Author's response to reviews

Title: Doublet method for very fast autocoding

Authors:

Jules J Berman (bermanj@mail.nih.gov)

Version: 2 Date: 27 August 2004

Author's response to reviews:

Both reviewers had excellent comments, and I am particularly thankful that both reviewers provided a fast turn-around.

Chris Chute's review:

1. The reviewer notes that doublet coding is well recognized in the computational linguistic literature, which does not appear to be referenced. The reviewer goes on to describe literature attempts to use different term lengths to try to achieve optimal concept disambiguation.

Reply: I'm pretty sure that Dr. Chute is referring to the use of n-grams to assist natural language approaches to coding. This is an area that I know very little about, but it seems that in this approach people made collections n-grams of chunks of text (e.g. bi-grams of two word combinations, tri-grams of 3-word combination, etc.) and based on the frequency of reference terms in the full text corpus and some measure of "closeness" to the n-grams, they could make statistically infer codes for terms or for [n-gram] decompositions of terms. A similar approach could be used to find candidate new terms [for a vocabulary] by parsing text. The n-gram approach begins with the same observation that I made in the introduction to the paper, (that n-grams carry more meaning than singletons), but the similarities between the techniques end there. I am not trying to get any meaning from the n-grams. I am only trying to get an algorithmic speed advantage by narrowing my look-ups to doublets (rather than building phrases from singletons). My approach does not use statistical methods and does not use natural language processing because I am not trying to extract any meaning from the text. I'm only trying to match terms terms appearing in the text against a nomenclature, as fast as possible. I had expanded Background to include a brief mention of n-gram statistical techniques, with added references.

"Medical autocoding can be considered a specialized form of machine translation. Medical autcoders transform text into an index of coded nomenclature terms (sometimes called a "concept index" or "concept signature"). Several innovative approaches to autocoding have used the higher information content of multiword terms (also called word n-grams) to match terms in text with terms in vocabularies or to enhance the content of vocabularies by identifying n-grams occurring in text that qualify as new nomenclature terms [2,3,4]. Unlike prior studies with n-grams, the method developed for this study does not use statistical inferencing or the information content of bi-grams to infer semantic meaning from natural language. The author has used the higher term specificity of doublets [bi-grams] to construct a simple and fast lexical parser. Lexical parsers are types of string-matching algorithms. In general, the overall speed of lexical parsers is determined by the speed with which the parser can prepare an array of all possible words and phrases contained in a block of text, coupled with the speed with which each of these phrases can be compared against all the terms in the nomenclature."

2. The reviewer addresses the question of word order dependence within the phrases that are being indexed. The reviewer indicates that long terms can be expressed with many different word orders.

The reviewer has touched on one of my favorite subjects, but one that I dealt with in another submission that describes the neoplasm taxonomy [used as the nomenclature for this paper]. Basically, I have come to believe that there is no longer any computational penalty to creating nomenclatures that contain every in-use word-order for a term. And that's what I tried to do when creating the nomenclature. For example, we have about 40 variants of the same nomenclature term, many of which are word-order variants.
tendon sheath with localized giant cell neoplasm
localized giant cell neoplasm arising in tendon sheath
localized giant cell neoplasm arising from tendon sheath
localized giant cell neoplasm of tendon sheath
tenosynovium with localized giant cell neoplasm
localized giant cell neoplasm arising in tenosynovium
localized giant cell neoplasm arising from tenosynovium
localized giant cell neoplasm of tenosynovium
localized giant cell neoplasm arising in the tendon sheath
localized giant cell neoplasm arising from the tendon sheath
localized giant cell neoplasm arising in the tenosynovium
localized giant cell neoplasm arising from the tenosynovium
tendon sheath with localized giant cell tumor
localized giant cell tumor arising in tendon sheath
localized giant cell tumor arising from tendon sheath
localized giant cell tumor of tendon sheath
tenosynovium with localized giant cell tumor
localized giant cell tumor arising in tenosynovium
localized giant cell tumor arising from tenosynovium
localized giant cell tumor of tenosynovium
localized giant cell tumor arising in the tendon sheath
localized giant cell tumor arising from the tendon sheath
localized giant cell tumor arising in the tenosynovium
localized giant cell tumor arising from the tenosynovium
tendon sheath with localized giant cell tumour
localized giant cell tumour arising in tendon sheath
localized giant cell tumour arising from tendon sheath
localized giant cell tumour of tendon sheath
tenosynovium with localized giant cell tumour
localized giant cell tumour arising in tenosynovium
localized giant cell tumour arising from tenosynovium
localized giant cell tumour of tenosynovium
localized giant cell tumour arising in the tendon sheath
localized giant cell tumour arising from the tendon sheath
localized giant cell tumour arising in the tenosynovium
localized giant cell tumour arising from the tenosynovium
localized tenosynovial giant cell neoplasm
localized tenosynovial giant cell tumor
localized tenosynovial giant cell tumour
nodular tenosynovitis
tendon sheath localized giant cell neoplasm
tenon sheath localized giant cell tumor
tenosynovium localized giant cell neoplasm
tenosynovium localized giant cell tumor

This is why we have over 100,000 terms in a nomenclature exclusively devoted to the names of neoplasms. The approach that I have come to prefer is that the natural language issues be solved by the person who is making the nomenclature, not by the person who is trying to write fast text parsers. If we were to expand the size of the nomenclature 10-fold by listing lots more word-order variants of terms, it would not have major effects on the speed of the parser. On the other hand, it has been my personal experience that if we program the parser with language rules to catch word-order variants, any chance of creating a fast parser is lost.

The cancer nomenclature used for this paper was described in a previously published article [http://www.biomedcentral.com/1471-2407/4/10]. A second paper has been submitted on the taxonomy (i.e. the nomenclature used in this paper), and a third paper is being prepared to discuss computational methods that can expand the nomenclature (to include almost all in-use word orders for terms). I think that Dr. Chute will enjoy the paper that's already been published, and it may convince him that my long-term goals for cancer informatics makes some sense (1. build a nomenclature that is better than the existing cancer nomenclatures, 2. build a fast and powerful autocoder, 3. organize cancer-related text in a standard way using XML-based data specifications).

I have added text to the manuscript that briefly states that the nomenclature includes word-order variants of terms.
"The nomenclature used is neocl.xml, previously described by the author and currently designated as "The developmental lineage classification and taxonomy of neoplasms"[1]. In the context of this manuscript, the purpose of the taxonomy is to provide a listing of all names of neoplasms, with synonyms grouped under a common code number. The current version of the neocl.xml file contains 102,271 unique names of neoplasms. In constructing the taxonomy, enormous effort was made to include every variant name for every known neoplasm of man. Variant names included different terms for the same concept and different ways of expressing an individual term (e.g. variations in word order)."

3. The reviewer suggests that code snippets might not be appropriate for this journal, indicating that it is more important to present a more detailed description of the underlying algorithm.

Reply. The reason I think that the code snippets are valuable, in this case, is that this paper was submitted as a software article. However, if the reviewer finds these snippets inappropriate for a scholarly journal, it's easy enough to remove them.

The code snippets have been removed. I have expanded discussion of the underlying algorithm and included extra text in "Results" and "Discussion."

In Results:

"The difference between the two coders occurs when the chunks of text are parsed. The phrase coder takes the text and creates an array of every ordered combination of 1,2,3,4 and 5-word phrases contained in the record. This number of elements in the phrase array is about 5 times larger than the number of words in the record. The doublet coder parses the record into the set of all ordered doublet terms. This number of elements in the array of doublets is about the same as the number of words in the record. The phrase coder must try to match about 5 times as many terms as the doublet coder. Also, the phrase coder tries to match each phrase in the record array against the entire nomenclature. The doublet coder tries to match each doublet encountered in the input text against the collection of doublet terms extracted from the nomenclature. Because doublets of text, unlike individual words, tend to have unique meanings, only a small subset of the doublets encountered in the input text will likely match the set of doublets extracted from the nomenclature. Text doublets that do not match any doublets in the nomenclature are "skipped." Text doublets that match doublets from the nomenclature are concatenated to consecutive matching doublets until a non-matching doublet is encountered. The length of matching doublets sets the length of the candidate term. The algorithmic strength of the doublet method is that it eliminates the need to create and match [against a nomenclature] an array of all possible phrases of all possible lengths found in a textual record."

In Discussion:

"Three algorithmic properties contribute to the speed advantage of the doublet coder over the phrase coder: 1) The doublet coder parses text into a number of phrases that is 5-fold smaller than the number of phrases produced by the phrase coder; 2) Doublets that are not found in any of the nomenclature terms can be quickly excluded, 3) Successive text doublets that match doublets from the nomenclature can be quickly concatenated and tested for matches in the nomenclature. These are the algorithmic differences between the phrase coder and the doublet coder and presumably account for the improved speed of the doublet method."

4. The reviewer notes that there are spots where the language is poor. He is correct. I made a sentence-by-sentence sweep of the paper and have made many changes to clarify meaning and improve the flow of the text. I also corrected a number of typographic errors.

David Johnson's review:

1. This reviewer, like the first reviewer, asked that the citations be expanded. He particularly wanted a literature citation for the discussion of the limitations of precision and recall.

I added 5 references, including three related prior works on natural language processing. The added reference discussing the limitations of precision and recall is:

I'd like to note that the kind of lexical parsing that I do in this paper is not what most people would consider to be natural language processing. True, it uses natural language (free-text) and true, it does processing. But most people think of natural language processing as a field where people write software that can "calculate" meaning from text. This usually means using linguistic rules or semantics. My technique is much closer to simple sequence matching.

2. The reviewer noted that execution time for software is influenced by factors other than choice of programming language (such as choice of data structures). The reviewer is correct, but I did everything feasible to minimize such differences. Basically, the algorithms only differ once they've gotten the final chunk of text for parsing. I added the following to the text:

"Both the phrase method and the doublet method coders used the same Programming Language, the same nomenclature, and the same data structure for the nomenclature (a simple associative array). Both coders grab sequentially occurring chunks of text, each chunk delimited by a double-newline delimiter (roughly the ascii equivalent of a paragraph delimiter) and place the grabbed chunk of text into a temporary scalar variables.

The difference between the two coders occurs when the chunks of text parsed...."

3. The reviewer also noted that readers unfamiliar to Perl would not understand the code snippets. Chris Chute had suggested that the code snippets could be deleted and the underlying algorithm expanded. That's what I did, and I hope this satisfies David Johnson's very similar comment. I added a few more words about how the reader can get more information about the Perl language.

4. The reviewer's last question is the most difficult and required a revision in the software. The reviewer is interested in how the software deals with special cases such as sentence boundaries and adjacent or overlapping terms.

The first part of the question is easy. The software totally ignores sentence boundaries. This flies in the face of common sense because anyone would think that there would be occasions that a multi-word term might occur across a sentence boundary, but that this term would create a false positive match. This is what I thought for several years, and I devoted considerable energy to creating a sentence parser to pre-process text into defined sentences prior to the autocoding step. The documentation is available at: [http://12.183.10.150/jjb/parsepod.htm]

Much to my surprise, it turns out that pseudo-terms occurring across sentence boundaries virtually never occur. Using a 4 megabyte text of abstract, and chunking at a "period-space-space" delimiter (that will catch most sentence boundaries) and comparing it with a medline record delimiter (which will effectively obliterate all sentence boundaries), there simply is no difference in matched terms. That is, pseudo-terms occurring across sentence boundaries do not occur.

The reviewer's second exception is more problematic. If you have two (or more) adjacent or overlapping terms, the parser will concatenate all of them into one large term. This rare occurrence actually accounts for the small reduction in term matches in the doublet coder compared to the phrase coder. For the first version of the paper, I was willing to accept a minor coding deficiency for the enhanced speed of the doublet coder. But the reviewer's question prompted me to add a few lines to the code that takes the fully concatenated doublet string and chops it into ordered pieces, looking for substring matches. This had the effect of completely wiping out the matching deficiency of the doublet coder. In the revised version of the paper, there are no matches found by the phrase method that are not also found by the doublet method. However, as in the first version of the paper, there are matches found by the doublet method that are missed by the phrase method.

I added a some text that briefly explains how the doublet method handles boundaries.

Boundaries:

"The doublet autocoder creates a string of words from a chunk of text. In the case of the example corpus (PubMed abstracts), all of the sentences from the abstract are squeezed into a single string of words, obliterating sentence boundaries. This means that if a sequence of words crossing a sentence boundary happens to match a term in the reference nomenclature, the coder will register a "pseudo-positive" term. Empirical evidence suggests that this theoretical error in the doublet autocoder simply does not occur. A 4 megabyte collection of abstracts chunked as whole abstracts or as chunked sentences (delimited by period-space-space) contained no instances of pseudo-terms created by the obliteration of sentence
boundaries in the whole-abstract text chunks. For fastidious developers who wish to ensure that their parsers respect sentence boundaries, it is possible to pre-process text into sentences with a sentence parser [11,12]."

I also expanded the algorithm to indicate that it now looks for subsumed terms within a run of doublets.

Overlapping terms:

"5. The runs of matching doublets are tested to see if they match any of the runs of doublets that compose nomenclature terms or if they contain any subsumed terms that match nomenclature terms."

I very much appreciate the comments of the reviewers and feel that their input has greatly improved the manuscript.