Rule-Based Classifiers

1 Propositional Logic

In Propositional Logic, we're interested in analysing *statements* about the world, and these statements will be either T (true) or F (false). (This is a major simplifying assumption made in Propositional Logic: it excludes the possibility of statements that are, e.g., only partly true. To handle partly-true statements, you have to abandon Propositional Logic and turn to one of the many other varieties of logic, e.g. Fuzzy Logic — covered later in this module.)

For conciseness, we do not use English to write atomic statements such as 'There is an obstacle in front of the robot'. Instead, we give atomic statements short names such as $p, p_0, p_1, \ldots, q, q_0, q_1, \ldots$ We refer to these as *propositional symbols*.

We can construct compound statements from simpler statements by combining propositional symbols into larger expressions using operators that we call *truth-functional connectives*.

Syntax. We define the *well-formed formulae* (wffs) of propositional logic as follows:

1. true and false are wffs.
2. Every propositional symbol is a wff.
3. For any wffs W, W_1 and W_2, the following are also wffs:

(W)	
$\neg W$	negation
$W_1 \wedge W_2$	conjunction; W_1 and W_2 are the conjuncts
$W_1 \vee W_2$	disjunction; W_1 and W_2 are the disjuncts
$W_1 \Rightarrow W_2$	conditional; W_1 is the antecedent; W_2 is the consequent
$W_1 \Leftrightarrow W_2$	biconditional

4. No other string of symbols is a wff.

Semantics. Wffs may be T or F.

1. The wffs true and false are always T and F, respectively.
2. The truth-value of a propositional symbol will be stipulated. A stipulation of truth-values for propositional symbols is called an *interpretation*, and is written $\mathcal{I}$, e.g. $\mathcal{I}(p) = \text{true}$. Often, we'll consider all possible interpretations, shown as a *truth-table*.
3. The truth-values of a compound wff can be determined from the truth-values of its component wffs.

   Here are all 16 possible logic functions of two wffs:

W_1	W_2	true	$W_1 \vee W_2$	$W_1 \Downarrow W_2$	W_1	$W_1 \Uparrow W_2$	W_2	$W_1 \Updownarrow W_2$	$W_1 \wedge W_2$	$W_1 \mid W_2$	$W_1 \oplus W_2$	$\neg W_2$	$W_1 > W_2$	$\neg W_1$	$W_1 < W_2$	$W_1 \downarrow W_2$	false
T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F

For completeness, the table includes all 16 possible logic functions of two variables. However, we continue to focus on $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$.

Meta-theory. Inspection of a truth-table allows us to say things about a wff of propositional logic:

- A wff is *satisfiable* iff it is T in *at least one interpretation* (at least one row in the truth-table).
- A wff is *unsatisfiable* iff it is T in *no interpretation* (no row).
- A wff is *valid* iff it is T in *every interpretation* (every row).
- Two wffs, W_1 and W_2, are *logically equivalent*, written $W_1 \equiv W_2$, iff they are T in exactly the same interpretations (exactly the same rows).
- A set of wffs Φ has wff W as a *logical consequence*, written $\Phi \models W$, iff in every interpretation in which all members of Φ are T then W is also T. (In every row where all the members of Φ are T, W is also T.)

The last of these is particularly important because it is the basis of logical reasoning. So we will give an example.

Let

$$\begin{array}{lll} p & =_{\text{def}} & \text{It is raining} \\ q & =_{\text{def}} & \text{I get wet} \\ r & =_{\text{def}} & \text{I catch cold} \end{array}$$

You are given the following premisses:

1. If it's raining, I get wet: $p \Rightarrow q$
2. If I get wet, I catch cold: $q \Rightarrow r$
3. It is raining: p

Show I catch cold.

What we want to show is

$$\{p \Rightarrow q, q \Rightarrow r, p\} \models q$$

We can draw up a truth-table and then check that the definition of logical consequence holds.

			premisses			conclusion
p	q	r	$p \Rightarrow q$	$q \Rightarrow r$	p	q
T	T	T	T	T	T	T
T	T	F	T	F	T	T
T	F	T	F	T	T	F
T	F	F	F	T	T	F
F	T	T	T	T	F	T
F	T	F	T	F	F	T
F	T	T	T	T	F	F
F	T	F	T	T	F	F

In every interpretation in which all the premisses are true (in this case, just row 1), the conclusion is also true.

Proof Theory. A proof theory enables us to derive conclusions from a set of wffs by *syntactic operations* alone. We manipulate the wffs without reference to their semantics. If the manipulation rules are 'right', the new wffs we generate will, in fact, also be exactly the logical consequences of the original set of wffs.

A proof theory comprises a finite set of *inference rules* (and a finite set of *logical axiom schemata*, which we will ignore here). An inference rule comprises a set of patterns called *conditions* and another pattern called the *conclusion*. Here are the only rules we will use in this lecture. The first is known as $\Rightarrow$-elimination and also as Modus Ponens:

$$\frac{W_1, W_1 \Rightarrow W_2}{W_2}$$

The second is known as $\wedge$-introduction:

$$\frac{W_1, W_2}{W_1 \wedge W_2}$$

Above the line are the conditions; below is the conclusion. If you have wffs that match the conditions, then you can, in a single step, derive a wff that matches the conclusion.

Given a set of inference rules and a set of premisses, Φ, we can try to *derive* a wff W from Φ by repeated applications of the inference rules. If there exists a derivation of W from Φ then we say that W is *derivable* from Φ, and we write this as:

$$\Phi \vdash W$$

We can illustrate this idea, using the same example from above:

1. $p \Rightarrow q$ (premiss: If it's raining, I get wet)
2. $q \Rightarrow r$ (premiss: If I get wet, I catch cold)
3. p (premiss: It is raining)
4. q ($\Rightarrow$-elimination using (1) and (3))
5. r ($\Rightarrow$-elimination using (2) and (4))

2 Propositional Definite Clauses

In the rest of the lecture we will be using a restricted form of propositional logic. This logic is less expressive than full propositional logic, but still very useful and therefore widely-used.

Syntax. All wffs must be *propositional definite clauses*.

1. Every propositional symbol is a propositional definite clause. These simple propositional definite clauses are informally referred to as *facts*.
2. If $p_1, \ldots, p_n$ and q are propositional symbols, then

   $$(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$$

   is a propositional definite clause. These propositional definite clauses are informally referred to as *rules*.

Class exercise. You can see that one of the restrictions is that we are only allowed conjunction in the antecedents of rules. For example, the following is not a propositional definite clause:

$$(p_1 \vee p_2) \Rightarrow q$$

Why is it no great loss that antecedents cannot be disjunctions?

3 Classification using Rules

A rule-based classifier will comprise a knowledge base that contains facts and rules and an inference engine for drawing conclusions from the knowledge base.

3.1 Knowledge Base

A rule-based classifier will have a *knowledge base* that contains propositional definite clauses (i.e. facts and rules) about the subject at hand. Suppose we were building a medical rule-based classifier for deciding whether a patient should be admitted into hospital or not. The rules would encode associations between symptoms and medical conditions. The facts would encode knowledge about the current patient.

We'll see how to use a knowledge base of propositional definite clauses to perform a classification task. To begin with, we'll assume there are only two classes. The rules will determine whether an object belongs to one of the two classes. If it cannot be proved that an object belongs to that class, we assume it belongs to the other class.

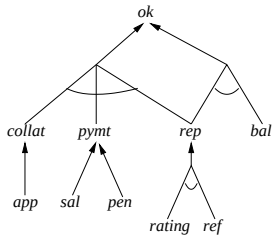
We'll give an example knowledge base for deciding whether to grant a loan to an individual.[1] For readability, it doesn't uses p's and q's! It uses the following propositional symbols:

ok	$=_{\text{def}}$	the loan should be approved
collat	$=_{\text{def}}$	the collateral for the loan is satisfactory
pymt	$=_{\text{def}}$	the applicant is able to make the loan repayments
rep	$=_{\text{def}}$	the applicant has a good financial reputation
app	$=_{\text{def}}$	the appraisal on the collateral is sufficiently greater than the loan amount
rating	$=_{\text{def}}$	the applicant has a good credit rating
ref	$=_{\text{def}}$	the applicant has a good reference from a financial institution
sal	$=_{\text{def}}$	the applicant has a good, steady income
pen	$=_{\text{def}}$	the applicant has a high pension
bal	$=_{\text{def}}$	the applicant has an excellent balance sheet

Here are the rules:

$$(collat \wedge pymt \wedge rep) \Rightarrow ok$$
$$(bal \wedge rep) \Rightarrow ok$$
$$app \Rightarrow collat$$
$$(rating \wedge ref) \Rightarrow rep$$
$$sal \Rightarrow pymt$$
$$pen \Rightarrow pymt$$

Notice how the rules chain together: the consequent of one rule may appear in the antecedent of another. We can depict this chaining graphically as an AND-OR graph.



[1] This is a minor extension of the knowledge base given in N.J. Nilsson: *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann, 1998.

In the graph, there are two types of node:

AND-nodes: Incoming links are joined by an arc. This indicates conjunction —every link must be proved.

OR-nodes: If there are multiple incoming links, they are not joined by an arc. This indicates disjunction —one of the links must be proved.

3.2 The Inference Engine

The inference engine will draw conclusions using the knowledge in the knowledge base. The work done by the inference engine of a rule-based system is referred to as *rule-based reasoning* (RBR). From a logic point of view, it derives wffs using the two inference rules mentioned earlier ($\Rightarrow$-elimination and $\wedge$-introduction).

Another perspective on rule-based reasoning is that it is doing a search of the AND-OR graph. Effectively, we are looking for an AND-OR tree within the AND-OR graph. The tree must connect the root of the graph with some of the leaves of the graph, as follows:

- If a node is an OR-node, it is sufficient for the AND-OR tree to include only one of the node's descendants.
- If a node is an AND-node, it is necessary for the AND-OR tree to include each of the node's descendants.
- If a node is a leaf, the node must represent a fact that is in the knowledge base.

The inference engine searches for this AND-OR tree using either forwards-chaining or backwards-chaining (or both).

3.2.1 Forwards-chaining

The first method we will look at goes by the name of *forwards-chaining*, although it is also called *bottom-up reasoning* and *data-driven reasoning*.

In terms of the AND-OR graph, this kind of reasoning starts at those leaves that match facts in the knowledge base and tries to build a tree from those leaves to the root of the graph.

Here is a simple (not necessarily very efficient) algorithm for forwards-chaining. To invoke the algorithm, you supply it with a list of goals. This list should include just the symbol at the root of the AND-OR graph. (Clearly then, it does not need to be a *list*: it includes only one symbol. Having it as a list gives consistency with the algorithm given in section 3.2.2 and is exploited in section 3.3.1.)

Algorithm: FORWARDSCHAINING(*GoalList*)

```
do
{   new := { };
    for each rule (p1∧, ..., ∧pn) ⇒ q ∈ KB
    {   if {p1, ..., pn} ⊆ KB
        {   if q ∉ new ∧ q ∉ KB
            {   insert q into new;
                if q is a member of GoalList
                {   return true;
                }
            }
        }
    }
    copy the members of new into KB;
}
while new ≠ ∅;
return false;
```

KB is the knowledge base.

Let's assume that four facts are inserted into the loans knowledge base:

sal
rating
ref
app

i.e. the current applicant has a good salary, a good credit rating, a good reference. and the appraisal on the collateral is sufficiently high. Does s/he get a loan?

In the lecture, we will trace the operation of the algorithm. We will invoke FORWARDSCHAINING([*ok*]) to see whether this applicant gets a loan. From this you should understand why phrases such as forwards-chaining, bottom-up reasoning and data-driven reasoning are used.

3.2.2 Backwards-chaining

The second method we look at goes by the names *backwards-chaining*, *top-down reasoning*, *goal-driven reasoning* and *hypothesis-driven reasoning*.

In terms of the AND-OR graph, this kind of reasoning starts at the root of the graph and tries to build a tree from the root to leaves that match facts in the knowledge base.

Here is a simple (not necessarily very efficient) algorithm for backwards-chaining. To invoke the algorithm, you supply it with a list of goals. Initially, this list should include just the symbol at the root of the AND-OR graph. However, the algorithm is recursive. The propositions needed to establish the original goal become a list of subgoals, which replace the original goal, and this list is fed into a recursive invocation of the algorithm. Each of these subgoals will be replaced in turn by their own subgoals, and so forth. If a subgoal matches a fact, then it can be removed from the list of (sub)goals. If the list of (sub)goals becomes empty, then we have succeeded in establishing the original goal.

Algorithm: BACKWARDSCHAINING(*GoalList*)

if *GoalList* is empty
{ **return** true;
}
currentGoal := head of *GoalList*;
for each clause $c \in KB$
{ **if** c is a fact and $c =$ *currentGoal*
{ *newGoalList* := tail of *GoalList*;
if BACKWARDSCHAINING(*newGoalList*)
{ **return** true;
}
}
else if c is a rule of the form $(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$ and $q =$ *currentGoal*
{ *newGoalList* := $p_1, \ldots, p_n$ and tail of *GoalList*;
if BACKWARDSCHAINING(*newGoalList*)
{ **return** true;
}
}
}
return false;

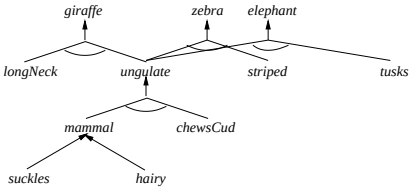
In the lecture, we'll trace the operation of the algorithm using the loan knowledge base, including the four facts given above. Again we will invoke BACKWARDSCHAINING([*ok*]). From this you should understand why phrases such as backwards-chaining, top-down reasoning, goal-driven reasoning and hypothesis-driven reasoning are used.

Class Exercise. Suppose the knowledge base contains a rule of the form $p \Rightarrow p$ or a chain of rules with a similar effect. Does such a rule cause a problem for (a) this backwards-chaining algorithm; (b) for the forwards-chaining algorithm?

3.3 Discussion

3.3.1 More than two classes

A rule-base can contains rules that establish membership of as many classes as you wish. For example, here is a simple animal identification rule-base, which can classify animals as giraffes, zebras or elephants. Note that the AND-OR graph has more than one root.



suckles $\Rightarrow$ *mammal*
hairy $\Rightarrow$ *mammal*
(*mammal* $\wedge$ *chewsCud*) $\Rightarrow$ *ungulate*
(*ungulate* $\wedge$ *longNeck*) $\Rightarrow$ *giraffe*
(*ungulate* $\wedge$ *striped*) $\Rightarrow$ *zebra*
(*ungulate* $\wedge$ *tusks*) $\Rightarrow$ *elephant*

The forwards-chaining algorithm easily accommodates this example. We invoke it with a list of the classes as its goals, FORWARDSCHAINING([*giraffe*, *zebra*, *elephant*]). It terminates as soon as any one of these gets established, and we can check to see which one was established.

The easiest way to use the backwards-chaining algorithm is to invoke it repeatedly. First invoke BACKWARDSCHAINING([*giraffe*]); if it returns true, then you can conclude the animal is a giraffe. If it return false, then invoke BACKWARDSCHAINING([*zebra*]); if it returns true, then you can conclude the animal is a zebra. If it returns false, finally invoke BACKWARDSCHAINING([*elephant*]); if it returns true, then the animal is an elephant. If it returns false, this animal could not be classified using these rules.

3.3.2 Syntactic variations

You can use attribute-value pairs in places where we used propositional symbols, e.g.:

$$(\textit{legs} = 4 \wedge \textit{tusks} = 2 \wedge \textit{hairy} = \textit{yes}) \Rightarrow \textit{class} = \textit{elephant}$$

This might look better, but it adds no real expressive power.

There are, of course, other variations which *would* add expressive power. Some of them might even take us beyond propositional logic into predicate logic. We could not then expect our forwards- and backwards-chaining algorithms to work without modification.

3.3.3 Extralogical variations

Once you move to attribute-value pairs, you might be tempted to allow relations other than equality, especially in the antecedents of the rules, e.g.:

$$\textit{sal} > 30000 \Rightarrow \textit{pymt}$$

Suppose one of the facts about the current loan applicant is $\textit{sal} = 32000$, then you will presumably intend that the system can derive *pymt*. This is going beyond what $\Rightarrow$-elimination can do for you! $\Rightarrow$-elimination requires an exact match.

To see that the rule's antecedent, $\textit{sal} > 30000$, and this fact, $\textit{sal} = 32000$, do 'match' requires you to 'step outside' of logic and exploit the arithmetic facilities of your machine. Your system will no longer be purely logical; it uses some *extralogical* facilities (where 'extra-logical' means 'outside of logic'). This might be very convenient in your task domain, and there's nothing to stop you from building a system that works in this way.

3.3.4 Interactive systems

Rule-based systems have been successfully used in practice, and for tasks other than classification too. In practice, they are most usually interactive systems. This means that, while we will assume that all the *rules* we need are already in the knowledge base, we do not assume that all the *facts* are known and stored already in the knowledge base. Instead, the user will supply the facts interactively during the session.

In forwards-chaining systems, the user is asked to supply some facts. The system then does as much forwards-chaining using these as it can. If the desired conclusion has not been established, then the user might be prompted for more facts, and another round of forwards-chaining begins.

In backwards-chaining systems, when the system is trying to see whether a leaf node (fact) is true or not, it will check the knowledge base but, if it is not in the knowledge base, then it will ask the user a question to determine whether the fact is true or false. The user's answer can be added to the knowledge base.

Some systems work exclusively using one or other of backwards-chaining or forwards-chaining. But many interleave the two. This can be very natural for an interactive system. Consider a consultation with a human doctor. The patient describes some symptoms. Conclusions are drawn (maybe only tentatively) using forwards-chaining. The doctor selects a hypothesis and, through backwards-chaining, arrives at a question that is addressed to the patient. The patient speaks again, perhaps answering the question and/or volunteering other information. And so the process repeats.

3.3.5 Explanations

It is often important that a system be able to explain its reasoning. Rule-based systems usually offer two kinds of explanation, known as *why* and *how* explanations.

A *why* explanation is an *ascent* of the AND-OR graph. It shows why it is useful to establish that a node in the graph is true: it shows which rules could fire.

A *how* explanation is a *descent* of the successful parts of the AND-OR graph. It shows how a node was established to be true: it shows which rules did fire in reaching that conclusion.

3.3.6 Where do the rules come from?

Rules are little nuggets of knowledge. It was originally thought that human experts could easily provide us with knowledge expressed in rule form. On the whole, this has proved not to be true. The first handful of rules might be easily acquired, but it becomes increasingly hard to obtain more rules to improve the coverage of the knowledge base. Different experts may disagree. Experts may be unable to articulate their knowledge. The brake this put onto the development of expert systems became known as the *knowledge acquisition bottleneck*.

Increasingly then, rules are learned from datasets. Unfortunately, we do not have time to look at the algorithms for doing this.