A quick approach to Summarizing

Magnus Skog, Jacob Persson

23rd January 2006

Abstract

This project deals with the difficult problem of automatic text summarization. This is an interesting and open-ended problem that has a multitude of uses. We attack the problem by using both a Linguistic Summarizer written in Prolog and a simpler sentence pruning algorithm written in Perl. The basic idea is to first cut the unnecessary parts out of the sentences and then prune the less important sentences creating a summary. The results were evaluated by a group of testers that compared the original articles with the summaries. The overall results were satisfying but there is more work to be done.

1 Introduction

Text summarizing is an interesting and difficult problem. In this project we attempt to create good summaries of shorter texts of about 1000 characters. We based this project on the work of Polanyi,Culy, van den Berg, Thione and Ahn(2004) and the the work of Thione, van den Berg, Polanyi and Culy(2004). We also use a parser created by Collins(1998). The system is divided into two parts: A discourse parser and a sentence pruner.

2 Linguistic summarizing

This part of our system uses a linguistic approach to remove overflow information inside sentences. The advantage of the linguistic models is that they often rely on both syntactic and semantic knowledge.

A linguistic model for summarizing usually builds up a tree/structure of the discourse to be summarized according to a discourse model. The actual summarizing is then performed by pruning branches with less interesting information.

2.1 Linguistic discourse models

A model we looked at was the linguistic discourse model(LDM) by Polanyi and Scha (1984, 1988). It starts with segmenting the discourse into basic discourse units (BDU) using a set of rules. Then a a tree of DCU's(discourse constituent unit) are build up from the BDU's and according to the structure of the discourse.

Between the DCU's are relations that can be divided into three classes: discourse coordination, discourse subordination and n-ary constructions. Discourse coordinations are relations where the parts have equal importance, like *"he shot the president and he got away with it"*. Discourse subordination express relations where something is a elaboration of something, for example *"jurgen is the VD of Janco, which is a large company on Iceland"*. N-ary construction are lists or enumerations of things.

A similar model is that of the rhetorical structure theory(RST) by Mann and Thompson (1988). There the discourse is segmented into nucleuses and satellites, where the nucleus is the main segment and the satellites hold some relation to it. Relations can for example be elaboration, cause, motivation and concession.

2.2 A simplified linguistic discourse model

Our model starts from a tree created by the August parser, which output a tree in the Penn tree-bank style. We don't segment our discourse instead segmenting is included into the rules.

Since we also use a statistical summarizer on a per sentence basis we decided to let our LDM handle only one sentence at a time. This means of course that it can't solve co references or deal with any relationships that span across sentences.

With these simplification made the difficulty in implementing this model is to write the rules that find relations between discourse segments.

2.3 Implementation

We choose to implement our LDM in Prolog because its build in search mechanism make rules easy to apply.

The output from the August parser is converted to a tree of Prolog lists. On this tree we do a bottom up search and match rules. The relations we are interested in are mostly those of the elaboration type and since this is the last and only step we prune the tree at the same time.

A rule for identifying a elaboration can look like this. Here the elaboration is identified by the cue word 'which'. You can also see how we prune the tree by only saving what's matched to X.

```
match(Tree, ['NP', X]) :- ['NP' | Y] = Tree, append(X, [[(','),
[(',')]], ['SBAR', ['WHNP', ['WDT', [which]]], _], [(',')], [(',')]]],
Y).
```

Taking the leafs of the pruned tree in a inorder walk results in the summarized sentence. This is applied to every sentence in the discourse. A couple of post processing steps are finally used to give a proper output.

3 Sentence Pruner

3.1 Introduction

Unlike the Prolog summarizer, the sentence pruner only works with whole sentences. It prunes unwanted sentences using an algorithm that is based on word frequency and the length of the sentences

3.2 The Algorithm

The algorithm starts by splitting the text into sentences. It also makes a list of all the words in the text together with a score which is equivalent to the number of times the word appears in the text. Each sentence is then processed by adding the scores for all the words in the sentence. If the sentence contains words that appear in the title of the text, then the score of the sentence is improved. Also, the first sentence in the text is given a bonus to it's score due to the general importance of the very first sentence after the title. When this has been done for all the sentence in the text, the algorithm then prunes the unwanted sentences. Input to the algorithm is a percentage. This is the amount of sentences that the algorithm won't prune. The top scoring sentences are kept while the rest are removed.

3.3 Implementation

We chose to implement the algorithm in Perl. Perl is the natural choice for this kind of simple text manipulation.

3.4 Improvements

The algorithm does several passes through the text. It might be possible to score the sentences and count the words at the same time, thus saving time by doing less passes. Also, the implementation uses regular expressions to prune the sentences. It does this one by one, thus doing several unnecessary passes through the text. A good improvement would be to make one pass during the text that prunes all unwanted sentences.

4 Evaluation

4.1 Introduction

The program was evaluated using a test-set of the corpus. Five texts were selected from the text-set and we ran the program on them. We pruned 75 percent of the sentences. The original text and the results were posted on a website where a number of people read the texts and scored them. An interesting result was that we accidentally forgot to mention that the texts were not summarized by hand, so most testers believed that the summaries were done by a human.

4.2 Expected Results

From the results of the training corpus we expected some trouble with co-references. The program would prune sentences containing a name of a person and then later on keep a sentence containing âHe went to the marketâ and no reference to what âheâ was could be found in the summary. We found this to be a major problem but we simply did not have the time to implement a co-reference solver to fix

4.3 Results

As expected, most of the negative comments were that there were numerous problems with co-references. Most texts contained these problems. Other than that, most testers were fooled by the program and still believed that the summaries had been written by hand. One tester commented, and I quote: 'It looks like you have been a bit lazy when you summarized these texts'. All testers thought that the readability of the texts were good to

excellent. Most of them, if not all, were very surprised when they later found out that the summaries were actually made by a computer program

4.4 Comments

It seems as the expected results were indeed reasonable. A co-reference solver would have remedied most of the problems and improved readability significantly.

5 Conclusion

The overall performance of our system was very good. The summaries were most of the time very easy to read and they often looked hand-made. In order to create better summaries we would have to start by implementing a co-reference solver. That is the beyond the scope of this short project. Another problem is the extremely bad performance of the summarizer. This is however caused by the parser. A faster parser would improve performance but this could never be used in real-time application such as websites. This is unfortunate because a website would have been a very interesting application for this project. We could for example summarize news articles for quick and easy reading. This would however require a much faster parser and most likely a dedicated high-performance server.

6 References

Gian Lorenzo Thione, Martin van den berg, Livia Polanyi and Chris Culy. 2004. Hybrid Text Summarization: Combining External Relevance Measures with Structural Analysis

Livia Polanyi, Chris Culy, Martin van den Berg, Gian Lorenzo Thione, David Ahn. 2004. A rule based Approach to Discourse Parsing

Pierre Nugues. 2005. An introduction to Language Processing with Perl and Prolog

7 Appendix A: Running Instructions

7.1 Requirements

The program requires SWI-Prolog and Perl to be able to run

7.2 Instructions

In the root directory of the program there is a Perl script called "run.pl". The script runs the entire program, automizing most of the tedious longer run scripts. The corpus is available in the corpus directory. The parser must be available in the root directory of the program.
The usage is "Perl run.pl text percentage "title words"".

Example 1: Perl run.pl corpus/dev-set/prohibition 25 "prohibition"
Example 2: Perl run.pl corpus/test-set/wikipedia 15 "wikipedia"

You don't have to put in any title words but the results will be better if you do.

8 Appendix B: Links

The project files,including the parser and all of the articles used in the evaluation, are available at:
http://www.snakeeyes.nu/project/