Chapter 12

The algorithms of EMILE

This chapter attempts to give some insight into the reasoning underlying the algorithms of EMILE. We will start with a very simple version of the basic algorithm, and in several steps change it to the full algorithm, at each step elaborating on the motivations for the change.

12.1 1-dimensional clustering

Given a sample of sentences, we want to obtain sets of expressions and contexts that correspond to grammatical types. A simple clustering technique is to extract all possible context/expression combinations from a given sample of sentences, and group together expressions that appear with the same context.

12.1.1. EXAMPLE. If we take the sample sentences 'John makes tea' and 'John likes tea', we get the following context/expression *matrix*:

	(.) makes tea	John (.) tea	John makes (.)	(.) tea	John (.)	(.)	(.) likes tea	John likes (.)
John	x						x	
makes		x						
tea			x					x
John makes				x				
makes tea					x			
John makes tea						x		
likes		x						
John likes				x				
likes tea					x			
John likes tea						x		

from which we can obtain the clusters

['John (.) tea', {'makes', 'likes'}]
['(.) tea', {'John makes', 'John likes'}]
['John (.)', {'makes tea', 'likes tea'}]
['(.)', {'John makes tea', 'John likes tea'}]

Next, we can group contexts together if they appear with exactly the same expressions.

12.1.2. Example. If we add the sentences 'John makes coffee', 'John likes coffee' to the previous sample, the relevant part of the context/expression matrix looks like

	John (.) tea	John (.) coffee	John makes (.)	John likes (.)
makes	x	x		
likes	x	x		
tea			x	x
coffee			x	x

which yields the clusters

[{'John (.) tea', 'John (.) coffee'}, {'makes', 'likes'}]
[{'John makes (.)', 'John likes (.)'}, {'tea', 'coffee'}]

As stated before, a grammatical type can be characterized by the expressions that are of that type, and the contexts in which expressions of that type appear. Hence the clusters we find here can be interpreted as grammatical types. For instance, the clusters in the above example could be said to correspond to the grammatical types of 'verbs' and 'nouns', respectively.

12.2 2-dimensional clustering

One of the flaws in this technique is that it doesn't properly handle contexts whose type is ambiguous.

12.2.1. Example. If we add the sentences 'John likes eating' and 'John is eating' to the previous example, the relevant part of the context/expression matrix will look like this:

	John (.) tea	John (.) coffee	John (.) eating	John makes (.)	John likes (.)	John is (.)
makes	x	x				
likes	x	x	x			
is			x			
tea				x	x	
coffee				x	x	
eating					x	x

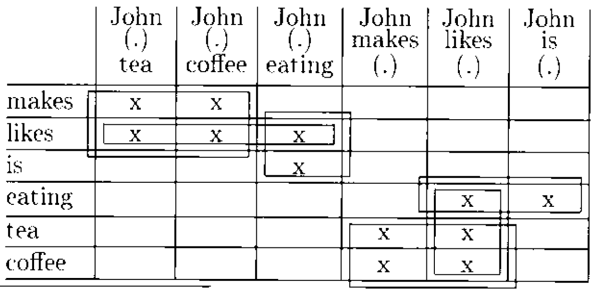
Here we can intuitively identify four grammatical types: noun-phrases, verb-phrases, 'ing'-phrases, and 'verbs-appearing-with-ing-phrases'-phrases. The context 'John likes (.)' is ambiguous, in the sense that it appears with both noun-phrases and 'ing'-phrases. If we proceed as before, we get the following clusters

[{'John (.) tea', 'John (.) coffee'}, {'makes', 'likes'}]
[{'John (.) eating'}, {'likes', 'is'}]
[{'John makes (.)'}, {'tea', 'coffee'}]
[{'John likes (.)'}, {'tea', 'coffee', 'eating'}]
[{'John is (.)'}, {'eating'}]

i.e. the context 'John likes (.)' is assigned a separate type.

Assigning ambiguous contexts a separate type not only results in a less natural representation, in a later step it will prevent us from correctly identifying the characteristic expressions of a type (as will be demonstrated in Example 12.4.2). A more natural representation would be to allow ambiguous contexts and expressions to belong to multiple types. For this, we need to use a different clustering method. The clustering method EMILE uses is to search for maximum-sized blocks in the matrix. This could be termed *2-dimensional clustering*.

12.2.2. Example. The following picture shows the matrix of the previous example, with the maximum-sized blocks indicated by rectangles.[32]



	John (.) tea	John (.) coffee	John (.) eating	John makes (.)	John likes (.)	John is (.)
makes	x	x				
likes	x	x	x			
is			x			
eating					x	x
tea				x	x	
coffee				x	x	

[32]Please note that the expressions and contexts have been arranged to allow the blocks to be easily indicated: in general, blocks will *not* consist of adjacent context/expression pairs.

These blocks correspond to the clusters

[{'John (.) tea', 'John (.) coffee'}, {'makes', 'likes'}]
[{'John (.) eating'}, {'likes', 'is'}]
[{'John makes (.)', 'John likes (.)'}, {'tea', 'coffee'}]
[{'John is (.)', 'John likes (.)'}, {'eating'}]
[{'John (.) tea', 'John (.) coffee', 'John (.) eating'}, {'likes'}]
[{'John likes (.)'}, {'eating', 'tea', 'coffee'}]

The last two clusters correspond to sets of context/expression pairs which are already 'covered' by the other blocks. In a sense these blocks are superfluous.

The algorithm to find these blocks is very simple: starting from a single context/expression pair, EMILE randomly adds contexts and expressions while ensuring that the resulting block is still contained in the matrix, and keeps adding contexts and expressions until the block can no longer be enlarged. This is done for each context/expression pair that is not already contained in some block. Once all context/expression pairs have been 'covered', the superfluous blocks (those completely covered by other blocks) are discarded.

12.3 Allowing for imperfect data

In the previous section, the requirement for a block was that it was entirely contained within the matrix, i.e. the clustering algorithm did not find a type unless every possible combination of contexts and expressions of that type had actually been encountered and stored in the matrix. This only works if a perfect sample has been provided. In practical use, we need to allow for imperfect samples. There are many context/expression combinations, such as for instance 'John likes evaporating', which are grammatical but nevertheless will appear infrequently, if ever.

To allow EMILE to be used with imperfect samples, two enhancements have been made to the algorithm. First, the requirement that the block is completely contained in the matrix, is weakened to a requirement that the block is *mostly* contained in the matrix. Specifically, a certain percentage of the context/expression pairs of the block as a whole should be contained in the matrix, as well as a certain percentage of the context/expression pairs in each individual row or column. We can express this as

$$\begin{array}{rlll} & \#(M \cap (T_C \times T_E)) \geq & \#(T_C \times T_E) & \cdot\,\texttt{total\_support\%} \\ \forall c \in T_C: & \#(M \cap (\{c\} \times T_E)) \geq & \#T_E & \cdot\,\texttt{context\_support\%} \\ \forall e \in T_E: & \#(M \cap (T_C \times \{e\})) \geq & \#T_C & \cdot\,\texttt{expression\_support\%} \end{array}$$

where M is the set of all encountered context/expression pairs, and the values `XXX_support%` are constants that can be set by the user.

12.3.1. EXAMPLE. Suppose that the matrix of context/expression pairs EMILE has encountered has the following sub-matrix:

	John makes (.)	John likes (.)	John drinks (.)	John buys (.)
tea	x	x	x	x
coffee	x	x	x	x
lemonade	x	x	x	
soup	x	x	x	x
apples				x

If the settings `context_support%` and `expression_support%` have been set to 75%, and `total_support%` has been set to 80%, then the type represented by the cluster

$$\left[\begin{array}{c}\{\text{`John makes (.)', `John likes (.)', `John drinks (.)', `John buys (.)'}\},\\ \{\text{`tea', `coffee', `lemonade', `soup'}\}\end{array}\right]$$

will be identified, in spite of the fact that one of the context/expression pair of the block, ('John buys (.)', 'lemonade'), does not appear in the matrix. However, the expression 'apples' will not be added to the above type, since it appears with less than `expression_support%` of the contexts.

Secondly, note that of the different expressions and contexts belonging to a grammatical type, it can be expected that the short and medium-length ones (in terms of number of words) will be encountered more often than the long ones. In other words, if we restrict the sample to short and medium-length contexts and expressions, it will be closer to a perfect sample. Implementing this notion, EMILE uses only short and medium-length contexts and expressions when searching for grammatical types.

12.4 Characteristic and secondary expressions and contexts

To search for longer expressions and contexts associated with types, EMILE uses characteristic expressions and contexts. As defined in Definition 11.2.5, a *characteristic* expression of a type T only appears with contexts that are of type T.[33] Since the types involved usually have not been fully identified yet, EMILE relaxes this requirement to also allow untyped contexts.

[33]Note that a context may have more than one type, so a context appearing with a expression characteristic for a type T may be of other types in addition to being of type T.

Occasionally, a type has no characteristic expressions (due to imperfections in the sample or the inherent ambiguity of the type): in such cases, the primary expressions of the type are used in place of the characteristic expressions. We call these the *characteristic** expressions of T, i.e. the *characteristic** expressions of T are defined as the characteristic expressions of T if there are any, and as the primary expressions of T otherwise.
The definitions of *characteristic* and *characteristic** contexts of a type T are analogous.
Any untyped context appearing with an characteristic expression of a type T is likely to belong to T as well. Contexts which appear with (a certain percentage of the) characteristic* expressions of T are called *secondary* contexts of T, as opposed to the *primary* contexts found by the clustering algorithm. Analogous for *secondary* expressions. Note that the constraint on the length of primary contexts and expressions does not apply to secondary contexts and expressions, and hence this allows for long contexts and expressions to be associated with types.

12.4.1. EXAMPLE. In the previous example, for the type represented by the cluster

[{'John likes (.)'}, {'eating', 'tea', 'coffee'}]

'John likes (.)' only appears with 'eating', 'tea' and 'coffee', so it is a characteristic (and hence characteristic*) context for this type. The expression 'eating' also appears with the context 'John is (.)', so it is not a characteristic expression. A similar condition obtains for 'tea' and 'coffee', so the type has no characteristic expressions at all. Consequentially, its primary expressions 'eating', 'tea' and 'coffee' are also its characteristic* expressions.
For the type represented by the cluster

[{'John makes (.)', 'John likes (.)'}, {'tea', 'coffee'}]

all its expressions and contexts are characteristic.

12.4.2. EXAMPLE. In Example 12.2.1, we used 1-dimensional clustering to obtain the cluster

[{'John makes (.)'}, {'tea', 'coffee'}]

Here, 'tea' and 'coffee' are not characteristic expressions, since they appear with the context 'John likes (.)', which here is not a context belonging to the type. So the type has no characteristic expressions. It is easy to see that when using 1-dimensional clustering, whenever a context is ambiguous[34], all types involved will lack characteristic expressions.

[34]'Ambiguous' in the sense that the set of expressions it appears with is the union of several smaller sets associated with other contexts

12.4.3. Example. Assume that primary expressions are constrained to be at most 5 words long. If we add the sentence 'John makes really really really really strong coffee' to the sample of the previous example, then the expression 'really really really really strong coffee' will not be added as a primary expression to the type represented by the cluster

$$[\ \{\text{'John makes (.)', 'John likes (.)'}\},\ \{\text{'tea', 'coffee'}\}\ ]$$

However, since 'John makes (.)' is a characteristic expression of this type, the expression 'really really really really strong coffee' will be associated with the type as a secondary expression.

12.5 Finding rules

The EMILE program also transforms the grammatical types found into derivation rules. For reasons of simplicity, EMILE constructs a context-free grammar rather than a context-sensitive grammar. For this construction, only the sets of expressions associated with the types are needed: the sets of contexts associated with the types are not used in creating the derivation rules.
First, EMILE searches for rules that are *supported*. Obviously, if an expression e belongs to a type T (as a secondary expression), the rule

$$[T] \Rightarrow e$$

is supported. EMILE finds more complex rules, by searching for characteristic* expressions of one type that appear in the secondary expressions of another (or the same) type. For example, if the characteristic* expressions of a type T are

$$\{\text{dog}, \text{cat}, \text{gerbil}\}$$

and the type [0] contains the secondary expressions

$$\{\text{I feed my dog}, \text{I feed my cat}, \text{I feed my gerbil}\}$$

then EMILE will find the rule

$$[0] \Rightarrow \text{I feed my } [T]$$

This process of abstraction is repeated to obtain more abstract rules. Formally, a rule R is considered to be *supported* if it is of the form $[T] \Rightarrow e$ (with e being a secondary expression of T), or if it is of the form $[T] \Rightarrow s_0[T_1]s_1[T_2]\ldots s_k$, $k \geq 1$, and for some $i \in \{1,\ldots,k\}$,

$$\#\{e \in T_E^* \mid R \text{ with } [T_i] \text{ replaced by } e \text{ is supported}\} \geq \#T_E^* \cdot \texttt{rule\_support\%} \qquad (12.1)$$

In certain cases, using characteristic* and secondary expressions in this manner allows EMILE to find recursive rules. For instance, a characteristic* expression of the type of sentences S might be

Mary drinks tea

If the maximum length for primary expressions is set to 4 or 5, the sentence

John observes that Mary drinks tea

will be a secondary expression of S, but not a primary or characteristic one. So if there are no other expressions involved, EMILE would derive the rules

$$\begin{aligned} [S] &\Rightarrow \text{Mary drinks tea} \\ [S] &\Rightarrow \text{John observes that } [S] \end{aligned}$$

which would allow the resulting grammar to generate, for instance,

John observes that John observes that John observes that Mary drinks tea

EMILE creates a set of supported rules capable of generating all sentences in the original sample. To reduce the size of this grammar, the program discards from the final output rules which are superfluous, such as rules which are instantiations of other rules[35], and rules for types which aren't referred to in other rules.
Experiments showed that often, EMILE finds several types which where only slight variations of one another. If all these types are referred to in the rules, this results in a much larger rule-set than is necessary. The most recent incarnation of EMILE tries to prevent this by being actively conservative in the number of types used: a set of *used types* is maintained, and only rules using those types are considered for inclusion. This set initially contains only the whole-sentence type [0], and types are added only if this would result in a decrease in the size of the total rule-set.[36]

12.6 Future Developments

There is still a lot of room for improvement. The clustering algorithm could be extended to use negative samples (i.e. sentences which should not be constructible) as well as positive ones. Furthermore, a module can be added to EMILE which allows it to identify those sentences whose grammaticality is the most uncertain (from those sentences which EMILE considers grammatical but which are not in the original sample), which would allow it to query an oracle in a directed fashion.

[35]I.e. which can be obtained from other rules by replacing a type reference by a secondary expression of that type

[36]EMILE can also be set to allow a small increase; this often results in a more meaningful grammar at the expense of a slightly larger rule-set.

Another possible extension is to the algorithm constructing the derivation-rule grammars. Currently EMILE constructs a context-free grammar. It may be possible to adapt EMILE to produce a more sensible context-sensitive grammars, using the sets of contexts produced by the clustering algorithm.