Problem assignment for scientific programmer selection

April 14, 2015

Note: This programming assignment must be done in C++. Please submit your source code (see below) the day before your interview.

For machine translation it is important to estimate and compare the fluency of different possible translation outputs for the same source (i.e., foreign) sentence. This is commonly achieved by using a language model, which measures the probability of a string (which is commonly a sentence). Since entire sentences are unlikely to occur more than once, this is often approximated by using sliding windows of words (n-grams) occurring in some training data.

Background

n-gram: An n-gram refers to a continuous sequence of n tokens. For instance, given the following sentence: *our neighbour , who moved in recently , came by .* If $n = 3$, then the possible n-grams of this sentence include:

'our neighbour ,'
'neighbour , who'
', who moved'
...
', came by'
'came by .'

Note that punctuation marks such as comma and full stop are treated just like any 'real' word and that all words are lowercased.

Objectives

For both parts below we are particularly interested in two aspects: How much working memory (RAM) does your program require to store all n-grams (Part 1) and how fast is your program in querying your data structure to return the frequencies of n-grams (Part 2). While the training data provided here is rather small, so you can easily run it on any machine/laptop, you should take scalability into account. We will also test your code on substantially larger training data sets locally.

Part 1

Given a text corpus (i.e., natural language data set), the purpose of this task is to count and store the frequencies (number of occurrences) of all n-grams up to length 5 that occur in this corpus. For this task, use the following corpus: `train.txt` (see below for ULR). This corpus is already tokenized, i.e., all words are already separated by white spaces, including punctuation marks. Also, each line in this file corresponds to one sentence. Do not consider n-grams that cross sentences, i.e., lines in the file. Your program should simply take the name of the corpus file as an argument and store all n-grams that occur in this corpus in memory.

Your program should store **all** different n-grams up to (and including) length 5, i.e., all 1-grams, 2-grams, 3-grams, 4-grams, and 5-grams, and their respective frequencies.

If you want to get a better feel of how much memory your approach takes, you can also experiment with the file `train_bitLarger.txt` (see below for ULR) which contain slightly less than 200,000 sentences.

Part 2

In Part 1, you basically constructed a simple n-gram model and stored it in memory. In Part 2, you apply this model to new data. To this end, your program should read in an additional file, `test.txt` (see below for URL), given as a second command line argument: `your_program train.txt test.txt`. The test file consists of a number of 5-grams, where each line in the file consists of one 5-gram. Your program should read in the test file and say for each 5-gram of the form `word_1 word_2 word_3 word_4 word_5` what the frequencies for all n-grams ($1 \leq n \leq 5$) ending in `word_5`, i.e.:

```
frequency(word_1 word2 word_3 word_4 word_5)=?
frequency(word_2 word_3 word_4 word_5)=?
frequency(word_3 word_4 word_5)=?
frequency(word_4 word_5)=?
frequency(word_5)=?
```

The output should simply be written to `stdout`, following the order of the test file. If an n-gram does not occur in your model, return a frequency of 0. Note, that it may be very common that a 5-gram has a frequency of 0, but that the shorter n-grams covering the rightmost part of this 5-gram have non-zero frequencies.

Your program should also provide CPU time statistics on how long it took to process the entire test file in order to return all frequencies.

It is not required to have two separate versions of your program, one just addressing Part 1 and another one addressing Part 2. It is entirely sufficient to submit one program that addresses both parts.

Include a short README file together with your source code that tells us how to compile the source code. It should be compilable on Linux, using gcc/g++ version 4.4.7. All files (not including any data files) should be sent to me (`c.monz@uva.nl`) as a gzipped tarball. It would be good if you could include some light documentation of your source code.

Data

Download the following files. Both files are fairly small. `train.txt.gz` contains 10,000 lines and `test.txt.gz` contains 1,960 5-grams (lines).

train: `http://staff.science.uva.nl/~christof/progassign/train.txt.gz`

test: `http://staff.science.uva.nl/~christof/progassign/test.txt.gz`

Optionally, you can monitor memory usage with
`http://staff.science.uva.nl/~christof/progassign/train_bitLarger.txt.gz`

Good luck!