Mining Linguistic and Molecular Biology Texts through Specialized Concept Formation

Gemma Bel-Enguix, Veronica Dahl and M. Dolores Jiménez-López

GRLMC-Research Group on Mathematical Linguistics
Rovira i Virgili University, 43002 Tarragona, Spain

Abstract. We present, discuss and exemplify a fully implemented specialization of the *Concept Formation* Cognitive model into a model of text mining that can be applied to human or molecular biology languages.

1 Introduction

In [2], an executable cognitive model of knowledge construction, *Concept Formation*, was proposed and developed into a working system, inspired by constructivist theory as well as by natural language processing methodologies. This system accommodates user definition of properties between concepts, as well as user commands to relax their enforcement under certain conditions. For instance, a language tutoring system might include gender agreement properties, but relax the most commonly violated ones in order to accept incorrect input while pointing out the source of the mistake.

In this article we specialize Concept Formation into a model of text mining that has application in two important families of natural languages: human languages per se, and the (also human albeit less overtly so) languages of molecular biology. Complete running programs are given in at http://www.geocities.com/ CHRPrograms/SCF.html.

2 Motivation

We can imagine the human mind as a dynamically evolving store of knowledge, which constantly updates itself from new information built from previous information through some kind of reasoning. In this model, problems, events, feelings etc. that are on focus (that is, in our consciousness at any given time) trigger a (partly or wholly) unconscious search in the knowledge store for those pieces of information that relate to the problem. Once found, these can be put together to draw new knowledge from them. A rule appropriate for modelling the formation of the new knowledge might look roughly as follows: c_1, c_2, c_i `-->` `newc.` where the c_is and newc are concepts expressed as logic atoms.

These rules, referred to as Concept Formation rules in [2], are at the core of our text mining methodology. As in that previous work, the concepts in question are represented through logic terms, and thus may contain arguments, which were there used mostly to represent values relevant to conditions to be tested. For instance, the concept of the likelihood of a speeding ticket could be formed by a rule such as

Bel-Enguix G., Dahl V. and Jiménez-López D. (2009).
Mining Linguistic and Molecular Biology Texts through Specialized Concept Formation.
In *Proceedings of the 6th International Workshop on Natural Language Processing and Cognitive Science* , pages 117-121
DOI: 10.5220/0002199101170121

```
speed_limit(X), current_speed(Y) => test(Y>X),  speeding_ticket(likely).
```

expressing that if the current speed Y exceeds the speed limit X, a speeding ticket is likely.

In this article we focus on such rules as well, but use the arguments mostly to store information about the text being mined- in particular, to store subtext with desired characteristics as we go along in the process of extracting it. We provide utilities specialized to text mining tasks, which are then used by the core rule in our model– the Power Matching rule– to combine in one stroke all the needed information previously scattered across the input text. This basic rule can be tailored into further uses than the ones described in this paper, with the same underlying utilities we provide.

3 Computational Preliminaries: CHR

Constraint Handling Rules (CHR) [3] have the format `Head ==>Guard|Body`

`Head` and `Body` are conjunctions of atoms and `Guard` is a test constructed from (Prolog) built-in or system-defined predicates. The variables in `Guard` and `Body` occur also in `Head`. If the `Guard` is the constant "true" (i.e., no tests need succeed in order for the rule to apply), then it is omitted together with the vertical bar. Its logical meaning is the formula $(Guard \rightarrow (Head \rightarrow Body))$ and the meaning of a program is given by conjunction. There are three types of CHR rules:

- *Propagation rules*, which add new constraints (body) to the constraint set.
- *Simplification rules*, which also add as new constraints those in the body, but remove as well the ones in the head of the rule.
- *Simpagation rules*, which combine propagation and simplification traits, and allow us to select which of the constraints mentioned in the head of the rule should remain and which should be removed from the constraint set.

The rewrite symbols for the first two rules are respectively: `==>`, `<=>` and for simpagation rules, the notation is `Head1\Head2<=>body`. Anything in `Head1` remains in the constraint set and anything in `Head2` is removed from the constraint set.

4 Our Methodology Explained through an Example

4.1 Mining Spoken Languages

Consider the problem of finding a string of words of any length which is common to three short sentences given as input. For instance, for the input corpus: {*The drought of March has pierced to the root; Alice has had enough of hares of March; Waters of March was written by Jobim*}, the output should include "of March" as one of the common sequences found, indicating moreover the position where the sequence starts within each sentence.

Our system's utilities first compile the sentences into Prolog definitions of each (named s1, s2, s3), done in terms of atoms of the form w(i,j,W), where i is the sentence number, j the word's position in that sentence, and W the word itself. The above given input, for instance, compiles into:

```
(1) s1:- w(1,1,the), w(1,2,drought), w(1,3,of), w(1,4,march), w(1,5,has),
         w(1,6,pierced), w(1,7,to), w(1,8,the), w(1,9,root).

(2) s2:- w(2,1,alice), w(2,2,has), w(2,3,had), w(2,4,enough), w(2,5,of),
         w(2,6,hares), w(2,7,of), w(2,8,march).

(3) s3:- w(3,1,waters), w(3,2,of), w(3,3,march), w(3,4,was), w(3,5,written),
         w(3,6,by), w(3,7,jobim).
```

If we now initialize the system by calling all three strings, i.e.:

```
(4)   ?- s1, s2, s3.
```

we are in a position to extract substrings from these sentences in a high level fashion, through the following two propagation rules:

```
(5) w(Row,C,N), w(Row,C1,N1) ==> C1 is C+1 | sub([N,N1],Row,C).

(6) w(Row,C,N), sub(S,Row,C1) ==> C1 is C+1 | sub([N|S],Row,C).
```

Rule (5) detects two subsequent words in the same sentence, or row, and records them through a new constraint sub/3 in list form (in the first argument of sub/2), keeping as well, in its second argument, a record of the row (or sentence) the substring was found in, and in its third argument, the column it starts at within that row. Rule (6) similarly identifies all other substrings in the input strings, by adding one more word at a time to an already found string.

Of course, for different problems we may specialize these rules further, so that they zoom onto some sufficient subset of the set of all substrings, e.g. on all those substrings of a given size.

We have now enough utilities for the first incarnation of our Power matching rule, which extracts a substring S that is common to all three strings, and records the position in each sentence where the substring appears:

```
(7) sub(S,1,C1), sub(S,2,C2), sub(S,3,C3) ==> common(S,[C1,C2,C3]).
```

This completes our formulation for this toy example. Among the results the system outputs, we have:

```
common([of,march],[3,5,2])
```

Notice that in their declarative reading, our system's rules form a specialized concept (e.g. substring, common string, etc.) and in their operational reading, they produce all instances of that concept with respect to given input.

4.2 Mining Molecular Biology Text

The same methodology can be directly used for mining sequences of nucleotides given as input, without touching the system itself. All we need to do is change the input so that the compiler will treat strings of nucleotides, e.g. from:

```
c a t g g c a a
t g g c a c t g
a c g t g g c a
```

the compiler will obtain (we now use "n" instead of "w" for mnemonics):

```
(1') s1:- n(1,1,c),n(1,2,a),n(1,3,t),n(1,4,g),n(1,5,g),n(1,6,c),n(1,7,a),n(1,8,a).

(2') s2:- n(2,1,t),n(2,2,g),n(2,3,g),n(2,4,c),n(2,5,a),n(2,6,c),n(2,7,t),n(2,8,g).

(3') s3:- n(3,1,a),n(3,2,c),n(3,3,g),n(3,4,t),n(3,5,g),n(3,6,g),n(3,7,c),n(3,8,a).
```

Calling all input strings through rule(1) results in the output:

```
common([t,g,g,c,a],[3,1,4])
```

being generated among others, indicating that t g g c a is a common substring, and that its start position in strings s1, s2 and s3 is respectively 3, 1 and 4.

So far we have only considered identical subsequences, i.e. there are no ambiguous elements in the vocabulary. Our formulation however has been designed to accommodate ambiguous input with minimum extra apparatus and computational overhead, as we discuss in section 5.1.

5 Three Special Cases of String Analysis

5.1 Ambiguous Matching

Whereas the basic nucleotide set consists of the nucleotides A,C,T,G, ambiguity (where a given string's position can take one value or another) is typically expressed by using extra names for the ambiguous nucleotides, so for instance a nucleotide denoted as R can materialize as either A or G.

Ambiguous matching usually introduces considerable extra work, both in terms of representing ambiguous strings, and of processing them.

In contrast, all our formulation needs in order to represent and process any ambiguous nucleotide is for the compiler to materialize all its incarnations locally when the ambiguous string is read in. For instance, a nucleotide of type R appearing in the third sequence, column 7, which following our notation will be input as as n(3,7,r), compiles into the two nucleotides n(3,7,a) and n(3,7,c). Non-ambiguous nucleotides in the same sequence remain represented as before, so that complexity-wise, the representation grows only linearly with respect to the number of ambiguous nucleotides. In order to process ambiguous strings, once we have compiled them as just described, no further modifications are needed to our system: it runs as is.

5.2 Finding a Substring's Frequency

In cryptanalysis [1], frequency analysis has been defined as the study of the frequency of letters or groups of letters in a ciphertext. Frequency analysis is based on the fact that, in any given stretch of written language, certain letters and combinations of letters occur with varying frequencies. It is clear that the methodology presented here can be used to identify the common combinations of letters and to assign them frequencies. In this section we exemplify with DNA strings, but as before, the same methodology is applicable to linguistic texts.

In molecular biology, finding a substrings' frequency can help, among others, to find DNA words. Those sequences more frequently repeated have a high probability of being meaningful in the genetic code.

For approaching this problem, we now modify our input to consist of just one sequence (which results in binary atoms compiling from the input, since we no longer need to record the sequence, or row, number), we introduce a parameter N in the call, which becomes go(N), with N being the length of the subsequences sought, and we calculate subsequences of that length. The Power Matching rule becomes (Max being the length of the substrings to be counted):

```
(8) n(C,N), sub(S,C1,L,Max) ==> L < Max, L1 is L+1, C1 is C+1 |
                                sub([N|S],C,L1,Max).

(9) sub(S,C1,Max,Max), sub(S,C2,Max,Max) ==> dif(C1,C2) |
                                  repeated(S,[C1,C2]).

(10) sub(S,C1,Max,Max) \  repeated(S,Where) <=>
                       notin(C1,Where) |
                       repeated(S,[C1|Where]).
```

This rendition of the Power Matching schema illustrates matching an unknown number of string occurrences. Rule (8) creates substrings of increasing length up to the maximum, rule (9) detects two equal such substrings, starting respectively in positions C1 and C2, and after checking that these two positions are different, records the fact that the string S appears in both those positions. Rule (10) finds one more occurrence of the same string, and updates the information accordingly, adding the new position in the list of positions where the string repeats.

5.3 Finding Gapped Patterns

Because of the existence of introns and junk, in some molecular biology contexts it is reasonable to search for patterns that repeat in different sequences, even though they may be interrupted by an arbitrary number of words. Several systems exist and are available on line to find these gapped patterns in molecular biology, like MOTIF (http://motif.stanford.edu/) or TEIRESIAS (http://www.research.ibm.com/bioinformatics/home.html). Finding the maximal (gapped) patterns in a text (phrases with discontinuities), combined with the study of frecuencies, can easily help to text summarization and text classification. Our methodology can be adapted to this task as well, by keeping also the end point of the subsequences found, and using equations on them in this version of the matching rule. For instance, for the input:

```
s1:- w(1,1,the), w(1,2,big), w(1,3,wolf).
s2:- w(2,1,the), w(2,2,big), w(2,3,bad), w(2,4,wolf).
s3:- w(3,1,the), w(3,2,big), w(3,3,ugly), w(3,4,silly), w(3,5,wolf).
```

our system produces as output:

```
pattern([[the,big],_B,[wolf]])
```

6 Concluding Remarks

We have presented a new text mining methodology as an extension, through specialization, of the cognitive model of concept formation presented in [2]. We have not, in this article, exploited the feature of property relaxation which is part of the Concept Formation general model, this is left for future work.

Our focus at this point is expressiveness and elegance of formulation rather than efficiency, but our results are nevertheless reasonably efficient considering the tasks at hand. With this initial incursion into our methodology and its applications, we hope to motivate further research on its suitability for many other different kinds of problems in string analysis.

Acknowledgements

Agostino Dovier and André Lesvesque useful comments on a previous draft of this article are grateful acknowledged. This work was supported by V. Dahl's Marie Curie Chair of Excellence from the European Commission, by Canada's NSERC, and the Universities of Simon Fraser and Rovira i Virgili.

References

1. Becket, B. (1988), *Introduction to Cryptology*, Blackwell.
2. Dahl V. and Voll K. (2004) "Concept Formation Rules: an executable cognitive model of knowledge construction", in proceedings of *First International Workshop on Natural Language Understanding and Cognitive Sciences*, INSTICC Press.
3. Fruhwirth, T. (1998), Theory and Practice of Constraint Handling Rules, *Journal of Logic Programming*, Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriot, Eds.), 37(1-3), pp. 95-138.
4. Zahariev, M., Dahl, V., Chen, W. and Levesque, A. (2009), Efficient Algorithms for the Discovery of DNA Oligonucleotide Barcodes for DNA Sequences and Groups of Sequences. To appear in *Journal of Molecular Ecology Resources's Special Issue on DNA Barcoding*.