Pattern Analysis and Machine Intelligence

Lecture Notes on Machine Learning

Matteo Matteucci

matteucci@elet.polimi.it

Department of Electronics and Information

Politecnico di Milano

Probability for Dataminers

– Information Gain –

Information and Bits

Your mission, if you decide to accept it, will be:

"Transmit a set of independent random samples of X over a binary serial link."

1. Starring at X for a while, you notice that it has olny four possible values: A, B, C, D
2. You decide to transmit the data encoding each reading with two bits:

$$A = 00, B = 01, C = 10, D = 11.$$

Mission Accomplished!

Information and "Fewer Bits"

Your mission, if you decide to accept it, will be:

"The previous code uses 2 bits for symbol. Knowing that the probabilities are not equal: P(X=A)=1/2, P(X=B)=1/4, P(X=C)=1/8, P(X=D)=1/8, invent a coding for your transmission that only uses 1.75 bits on average per symbol."

1. You decide to transmit the data encoding each reading with a different number of bits:

$$A = 0, B = 10, C = 110, D = 111.$$

Mission Accomplished!

Information and Entropy

Suppose X can have one of m values with probability

$$P(X = V_1) = p_1, \ldots, P(X = V_m) = p_m.$$

What's the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from X's distribution?

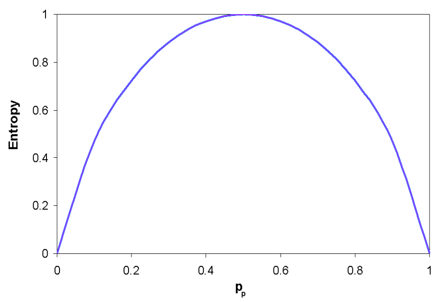
$$\begin{aligned} H(X) &= -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_m \log_2 p_m \\ &= -\sum_{j=1}^{m} p_j \log_2 p_j = \textit{Entropy} \text{ of } X \end{aligned}$$

"Good idea! But what is entropy anyway?"

Entropy: "What is it anyway?"

Simple Case:

- X has 2 values $\oplus$ and $\ominus$
- $p_\oplus$ probability of $\oplus$
- $p_\ominus = 1 - p_\oplus$ probability of $\ominus$



$$H(X) = -p_\ominus \log_2 p_\ominus - p_\oplus \log_2 p_\oplus$$

Entropy measures "disorder" or "uniformity in distribution"

1. *High Entropy*: X is very "disordered" $\rightarrow$ "interesting"
2. *Low Entropy*: X is very "ordered" $\rightarrow$ "boring"

Useful Facts on Logarithms

Just for you to know it might be useful to review a couple of formulas to be used in calculation:

- $\ln x \times y = \ln x + \ln y$
- $\ln \frac{x}{y} = \ln x - \ln y$
- $\ln x^y = y \times \ln x$
- $\log_2 x = \frac{\ln x}{\ln 2} = \frac{\log_{10} x}{\log_{10} 2}$
- $\log_a x = \frac{}{\log_b a}$
- $\log_2 0 = -\infty$ (the formula is no good for a probability of 0)

Now we can practice with a simple example!

Specific Conditional Entropy

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Hystory	No
Math	Yes

Suppose we are interested in predicting output Y from input X where

- X = University subject
- Y = Likes the movie “Gladiator”

From this data we can estimate

- P(Y = Yes) = 0.5
- P(X = Math) = 0.5
- P(Y = Yes | X = History) = 0

Definition of Specific Conditional Entropy:

- H(Y|X=v): *the entropy of Y only for those records in which X has value v*
  - H(Y|X=Math) = 1
  - H(Y|X=History) = 0

Conditional Entropy

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Hystory	No
Math	Yes

Definition of Conditional Entropy H(Y|X):

- *The average Y specific conditional entropy*
- *Expected number of bits to transmit Y if both sides will know the value of X*
- $\sum_j P(X = v_j) H(Y|X = v_j)$

Definition of Conditional Entropy H(Y|X):

- $\sum_j P(X = v_j) H(Y|X = v_j)$

v_j	$P(X = v_j)$	$H(Y\|X = v_j)$
Math	0.5	1
Hystory	0.25	0
CS	0.25	0

H(Y|X) = ?

Information Gain

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Hystory	No
Math	Yes

I must transmit Y on a binary serial line.
How many bits on average would it save me if both ends of the line knew X?

$$
\begin{aligned}
IG(Y|X) &= H(Y) - H(Y|X) \\
&= 1 - 0.5 = 0.5
\end{aligned}
$$

Information Gain measures the "information" provided by X to predict Y

This IS about Machine Learning!

Relative Information Gain

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Hystory	No
Math	Yes

I must transmit Y on a binary serial line.
What fraction of the bits on average would it save me if both ends of the line knew X?

$$
\begin{aligned}
RIG(Y|X) &= (H(Y) - H(Y|X))/H(Y) \\
&= (1 - 0.5)/1 = 0.5
\end{aligned}
$$

Well, we'll find soon Information Gain and Relative Information gain talking about supervised learning with Decision Trees ...

Why is Information Gain Useful?

Your mission, if you decide to accept it, will be:

"Predict whether someone is going live past 80 years."

From historical data you might find:

- IG(LongLife | HairColor) = 0.01
- IG(LongLife | Smoker) = 0.2
- IG(LongLife | Gender) = 0.25
- IG(LongLife | LastDigitOfSSN) = 0.00001

What you should look at?

Classification Algorithms

– Decision Trees –

Decision Trees: Definition

A Decision Tree is an arrangement of tests that prescribes an appropriate test at every step in an analysis:

- a method for approximating discrete-valued target functions,
- capable of learning disjunctive expressions,
- robust to noisy data.

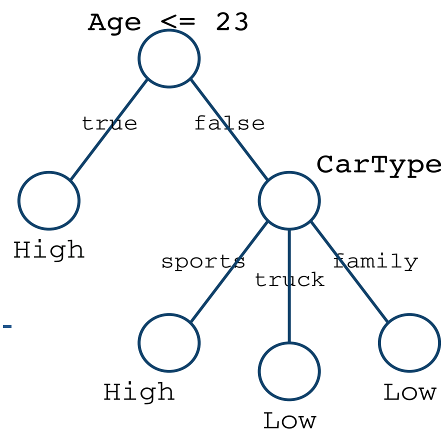
This lectures has been (heavily) inspired by:

- `www.autonlab.org/tutorials/`
- `www.cs.uregina.ca/˜hamilton/courses/831/index.html`
- T. Mitchell, “Decision Tree Learning”, in T. Mitchell, Machine Learning, The McGraw-Hill Companies, Inc., 1997, pp. 52-78
- P. Winston, “Learning by Building Identification Trees”, in P. Winston, Artificial Intelligence, Addison-Wesley Publishing Company, 1992, pp. 423-442.

Decision Trees: Representation

Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.

- A Node represent a test on attributes' values
- An Arch represent the result of the test
- Nodes with no outgoing arches are called Leaves
- Leaves represent the resulting classification



Decision trees classify instances by sorting them down the tree from the root node to some leaf node, which provides instance classification.

When should I use a Decision Tree?

Decision tree learning is generally best suited to problems with:

- Instances represented by attribute-value pairs.
  - Records are described by a fixed set of attributes (e.g., temperature) and their values (e.g., hot).
  - Each attribute takes on a small number of disjoint possible values (e.g., hot, mild, cold).
  - Extensions to the basic algorithm allow handling real-valued attributes as well (e.g., a floating point temperature).
- The target function has discrete output values.
  - A decision tree assigns a classification to each example.
  - Extensions allow learning target functions with real-valued outputs, although the application of decision trees in this setting is less common.
- The training data may contain errors or missing attribute values
- Disjunctive descriptions may be required

A (Very) Small Example Dataset

Suppose you work for *TenenTel Insurance Ltd.* as a consultant:

Age	CarType	Risk
17	sprots	High
43	family	Low
68	family	Low
32	truck	Low
23	family	High
18	family	High
20	family	High
45	sports	High
50	truck	Low
64	truck	High
46	family	Low
40	family	Low

- An expert already assigned clients to a "Risk" level
- The company would like an automatic procedure to obtain the same result on new records
- The company is also looking for a compact representation of the classification process to be used by a computer

Can we learn a Decision Tree out of these data?

Learning Decision Trees: A Simple Idea

A record is classified by starting at the root node of the decision tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute.

It is computationally impractical to find the smallest possible Decision Tree, so we use a procedure that tends to build small trees.

- Start from the root
- Select an attribute
- Split the data according to that attribute
- Recurse . . .

How should I choose the attribute for splitting?
Information Gain! (That's too obvious)
What about real values?
Oppps, this is less obvious . . .
When should I stop recursion?
Hmmm . . . let me think a little bit

Dealing With Real Attributes

Use a thresholded split to compute Entropy

- Suppose X is the real-valued attribute
- Define $IG(Y|X:t) = H(Y) - H(Y|X:t)$
- Define

$$\begin{aligned} H(Y|H:t) &= H(Y|X \leq t)P(X \leq t) + \\ &+ H(Y|X > t)P(X > t) \end{aligned}$$

- Define $IG^*(Y|X) = \max_t IG(Y|X:t)$
- For each real-valued attribute, use $IG^*(Y|X)$ for selecting the split

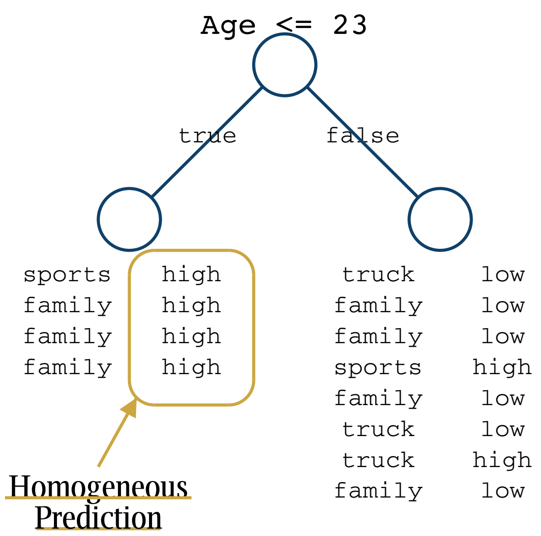
Consider each test on a real-valued attribute as a boolen test on the attribute being less or equal than such threshold value.

When should I stop splitting?

Stopping Recursion: Case Base 1

Each split generates new datasets:

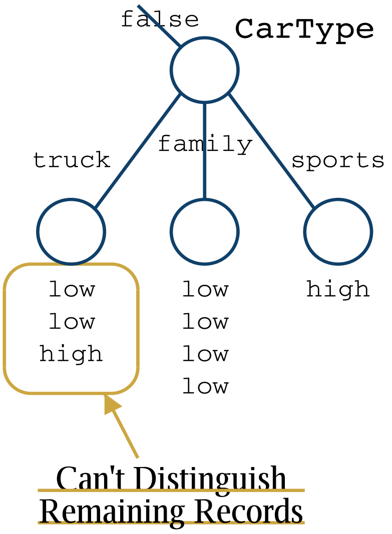
- stop if the new dataset has a homogeneous prediction
- otherwise recurse on the reduced dataset



Stopping Recursion: Case Base 2

Suppose, for this example, you treat real-valued attributes as categorical ones and you remove them when splitting:

- stop if you can't distingush the records in the new dataset
- predict the most common class or randomly if equally likely



Stopping Recursion: "What if Information Gain is 0?"

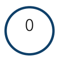
Consider the following example: $Y = A$ xor B

$$\begin{aligned} H(Y) &= 1 \\ H(Y|A) &= P(\bar{A})H(Y|\bar{A})P(A)H(Y|A) \\ &= 1/2 \times 1 + 1/2 \times 1 = 1 \\ H(Y|B) &= P(\bar{B})H(Y|\bar{B})P(B)H(Y|B) \\ &= 1/2 \times 1 + 1/2 \times 1 = 1 \end{aligned}$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Should I stop Recursion?

- if I stop recursion when Information Gain is zero
- randomly predict one of the output
- 50% Error Rate



Stopping Recursion: "What if Information Gain is 0?"

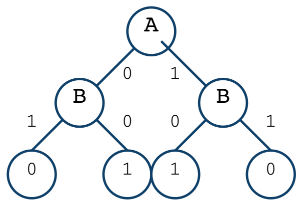
Consider the following example: $Y = A$ xor B

$$\begin{aligned} H(Y) &= 1 \\ H(Y|A) &= P(\bar{A})H(Y|\bar{A})P(A)H(Y|A) \\ &= 1/2 \times 1 + 1/2 \times 1 = 1 \\ H(Y|B) &= P(\bar{B})H(Y|\bar{B})P(B)H(Y|B) \\ &= 1/2 \times 1 + 1/2 \times 1 = 1 \end{aligned}$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Should I stop Recursion?

- if I randomly split when Information Gain in zero
- 0% Error Rate



Learning Decision Tree: The Algorithm

`Node BuildTree(Dataset, Output)`

- `if` all output values are the same in Dataset, `return` a leaf node that says "predict this unique output"
- `if` all input values are the same, `return` a leaf node that says "predict the majority output"
- `else` find attribute X with Highest Information Gain
- Suppose X has n_X distinct values
  - Create and return a node with n_X children
  - The i^{th} children is given by

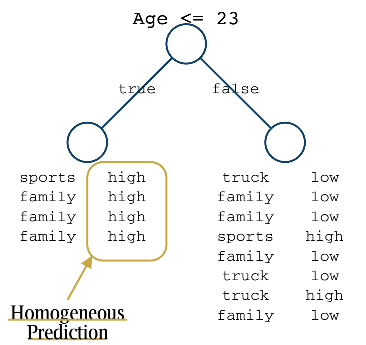
`BuildTree(Dataset`$_i$`, Output)`

Decision Trees: An Example (I)

Build the Decision Trees to classify the `Risk` level from `Age` and `CarType`.

Age	CarType	Risk
17	sprots	High
43	family	Low
68	family	Low
32	truck	Low
23	family	High
18	family	High
20	family	High
45	sports	High
50	truck	Low
64	truck	High
46	family	Low
40	family	Low

- What's the Information Gain for `CarType` Attribute?
- What's the Information Gain for `Age` Attribute?
- Which attibute should I split first?

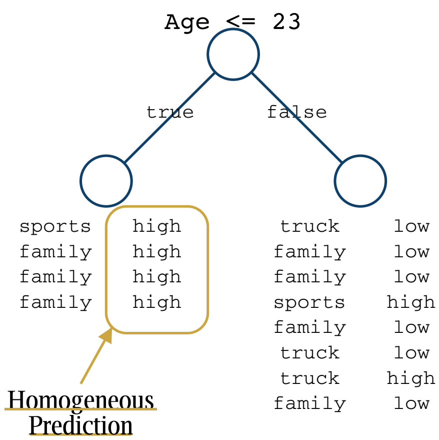


Decision Trees: An Example (II)

Build the Decision Trees to classify the `Risk` level from `Age` and `CarType`.

Age	CarType	Risk
17	sprots	High
43	family	Low
68	family	Low
32	truck	Low
23	family	High
18	family	High
20	family	High
45	sports	High
50	truck	Low
64	truck	High
46	family	Low
40	family	Low

- Should I compute the Information Gain?
- Should I split the `CarType` Attribute?

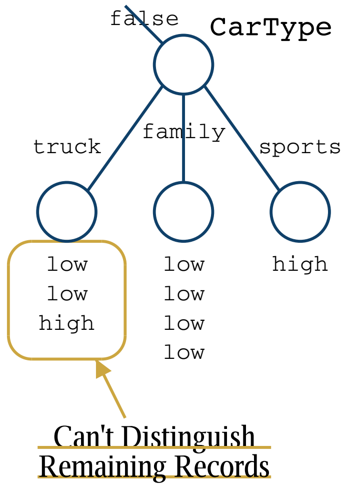


Decision Trees: An Example (III)

Build the Decision Trees to classify the `Risk` level from `Age` and `CarType`.

Age	CarType	Risk
17	sprots	High
43	family	Low
68	family	Low
32	truck	Low
23	family	High
18	family	High
20	family	High
45	sports	High
50	truck	Low
64	truck	High
46	family	Low
40	family	Low

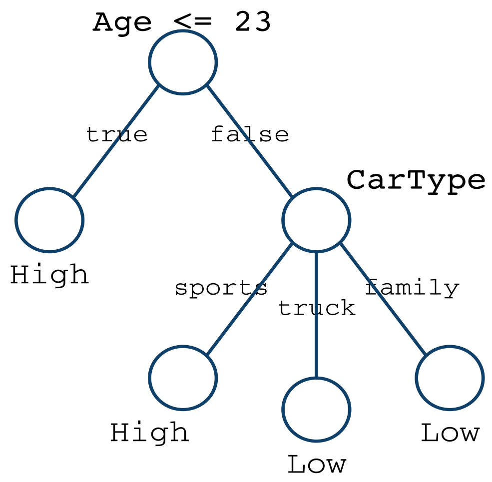
- Can I split again?



Decision Trees: An Example (IV)

Build the Decision Trees to classify the `Risk` level from `Age` and `CarType`.

Age	CarType	Risk
17	sprots	High
43	family	Low
68	family	Low
32	truck	Low
23	family	High
18	family	High
20	family	High
45	sports	High
50	truck	Low
64	truck	High
46	family	Low
40	family	Low



Overfitting in Decision Trees

Consider the following artificial dataset:

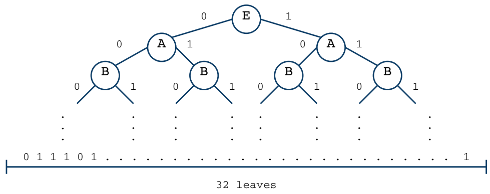
- the outpt Y is a copy of E plus a 25% noise
- attributes A, B, C, D are irrelevant to predict Y

A	B	C	D	E	Y
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
.	.	.	.	.	.
.	.	.	.	.	.
1	1	1	1	1	1

32

Y = E + noise
noise = flip 1 out of 4

We can learn a Decision Tree perfetly predicting the output Y, but it will be corrupted by noise 25% of the times



What if we get new data from the same process?
You'll get 1 out of 4 errors!

Overfitting

Overfitting: we have overfitting when the model we learnt fits the noise in the data, thus it does not generalize on new samples (it just memoryzes the training set).

Can we measure generalization?

- Hide some data before learning the tree (*Test Set*)
- Estimate how well the tree predict on "new" data (*Test Set Error*)

Can we avoid overfitting?

- Statistical tests on the data
- Use cross-validation techniques
- Introduce a bias into the model

General rule: Use Occam's razor!
"Entia non sunt multiplicanda praeter necessitatem"

Avoiding Overfitting

How can we avoid overfitting in Decision Trees?

- Statistical Test or Cross-Validation
  - Stop growing the tree when data split not statistically significant
  - Stop growing the tree when generalization error increase
- Post Pruning
  - Grow full tree, then post-prune the tree
  - Grow full tree, transform it into decision rules, post-prune the rules

Converting a Decision Tree to rules before pruning has some advantages:

- Big Decision Trees might be difficult to understand from a human user
- They can be translated into an equivalent representation known as Decision Rules
  - `IF Age<=23 OR CarType IS sports`
    `THEN Risk IS High`
  - `IF Age>23 AND CarType IS family OR truck`
    `THEN Risk IS Low`

Rule Generation Advantages

To generate rules, trace each path in the decision tree, from root node to leaf node, recording the test outcomes as antecedents and the leaf-node classification as the consequent. This has three advantages:

- Allows distinguishing among the different contexts in which a decision node is used
  - pruning decision regarding an attribute test can be made differently for each path.
  - if the tree itself were pruned, the only two choices would be remove the decision node completely, or retain it
- Removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves (i.e., avoiding to reorganize the tree if the root node is pruned while retaining part of the subtree below this test)
- Converting to rules improves readability.

Rule Pruning

Once a rule set (i.e., all the paths to the leaves) has been devised:

1. Eliminate unecessary rule antecedents to simplify the rules
   - To simplify a rule, eliminate antecedents that have no effect on the conclusion reached by the rule
   - Independence from an antecendent is verified using a test for independency
2. Eliminate unecessary rules to simplify the rule set.
3. Replace those rules that share the most common consequent by a default rule that is triggered when no other rule is triggered.

We can use the following independence tests:

- Chi-Square (cell frequencies $m > 10$): $\chi^2 = \sum_i \sum_j \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$
- Yates' Correction ($5 \leq m \leq 10$): $\chi^2 = \sum_i \sum_j \frac{(|o_{ij} - e_{ij}| - 0.5)^2}{e_{ij}}$
- Fisher's Exact Test ($m < 5$): *see Winston, pp. 437-442*

Contingency Table

A contingency table is the tabular representation of a rule:

	C_1	C_2	
R_1	x_{11}	x_{12}	$R_{1T} = x_{11} + x_{12}$
R_2	x_{21}	x_{22}	$R_{2T} = x_{21} + x_{22}$
	$C_{T1} = x_{11} + x_{21}$	$C_{T2} = x_{12} + x_{22}$	$T = x_{11} + x_{12} + x_{21} + x_{22}$

- R_1 and R_2 represent the Boolean states of an antecedent for the conclusions C_1 and C_2 (C_2 is the negation of C_1)
- x_{11}, x_{12}, x_{21}, and x_{22} represent the frequencies of each antecedent-consequent pair.
- R_{1T}, R_{2T}, C_{T1}, and C_{T2} are the marginal sums of the rows and columns, respectively.

The marginal sums and T, the total frequency of the table, are used to calculate expected cell values in test for independence.

Test for Independence

Given a contingency table of dimensions r by c (rows x columns):

1. Calculate the marginal sums.
2. Calculate the total frequency, T, using the marginal sums.
3. Calculate the expected frequencies for each cell: $e_{ij} = R_{iT} \cdot C_{Tj}/T$
4. Select the test to be used based on the highest expected frequency m
5. Calculate χ^2 using the chosen test
6. Calculate the degrees of freedom: $df = (r-1)(c-1)$
7. Use a chi-square table with χ^2 and df to determine if the conclusions are independent from the antecedent at the selected level of significance α (usually $\alpha = 0.05$).
   - $\chi^2 > \chi^2_\alpha$: reject the null hypothesis of independence (keep the antecedents)
   - $\chi^2 \leq \chi^2_\alpha$: accept the null hypothesis of independence (discard the antecedents)

A Complete Example

Suppose you get made and interview among you friends after their Summer holidays to classify sunburn risk factors:

Name	Hair	Height	Weight	Lotion	Result
Sarah	blonde	average	light	no	sunburned
Dana	blonde	tall	average	yes	none
Alex	brown	short	average	yes	none
Annie	blonde	short	average	no	sunburned
Emily	red	average	heavy	no	sunburned
Pete	brown	tall	heavy	no	none
John	brown	average	heavy	no	none
Katie	blonde	short	light	yes	none

- Can you set up a Decision Tree to predict if some one will get a sunburn? Can you prune the resulting tree into a few simple rules?
- Can you do that without using the **C4.5 free software**?

Classification Rules

What are Classification Rules?

Classification Rules or Decision Rules are classical IF-THEN rules:

- The IF part states a condition over the data
- The THEN part includes a class label

```
IF (lotion = no) THEN sunburn
```

Depending on the conditions we can have different kind of knowledge representation:

- Propositional, with attribute-value comparisons

  (Overcast = sunny) $\wedge$ (Temperature > 30) $\rightarrow$ PlayGolf

- First order Horn clauses, with variables: $L1 \wedge L2 \wedge L3 \wedge \ldots \wedge Ln \rightarrow H$
  - $H, L1, \ldots, Ln$ are positive literals (predicates applied to terms)
  - H is called head or consequent
  - $L1 \wedge L2 \wedge L3 \wedge \ldots \wedge Ln$ is called body or antecedents

  father(y,x) $\wedge$ female(x) $\rightarrow$ daughter(x,y)

Why shoud we use Classification Rules?

Inductive Learning Hypothesis: any hypothesis (h) found to approximate the target function (t) over a sufficiently large set of training examples (e) will also approximate the target function (t) well over other unobserved examples.

One of the most **expressive** and most **human readable** representation for hypotheses (i.e., models) is sets of IF-THEN rules.
They are also easy to use in Expert Systems.

Learning can be seen as exploring the Hypothesis Space

- **General to Specific:** Start with the most general hypothesis and then go on through specialization steps
- **Specific to General:** Start with the set of the most specific hypothesis and then go on through generalization steps

We'll come back to this soon! ;-)

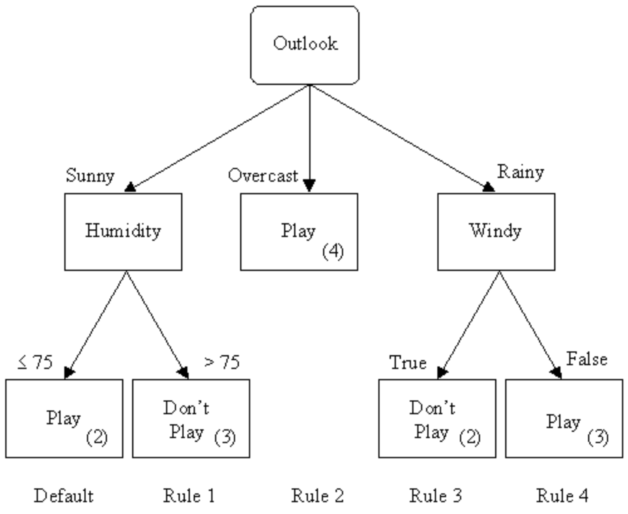
An Example: "Is this a nice day to play golf?"

Outlook	Temp	Humid.	Windy	Play
sunny	85	85	false	No
sunny	80	90	true	No
overcast	83	78	false	Yes
rain	70	96	false	Yes
rain	68	80	false	Yes
rain	65	70	true	No
overcast	64	65	true	Yes
sunny	72	95	false	No
sunny	69	70	false	Yes
rain	75	80	false	Yes
sunny	75	70	true	Yes
overcast	72	90	true	Yes
overcast	81	75	false	Yes
rain	71	80	true	No

How can we learn a set of rule from this data?

- Learn a Decision Tree, from the data then convert it into rules
- Use a Sequential Covering Algorithm

Extracting Rules from the Decision Tree



1. `if (outlook = overcast) then play`
2. `if (outlook = rain) and (windy = false) then play`
3. `if (outlook = sunny) and (humidity = high) then don't play`
4. `if (outlook = rain) and (windy = true) then don't play`
5. `Default class: play`

This is old stuff! Let's try something new!

Sequential Covering Algorithm

In order for a rule to be useful there are two pieces of information to be considered:

- **Accuracy:** How often is the rule correct?
- **Coverage:** How often does the rule apply?

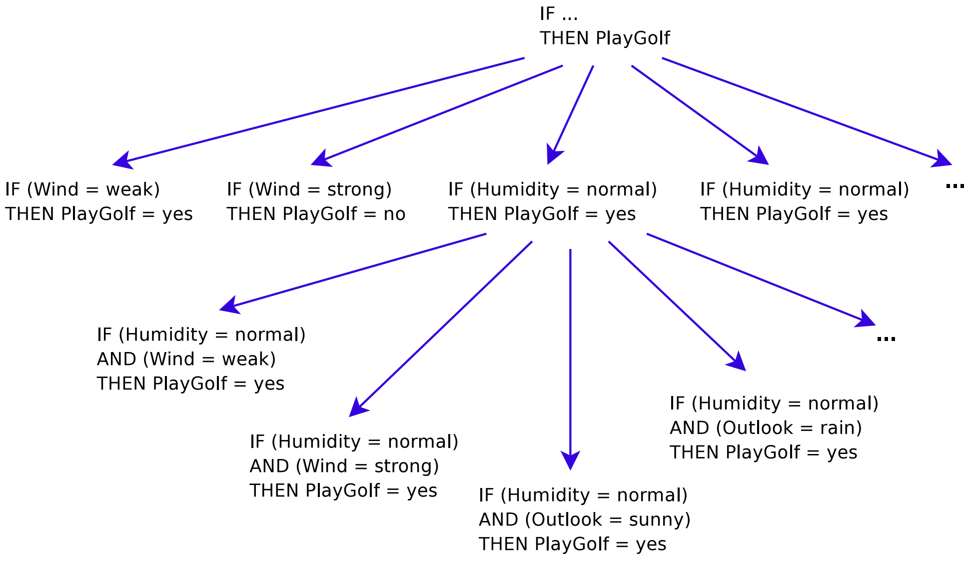
Considering this informations we can use a simple **Sequential Covering** algorithm:

- Consider the set E of positive and negative examples
- Repeat
  - Learn one rule with high accuracy, any coverage
  - Remove positive examples covered by this rule

  Until all the examples are covered

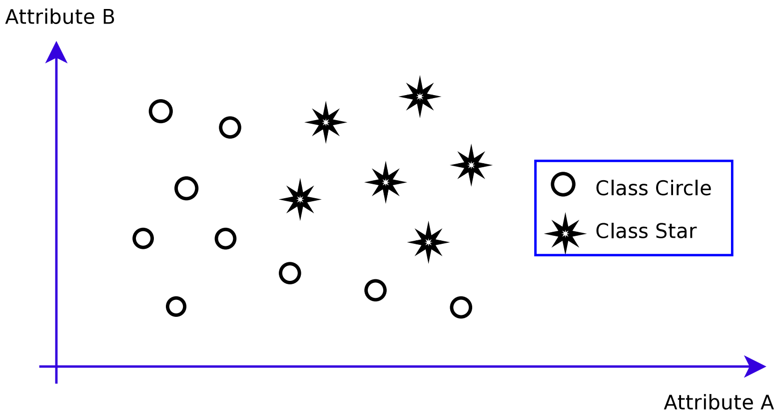
Let's figure out the algorithm!

Learning One Rule



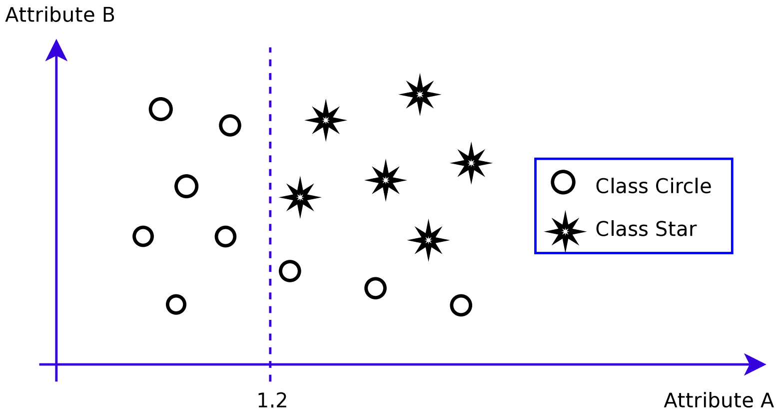
- Start from the most general rule, consisting of an empty condition
- Add tests on single attributes until the accuracy improves . . .

Learning One Rule Example: The `Star` Class



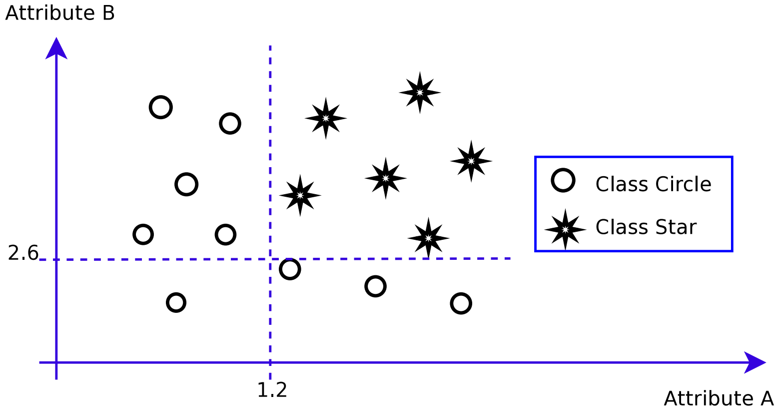

```
IF True
THEN Star
```

Learning One Rule Example: The Star Class



IF A>1.2
THEN Star

Learning One Rule Example: The Star Class



IF A>1.2
AND B>2.6
THEN Star

- Rule for `Star` class:
  - `IF A > 1.2 AND B > 2.6 THEN Star`
- Possible rules for `Circle` class:
  - `IF A < 1.2 THEN Circle`
  - `IF A > 1.2 AND B < 2.6 THEN Circle`

Contact Lens Example (I)

age	prescription	astigmatism	Tear production	lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
middle-aged	myope	no	reduced	none
middle-aged	myope	no	normal	soft
middle-aged	myope	yes	reduced	none
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	no	reduced	none
middle-aged	hypermetrope	no	normal	soft
middle-aged	hypermetrope	yes	reduced	none
middle-aged	hypermetrope	yes	normal	none
old	myope	no	reduced	none
old	myope	no	normal	none
old	myope	yes	reduced	none
old	myope	yes	normal	hard
old	hypermetrope	no	reduced	none
old	hypermetrope	no	normal	soft
old	hypermetrope	yes	reduced	none
old	hypermetrope	yes	normal	none

- The rule we seek:
  `IF ?`
  `THEN Recommendation = hard`
- Possible test:

`Age = Young`	2/8	
`Age = middle-aged`	1/8	
`Age = old`	1/8	
`Prescr. = myope`	3/12	
`Prescr. = hypermetrope`	1/12	
`Astigmatism = no`	0/12	
`Astigmatism = yes`	4/12	←
`Tear prod. = reduced`	0/12	
`Tear prod. = normal`	4/12	←

- With ties pick the one with higher coverage or random . . .

Contact Lens Example (II)

Actual rule: `IF Astigmatism = Yes THEN Recommendation = hard`

age	prescription	astigmatism	Tear production	lenses
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
middle-aged	myope	yes	reduced	none
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	yes	reduced	none
middle-aged	hypermetrope	yes	normal	none
old	myope	yes	reduced	none
old	myope	yes	normal	hard
old	hypermetrope	yes	reduced	none
old	hypermetrope	yes	normal	none

- Try to get something more accurate:
  `IF Astigmatism = Yes`
  `AND ?`
  `THEN Recommendation = hard`
- Possible test:

`Age = young`	2/4	
`Age = middle-aged`	1/4	
`Age = old`	1/4	
`Prescr. = myope`	3/6	
`Prescr. = hypermetrope`	1/6	
`Tear prod. = reduced`	0/6	
`Tear prod. = normal`	4/6	←

Contact Lens Example (III)

```
IF Astigmatism = Yes AND Tear_Production = normal
THEN Recommendation = hard
```

age	prescription	astigmatism	Tear production	lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	yes	normal	none
old	myope	yes	normal	hard
old	hypermetrope	yes	normal	none

- Try to get something even more accurate:

Age = young	2/2
Age = middle-aged	1/2
Age = old	1/2
Prescr. = myope	3/3 ←
Prescr. = hypermetrope	1/3

With ties prefer the attribute with higher coverage:

```
IF Astigmatism = Yes
AND Tear_Production = normal
AND Prescription = myope
THEN Recommendation = hard
```

age	prescription	astigmatism	Tear production	lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
middle-aged	myope	yes	normal	hard
old	myope	yes	normal	hard

Contact Lens Example (IV)

We can now check the coverage of the rule we have found

age	Spectacle prescription	astigmatism	Tear production rate	Recommended lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
middle-aged	myope	no	reduced	none
middle-aged	myope	no	normal	soft
middle-aged	myope	yes	reduced	none
middle-aged	myope	yes	normal	hard
middle-aged	hypermetrope	no	reduced	none
middle-aged	hypermetrope	no	normal	soft
middle-aged	hypermetrope	yes	reduced	none
middle-aged	hypermetrope	yes	normal	none
old	myope	no	reduced	none
old	myope	no	normal	none
old	myope	yes	reduced	none
old	myope	yes	normal	hard
old	hypermetrope	no	reduced	none
old	hypermetrope	no	normal	soft
old	hypermetrope	yes	reduced	none
old	hypermetrope	yes	normal	none

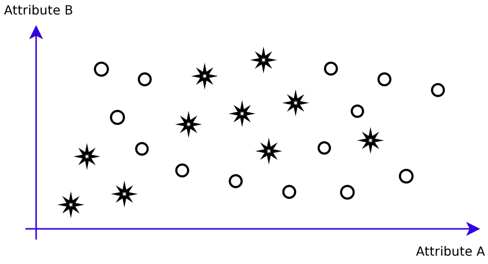
```
IF Astigmatism = Yes
AND Tear_Production = normal
AND Prescription = myope
THEN Recommendation = hard
```

Remove these cases from the dataset and a pply the learning process to the new dataset

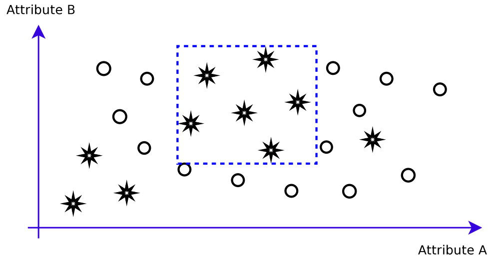
```
IF Age = young
AND Astigmatism = Yes
AND Tear_Production = normal
THEN Recommendation = hard
```

Can guess any flow of this algorithm?
Overfitting is behind the corner!

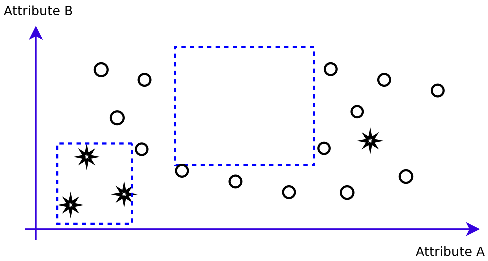
Overfitting the `Star` Class



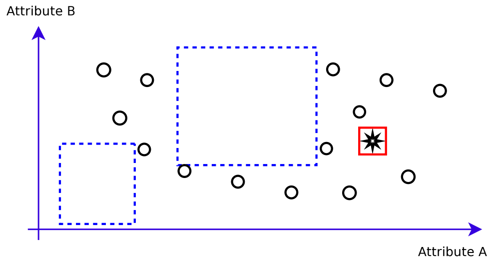
Suppose we have these data



Learn a rule



Remove covered and learn a new rule



Remove covered and learn a new rule

Watch out for too specific rules with low coverage!

Containing Overfitting

The learn-one-rule algorithm on the previous slides tries to learn a rule as accurate as possible. Such a rule may:

- be very complicated (containing many conjuncts)
- exhibit low predictive accuracy on unseen examples
- fit into noise

Use approximated, but simpler rules:

- Pre-pruning: Stop refinement of rules although still not accurate
  - **Minimum Purity Criterion:** Requires a certain percentage of the examples covered by the rules is positive
  - **Significance Testing:** Verify significant differences between the distribution of instances covered by a rule and its direct predecessor.
- Post-pruning: Learn “pure” rules than remove some attributes

The latter used to work better, let’s focus on it!

Post-Pruning of Classification Rules

The general idea of Post-Pruning is to learn a “pure” rule first then remove some attribute value pairs from the rule to make it more general:

- Method 1: Separate the training data into training set and validation set; learn rules from the training set and test the accuracy of the pruned rule on a validation set.
- Method 2: Use a rule quality measure, test the quality of the pruned rule on the same training set from which is rule was learned:
  - Compute a rule quality value: Q(R)
  - Check each attribute-value pair L in R to see if removal of L decreases Q(R)
  - If not, L is removed (i.e., R is generalized)
  - Repeat the process until no further improvement is possible

For instance, ELEM2 uses rule quality formula: $Q(r) = \log \frac{P(r|c)(1-P(r|\overline{c})}{P(r|\overline{c})(1-P(r|c)}$.

Using Classification Rules

When matching a new example with a set of rules:

1. Single-match: only one rule is matched with the example then classify the example into the class indicated by the rule.
2. Multiple-match: multiple rules are matched with the example:
   - If rules indicate the same class then classify into that class.
   - When matched rules indicate different classes:
     - Method 1: Rank the rules, use the first matched rule to classify
     - Method 2: Compute a decision score for each of the involved classes: $DS(C) = \sum_i Q(r_i)$ where $Q(r_i)$ is a quality measure of rule r_i choose the one with the highest decision score.
3. No-match: there is no matching rule
   - Method 1: Use a default (majority class) to classify
   - Method 2: Partial matching is performed. Calculate a matching score for each partially matched rule and a decision score for each involved class. Choose the class with the highest decision score.

Classification Rules vs. Decision Trees

They can represent the same kind of knowledge:

- Decision rules are easier to understand (human readable)
- Rules are more flexible than decision trees
  - No overlapping among branches in a decision tree so they do not suffer from replicated subtrees
  - Branches in a decision tree share at least one attribute (you can always translate a tree into a rule but not the viceversa)
- In multi-class situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account
- Sequential covering maximizes single rule's accuracy reducing its coverage while decision tree maximizes overall purity
- If-then rules can be used with expert systems

Still remains for both of them the issue of selecting the *best attribute* and dealing with *missing values* or *continuous attributes*.

Selecting Attribute-Value Pairs

Various methods for selecting attributes have been proposed with the goal of improve rule accuracy

- Training accuracy of the rule on the training example (AQ15): p/t
  - t instances covered by rule, p number of these that are positive
  - Produce rules for positive instances as quickly as possible
  - May produce rules with very small coverage
- Information gain (CN2, PRISM): $\log_2 p/t - \log_2 P/T$
  - P and T are the positive and total number of examples that satisfied the previous rule
  - Equivalent to p/t
- Information gain with coverage (FOIL): $p(\log_2 p/t - \log_2 P/T)$
  - P and T are the positive and total number of examples that satisfied the previous rule
  - *Coverage* is also considered in the evaluation.

Dealing with Missing Values

In certain cases, the available data may be missing values for some attributes, these records will fail any test.

There a few strategies for dealing with the missing attribute value:

- Treat "missing" as a separate value
- Assign it the value that is most common among training examples
- Assign it the most common value among examples that have the same classification
- Assign a probability to each of the possible values rather than simply assigning the most common one. These probabilities can be estimated based on the observed frequencies among the examples.

Given a Boolean attribute A, if we have 6 known examples with A = 1 and 4 with A = 0, then we would say the probability that A(x) = 1 is 0.6, and the probability that A(x) = 0 is 0.4. A fractional 0.6 of records can be assumed to have A = 1 and the rest A = 0. (C4.5)

How can I use the classifiers learned this way?

Dealing with Continuous Values (I)

Discretize numeric attributes by dividing each of them into intervals:

- Sort instances according to attribute's values
- Place breakpoints where the class changes (the majority class)
- This minimizes the total error

Example: Temperature from weather data

Outlook	Temp.	Humid.	Windy	Play
sunny	85	85	false	No
sunny	80	90	true	No
overcast	83	86	false	Yes
rain	75	80	false	Yes
...	...	...	...	...

- Very sensitive to noise
- Unique ID attributes will have zero errors
- Incorrect exmaple classisi-cations induce splits

64 | 65 | 68 69 70 | 71 72 72 | 75 75 | 80 | 81 83 | 85
Yes | No | Yes Yes Yes | No No Yes | Yes Yes | No | Yes Yes | No

Dealing with Continuous Values (II)

Example: Temperature from weather data

Outlook	Temp.	Humid.	Windy	Play
sunny	85	85	false	No
sunny	80	90	true	No
overcast	83	86	false	Yes
rain	75	80	false	Yes
...	...	...	...	...

- Very sensitive to noise
- Unique ID attributes will have zero errors
- Incorrect exmaple classisi-cations induce splits

64 | 65 | 68 69 70 | 71 72 72 | 75 75 | 80 | 81 83 | 85
Yes |No |YesYesYes|NoNoYes |YesYes |No |YesYes |No

Simple solution: enforce a minimum number of majority class instances per each interval; for instance, with min = 3) we get:

64 65 68 69 70 | 71 72 72 75 75 | 80 81 83 85
YesNoYesYesYes|NoNoYesYesYes |NoYesYesNo

Rule Learning Alternatives

The Sequential Learning Algorithm is non the only learning algorithm we can use to learn rules from a dataset of examples:

- The **1R** algorithm learns a 1-level decision tree, i.e., rules that all test one particular attribute
  - One branch for each value, each branch assigns most frequent class
  - Choose attribute with lowest error rate (i.e., proportion of instances that do not belong to the majority class of their corresponding branch)
- The **RISE** algorithm (Rule Induction from a Set of Exemplars) works in a specific-to-general approach:
  - Initially, it creates one rule for each training example
  - Then it goes on through elementary generalization steps until the overall accuracy does not decrease

Check for their pseudo-code!