決定木　その4
（改めて）決定木の作り方

慶應義塾大学理工学部
櫻井彰人

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）、再帰的かつ分割統治（　　）
  - 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

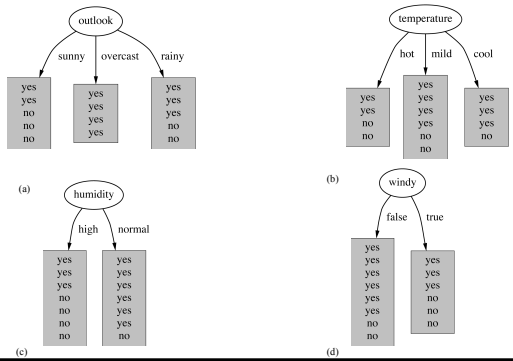
これでいいのか？

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？



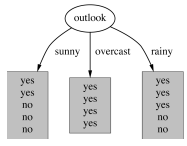
属性選択の基準

- どの属性が最適化?
- できあがる決定木が最小のものがよい
  - ヒューリスティック: “純度” 最高の属性を選ぶ
  - 「最小」のものを選ぶことに関し、深遠な議論がある
- 良く使われる “不純度” の基準: （ノードの）エントロピー
  - エントロピーが低いほど、ノードの “純度” は高い.
- 方略: 子供のノードのエントロピーが最小となる属性を選べ.

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



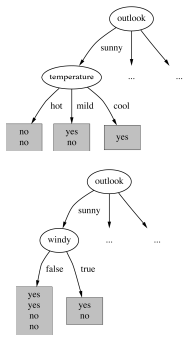
情報量増分 Information gain

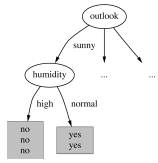
- ただし、通常は、ノードのエントロピーを直接用いることはない。情報量増分を用いる.

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。

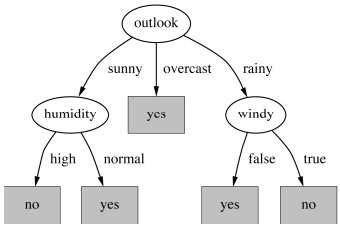
分割を続ける





$$\text{gain("Temperature")} = 0.571\,\text{bits}$$
$$\text{gain("Humidity")} = 0.971\,\text{bits}$$
$$\text{gain("Windy")} = 0.020\,\text{bits}$$

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

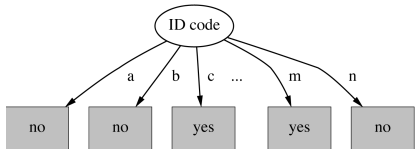
情報量増分の問題点

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 ）

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[redacted]属性を選ぶようにバイアスしている
  - この結果、過学習[redacted](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。

増分比

どんなバイアスでしたか？

- 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
これが14個あるゆえ、14倍
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

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

増分比について

- “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（直し過ぎ。治療が過剰）
  - 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.

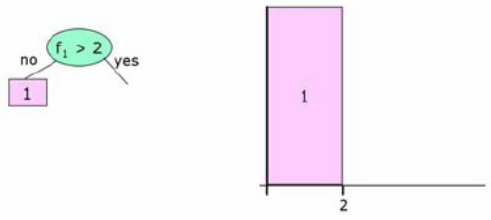
補足

- 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)

他の型の属性の取扱い

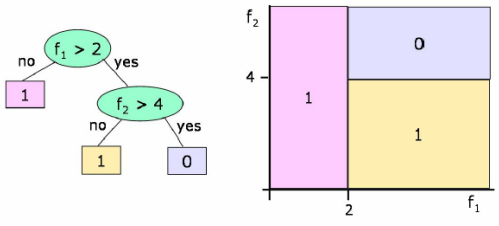
数値属性

- 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



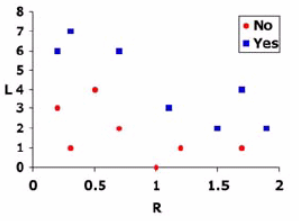
数値属性

- 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

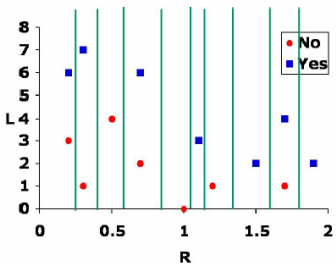
L	R	B
3	0.2	No
1	0.3	No
4	0.5	No
2	0.7	No
0	1.0	No
1	1.2	No
1	1.7	No
6	0.2	Yes
7	0.3	Yes
6	0.7	Yes
3	1.1	Yes
2	1.5	Yes
4	1.7	Yes
2	1.9	Yes



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

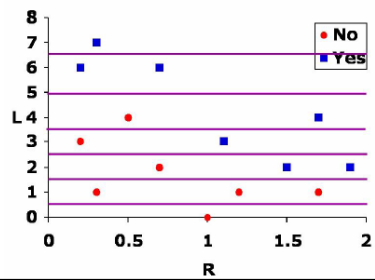
- 各属性ごとに、分割することを考えよう



- 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.

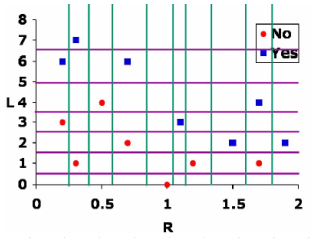
分割その II

- 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

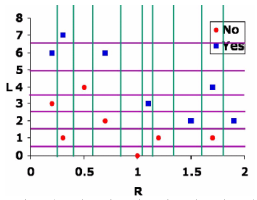


エントロピー	1.00	1.00	0.98	0.98	0.94	0.98	0.92	0.98	0.92
境界	0.25	0.40	0.60	0.85	1.05	1.15	1.35	1.60	1.80

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



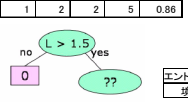
エントロピー	1.00	1.00	0.98	0.98	0.94	0.98	0.92	0.98	0.92
境界	0.25	0.40	0.60	0.85	1.05	1.15	1.35	1.60	1.80

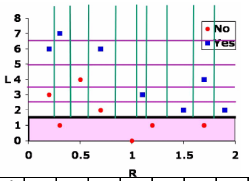
- たまたま、L 軸で、境界を 1.5 とした場合、片側が No だけになることがわかった（エントロピーも最小）

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



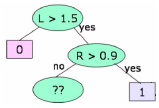


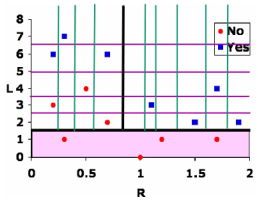
エントロピー	0.85	0.88	0.79	0.60	0.69	0.76	0.83
境界	0.25	0.40	0.60	0.90	1.30	1.60	1.80

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: