MaxMatch Tokenizer Implementation

Lorenzo Tosi

November 2, 2018

Contents

1 Introduction

This tokenizer is able, given a sentence list and a dictionary file, to divide sentences in words using the MaxMatch algorithm. This algorithm is well suited for languages which do not use word separators (e.g. Chinese, Taiwanese), and has been tested with Japanese. The tokenizer is written fully in Python3, the package *re* needs to be installed to work properly. The program itself is composed of 3 scripts, that are able to prepare the files needed to make the program work properly, plus the MaxMatch algorithm implementation. The files are the following:

dictionary_extractor.py This file is able to extract and filter a list of words and to tokenize them, in order to fed them to the algorithm as the dictionary. The test file used to extract the dictionary is *ja_gsd-ud-train.conllu*, included in the package. The file returned by the script is a *.txt* called *dictionary.txt*

sentencelist_extractor.py This file extracts a list of sentences that can be then tokenized in the algorithm. The file used to extract the sentence list is *ja_gsd-ud-test.conllu*, included in the package. The file returned by the script is a *.txt* called *sentencelist.txt*

testtokens_extractor.py This file extracts an already tokenized list of sentences in the same format as the file produced by the *maxmatch.py* program. This enables comparison and performance evaluation through algorithms like *WER*. The file used to extract the sentence list is *ja_gsd-ud-test.conllu*, included in the package. The file returned by the script is a *.txt* called *testtokens.txt*

maxmatch.py This is the MaxMatch tokenizer program. Detailed explanation of its functioning is included in a specific section.

2 How to Use the Text-Preparation Scripts

The script can be launched via the terminal by writing

```
$ python3 scriptname.py
```

The program will ask in input the name of the file containing the sentence list and the name of the file containing the dictionary.

```
Input filename here:
```

Test files produced by the scripts are included with the package.

3 How to Use *maxmatch.py*

The script can be launched via the terminal by writing

```
$ python3 maxmatch.py
```

The program will ask in input the name of the file containing the sentence list and the name of the file containing the dictionary.

```
Input sentence file here:
Input dictionary name here:
```

The program will produce in output a *.txt* file named *maxmatchtokens.txt*

4 *maxmatch.py* Algorithm Description

4.1 Functioning

The program replicates the *Viterbi Algorithm* through a decision tree approach. In the *Main* section, the program creates a list where every sentence (line) is appended as a single element. The max_match function then picks the first element of the list and starts analysing it.

```
senten = m.readlines()
        for inp in senten:
                while True:
                        if len(inp) > 0:
                                inp = max_match(inp, diction)
                        else:
                                break
```

The program reads the input letter-by-letter, appending them to a string that is checked through every iteration, to see if the word is in the dictionary.

```
for letter in text:
        inword += letter
        if inword in dictionary:
                outword = inword
```

This bit of code lets the program pick the longest word in the dictionary, with starting letter being the first character of the input, in a separate variable. The full string (*inword*) and longest dictionary word (*outword*) are then checked against a decision tree structured as follows:

1. If beginning of *inword* is enclosed in quotation marks (「」) and matches an ASCII printable *regex* (latin letters, digits, punctuation), attach the quotation marks and the word enclosed to the list of segmented words deriving from the sentence.

2. If beginning of *inword* is composed of matches an ASCII printable characters followed by an ASCII non-printable character, attach the printable string to the list of segmented words deriving from the sentence.

3. If *outword* is in the dictionary, attach it to the list of segmented words deriving from the sentence.

4. Else attach the first character to the list of segmented words deriving from the sentence.

The file then removes, through the *.replace* method, the first iteration of the word attached to the list from the input, and the program keeps cycling through the input until no characters are left.
Reached this point, the program breaks from the *While True* cycle included in Main, feed the next sentence to the input and repeats until the whole input file is processed. In the end, the program joins the list of segmented letters (that is already including line breaks to divide sentences), cleans the redundand whitespaces through a regex and write the output file

```
final = "␣".join(sentence)
final = re.sub("\s\\n\s", r"\n", final)
r.write(final)
```

4.2 Bug Fixes

The most important implementation that was made was to extend the decision tree in order to avoid segmenting digits and English words outside dictionary.
The first accuracy evaluation was made with a reduced decision tree, that was segmenting into single letters every word not included in the dictionary. This are the results obtained through the *apertium-eval-translator* script:

```
Number of words in reference: 12615
Number of words in test: 13657
Edit distance: 3042
Word error rate (WER): 24.11 %
Number of position-independent correct words: 11075
Position-independent word error rate (PER): 20.47 %
```

Including the first two steps of the decision tree made possible to improve both WER and PER scoring:

```
Number of words in reference: 12615
Number of words in test: 13555
Edit distance: 2969
Word error rate (WER): 23.54 %
Number of position-independent correct words: 11067
Position-independent word error rate (PER): 19.72 %
```

4.3 Possible Improvements

The algorithm main problem is the computational time. The program processed 557 sentences in 133.6 seconds, making it slow on a big corpus. The CProfiler statistics with the related call graph are contained in the file *maxmatch.pstat* available in the package.
The recursive analysis of the whole string against the dictionary could be identified as the main bottleneck. A possible improvement can be breaking the loop:

```
for letter in text:
        inword += letter
        if inword in dictionary:
                outword = inword
```

As soon as the program does not find an inword for 2 times in a row the look should exit and enter the decision tree phase, reducing the computational time with a slight reduction in precision.

4.4 Dictionary Scripting Choice

The dictionary script was tested in two different environments: *bash* with *Perl* scripting and *Python3*. The first script is shown hereunder:

```
cat ja_gsd-ud-train.conllu | grep -P "^[0-9]\t" |
cut -f2 | LC_ALL=C sort -u > dictionarybash.txt
```

and produces a dictionary of 13789 unique entries while the *Python3* script produces 22313 unique entries. The cut from 22313 is made during the *sort* command of *bash*, as proved by sorting the Python dictionary through this command.
A quick lookup shows that words were cut regardless of their forms: Latin written words as **"FeelingHeart"** and **"Xainctes"** , numbers as **"20.7"**, Japanese words written in different alphabets as "ウェストファル" or "料理番組". The accuracy test shows that running the same sentence input files with the bash-sorted dictionary produces worse results as expected:

```
Number of words in reference: 12615
Number of words in test: 14371
Edit distance: 4039
Word error rate (WER): 32.02 %
Number of position-independent correct words: 10781
Position-independent word error rate (PER): 28.46 %
```

5 Files Included in the Package

MaxMatch_Documentation.pdf	
MaxMatch_Documentation.tex	
dictionary.txt	*Python3* script dictionary
dictionary_extractor.py	
dictionarybash.txt	*bash* script dictionary
ja_gsd-ud-test.conllu	
ja_gsd-ud-train.conllu	
maxmatch.pstat	
maxmatch.py	
maxmatchtokens_bashdict.txt	maxmatch produced with the *bash* script dictionary
maxmatchtokens_pythondict.txt	maxmatch produced with the *Python3* script dictionary
sentencelist.txt	
sentencelist_extractor.py	
testtokens.txt	
testtokens_extractor.py	