知的情報処理
6. 簡単便利な決定木: 作るのは少々難しい

櫻井彰人
慶應義塾大学理工学部

今日の目標

- 決定木の作り方を理解する
  - 構築には、greedy アルゴリズム
    - ノードに置く属性の選択: 情報量増分
      - 「増分比」が必要な局面がある。
    - 復習: 情報量について
  - 回帰もできる(回帰木)
  - R では、tree, rpart を試す

目次

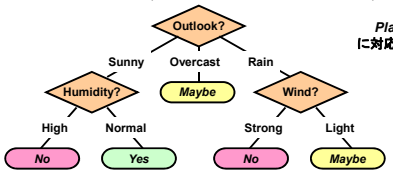
- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

決定木とは

- 決定木は、他の学習器とかなり異なる
  - 境界は、綺麗な関数ではかけない
  - 何を目標としているか(何が目標関数か)明確ではない
  - アルゴリズムは、OR的なものとはかけ離れている
- けれども、最終ユーザに非常に分かりやすく、誤差もそこそこに小さいため、重要なツールである
- 単純かつ有効なだけに、様々な工夫がされてきている。
  - あなどってはいけない。
- 出来上がった決定木は理解しやすい(すぐ分かる。だから使われる)。しかし、作るのは、結構、難しい。
  - 最近流行りの random forest の要素であるが、これだけ集まると(森になると)何をやっているのかわからない。

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)



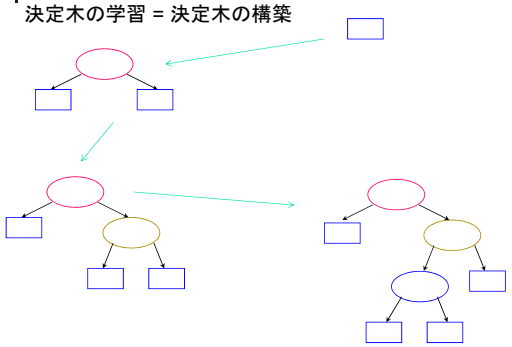
PlayTennis に対応する決定木

目次

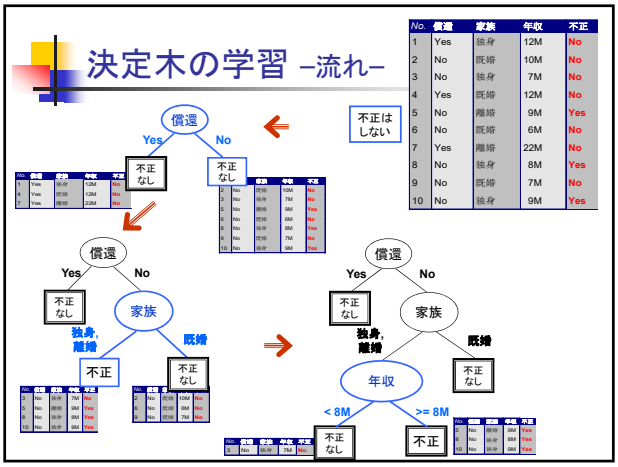
- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

決定木の学習 –流れ–

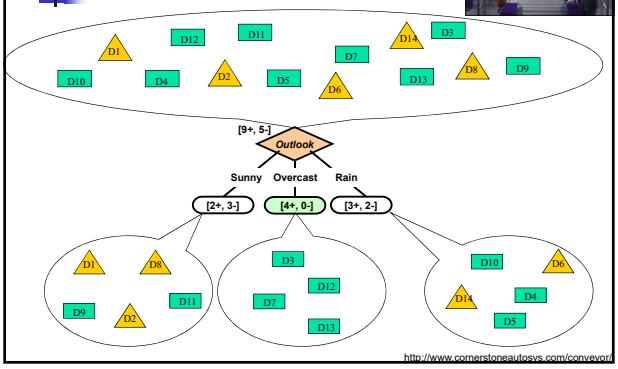
決定木の学習 = 決定木の構築



決定木の学習 –流れ–



決定木の学習 –1ノード–



目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

決定木の作成 –グリーディ方略–

- グリーディな方略.
  - 一般: いったん決めたら、心変わりしない
    - 迷路を進むときに、後戻りしない。一度掴んだら離さない。
    - 最適ではないが、後戻りしない分、速い。
  - 決定木: 訓練データを、ある評価基準を最適化するように、ある属性で分割する.
    - 一度分割してある枝を作ったら、それを取りやめることはない

目次代わり

- 課題
  - 訓練データの分割方法を決定する
    - 属性テスト方法をどう定めるか？
    - 最良の分割をどう決めるか？
  - (分割の)止め時の決め方

決定木の作成 –グリーディ方略–

- グリーディな方略.
  - 一般: いったん決めたら、心変わりしない
    - 迷路を進むときに、後戻りしない。一度掴んだら離さない。
    - 最適ではないが、後戻りしない分、速い。
  - 決定木: 訓練データを、ある評価基準を最適化するように、ある属性で分割する.
    - 一度分割してある枝を作ったら、それを取りやめることはない

目次代わり

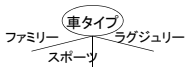
- 課題
  - 訓練データの分割方法を決定する
    - 属性テスト方法をどう定めるか？
    - 最良の分割をどう決めるか？
  - (分割の)止め時の決め方

ノード上の属性の決め方

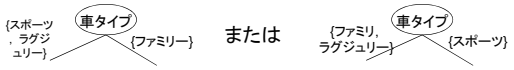
- 属性タイプによって異なる
  - 名義変数
  - 順序変数
  - 数値変数
- いくつに分割するかによって異なる
  - 2分割
  - 多分割

名義変数による分割

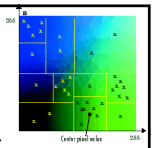
- 多分割: 当該変数の変数値の「異なり数分」、分割する.



- 2分割: 変数値を2個に分割する.
  最適な分割を求める必要あり.



数値変数に基づく分割



- 身長、体重、血圧、コレステロール値、、、
  - 1単位ずつ分けると分けすぎ。
  - 離散化: いくつかの境(閾値ともいう)を設けて、いくつかに分ける。
- いくつかの方法がある
  - 離散化 して順序属性として扱う
    - 静的 – 最初に一回だけ離散化
    - 動的 – 等幅区間、等頻度区間(パーセンタイル)、クラスタリング
  - 2値判別: (A < v) または (A ≥ v)
    - すべての可能な分割を考え、ベストなものを見出す
    - 計算が一層必要となることも

決定木の作成 –グリーディ方略–

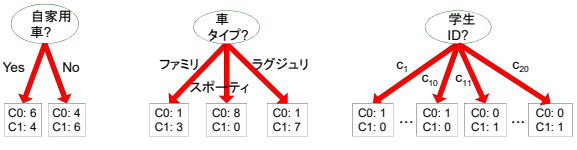
- グリーディな方略.
  - 一般: いったん決めたら、心変わりしない
    - 迷路を進むときに、後戻りしない。一度掴んだら離さない。
    - 最適ではないが、後戻りしない分、速い。
  - 決定木: 訓練データを、ある評価基準を最適化するように、ある属性で分割する.
    - 一度分割してある枝を作ったら、それを取りやめることはない

目次代わり

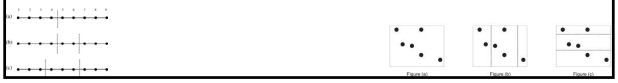
- 課題
  - 訓練データの分割方法を決定する
    - 属性テスト方法をどう定めるか？
    - 最良の分割をどう決めるか？
  - (分割の)止め時の決め方

最良な分割はどうやって見つける？

分割前: C0 (クラス0)に 10 データ,
C1 (クラス1)に 10 データ



どの条件が最適か?

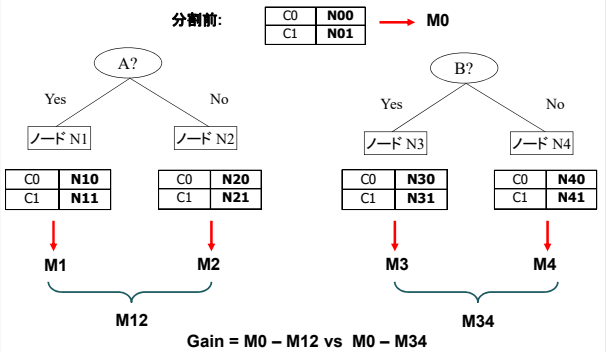


最良な分割はどうやって見つける？

- グリーディ方略:
  - 新ノード内のクラス分布が 同質となる 分割がベター
  - どこかのクラスが圧倒的な多数となる(これが同質)ということは、それだ！といっても間違いが少ない
- (ノードの)同質さの物差しが必要:

C0: 5 C1: 5	C0: 9 C1: 1
非-同質, 純度が低い **不純度が高い**	**同質, 純度が高い** **不純度が低い**

最良な分割はどうやって見つける？



Gain = M0 – M12 vs M0 – M34

M0 = N00とN01のエントロピー

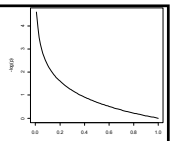
不純度のものさし

- エントロピー
- ジニ・インデックス Gini Index
- 誤分類率

目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

エントロピー



- 平均情報量とも呼ばれる。式で書くと

$$H(p_1,...,p_m) = -p_1 \log_2 p_1 - ... - p_m \log_2 p_m$$
$$= p_1(-\log_2 p_1) + ... + p_m(-\log_2 p_m)$$

- （比較のために）サイコロの出る目の平均

$$+ p_1 * 1 + p_2 * 2 + ... + p_6 * 6$$

- つまり、平均情報量が情報量の平均だとすると $-\log_2 p_i$ が「情報量」ということになる

負の符号「–」がついているのは、p<1 故 log p <0 となるが、負の数はいろいろと不便なため、符号反転しているから OR ½ のべき乗を考えるから（2のべき乗ではない）

情報量

ある事象の情報量は、
その事象が起こったということを（他の皆が知らないときに）知ることの価値

「事象」として「コインの表が出ること」（確率1/2））としよう。
「表が出たこと」を知る価値を a としよう。
「コイン1が表」「コイン2が表」という2つの情報を知る価値は a + a = 2a だろう（一つずつ、2回、聞く場合を考えればよい）。
「コイン1が表」「コイン2が表」の二つの事象が起こる確率は ½ * ½ =1/4 。

「事象」として「サイコロの1が出ること」（確率1/6））としよう。
「1が出たこと」を知る価値を b としよう。
「サイコロ1が1」「サイコロ2が1」という2つの情報を知る価値は b + b = 2b だろう（一つずつ、2回、聞く場合を考えればよい）。
「コイン1が表」「コイン2が表」の二つの事象が起こる確率は 1/6 * 1/6 =1/36 。
つまり、事象が起こる確率が2乗になると、価値は2倍になる

情報量を表す関数

事象が起こる確率が2乗になると、価値は2倍になる

事象が起こる確率 p が p^2 になると、価値 v は 2v になる

予想屋を想像して下さい。一度予想して正解して報酬を得る。再び予想して正解して報酬を得る。2回とも正解の確率は積、報酬は和でしょう？

事象が起こる確率 p が p^2 になると、価値 v(p) は $v(p^2) = 2v(p)$ になる

上記のような関数は log しかないことが示せる（底は決まらない。報酬による）

そこで、底を2とし価値が正になるように符合反転すると（底を1/2 にしたのと同じ）、生起確率 p の事象が生起したことを知るという情報の価値は、-log p とすればよいことが分る。

$$情報量(p) = -\log_2 p = \log_{1/2} p$$

不公平かもしれないコイン

- 表が出る確率 p, 裏が出る確率が 1-p であるコインのコイン投げを考える。
- このコインを1回投げたときに出た「表・裏」を知る情報の価値はどのくらいであろうか？
- 「表が出る」という情報の価値は、-log p,「裏が出る」という情報の価値は、-log(1-p)である。
- 表が出る確率は p, 裏が出る確率は 1-p であるので、この確率に基づく（情報価値の）平均値を考えよう

$$H(p,1-p) = p(-\log_2 p)+(1-p)(-\log_2(1-p))$$
$$= -p\log_2 p-(1-p)\log_2(1-p)$$

不公平かもしれないサイコロ

- 「目iが出る」確率 p_i であるサイコロを考える。
- このサイコロを1回投げたときに出た目を知る情報の価値はどのくらいであろうか？
- 「目iが出る」という情報の価値は、-log p_iである。
- この確率に基づく（情報価値の）平均値を考えよう

$$H(p_1,p_2,\ldots,p_6)$$
$$= p_1(-\log_2 p_1)+p_2(-\log_2 p_2)+\cdots+p_6(-\log_2 p_6)$$
$$= -p_1\log_2 p_1-p_2\log_2 p_2-\cdots-p_6\log_2 p_6$$

目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

不純度を測る物差しとしての情報量

- 不純度
  - 2種類の個体が混在している場合を考える。割合をpと1-pとする。p=0またはp=1のときは最も純粋であるので、このとき0、p=1/2のとき最も純度が低いので、このとき1になるような関数があるとよい。明らかにエントロピーがその性質を満たす。
  - 一般にn種類の個体が混在している場合はどうだろうか。割合を$p_1,\ldots p_n$ とする。p_i のいずれかが1で他が0というときが最も純度が高い。逆にp_i のすべてが等しいとき（1/nの時）最も純度が低い。明らかにエントロピーはこの性質をもつ。そこで、

$$\textit{non-Information}(D) \equiv H(D) = \sum_{c \in classes(D)} \left[-\frac{|D_c|}{|D|} \cdot \log \frac{|D_c|}{|D|} \right]$$

  - 補足: 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

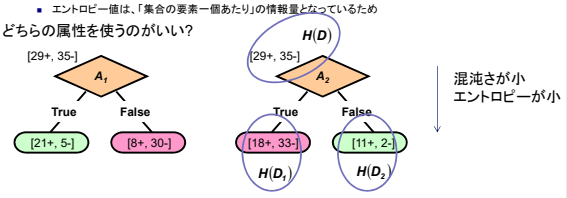
情報量増分

要は、逆を考えて、
下るに従い、
「混沌さ」が減ればよい
「クリア」になればよい

- 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:

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

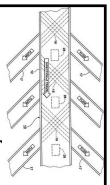
  - 但し D_v は $\{x \in D \mid x.(A) = v\}$, すなわち、$D$ 中の事例で属性 A の値が v であるものの集合
  - 補足: A による分割によって生じる部分集合 D_v の大きさに従ってエントロピーの大きさを調整
    - エントロピー値は、「集合の要素一個あたり」の情報量となっているため
- どちらの属性を使うのがいい？



目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

あらためて: 決定木の構築



- 通常の手順: 上から下に(根から葉へ)そしてgreedy. 再帰的かつ分割統治(divide-and-conquer)
  - まずは: 一つの属性を選び根とする。属性値ごとに枝を作る
  - 次は: 訓練データを部分集合に分割 (枝一本につき一個)
  - 最後に: 同じ手順を、個々の枝について行う。その場合、個々の枝に割り当てられた訓練データのみを用いる(全体は用いない)
- ノードに(それへの枝に)割り当てられた訓練データがすべて同じクラスになったら、終了

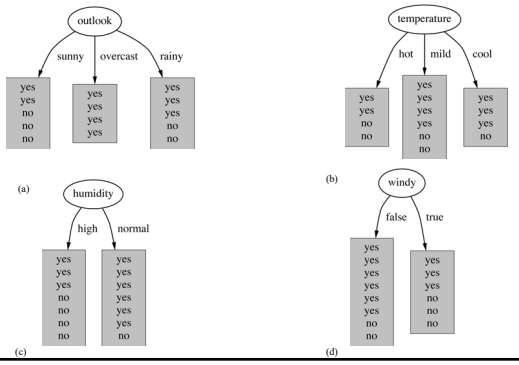
http://www.freepatentsonline.com/7086519.html

テニスをするや否や

Outlook	Temp.	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Tom Mitchell "Machine Learning" の例題. よく使われる

どの属性がいいのか?

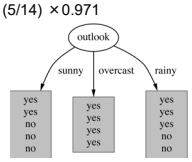


計算例: 属性 "Outlook"

"Outlook" = "Sunny":
info([2,3]) = entropy(2/5,3/5) = – (2/5)log(2/5) – (3/5)log(3/5) = 0.971
"Outlook" = "Overcast":
info([4,0]) = entropy(1,0) = – 1 log(1) – 0 log(0) = 0
"Outlook" = "Rainy":
info([3,2]) = entropy(3/5/,2/5) = – (3/5)log(3/5) – (2/5)log(2/5) = 0.971

この属性を用いたときの情報量は
info([3,2],[4,0],[3,2]) = (5/14) × 0.971 + (4/14) × 0 + (5/14) × 0.971
= 0.693 bits

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



http://www.space.com/4700-weather-outlook-improves-thursday-shuttle-launch.html

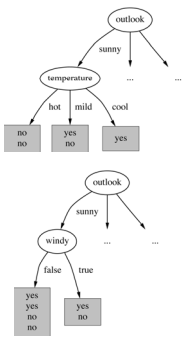
計算例: 情報量増分

- ただし、通常は、ノードのエントロピーを直接用いることはない。情報量増分を用いる.

情報量増分: 分割前の情報量 – 分割後の情報量
gain("Outlook") = info([9,5]) – info([2,3],[4,0],[3,2]) = 0.940 – 0.693
= 0.247 bits
同様に計算すると
gain("Outlook") = 0.247
gain("Temperature") = 0.029
gain("Humidity") = 0.152
gain("Windy") = 0.048

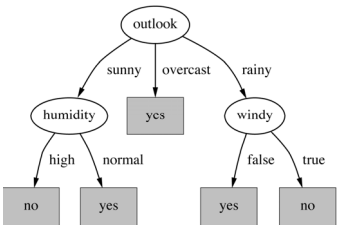
- 情報量増分が多いほど、純度が高い。従って、"Outlook" を選ぶことにする。

分割を続ける



$$gain("Temperature") = 0.571 bits$$
$$gain("Humidity") = 0.971 bits$$
$$gain("Windy") = 0.020 bits$$

最終的に得られる決定木



- 注: すべての葉が“純”である必要はない; というのも、同じデータなのにクラスが違うことがあるから(ノイズのせい)

⇒ データがそれ以上分割しない方がよくなったら、やめ

http://www.mochadad.com/2011/01/the-importance-of-family-goal-setting

目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI
  - 回帰木
  - Rにおいては

ちょっと問題が

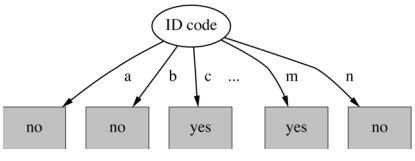
- 属性値の多い属性(例えば、ID)があると、おかしくなる
  - 属性値の多い属性が選ばれてしまう
  - 選んでみると、おかしい！

枝数が非常に多くなる属性があると、、、

- IDコードをつけてみよう

ID code	Outlook	Temp.	Humidity	Windy	Play
A	Sunny	Hot	High	False	No
B	Sunny	Hot	High	True	No
C	Overcast	Hot	High	False	Yes
D	Rainy	Mild	High	False	Yes
E	Rainy	Cool	Normal	False	Yes
F	Rainy	Cool	Normal	True	No
G	Overcast	Cool	Normal	True	Yes
H	Sunny	Mild	High	False	No
I	Sunny	Cool	Normal	False	Yes
J	Rainy	Mild	Normal	False	Yes
K	Sunny	Mild	Normal	True	Yes
L	Overcast	Mild	High	True	Yes
M	Overcast	Hot	Normal	False	Yes
N	Rainy	Mild	High	True	No

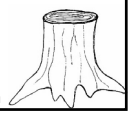
IDコードを根にもってくると、“切株”



この分割のエントロピー

info(“IDcode”) = info([0,1]) + info([0,1]) + … + info([0,1]) = 0 bits

⇒ 情報量増分は最大となる(すなわち、0.940 bits)



http://www.extension.umn.edu/yardandgarden/ygbriefs/h446removetree.html

枝分かれの多い属性

従って,

- 属性値が多いと、訓練データの部分集合は“純”になりやすい
  - 情報量増分は、属性値の多い属性の方にバイアスしている
  - この結果、過学習 overfitting (過去のデータの学習という意味では素晴らしいが、予測のためには最適でない属性を選んでしまう)になってしまう。

一つの解決法：増分比

- 増分比 Gain ratio: 情報量増分のもつバイアスを減少させる
- 増分比は、枝の本数とそれに割り当てられる訓練データの大きさの両方を勘定に入れる
  - 情報量増分の修正は、訓練データの集合をどのような（大きさと要素数の）部分集合に分割するかという分割の情報量を用いて、行われる

増分比の計算例

計算例: IDコードの分割情報量（split information）
info([1,1,…,1]) = 14 × (- (1/14) log(1/14)) = 3.807 bits

増分比の定義
gain_ratio(“Attribute”) = gain(“Attribute”) / split_info(“Attribute”)
計算例:
gain_ratio(“IDcode”) = 0.940 bits / 3.807 bits = 0.246

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

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

$$= H\left(\frac{|D_1|}{|D|}, \frac{|D_2|}{|D|}, \ldots, \frac{|D_{|A|}|}{|D|} \right)$$

他の属性に関する増分比

Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.362
Gain ratio: 0.247/1.577	0.156	Gain ratio: 0.029/1.362	0.021
Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

```
ID
Info:           0
Gain:           0.940
Split info:     3.807
Gain ratio:     0.246
```

解決した？解決しない？

- “Outlook” がトップであるが、今度は “Humidity” が肉薄している。というのも、“Humidity” は2個に分割するため、増分比が相対的に良くなるためである.
- 見ればわかるように: “ID code” の増分比が最大！. もっともそのアドバンテージは大分と減少したが.
- 増分比の問題点: 過補償となるおそれがあること
  - 分割情報量が小さいために、不適当な属性が選ばれる可能性
  - よくある修理方法: 増分比が最大のものを選ぶのだが、当該属性の情報量増分は、少なくとも、情報量増分の平均値（全属性で考えて）はあるものという条件を課す.

補足

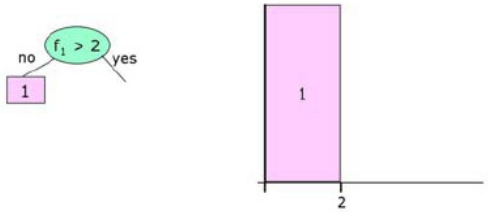
- 決定木のトップダウン（根から葉へ）アルゴリズム(“ID3”)は、Ross Quinlan (University of Sydney Australia) が開発
- 増分比は、このアルゴリズムの基本的な改良の一つ
  - これに引き続き開発されたのが C4.5。数値属性、欠測値、ノイズのあるデータが扱える
- 属性選択には他の方法がたくさんある! (といっても、結果の精度にはあまり違いがない)

目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI index
  - 回帰木
  - Rにおいては

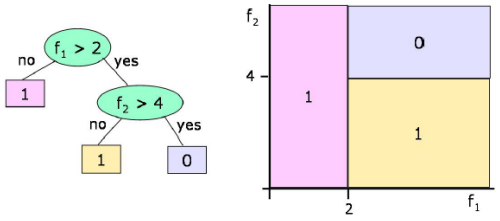
数値属性

- 属性テストは次の形をとる x_j > ある定数
- 属性値のなす空間を短冊に分割する



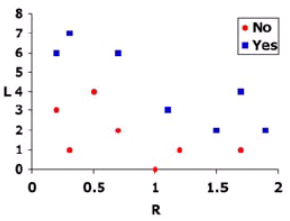
数値属性

- 勿論、これでもいい x_j > ある定数
- 短冊への分割は同じ



破産の予測

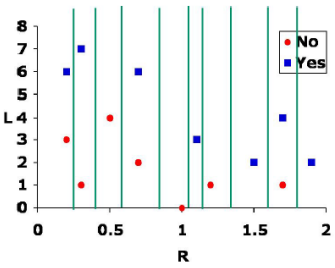
L	R	B
3	0.2	No
1	0.3	No
4	0.5	No
2	0.7	No
0	1.0	No
1	1.2	No
1	1.7	No
6	0.2	Yes
7	0.3	Yes
6	0.7	Yes
3	1.1	Yes
2	1.5	Yes
4	1.7	Yes
2	1.9	Yes



L: 一年あたりの支払い遅延回数
R: 支出/ 収入
B: 破産

分割を考えよう

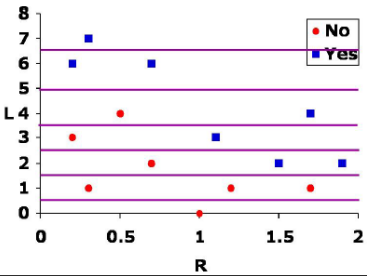
- 各属性ごとに、分割することを考えよう



- 今回の例では、R軸 に沿っての分割の仕方は、高々 9 方法ある
  - 一般に, 訓練データが m 個あれば、m – 1 方法ありそう
  - しかし今回の場合は、R 軸の値が同じデータがあるので、その分、減った.

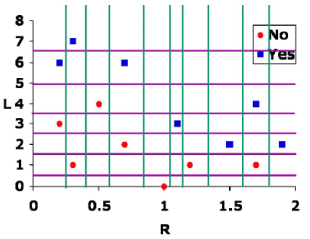
分割その II

- L 軸では高々 6 方法ある
  - L 軸は整数値をとるので、値が重複するデータは多い.



分割によるエントロピーを計算

境界	下方にあるNoの個数	下方にあるYesの個数	上方にあるNoの個数	上方にあるYesの個数	エントロピー
6.5	7	6	0	1	0.93
5.0	7	4	0	3	0.74
3.5	6	3	1	4	0.85
2.5	5	2	2	5	0.86
1.5	4	0	3	7	0.63
0.5	1	0	6	7	0.93

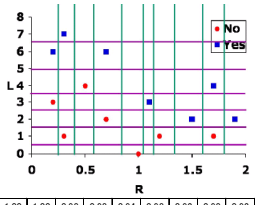


エントロピー	1.00	1.00	0.98	0.98	0.94	0.98	0.92	0.98	0.92
境界	0.25	0.40	0.60	0.85	1.05	1.15	1.35	1.60	1.80

承前

- それぞれの軸でのすべての可能性を考え、分割した場合のエントロピーを計算した

境界	下方にあるNoの個数	下方にあるYesの個数	上方にあるNoの個数	上方にあるYesの個数	エントロピー
6.5	7	6	0	1	0.93
5.0	7	4	0	3	0.74
3.5	6	3	1	4	0.85
2.5	5	2	2	5	0.86
1.5	4	0	3	7	0.63
0.5	1	0	6	7	0.93



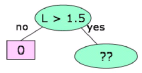
エントロピー	1.00	1.00	0.98	0.98	0.94	0.98	0.92	0.98	0.92
境界	0.25	0.40	0.60	0.85	1.05	1.15	1.35	1.60	1.80

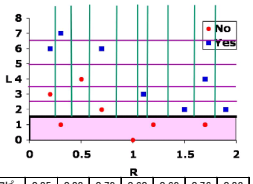
- たまたま、L 軸で、境界を 1.5 とした場合、片側が No だけになることがわかった（エントロピーも最小）

承前

- 残りの空間のすべての分割を考える.
- エントロピーは再計算が必要. すでに葉に割り当てられた訓練データは取り除いて考えなければならないから.

境界	下方にあるNoの個数	下方にあるYesの個数	上方にあるNoの個数	上方にあるYesの個数	エントロピー
6.5	3	6	0	1	0.93
5.0	3	4	0	3	0.74
3.5	2	3	1	4	0.85
2.5	1	2	2	5	0.86



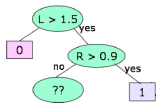


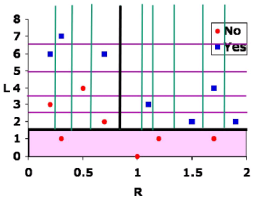
エントロピー	0.85	0.88	0.79	0.60	0.69	0.76	0.83
境界	0.25	0.40	0.60	0.90	1.30	1.60	1.80

承前

- 今度の最適な分割は R > 0.9 である. しかも、すべて Yes であるので、葉を作ることができる.

境界	下方にあるNoの個数	下方にあるYesの個数	上方にあるNoの個数	上方にあるYesの個数	エントロピー
6.5	3	6	0	1	0.93
5.0	3	4	0	3	0.74
3.5	2	3	1	4	0.85
2.5	1	2	2	5	0.86

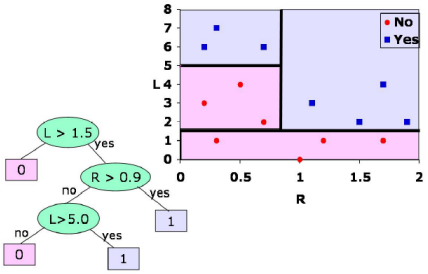




エントロピー	0.85	0.88	0.79	0.60	0.69	0.76	0.83
境界	0.25	0.40	0.60	0.90	1.30	1.60	1.80

承前

- これを続ければ次のものが得られる:



目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI index
  - 回帰木
  - Rにおいては

GINI に基づく分割基準

- これまで説明してきた分割基準はエントロピーであった:

$$Entropy(t) = -\sum_j p(j|t)\log p(j|t)$$

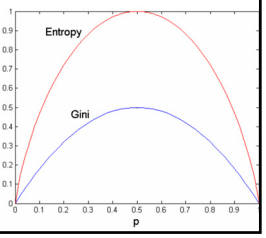
(注: $p(j|t)$ はノード t におけるクラス j データの相対頻度).

- 別法に GINI インデックスを用いるものがある:

$$GINI(t) = 1 - \sum_j [p(j|t)]^2$$

- 両者とも:
  - 最大値 ($\log n_c$ または $1 - 1/n_c$) が得られるのは、当該データがどのクラスにも等分に分配されているときである.「等分である」ということは何の面白さもない. しかし、この状態で、ずっと「実はこれ！」と教わり続けることは結構価値のあることである。
  - 最小値 (0.0) に近い値が得られるのは、ほとんどすべてのデータが同一のクラスに属するとき. 少数派が発生する場合は、非常に面白い。けれども、たいていは多数派が発生するので、まったく面白くない。

2個のクラスに分ける場合:

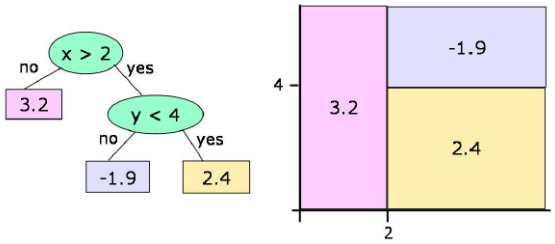


目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI index
  - 回帰木
  - Rにおいては

回帰木 Regression Trees

- 決定木と同じ, 但し 葉において、実数値定数を出力する.



葉における値

- 今いる葉ノードには複数個のデータがあると仮定しよう。なおかつ、何らかの理由により、このノードはこれ以上分割しないものとする.
- 離散値の場合（これまでの場合）, 葉における値（出力値）は、その葉における多数派の値としていた.
- 数値属性の場合, 妥当な値は平均値であろう.
- 従って, もし葉ノードにおける出力値として平均値を用いるならば, （これからノードを分割して子供が葉ノードになろうというときには）枝分かれして作られる新たな葉ノードにおいて、データのもつ値が、当該葉ノード内の値の平均値よりあんまり離れていない方がよかろう.
- 統計学には、数値の集合がどのくらい分散しているかを表す尺度がある
  - (言い換えれば, 個々の数値が平均値からどれだけ離れているか);
  - ご存じの分散である.

目次

- 決定木
  - 決定木とは
  - 決定木の学習 – 流れ、1ノード、Greedy方略 –
  - ノード上の属性の決め方 –名義属性、数値属性–
  - 復習: エントロピー
  - 不純度を測る物差しとしての情報量、情報量増分
  - あらためて、決定木の構築
  - ちょっと問題が –枝分かれの多い属性–
  - 数値属性
  - GINI index
  - 回帰木
  - Rにおいては

Rにおける決定木

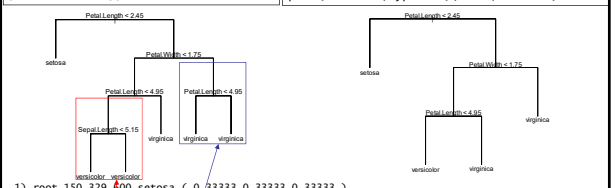
- R には、決定木関連のパッケージとして、tree、rpart、及び rpart を多変量回帰木 (multivariate regression trees)に拡張させた mvpart がある。

分類木の例（tree）

library(tree)

```
data(iris)
(iris.tr<-tree(Species~.,data=iris))
plot(iris.tr,type="u"); text(iris.tr)
```

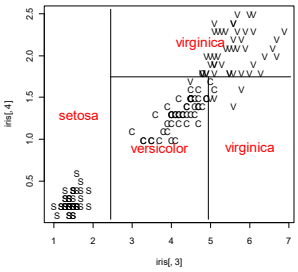
```
(iris.tr1<-snip.tree(iris.tr,nodes=c(12,7)))
plot(iris.tr1,type="u");text(iris.tr1)
```




```
1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
  2) Petal.Length < 2.45 50   0.000 setosa ( 1.00000 0.00000 0.00000 ) *
  3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
    6) Petal.Width < 1.75 54  33.320 versicolor ( 0.00000 0.90741 0.09259 )
      12) Petal.Length < 4.95 48   9.721 versicolor ( 0.00000 0.97917 0.02083 )
        24) Sepal.Length < 5.15 5   5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
        25) Sepal.Length > 5.15 43   0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
      13) Petal.Length > 4.95 6   7.638 virginica ( 0.00000 0.33333 0.66667 ) *
    7) Petal.Width > 1.75 46   9.635 virginica ( 0.00000 0.02174 0.97826 )
      14) Petal.Length < 4.95 6   5.407 virginica ( 0.00000 0.16667 0.83333 ) *
      15) Petal.Length > 4.95 40   0.000 virginica ( 0.00000 0.00000 1.00000 ) *
```

分類木の例（tree）

```
library(tree)
(iris.tr1<-snip.tree(iris.tr,nodes=c(12,7)))
iris.label<-c("S", "C", "V")[iris[, 5]]
plot(iris[,3],iris[,4],type="n")
text(iris[,3],iris[,4],labels=iris.label)
partition.tree(iris.tr1,add=T,col=2,cex=1.5)
```

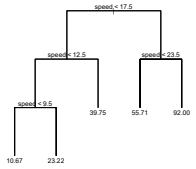


この図は分かりやすくてよいのだが、
rpartには用意されていない。

回帰木の例（tree）

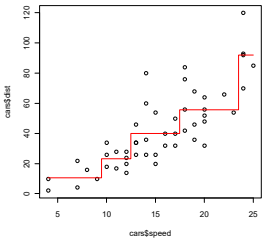
```
> Library(tree)
> data(cars)
> cars.tr<-tree(dist~speed,data=cars)
> print(cars.tr)
node), split, n, deviance, yval
      * denotes terminal node

1) root 50 32540.0 42.98
  2) speed < 17.5 31  8307.0 29.32
    4) speed < 12.5 15  1176.0 18.20
      8) speed < 9.5 6   277.3 10.67 *
      9) speed > 9.5 9   331.6 23.22 *
    5) speed > 12.5 16  3535.0 39.75 *
  3) speed > 17.5 19  9016.0 65.26
    6) speed < 23.5 14  2847.0 55.71 *
    7) speed > 23.5 5  1318.0 92.00 *
> plot(cars.tr,type="u")
> text(cars.tr)
> plot(cars.tr,type="u")
> text(cars.tr)
>
```

```
Library(tree)
data(cars)
cars.tr<-tree(dist~speed,data=cars)
print(cars.tr)
plot(cars.tr,type="u")
text(cars.tr)
plot(cars.tr,type="u")
text(cars.tr)
```



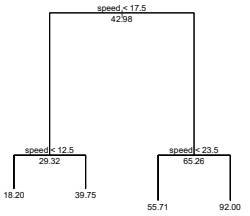
回帰木の例（tree）

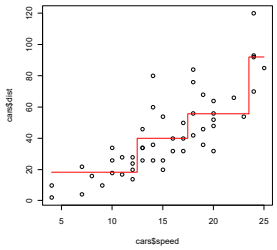
```
> plot(cars$speed,cars$dist)
> partition.tree(cars.tr,add=T,col=2)
>
```



回帰木の例（tree）

```
(cars.tr1<-prune.tree(cars.tr,best=4))
plot(cars.tr1); text(cars.tr1,all=T)

plot(cars$speed,cars$dist)
partition.tree(cars.tr1,add=T,col=2)
```





では、別のデータで

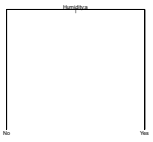
- 例によって、テニスのデータを用いてみよう
- このデータの特徴は、すべての属性が離散値であること

Outlook	Temp.	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

説明が必要です。なお、漢字もOK

library(tree) としたあと

```
> setwd("D:/R/Sample")
> playTennis <- read.csv("PlayTennis.csv", header=T)
> (playTennis.tr<-tree(Play~., data=playTennis))
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

1) root 14 18.250 Yes ( 0.3571 0.6429 )
  2) Humidity: High 7  9.561 No ( 0.5714 0.4286 ) *
  3) Humidity: Normal 7  5.742 Yes ( 0.1429 0.8571 ) *
> plot(playTennis.tr); text(playTennis.tr)
```

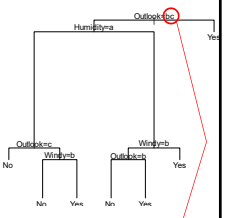


これは失敗と言っていいでしょう。なぜこうなってしまったのでしょうか？
それは、枝分かれするときの条件が厳しく（つまり、枝分かれしないよう）なっているからです。
それ（つまり、制御の仕方）を調べてみましょう。
?tree
として下さい。"tree" の説明書が得られます。しかし、木を生成するときの制御の仕方についての記述は見つかりません。こういうときは、control というキーワードを探してみます。下の方に control.tree という文言があります。ここをクリックするか
?control.tree
としてみてください。tree.control(nobs, mincut = 5, minsize = 10, mindev = 0.01) が制御方法であり、default値であることが分ります。多少試行錯誤すると、treeにおいては、mincut = 1, minsize = 2 が最小値、つまり、最も木が発達しやすいパラメータであることが分ります。そこで、
tree.control(length(playTennis[, 1]), mincut = 1, minsize = 2)
としてみますが、結果は変わりません。
理由は分りません。
やむをえず、別のライブラリを使うことにします。

```
> library(rpart)
> setwd("D:/R/Sample")
> playTennis <- read.csv("PlayTennis.csv", header=T)
> (playTennis.tr <- rpart(Play~ ., playTennis ) )
n= 14

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 14 5 Yes (0.3571429 0.6428571) *
> plot(playTennis.tr); text(playTennis.tr)
  以下にエラー plot.rpart(playTennis.tr) : fit is not a tree, just a root
```

これはもっと悪い。枝分かれせず、根のみとなってしまった。
先ほどと同様に
?rpart
としてみましょう。今度は引数に control というものがあります。その下にある例題を見ると、rpart.control を使えばよいことが分ります。rpart.control をクリックするか ?rpart.control としてみましょう。Minsplit を小さくすれば良さそうなことが想像できます。試してみましょう。

```
> library(rpart)
> setwd("D:/R/Sample")
> playTennis <- read.csv("PlayTennis.csv", header=T)
> (playTennis.tr <- rpart(Play~ ., playTennis,
+    control=rpart.control(minsplit=1)) )
n= 14

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 14 5 Yes (0.3571429 0.6428571)
   2) Outlook=Rainy,Sunny 10 5 No (0.5000000 0.5000000)
     4) Humidity=High 5 1 No (0.8000000 0.2000000)
       8) Outlook=Sunny 3 0 No (1.0000000 0.0000000) *
       9) Outlook=Rainy 2 1 No (0.5000000 0.5000000)
        18) Windy=True 1 0 No (1.0000000 0.0000000) *
        19) Windy=False 1 0 Yes (0.0000000 1.0000000) *
     5) Humidity=Normal 5 1 Yes (0.2000000 0.8000000)
      10) Windy=True 2 1 No (0.5000000 0.5000000)
        20) Outlook=Rainy 1 0 No (1.0000000 0.0000000) *
        21) Outlook=Sunny 1 0 Yes (0.0000000 1.0000000) *
      11) Windy=False 3 0 Yes (0.0000000 1.0000000) *
   3) Outlook=Overcast 4 0 Yes (0.0000000 1.0000000) *
> plot(playTennis.tr); text(playTennis.tr)
```




```
> levels(playTennis$Outlook)
[1] "Overcast" "Rainy"    "Sunny"
> levels(playTennis$Temp.)
[1] "Cool" "Hot"  "Mild"
> levels(playTennis$Windy)
[1] "False" "True"
```

今度はうまく行ったようである。では、未知データがどう分類されるか見てみよう。
"predict" について rpart の説明書中には記述がない。
こういったときは、?predict.rpart としてみる（つまり、クラス rpart のメソッド predict）。
パッケージ e1071の naiveBayes とは異なり、次のように簡単にテストできる。

```
PlayTennisTest02 <- read.csv("PlayTennisTest02.csv",header=TRUE)
predict(playTennis.tr, PlayTennisTest02)
```

```
> playTennisTest02 <- read.csv("PlayTennisTest02.csv",header=TRUE)
> predict(playTennis.tr, playTennisTest02)
     No Yes
[1,]  1   0
[2,]  0   1
> playTennisTest02
  Outlook Temp. Humidity Windy Play
1   Sunny  Cool     High  True   No
2   Rainy  Mild   Normal False  Yes
```

結果は勿論、想定通り。なお、パラメータに type があり、type="prob" とすれば確率値の出力が可能です。

```
> predict(playTennis.tr, PlayTennisTest02, type="prob")
     No Yes
[1,]  1   0
[2,]  0   1
> # level number, class frequencies, probabilities
> predict(playTennis.tr, PlayTennisTest02, type="matrix")
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    0    1    0
[2,]    2    0    3    0    1
```

気がついたかもしれませんが、tree も rpart も 2分木しか作りません。
その点では、weka の J48 の方がよくできています。

今日の課題

- Naïve Bayes のときの「今日の課題」とデータは同じです。
- rpart を用いて、下図左の訓練データが与えられたとき、下図右のテストデータの属性「スキー」の値を推定せよ。
- Rを使ってください。

雪	天気	シーズン	体調	スキー
ベタ	霧	ロー	回復	no
新雪	晴	ロー	回復	yes
新雪	霧	ロー	回復	yes
ざらめ	霧	ロー	怪我	no
新雪	晴	ロー	怪我	no
ベタ	晴	ロー	回復	yes
新雪	霧	ロー	回復	yes
ベタ	晴	半ば	回復	yes
新雪	晴	ハイ	回復	yes
新雪	風	ロー	回復	yes
ざらめ	霧	半ば	回復	no
新雪	風	ロー	回復	yes
新雪	晴	半ば	回復	yes
ざらめ	風	ハイ	疲労	no

雪	天気	シーズン	体調	スキー
ベタ	風	半ば	疲労	?

時間が余る人向け

- 「今日の課題」について、学習データの confusion matrix を作ってください。
- 面倒なことは、rpart木の予測値（predictの出力値）が、no, yes の2列の matrix（行は各データ）になることです。
  - 一方、正解として用意しているのは、yes, no が一列に並んだ配列です。
- 次のような方法が考えられます（いくつもあると思います）
- predict の出力を yes, no の列に変える
  - 例えば、no の列をみて、要素が 0.5より大 であれば、"no" にそうでなければ、"yes" にする。lapply が使えます。なお、結果は list になりますので、as.character を使って character に変えます
  - factor を、no の列（またはyesの列）に適用してもよい。labels を指定して、level の名称を "yes" と "no" にする。
- 正解値（ no, yes を値とする列）を 0, 1 の列にし、predict の出力の no の列（yes の列でもよい）が 0,1 の列であることを利用する。

なお、最も簡単な解は predict で, type="class" を指定することです。上の問題は「それをしない」方法を考えてみようという、Rの練習問題です。

まとめ

- 決定木の作り方
  - 分りやすく、使いやすい。誤差は大きめ。
  - 構築には、greedy アルゴリズム
    - ノードに置く属性の選択: 情報量増分
      - 「増分比」が必要な局面がある。
    - 復習: 情報量について
  - 一つの発展形として、回帰木がある
  - R では、tree, rpart を試してみた