Changelog: 14 Oct 2016, 30 Oct 2016

Decision Trees

Entropy, Information Gain, Gain Ratio

Lecture 3: Part 2

Marina Santini

Acknowledgements
Slides borrowed and adapted from:
Data Mining by I. H. Witten, E. Frank and M. A. Hall

Outline

- Entropy
- Information gain
- Gain ratio

Trees

- "Divide-and-conquer" approach produces tree
- Nodes involve testing a particular attribute
- Usually, attribute value is compared to constant
- Other possibilities:
  - Comparing values of two attributes
  - Using a function of one or more attributes
- Leaves assign classification, set of classifications, or probability distribution to instances
- Unknown instance is routed down the tree

Nominal and numeric attributes

- Nominal:
  number of children usually equal to number values
  ⇒ attribute won't get tested more than once
  - Other possibility: division into two subsets
- Numeric:
  test whether value is greater or less than constant
  ⇒ attribute may get tested several times
  - Other possibility: three-way split (or multi-way split)
    - Integer: *less than, equal to, greater than*
    - Real: *below, within, above*

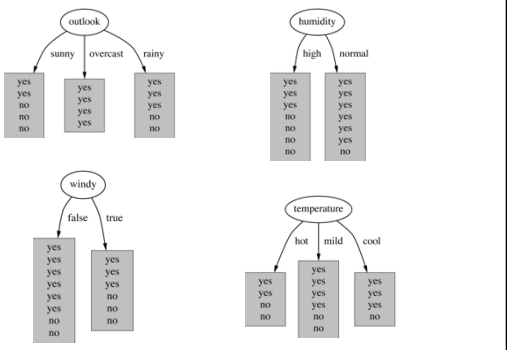
Missing values

- Does absence of value have some significance?
- Yes ⇒ "missing" is a separate value
- No ⇒ "missing" must be treated in a special way
  - Solution A: assign instance to most popular branch
  - Solution B: split instance into pieces
    - Pieces receive weight according to fraction of training instances that go down each branch
    - Classifications from leave nodes are combined using the weights that have percolated to them

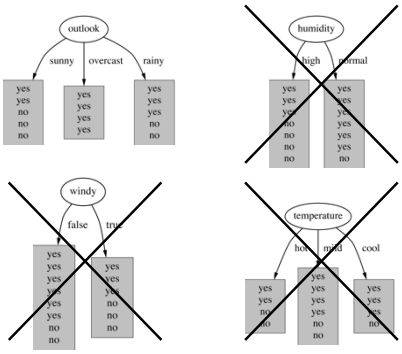
Constructing decision trees

- Strategy: top down
  Recursive *divide-and-conquer* fashion
  - First: select attribute for root node
    Create branch for each possible attribute value
  - Then: split instances into subsets
    One for each branch extending from the node
  - Finally: repeat recursively for each branch, using only instances that reach the branch
- Stop if all instances have the same class

Which attribute to select?



Which attribute to select?



Criterion for attribute selection

- Which is the best attribute?
  - Want to get the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular *impurity criterion*: *information gain*
  - Information gain increases with the average purity of the subsets
- Strategy: choose attribute that gives greatest information gain

Computing information

- Measure information in *bits*
  - Given a probability distribution, the info required to predict an event is the distribution's *entropy*
  - Entropy gives the information required in bits (can involve fractions of bits!)
- Formula for computing the entropy:

$$\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n$$

Example: attribute *Outlook*

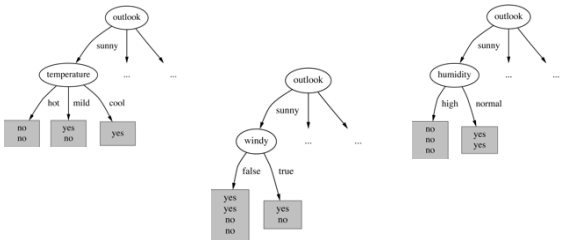
- *Outlook = Sunny* :
  $\text{info}([2,3]) = \text{entropy}(2/5, 3/5) = -2/5 \log(2/5) - 3/5 \log(3/5) = 0.971 \text{ bits}$
- *Outlook = Overcast* :
  $\text{info}([4,0]) = \text{entropy}(1,0) = -1 \log(1) - 0 \log(0) = 0 \text{ bits}$
  Note: this is normally undefined.
- *Outlook = Rainy* :
  $\text{info}([2,3]) = \text{entropy}(3/5, 2/5) = -3/5 \log(3/5) - 2/5 \log(2/5) = 0.971 \text{ bits}$
- Expected information for attribute:
  $\text{info}([3,2],[4,0],[3,2]) = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971 = 0.693 \text{ bits}$

Computing information gain

- Information gain: information before splitting – information after splitting
  gain(*Outlook*) = info([9,5]) – info([2,3],[4,0],[3,2])
  = 0.940 – 0.693
  = 0.247 bits
- Information gain for attributes from weather data:

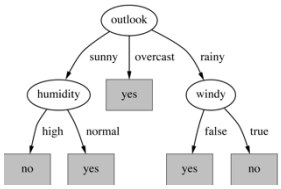
  gain(*Outlook*) = 0.247 bits
  gain(*Temperature*) = 0.029 bits
  gain(*Humidity*) = 0.152 bits
  gain(*Windy*) = 0.048 bits

Continuing to split



gain(*Temperature*) = 0.571 bits
gain(*Humidity*) = 0.971 bits
gain(*Windy*) = 0.020 bits

Final decision tree



- Note: not all leaves need to be pure; sometimes identical instances have different classes
  - ⇒ Splitting stops when data can't be split any further

Wishlist for a purity measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property* (i.e. decisions can be made in several stages):

$$measure([2,3,4]) = measure([2,7]) + (7/9) \times measure([3,4])$$

- Entropy is the only function that satisfies all three properties!

Properties of the entropy

- The multistage property:

$$entropy(p,q,r) = entropy(p,q+r) + (q+r) \times entropy\left(\frac{q}{q+r}, \frac{r}{q+r}\right)$$

- Simplification of computation:

$$info([2,3,4]) = -2/9 \times \log(2/9) - 3/9 \times \log(3/9) - 4/9 \times \log(4/9)$$
$$= [-2 \times \log 2 - 3 \times \log 3 - 4 \times \log 4 + 9 \times \log 9]/9$$

- Note: instead of maximizing info gain we could just minimize information

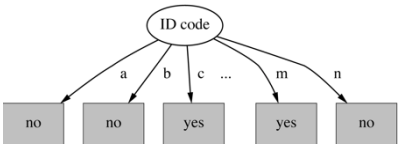
Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - ⇒ Information gain is biased towards choosing attributes with a large number of values
  - ⇒ This may result in *overfitting* (selection of an attribute that is non-optimal for prediction)
- Another problem: *fragmentation*

Weather data with *ID code*

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

Tree stump for *ID code* attribute



- Entropy of split:

$$\text{info}(\text{ID code}) = \text{info}([0,1]) + \text{info}([0,1]) + \ldots + \text{info}([0,1]) = 0 \text{ bits}$$

⇒ Information gain is maximal for ID code (namely 0.940 bits)

Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account
- Intrinsic information: entropy of distribution of instances into branches (i.e. how much info do we need to tell which branch an instance belongs to)

Computing the gain ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log(1/14)) = 3.807 \text{ bits}$$

- Value of attribute decreases as intrinsic information gets larger
- Definition of gain ratio:

$$\text{gain\_ratio}(\textit{attribute}) = \frac{\text{gain}(\textit{attribute})}{\text{intrinsic\_info}(\textit{attribute})}$$

- Example:

$$\text{gain\_ratio}(\text{ID code}) = \frac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246$$

Gain ratios for weather data

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])	1.557
Gain ratio: 0.247/1.577	0.157	Gain ratio: 0.029/1.557	0.019
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

More on the gain ratio

- "Outlook" still comes out top
- However: "ID code" has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix: only consider attributes with greater than average information gain

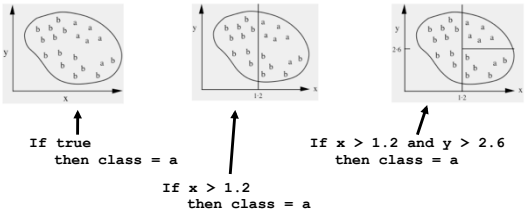
Discussion

- Top-down induction of decision trees: ID3, algorithm developed by Ross Quinlan
  - Gain ratio just one modification of this basic algorithm
  - ⇒ C4.5: deals with numeric attributes, missing values, noisy data
- Similar approach: CART
- There are many other attribute selection criteria!
  (But little difference in accuracy of result)

Covering algorithms

- Convert decision tree into a rule set
  - Straightforward, but rule set overly complex
  - More effective conversions are not trivial
- Instead, can generate rule set directly
  - for each class in turn find rule set that covers all instances in it
    (excluding instances not in the class)
- Called a *covering* approach:
  - at each stage a rule is identified that "covers" some of the instances

Example: generating a rule



- Possible rule set for class "b":

```
If x ≤ 1.2 then class = b
If x > 1.2 and y ≤ 2.6 then class = b
```

- Could add more rules, get "perfect" rule set

Classification rules

- Popular alternative to decision trees
- *Antecedent* (pre-condition): a series of tests (just like the tests at the nodes of a decision tree)
- Tests are usually logically ANDed together (but may also be general logical expressions)
- *Consequent* (conclusion): classes, set of classes, or probability distribution assigned by rule
- Individual rules are often logically ORed together
  - Conflicts arise if different conclusions apply

From trees to rules

- Easy: converting a tree into a set of rules
  - One rule for each leaf:
    - Antecedent contains a condition for every node on the path from the root to the leaf
    - Consequent is class assigned by the leaf
- Produces rules that are unambiguous
  - Doesn't matter in which order they are executed
- But: resulting rules are unnecessarily complex
  - Pruning to remove redundant tests/rules

From rules to trees

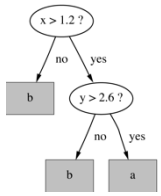
- More difficult: transforming a rule set into a tree
  - Tree cannot easily express disjunction between rules
- Example: rules which test different attributes

```
If a and b then x
If c and d then x
```

- Symmetry needs to be broken
- Corresponding tree contains identical subtrees (⇒ "replicated subtree problem")

Rules vs. trees

Corresponding decision tree:
(produces exactly the same predictions)



- But: rule sets *can* be more perspicuous when decision trees suffer from replicated subtrees
- Also: in multiclass situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account

The end