exteca

ERL Guide

15/09/2002 13:31

A Guide to the Exteca Rule Logic

Version History
3 July 2002 First Draft

Author
Llewelyn Fernandes

Contents

Introduction

This document provides some notes on using the ERL, which is the language for describing categorisation rules within an Exteca ontology.

Adding rules to an ontology

In order for an ontology to be used for categorisation of documents it requires a set of rules for each concept. These rules describe features of content that may relate to the concept, thereby prompting the categorisation of that content under that concept.

How to add rules

The simplest way to add rules to an ontology is to let the categorisation engine do it for you! This involves calling the generateDefaultRules() method of the CategorisationEngine class before you load the ontology. See the categorisation module documentation for further details.

Using default rules provides very limited categorisation accuracy and generally at some stage you will want to augment them or completely replace them. This involves adding rules statements to the concepts in the ontology. Here is an example

```
<concept-content id="boating">
   <rules content="text">
      <rule logic="'boat'"/>
      <rule logic="'life jacket'"/>
      <rule logic="'yacht'"/>
   </rules>
</concept-content>
```

Let’s look at this example in detail. First we declare that we are describing content for concept “boating”:

```
<concept-content id="boating">
```

Next we declare that we are writing rules for content identified by type “text”:

```
<rules content="text">
```

The categorisation engine makes no particular interpretation of what “text” is. The label merely allows you to segregate rules for different types of content.

Next we define the rules. Each rule identifies a textual feature that, if found, will contribute towards the categorisation of a document under the concept “boating”.

```
<rule logic="'boat'"/>
<rule logic="'life jacket'"/>
<rule logic="'yacht'"/>
```

The Genesis technology from Exteca will provide a means for generating quality rules automatically based on sample texts. Genesis is currently under development.

How rules get used

When the catgorisation engine processes a document it is performing the following steps:

1. Look at the document and find all the matching words in the rules of the ontology.
2. Apply the logic for the matched rules as defined by the rules in the ontology.
3. Calculate the categorisation confidence based on the matched rules.

In step 3 the categorisation confidence is found using a probabilistic calculation which is a measure of how much evidence there is that the document is about the given concept. So, when building rules the important point to bear in mind is that you are writing down the *evidence* that the engine should look for when making its categorisation decision. The more evidence that is found the more confident the engine can be in its decision making.

Rule types

This section describes the syntax of the rule logic string. Rule logic takes the form of a string of nested rule elements. For example "title('kite', 'flying')" looks for the words 'kite' and 'flying' in the title of a document. The following sections describe the syntax of each rule element.

Word

A word is specified within single quotes:

'rainbow'

It is only matched when the exact word is found in the document.

Morphology

The categorisation engine supports an extensible model for handling word morphology. New morphology handlers can be written to support the Morpher interface and plugged into the engine.

Currently three morphers are provided – a stemmer and a plural stemmer, and a case sensitive morpher. See the categorisation module documentation for details of how to code for them.

Stemmer

A stemmed word is either specified as a word to be stemmed using the '#stem' morpher:

'#stem(selling)'

or as a word that has already been stemmed using '#-stem

‘#-stem(sell)’

In both cases the rule will match against all stems of the word (e.g. sell,
selling, seller etc).

Plural stemmer

The plural stemmer matches singular and plural forms of the same word. It is specified using #plural on the plural form:

‘#plural(babies)’
‘#plural(donkeys)’

or #-plural on the singular form:

‘#-plural(baby)’
‘#-plural(donkey)’

Case morpher

Normally the categorisation engine will convert all text to lowercase before matching. The case morpher preserves the case so that case-sensitive matches can be made:

```
‘#case(Jack)’
‘#case(jack)’
```

Phrase

A phrase is specified by placing two or more words in quotes:

```
'a shot in the dark'
```

you can use morphers within a phrase:

```
'#stem(farm) supplies'
```

Sequence

A sequence is specified by placing subrules in a sequence grouping:

```
sequence(‘round’, ‘robin’)
```

Note that when the subrules are just words you have the equivalent of a phrase rule, so the above rule is equivalent to

```
‘round robin’
```

Sentence

A sentence rule is specfied by placing subrules in a sentence grouping:

```
sentence('win', 'lucky')
```

A sentence rule matches against cases where all the subrules occur in the same sentence.

Section match

Rules can be matched within a particular section of a document. Sections are defined in the document using XML tags.

A section match rule is specified by placing subrules in a grouping using the name of the section to match against:

```
title('rabbit', 'dog', 'chase')
cv(skills(‘presentation’, 'sales and marketing'))
```

In the first example we are looking for a title tag containing all the words
'rabbit', 'dog' and 'chase'. In the second example we are looking for a 'skills'
tag within a 'cv' tag that contains the word ‘presentation’ and the phrase 'sales marketing'.

Or

An or rule is specified by placing subrules in an or grouping:

```
or('cakes', 'biscuits', 'sweets')
```

The or rule will match if any of its subrules match. The score will be the score of the greatest scoring child.

And

An and rule is specified by placing subrules in an and grouping:

```
and('holiday', 'abroad')
```

The and rule will match if all of its subrules match. The score will be the score of the lowest scoring child.

Probability

A probability rule is specified by placing subrules in a probability grouping:

```
probability(and(‘cat’,’hat’), ‘mat’)
```

The probability rule will return a probabilistic combination of all the evidence from all the subrules. It is equivalent to specifying each subrule individually as <rule> elements in the ontology:

```
<rules content="text">
    <rule logic="and(‘cat’,’hat’)"/>
    <rule logic="’mat’"/>
</rules>
```

Applying weights to subrules

Each subrule can be weighted by prefixing the weight (in the range 0-100) on the front of the subrule:

```
30:'apple'
60:sentence('pear','cherry')
```

If no weight is supplied the default of 100 is assumed.

Further examples

30:title(‘#stem(healthy)’)

Contributes a score of 30 if a stem of the word ‘healthy’ is found in the title section of the document.

or(10:sentence(‘car’,’bike’), 20:title(‘car’,’bike’))

Contributes a score of 10 if the words ‘car’ and ‘bike’ are found in any sentence in the document, or a score of 20 if the same words are found in the title.

Effective rule-writing

As mentioned earlier, writing rules is about writing down evidence for a match against a concept. This writing down of evidence is somewhat different to just listing the vocabulary associated with a concept and is the key to writing good rules. Writing down the evidence is an exercise in

1.Listing the vocabulary associated with the concept
2.Specifying the ways in which this vocabulary is *characteristically* used in association with the concept (e.g. what phrases are formed from the words, how are words and phrases used within sentences, what stemming is significant, etc)
3.Specifying the strength of this evidence, or more accurately, specifying the degree to which this evidence distinguishes documents about the concept from documents about any other concept.

This last point is concerned with specifying the weighting associated with the rule and is discussed in the next section.

Weighting

Weighting is an important issue in rule building. Without weighting your rules the decisions made by the engine are very unsubtle – it is either 0% or 100% confident in its decision. What's more without weighting it only requires one rule to match for a 100% confidence to be recorded.

The weight on a rule specifies how likely the document is about the category given the evidence specified in the rule.

Let’s take the example above and add some weights to the rules:

```
<concept-content id="boating">
   <rules content="text">
      <rule logic="10:boat"/>
      <rule logic="20:life jacket"/>
      <rule logic="30:yacht"/>
   </rules>
</concept-content>
```

Now when the word “boat” is found it only contributes a score of 10 towards the overall score. Similarly for the other rules. In this way the more evidence is found the more confident the categorisation engine is in its decision.