Knowledge Engineering

Semester 2, 2004-05

Michael Rovatsos
mrovatso@inf.ed.ac.uk

School of informatics

Lecture 19 – All questions answered (?)
18th March 2005

Today' lecture

- First part: Summary of the course
- Second part: Some advice for revision
- Third part: Answer some questions

Objective of the Course

- To describe AI techniques that can be used to engineer knowledge-based systems
- Focus on methods that manipulate symbolic representations of knowledge
  - Acquire knowledge from examples
  - Represent knowledge and perform inference tasks
  - Reason over distributed sources of knowledge
  - Enable interaction between distributed reasoning systems
  - Modify existing knowledge using new information
- Concentrate on methods for automating these tasks using "intelligent" techniques

Method

- Covered a very wide range of material (naturally, this involves losing out on the "depth" side)
- Attempted to describe algorithms where possible, and to enable you to use them without the burden of detailed analysis and specification
- Usually this was done using illustrative examples and analysing what the algorithms would do on them
- However, there are also theoretical insights that involve a deep understanding of the methods introduced
  - Examples: the frame problem, non-monotonic reasoning, properties of auction protocols, differences btw. different methods for representing uncertainty, etc.

Introduction

- Foundation for understanding of the area of Knowledge Engineering
- What is knowledge?
- Discussed different types of knowledge
- Knowledge-based systems
- The Knowledge Engineering Process
- How does human activity come into play?

Knowledge Acquisition

- Focused on automated discovery of knowledge from examples
- Defined the inductive learning problem
- Decision Tree Learning
  - A simple method for attribute-based learning
- Version-Space Learning
  - A more advanced method for learning using generalisation hierarchies
- General issues: Ockham's razor, noise & overfitting, etc.

Knowledge Representation & Reasoning

- Started with a recap of foundations of logic
- Introduced ontological reasoning as a key element of KR & R
- Discussed a number of interesting aspects of KR
- Category reasoning systems
  - Multiple inheritance, default/non-monotonic reasoning
- Reasoning about change
  - Frame problems and their solution
- Reasoning with uncertainty
  - Probabilistic reasoning, fuzzy logic, Dempster-Shafer theory
- Model-based reasoning as a case study for practical KR & R systems

Knowledge Synthesis

- How do we integrate different kinds of knowledge?
- A case study for traditional, closed KBS: Amphion
- Then moved on to open systems
- Agents & Multiagent Systems
  - What are agents and what are suitable architectures for building them?
  - How do KBS interact with each other? What are appropriate models of interaction and agent communication?
  - Distributed rational decision making: how can we build systems that satisfy certain global properties despite agents being self-interested?
- KE & The Semantic Web
  - Reasoning about knowledge on the Web
  - Assessing Semantic Web languages wrt their use in KR & R
  - "Open world" assumptions

Knowledge Evolution

- How do we improve our existing knowledge with new information?
- Parallel to knowledge acquisition, but more general case
- Truth maintenance systems: how can we update our beliefs about the world in an efficient way?
- Knowledge in learning: can we extend inductive learning methods to make use of background knowledge?
- A simple method: explanation-based learning
- The "queen" of inductive learning methods: inductive logic programming
  - In particular: Making new discoveries with ILP

The Essence (of it all)

- Symbolic AI techniques offer a variety of methods that can be used to address key Knowledge Engineering issues
  - These can aid in the automation of all aspects of KE
  - Particular advantage over other computational methods: systems are understandable for humans!
- The shift from closed to open systems
  - Today's requirements are very different from those of the "expert systems" era
  - Focus on agents/multiagent systems and the Semantic Web
  - Networked world requires a whole new way of thinking!
- In the real world, you will still find that there is not sufficient acceptance for many methods presented here
  - "The AI curse" – if it works, it can't be AI and vice versa
  - Large initial investments scare industry
  - Many methods still restricted to the academic realm, don't scale to real-world problems

What is important

- Develop a basic understanding of the problems
  - When we build a system, how do we get the knowledge into the system, how is it used and how is it updated?
- Understand how the discussed techniques work
  - Ability to solve a simple problem using each of the algorithms
  - Discuss their properties (e.g. why is a certain strategy optimal in a certain interaction protocol?)
- Develop a critical view of each of the techniques
  - What are the pros and cons of each method?
  - When would you use which one?
  - Some are much more theoretical than others!
- View all that in the context of integrated KBS
  - Example: A KBS might consist of interacting agents who use ontologies to reason about Semantic Web content, their internal representations might require modelling uncertainty and belief revision, and they should also learn from experience

A Suggested Checklist

1. Create a "mental outline" of the material for yourself, make sure you understand the different aspects of KE and its foundational concepts
2. Make sure you can say something about each of the methods covered in the course (know what it is, informal description)
3. Try them out on simple examples (where algorithms were presented), don't worry about weird special cases or knowing complex formulae by heart
4. Think about advanced matters (properties of certain methods, advantages/disadvantages, connections between different methods and sub-areas)

Things you won't need

- You don't have to be able to write down pseudo-code definitions of complex algorithms
- You don't have to think about extreme, special cases
- Naturally, no single task will have to be performed that will take more than ten to fifteen minutes
- This also means that it is unlikely that you will be asked to solve an entire problem by applying some algorithm!
- Don't memorise complex formulae!

Question 1

Why are ontologies useful?

Question 1

- Ontologies allow for structuring knowledge about objects and relationships that are to be dealt with in reasoning about a particular application domain
- Require the knowledge engineer to think about a domain in a principled way
- Ontological engineering makes ontological commitments explicit
- This also means thinking about how to represent "special" kinds of knowledge
  - Categories and taxonomies, measurements, substances and object, action and change, beliefs and mental states, etc.

Question 1

- Ontologies enable knowledge reuse if they reflect a shared understanding of a domain
- More visionary, long-term goal: develop a widely accepted general ontology of human common-sense knowledge
- Philosophical perspective: an ontology is ultimately a theory of the things that exist and how they are connected to each other

Question 2

Why is $true \Rightarrow Grandfather(x, y)$ used as a most general hypothesis in the ILP example rather than $false \Rightarrow Grandfather(x, y)$?

Question 2: Most general hypotheses in ILP

Example: we are trying to learn the *Grandfather* relation

1. Split examples into positive and negative ones (12/388):
   +: $\langle Mum, Charles\rangle$, $\langle Elizabeth, Beatrice\rangle$
   -: $\langle Mum, Harry\rangle$, $\langle Spencer, Peter\rangle$
2. Construct a set of clauses, each with $Grandfather(x, y)$ as a head
   - Start with $true \Rightarrow Grandfather(x, y)$
   - This classifies negative examples as true, specialise it
   - Generate possible hypotheses by adding a literal to the LHS:
     $$Father(x, y) \Rightarrow Grandfather(x, y)$$
     $$Parent(x, y) \Rightarrow Grandfather(x, y)$$
     $$Father(x, z) \Rightarrow Grandfather(x, y)$$
   - Prefer the one that classifies most data correctly (here: the third one)
3. Repeat these steps until all data is correctly classified

Question 3

When is it best to use Decision Tree Learning rather than Version Space Learning and vice versa?

Question 3

- Prefer version space learning (VSL) if you want to use the hypothesis directly in your KBS (decision tree learning (DTL) generates trees that cannot be used directly, e.g., FOL-based reasoning systems)
- Unlike DTL, VSL is an incremental learning algorithm
  - A DT is generated with the entire data and is fixed after this training phase, hard to make use of new examples
- VSL fails if data is noisy or target concept is not in hypothesis space, DTL doesn't
  - Inconsistent data makes the version space collapse, in DTL we can still use majority vote or default classifications
- VSL requires a generalisation hierarchy
  - This may not be available of may allow for too many degrees of freedom, i.e. generate too many specialisations/generalisations in a single step

Decision Theory

- A theory of (single-agent) rational decision making
- Based on a set of alternatives, what is the optimal decision an agent may make?
- Informally speaking, this depends on how desirable an alternative see and how likely we think it is
  - decision theory = utility theory + probability theory
- Let $O = \{o_1, \ldots o_n\}$ a set of possible outcomes (e.g. possible "runs" of the system until final states are reached)
- A **preference ordering** $\succ_i \subseteq O \times O$ for agent i is an antisymmetric, transitive relation on O, i.e.
  - $o \succ_i o' \Rightarrow o' \not\succ_i o$
  - $o \succ_i o' \wedge o' \succ o'' \Rightarrow o \succ_i o''$
- Such an ordering can be used to express strict preferences of an agent over O (write $\succeq_i$ if also reflexive, i.e. $o \succeq_i o$)

Decision Theory

- Preferences are often expressed through a **utility function** $u_i : O \to \mathbb{R}$:

$$u_i(o) > u_i(o') \Leftrightarrow o \succ o', \quad u_i(o) \geq u_i(o') \Leftrightarrow o \succeq o'$$

- Principle of **expected utility maximisation**:

$$a^* = \arg\max_{a \in A} \sum_{o \in O} P(o|a)u(o)$$

  where $a \in A$ are the actions/decisions an agent may take
- Generally accepted criterion, but also problems:
  - Incomplete information (wrt outcomes, probabilities, preferences)
  - Risk aversion attitude (value of additional utility depending on current "wealth", e.g. money)
  - Quantification problem (optimal=maximising average utility?, comparability of different utility values)

Game Theory

- Application of decision-theoretic principles to interaction among several agents
- Basic model: agents perform simultaneous actions (potentially over several stages), the actual outcome depends on the combination of action chosen by all agents
- **Normal-form games**: final result reached in single step (in contrast to **extensive-form games**)
  - Agents $\{1, \ldots, n\}$, S_i=set of (pure) **strategies** for agent i, $S = \times_{i=1}^{n} S_i$ space of **joint strategies**
  - Utility functions $u_i : S \to \mathbb{R}$ map joint strategies to utilities
  - A probability distribution $\sigma_i : S_i \to [0, 1]$ is called a **mixed strategy** of agent i (can be extended to joint strategies)
- Game theory is concerned with the study of this kind of games (in particular developing solution concepts for games)

Dominance and Best Response Strategies

- Two simple and very common criteria for rational decision making in games
- Strategy $s \in S_i$ is said to **dominate** $s' \in S_i$ iff

$$\forall s_{-i} \in S_{-i} \quad u_i(s, s_{-i}) \geq u_i(s', s_{-i})$$

  ($s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$, same abbrev. used for S)
- Dominated strategies can be safely deleted from the set of strategies, a rational agent will never play them
- Some games are solvable in **dominant strategy equilibrium**, i.e. all agents have a single (pure/mixed) strategy that dominates all other strategies

Dominance and Best Response Strategies

- Strategy $s \in S_i$ is a **best response** to strategies $s_{-i} \in S_{-i}$ iff

$$\forall s' \in S_i, s' \neq s \quad u_i(s, s_{-i}) \geq u_i(s', s_{-i})$$

- Weaker notion, only considers optimal reaction to a *specific* behaviour of other agents
- Unlike dominant strategies, best-response strategies (trivially) always exist
- Strict versions of the above relations require that ">" holds' for at least one s'
- Replace s_i/s_{-i} above by σ_i/σ_{-i} and you can extend the definitions for dominant/best-response strategies to mixed strategies

Nash Equilibrium

- Nash (1951) defined the most famous equilibrium concept for normal-form games
- A joint strategy $s \in S$ is said to be in (pure-strategy) **Nash equilibrium** (NE), iff

$$\forall i \in \{1, \ldots n\} \forall s_i' \in S_i \quad u_i(s_i, s_{-i}) \geq u_i(s_i', s_{-i})$$

- Intuitively, this means that no agent has an incentive to deviate from this strategy combination
- Very appealing notion, because it can be shown that a (mixed-strategy) NE always exists
- But also some problems:
  - Not always unique, how to agree on one of them?
  - Proof of existence does not provide method to actually find it
  - Many games do not have pure-strategy NE

Example

Two men are collectively charged with a crime and held in separate cells, with no way of meeting or communicating. They are told that:

- if one confesses and the other does not, the confessor will be freed, and the other will be jailed for three years;
- if both confess, then each will be jailed for two years.

Both prisoners know that if neither confesses, then they will each be jailed for one year.

Example

The Prisoner's Dilemma: Nash equilibrium is not Pareto efficient (or: no one will dare to cooperate although mutual cooperation is preferred over mutual defection)

	2	C	D
1			
C		(3,3)	(0,5)
D		(5,0)	(1,1)

Problem: $DC \succ CC \succ DD \succ CD$ (from first player's point of view) and $u(CC) > \frac{u(DC)+u(CD)}{2}$