Why use a Classifier?
When will it help? And when will it not?

Alan Rector
Department of Computer Science, Univeristy of Manchester
Manchester M13 9PL, England
rector@cs.man.ac.uk

The availability of the Protégé-OWL plugin brings to the Protégé community the opportunity to use a description logic reasoner or "classifier" to help check and organise ontologies. However, Protégé users have been getting along without classifiers for years. So why should they need one now? When is a classifier not likely to be helpful? What does OWL make possible that could not be done before?

The Manchester group have been using classifiers to help build and manage ontologies for more than a decade and teaching the use of OWL and its predecessors for several years. On the one hand, we believe that building large, re-usable ontologies with rich multiple inheritance by manual effort unaided by formal checks is virtually impossible, and doubly so in any collaborative environment where work from several authors must be reconciled and integrated. On the other hand, we deliver most of the ontologies we build to applications in forms that do not require the applications to have classifiers or even know that classifiers have been used. .Furthermore, many of our users develop draft versions of ontologies without benefit of classifiers to which we propose "enrichments" with the help of classifiers and then return to the original authors for checking.

In short, there are several possible ontology life cycles of authoring, use, revision and maintenance. In most, classifiers bring benefits during the authoring and maintenance phases. Whether they are needed for at run time in the use phase depends on the application.

To understand the different use cases requires four general notions:

- *Composition* – the use of formal defintions to build more complex concepts out of simpler concepts – what we sometimes call "Conceptual Lego"
- *Normalisation and modularisation* – a specific set of techniques for managing multi-axial polyhierarchical ontologies
- *Pre- and Post- Coordination* – The difference between an ontology which provides a fixed vocabulary for run time applications and an ontology which provides the "words and grammar" to form allow run time applications to compose new concepts out of the expressions using concepts and to discover their relations with existing concepts (post-coordination)..
- *Open and closed world reasoning* – how the reasoning in ontologies and databases is different

Composition: The fundamental advantage of using classifiers is that they allow composition – *i.e.* they allow new concepts to be defined systematically from existing concepts. This can result in a dramatic reduction in the number of facts that have to be maintained explicitly.

For example, In defining an ontology of burns, we might consider the notion that burns can have any surface anatomic location (of which there might be, say, 300), three degrees of penetration, three categories of size, and two causations (heat and chemical). That is a total of at most a dozen new facts to maintain about burns. However, the total number of types of burns that can be defined from those dozen facts is 300*4*4*3 = 14,400. How many of those types are relevant to any application? How much effort is required to keep such a large number of types correctly classified by hand?

A special case of definition is *embedding* of concepts in expressions. OWL, and most related formalisms, allows complex expressions to be built up without having to name each intermediate concept before use. This embedding of 'anonymous concepts' allows more expressive representations to be built easily and naturally.

Normalisation and Polyhierarchy: Managing and maintaining a complex polyhierarchy of concepts is hard. Changes often need to be made in several different places, and keeping everything in step is error prone and laborious. Using a classifier, we have developed a mechanism for 'normalisation' of ontologies. In normalised ontologies, even the most complex polyhierarchies are built out of simple non-overlapping trees. In the example above, one large tree for anatomical locations and one simple tree each for the degree of penetration, size, and causation. Concepts bridging the trees are created by definitions. The relationships amongst defined concepts are maintained by the classifier. Changes are always made in exactly one place. If we need to change the anatomy, or add the possibility of ultra-violet light as a cause, we do it just once, not for some large and difficult to identify fraction of 14,400 potential concepts [1].

Pre- and Post- Coordination and the Ontology Life Cycle: Many applications need an ontology primarily as a fixed vocabulary with fixed relations. They do not expect to define new concepts in terms of existing concepts "on the fly" at run time. What is required at run time is a "pre-coordinated" ontology in which all concepts needed and the relations between them are explicitly pre-defined. That the pre-coordinated ontology was, or was not, derived through normalisation and classification is irrelevant at run time. What is required is a hierarchy (or polyhierarchy) of concepts to "fill the boxes" in the information or decision support model. In this case the life cycle of the ontology is that it is authored, maintained and demonstrated correct in OWL with the help of the classifier, but then "published" as a pre-coordinated form – in RDF(S), OWL-Lite, CLIPS, or some other proprietary form.

On the other hand, some applications require more concepts than can be conveniently enumerated. Indeed the number of concepts that can be defined from the elements of an ontology may be indefinitely large or even demonstrably infinite. If it cannot be predicted in advance which concepts are needed, then the classifier is needed at run time to create and classify the concepts found. In the example of burns above, it might well be worth the trouble to have a classifier if it meant that the user could see on the order of 15000 potential concepts but the system to be maintained needed to deal with only a dozen or so facts. The PEN&PAD clinical data entry system developed in Manchester used post-coordination to allow very large potential terminology to be maintained simply and tailored quickly to special needs. Discussion of how to use SNOMED-CT in similar ways is currently under way.

Classes and instances/Open and closed world reasoning/Ontologies and Databases. Ontology classifiers are designed primarily to reason about classes and schemas – *i.e.* the things that are necessarily true about all instances of given types in our conceptualisation. The goal is to find a stable structure for the domain true for all instances, make it completely explicit, and then use it repeatedly for each instance. Hence, it something should only be considered false if it is false of all possible instances – *i.e.* it is inconsistent. Classifiers therefore use "open world" reasoning in which negation means that something is provably false. This is a relatively expensive, but justified because it is relatively rarely performed and frequently re-used.

Databases and logic programming on the other hand are designed to reason about the things that are specific about each individual (instance). It is unreasonable, in general, to record everything that is false as well as everything that is true. Indeed, there are an indefinitely large number of things that might be false about any individual. Therefore, databases and logic programming use "closed world reasoning – if something cannot be proven to be true, then it is considered false. This is a much weaker form of reasoning, but much more efficient. Hence applications involving large numbers of individuals will almost certainly need to use databases or related "closed world" technologies.

Indexing: One of the characteristics of normalised ontologies is that they separate the different axes of classification cleanly. The result is a structure that is extremely effective at indexing information – be it interactions between drugs, resources to be used on the Semantic Web, or codes from external systems such as ICD or ICPC. A major advantage of such schemes is that they are fractal – they can be extended to smoothly to finer and finer levels of granularity. The complex, polyhierarchies required for indexing in such situations are precisely those that are difficult to maintain manually and easy to maintain with the aid of a classifier.

Summary: We suggest that there are four general use cases for ontology classifiers

- To help in authoring and maintaining large, multiaxial or polyhierarchical ontologies, which will be published in pre-cordinated form.
- To reason about schemas and guarantee that they are consistent with the ontologies
- To provide "fractal indexing" for complex information
- To allow post coordination in situations in which pre-coordination would lead to combinatorial explosions.

Classification is *not* usedful for:

- Dealing with large numbers of instances
- Dealing with numerical reasoning – at least not at the present time
- Dealing with condition-action rules about individuals, particularly if the information on which they are based may change. Classification is about what is true in all "worlds" at all times, not about what is true now but may change later.

1. Rector, A., Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. in *Knowledge Capture 2003*, (Sanibel Island, FL, 2003), ACM, 121-128.