフで見ることができる



テストデータへの適用

- 「Test」を選択すると、生成した決定木をテストデータへ適用した結果が表示される
- 「Test New Example」を選択すると、各要素を変更した際、結果がどう変わるかがわかる