Chapter 3

Basic Components of AIML

Abstract

Chapter 3 of Library Technology Reports *(vol. 49, no. 8), "Streamlining Information Services Using Chatbots," describes elements in AIML. The basic structure of AIML is simple; one can create a working chatbot using a small number of AIML tags.*

Categories, Patterns, and Templates

AIML is composed of three basic elements. The building block of AIML is the category. Each category represents a question/answer or input/response pair. Categories are themselves composed of patterns and templates. Patterns represent the input received by the AIML interpreter. Templates represent the response generated by the interpreter to a given input.

An example of a very simple AIML category is the following:

```
<category>
<pattern> WHAT ARE YOUR HOURS
   </pattern>
<template>We are open from nine to
   five, Monday through Friday.
   </template>
</category>
```

To construct a simple AIML category from scratch, we first need to identify the code format, (in this case XML with an XML version and encoding statement). Follow this with an open AIML statement with its associated version. End with a closed AIML statement as below. Next, add the category example above to the code, creating a meaningful question/answer pair.

```
<?xml version="1.0"
   encoding="UTF-8">
<aiml version="1.0">
<category>
<pattern> WHAT ARE YOUR HOURS</
   pattern>
<template>We are open from nine to
   five, Monday through Friday.</
   template>
</category>
</aiml>
```

That's it . . . you've created the first question to which your robot can respond with correct information.

Preprocessing Steps

Before your AIML code is processed, two important transmutations are performed by the AIML interpreter program. These are deperiodization and normalization. Deperiodization is the simple removal of periods from the query string sent to the interpreter. Thus, the statement "We want the library's books." is transformed to "We want the library's books."

The second step, normalization, is more complex. During normalization, any remaining punctuation is removed first. Then all text is changed to uppercase, and finally the string is run against a number of "reduction" files to identify and expand any contractions of the short forms of questions. In our example above, the string is now changed to "WE WANT THE LIBRARYS BOOKS". At this point, the transformed search string is ready to be matched against the AIML files that comprise the chatbots "brain." Table 3.1 provides additional examples of the results of preprocessing.

Original String	Don't you have a library café.	Do you have a library café?
Deperiodization	Don't you have a library café.	Do you have a library café?
Normalization	DO YOU HAVE A LIBRARY CAFE	DO YOU HAVE A LIBRARY CAFE

Table 3.1
Examples of deperiodization and normalization.

From the table, you can see how some different formulations of a basic query are simplified to a single category using just the preprocessing steps. However, there are still a number of different ways to ask the question that are not resolved by preprocessing. To deal with these, you need to create additional categories that will address these different question formulations.

Efficiencies

Wildcards

In programming your chatbot, it is important to realize the complexity of language that you will need to address using your AIML code. This is achieved by composing categories that will match each formulation of the question you desire to answer. Using simple text matching, this is indeed a daunting task. However, AIML provides a number of approaches to streamline your coding task.

The first of these is the introduction of wildcard characters to the category pattern. AIML uses two different wildcard characters, the asterisk * and the underscore character _. Both of these characters can be used to match either single words or multiple-word strings. The difference between them is the priority or order in which they are selected in a text-matching sequence.

An underscore is processed first, before any additional exact pattern match is sought. If no pattern with a matching underscore sequence is identified among the coded AIML categories, then an exactly matching text string is sought. Finally, if no matching text string is located, a pattern match using an asterisk is sought.

The following provides an example of how this could work. Let's say the customer asks the question, "Do you have ebooks?" You want to respond with the template message `<template>We do not currently offer ebooks.</template>`. You can use a number of different pattern formulations in order to generate the desired template.

First and most simply, you could program the pattern using two underscore characters around the abbreviation "EBOOK." This would cue a response with the desired template, but would also respond with this same template to all instances of comments or questions about e-books, not just about their availability. An alternative would be to use the asterisk in the same position. Changing the underscore to an asterisk allows you to continue to catch the same wildcard matches, but also allows you to program specific responses to different text matches should you wish. Table 3.2 illustrates this.

Suffice it to say, in order to avoid many pattern matching-problems, it's safer to use the asterisk than the underscore in most instances.

<star/>

The value provided for an asterisk wildcard character in the question pattern can also be repeated in the answer template using a `<star/>` tag. This is a single tag that is used to repeat whatever text is matched by the pattern asterisk in the template. This can be extremely valuable in clarifying the intent of the customer or providing options for further interaction.

In the following example, an asterisk is inserted to allow the question to ask about any person. By repeating the name in the template response, the potential search parameters are clarified.

```
<category>
<pattern>DO YOU KNOW WHO * IS
   </pattern>
<template>Would you like me to
   search for <star/>?</template>
</category>
```

In the above example, to the question "Do you know who Melvil Dewey is?" the chatbot will respond, "Would you like me to search for Melvil Dewey?"

It is also possible to implement pattern/template pairs containing multiple wildcard strings. In such cases, it is important to identify which asterisk string is to be repeated in the template. This is done by assigning each asterisk in the sequence an index value equal to its ordinal position in the sequence. The first asterisk will match index 1, the second index 2, etc. Using this scheme, it is possible to further simplify the above code as follows:

```
<category>
<pattern>* WHO * IS</pattern>
<template>Would you like me to
   search for <star index="2"/>?
   </template>
</category>
```

Question	Pattern	Template	Response
Do you have ebooks at the library?	<pattern>_ EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* google ebooks *</pattern>	<template>We do not currently offer ebooks for loan; however, you can use Google ebooks on library computers.</template>	(no response—pattern not matched)
	<pattern>* kindle ebooks *</pattern>	<template> We do not currently offer ebooks for loan; however, you can access your Amazon/Kindle ebooks on library computers. </template>	(no response—pattern not matched)
Can I access my google ebooks at the library?	<pattern>_ EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* google ebooks *</pattern>	<template>We do not currently offer ebooks for loan; however, you can use Google ebooks on library computers.</template>	We do not currently offer ebooks for loan; however, you can use Google ebooks on library computers.
	<pattern>* kindle ebooks *</pattern>	<template> We do not currently offer ebooks for loan; however, you can access your Amazon/Kindle ebooks on library computers. </template>	(no response—pattern not matched)
Can I access my kindle ebooks at the library?	<pattern>_ EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* google ebooks *</pattern>	<template>We do not currently offer ebooks for loan; however, you can use Google ebooks on library computers.</template>	(no response—pattern not matched)
	<pattern>* kindle ebooks *</pattern>	<template> We do not currently offer ebooks for loan; however, you can access your Amazon/Kindle ebooks on library computers. </template>	We do not currently offer ebooks for loan; however, you can access your Amazon/Kindle ebooks on library computers.
What is the average price of ebooks on websites?	<pattern>_ EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* EBOOKS * </pattern>	<template>We do not currently offer ebooks.</template>	We do not currently offer ebooks.
	<pattern>* google ebooks *</pattern>	<template>We do not currently offer ebooks for loan; however, you can use Google ebooks on library computers.</template>	(no response—pattern not matched)
	<pattern>* kindle ebooks *</pattern>	<template> We do not currently offer ebooks for loan; however, you can access your Amazon/Kindle ebooks on library computers. </template>	(no response—pattern not matched)

Table 3.2
Examples of pattern matching using wildcard characters.

<srai>

Another option for making coding more efficient is to use <srai> tags. SRAI stands for "symbolic reduction artificial intelligence." This tag option allows you to replicate a given template response to multiple pattern inputs without writing each of these out in full. To use SRAI, you must first create a unique pattern that is unlikely to be matched by a simple text-matching scheme. It is often best to use a sequence of letters and numbers to name these patterns. Having chosen a pattern sequence for your SRAI element, you then compose the template for this sequence. Once this base category is created, you can then refer to it in the template of any other category you wish to make respond the same way.

```
<category>
<pattern>LIB2FINES</pattern>
<template>Fines for overdue books
   are ten cents per day
   </template>
</category>
<category>
<pattern>What are library fines for
   books?</pattern>
<template><srai> LIB2FINES
   </srai></template>
</category>
<category>
<pattern>How much are library
   fines?</pattern>
<template><srai> LIB2FINES
   </srai></template>
</category>
```

<random>

The <srai> efficiency option focused on offering the same response (or template) to multiple questions (or patterns). The <random> option provides a way to offer multiple responses to a single question. It is, in design, similar to HTML list formatting. For a given pattern to which you wish to offer multiple responses, simply insert into the template a <random> tag followed by <li> tags for each formulation of the response you wish to offer. Be certain to close each <li> tag with , and the <random> tag with </random>.

```
<category>
<pattern>I like the library.
   </pattern>
<template>
<random>
<li>I'm glad you like our
   services!</li>
<li>Thank you! We aim to please.
   </li>
<li>I'm happy to hear that!</li>
</random>
<template>
</category>
```