Machine Learning

Decision Tree Learning

Simplified ID3

(Simplification: random attribute selection)

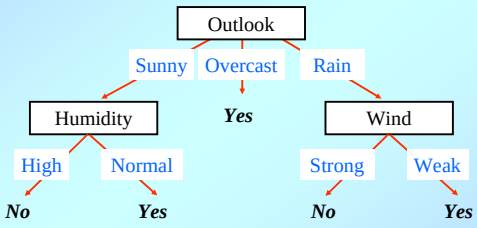
Machine Learning Basics

- Given data observed
- Attempt to find patterns (training)
- Use patterns to predict future (testing)
- Supervised learning
  - User tells machine what to find
- Unsupervised learning
  - Let the machine find "interesting" patterns, e.g. find clusters

Example Classification Problem: Play Tennis?

Day	Outlook	Temper.	Humid.	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Example Decision Tree



- Decision to make: *play tennis?*

Confusion Matrix

Prediction

Reality	–	+	
–	5	2	7
+	1	2	3
	6	4	10

Reality	Prediction
–	–
+	+
+	–
–	–
–	–
–	–
+	+
–	+
–	+
–	–

Performance Measures

Ideal Predictions

Reality	–	+	
–	7	0	7
+	0	3	3
	7	3	10

Actual Predictions, Example

	–	+	
–	5	2	7
+	1	2	3
	6	4	10

RC = (5+2) ÷10 = 70%

Precision = 2 ÷ 4 = 50%

Recall = 2 ÷ 3 = 67%

ID3 for machine learning

- ID3 performs supervised learning
- It builds decision trees
- Perfect fitting with training data
- Like other machine learning techniques:
  - No guarantee that it fits testing data
  - Danger of "over-fitting"

Prolog Implementation of Facts

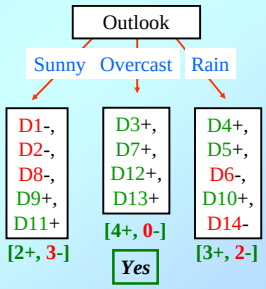
```
attribute( outlook, [sunny, overcast, rain] ).
attribute( temperature, [hot, mild, cool] ).
attribute( humidity, [high, normal] ).
attribute( wind, [weak, strong] ).

example( [outlook=sunny, temperature=hot, humidity=high,
    wind=weak], play=no ).
example( [outlook=sunny, temperature=hot, humidity=high,
    wind=strong], play=no ).
example( [outlook=overcast, temperature=hot, humidity=high,
    wind=weak], play=yes ).
```


Expected Output

```
?- run.
  1 outlook=sunny
    2 temperature=hot conclusion(play=no)
    2 temperature=mild
      3 humidity=high conclusion(play=no)
      3 humidity=normal conclusion(play=yes)
    2 temperature=cool conclusion(play=yes)
  1 outlook=overcast conclusion(play=yes)
  1 outlook=rain
    2 temperature=hot conclusion(category not seen before)
    2 temperature=mild
      3 humidity=high
        4 wind=weak conclusion(play=yes)
        .....
```

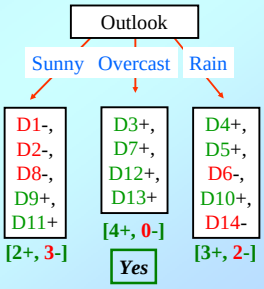

Example: ID3 Picks an attribute

- Pick an attribute *A*
- Compute Gain(S, *A*):
  - *Outlook*: 0.246
  - *Humidity*: 0.151
  - *Wind*: 0.048
  - *Temperature*: 0.029
- *Outlook* is picked



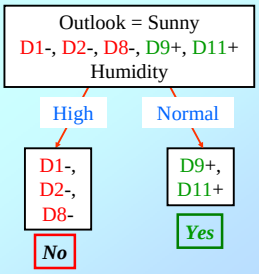
Simplified ID3 in Action (1)

1) Not all examples agree on conclusion
2) Pick one attribute
   - "*Outlook*" is picked
3) Divide examples according to values in *Outlook*
4) Build each branch recursively
   - "*Yes*" for "*Overcast*"



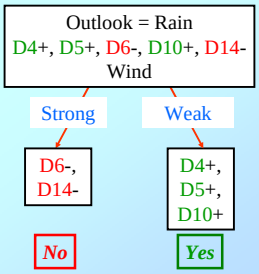
Simplified ID3 in Action (2)

1) Expand *Outlook=Sunny*
2) Not all examples agree
3) Pick one attribute
   - "*Humidity*" is picked
4) Divide examples
5) Build two branches
   - "*No*" for "*High*"
   - "*Yes*" for "*Normal*"

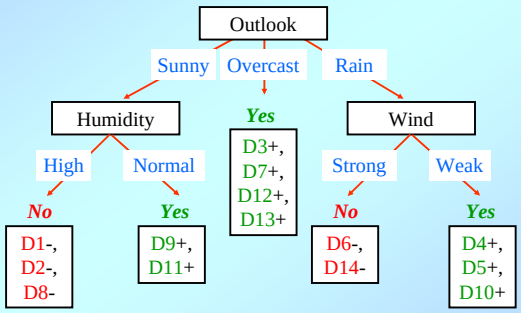


Simplified ID3 in Action (3)

1) Expand *Outlook=Rain*
2) Not all examples agree
3) Pick one attribute
   - "*Wind*" is picked
4) Divide examples
5) Build two branches
   - "*No*" for "*Strong*"
   - "*Yes*" for "*Weak*"



ID3 in Action – Tree Built



ID3 (top)

```
/*
   An induced Tree can be:
   (a) a Conclusion (see below); or
   (b) a list of subtrees, each of which takes the form:
         (Attribute=Value, Tree)
   (recursive definition).
   A Conclusion can be:
   (1) not enough experience to cover the newly encountered case;
   (2) conclusion (based on induction); or
   (3) inconclusive, as contradictory cases was found in the eg's.
*/
induce_tree( Tree ) :-
   findall( (AttVal, Class), example(AttVal, Class), Examples ),
   findall( Att, attribute( Att, _ ), Attributes ),
   induce_tree( Attributes, Examples, Tree ).
```

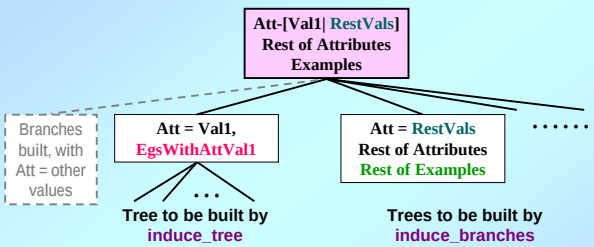

ID3 (simplified)

```
/*
   Four cases:
   1. no example covers this branch
   2. all examples under this subtree agree on the same conclusion
   3. no conclusions yet, but run out of attributes
   4. classify with unused attributes so far
   Note: the fourth clause picks the first unused attribute. ID3 picks
         the attribute that minimises "entropy" in the remaining data
*/
induce_tree( _, [ ], conclusion('category not seen before') ) :- !.
induce_tree( _, [(_, Class)| Examples], conclusion(Class) ) :-
   not( (member((_, Class1), Examples), Class1 \== Class) ), !.
induce_tree( [ ], _, conclusion('contradiction in examples') ) :- !.
induce_tree( [Att1 | RestAtts], Examples, ListOfSubtrees ) :-
   attribute( Att1, Values ),                /* retrieve data */
   induce_branches( Att1-Values, RestAtts, Examples, ListOfSubtrees ).
```


ID3 branches

```
/*
   induce_branches( Att-ValList, RestOfAtts, Egs, SubtreesToBuild)
   Example:
     Att-ValList: humidity-[high, normal]
     RestOfAtts: [wind]
     Egs: [([outlook=sunny, temp=hot, ...], no), ...]
     SubtreesToBuild: [(humidity=high, no), (humidity=normal, [...])]
*/
induce_branches( _-[ ], _, _, [ ] ).      /* no more values to deal with */
induce_branches( Att-[Val1| RestVals], RestAtts, Egs,
                 [(Att=Val1, Tree1)| Trees] ) :-
   classifyEgs( Att=Val1, Egs, EgsWithAttVal1, RestEgs ),
   induce_tree( RestAtts, EgsWithAttVal1, Tree1 ),
   induce_branches( Att-RestVals, RestAtts, RestEgs, Trees ).
```


Induce Branches




```
induce_branches( _-[ ], _, _, [ ] ).      /* no more values to deal with */
induce_branches( Att-[Val1| RestVals], RestAtts, Egs, [(Att=Val1, Tree1)| Trees] ) :-
   classifyEgs( Att=Val1, Egs, EgsWithAttVal1, RestEgs ),
   induce_tree( RestAtts, EgsWithAttVal1, Tree1 ),
   induce_branches( Att-RestVals, RestAtts, RestEgs, Trees ).
```

ID3 Classify Egs

```
classifyEgs( _, [ ], [ ], [ ] ).
classifyEgs( AttVal, [(AttValList, Class)| RestEgs],
   [(AttValList, Class)| InSet ], OutSet ) :-
        member(AttVal, AttValList), !,
        classifyEgs( AttVal, RestEgs, InSet, OutSet ).
classifyEgs( AttVal, [(AttValList, Class)| RestEgs],
   InSet, [(AttValList, Class)| OutSet ] ) :-
        classifyEgs( AttVal, RestEgs, InSet, OutSet ).
```


Remarks on ID3

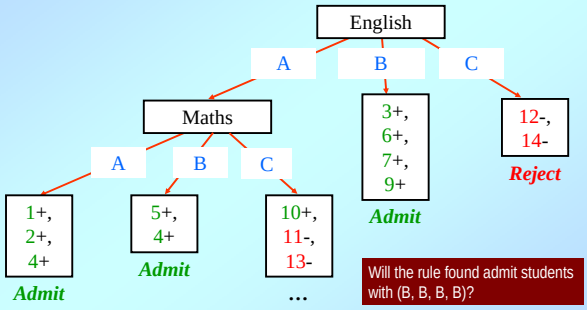
- Decision trees are easy to understand
- Decision trees are easy to use
- But what if data has noise?
  - I.e. under the same situation, contradictory results were observed
- Besides, what if some values are missing from the decision tree?
  - E.g. "Humidity = Low"
- These are handled by C4.5 and See5 (beyond our syllabus)

Exercise Classification Problem - Admit?

Student	Maths	English	Physics	IT	Exam
1	A	A	C	B	Admit
2	A	A	A	C	Admit
3	A	B	C	A	Admit
4	A	A	B	A	Admit
5	B	A	A	C	Admit
6	B	B	C	A	Admit
7	A	B	B	B	Admit
8	B	A	B	C	Admit
9	B	B	A	B	Admit
10	C	A	B	A	Admit
11	C	A	C	A	Reject
12	A	C	C	B	Reject
13	C	A	B	C	Reject
14	B	C	A	C	Reject

Suppose the rule is: "Only accept students with at least three (A or B)s, including at least one A". Could ID3 find it?

ID3 Exercise: "admit or reject"



Lessons from the exercise

- Without the right columns in the database, one cannot learn the underlying rule
  - It would help if there were a column "A or B"
- Rules generalise
  - they may not be correct
- Some branches may not be covered
  - Not shown in the above slide