# Project Proposal brainstorming

Neural Machine Translation output:

2 hypotheses:

I) NMT output uses more infrequent words (than previous MT paradigms (SMT and RBMT).

II) NMT output shows more variation (and inconsistency) (than previous paradigms)

e.g. en: district nl: wijk, district, buurt

How can I prove these hypotheses?

I: use word frequency list (SONAR)

--> devide in classes (less frequent, more frequent, very frequent, ...)

--> check for each word occuring in the MT output in which frequency class it is classified.

OR

* or work with a total score for every MT system: sum (all positions)/#content words 🡪 average frequency score of a content word in that system.

the MT system with the lowest value uses the most frequent words.

Sonar frequency list can be found on babel : /var/data/lt3/corpora/SoNaR500

II : calculate the type/token ratio of the content words

type = all different forms occuring in the text (=output)

token= all words occuring in the text (=output)

Make it fancier maybe by adding a POS tagger, so only content words are taken into account, which makes more sense to know the ratio.

(function words and articles will pollute/ annihilate the findings/ results otherwise)

# Problem analysis

## Data:

## input:

### **MT output files : (for NMT, SMT and RBMT)**

|  |  |
| --- | --- |
| format | txt |
| encoding | Utf-8 (without BOM) 🡪 is this problematic, should I change it to normal utf-8? |
| location | lt3 server |

1. **Sonar500 frequency list** : txt file (very large)

|  |  |
| --- | --- |
| format | txt |
| encoding | Utf-8 |
| location | Babel server |

## Output:

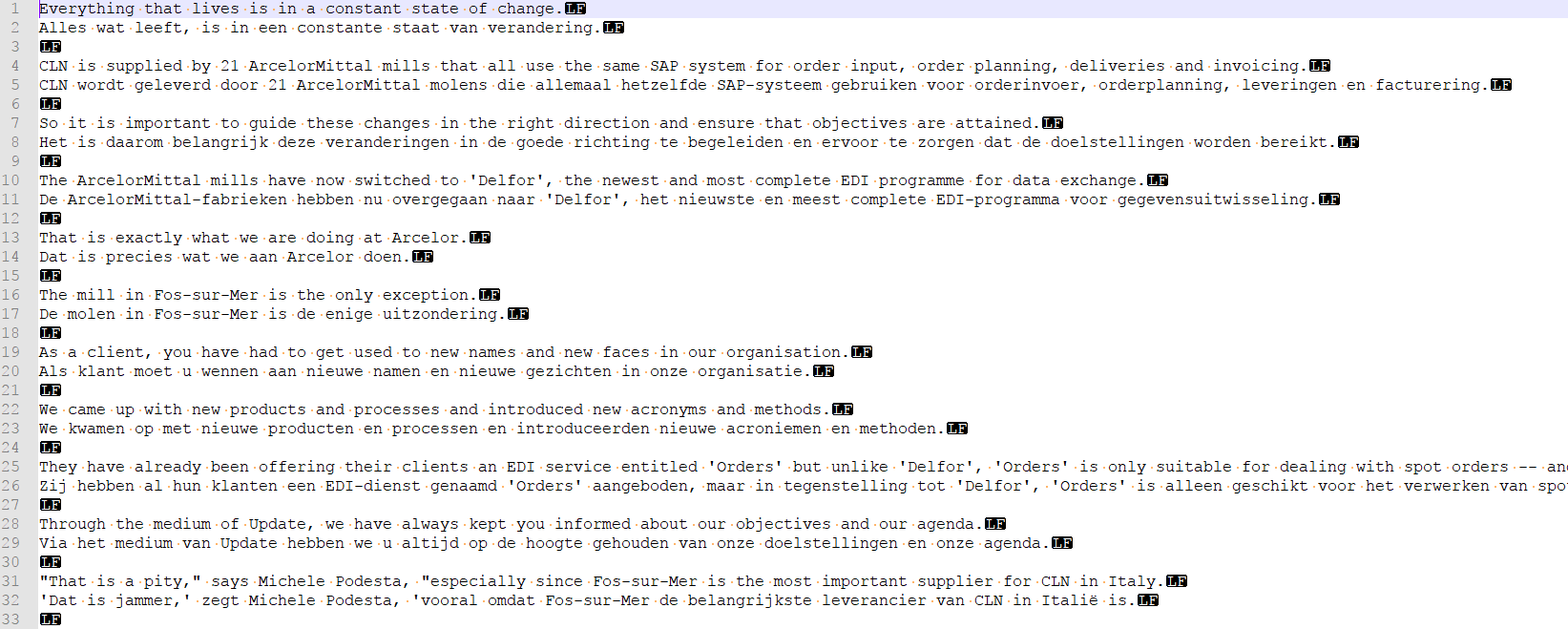
1) List with for every word in the MT output (NMT, SMT and RBMT), the frequenty in SoNaR 🡪   
🡪 visualized with Matplotlib as histograms?

2) Type token ratio for NMT, SMT and RBMT

## Processing:

#### **Step 1**

problem: The input files are bilingual (EN-NL) while for this research I only need the NL translations.  
Solution: write a python script that only contains the Dutch sentences.

Figure 1: bilingual txt file from corpus

I will write a script to only import line 2, 5, 8, … of each file (which as you can see on the screenshot above, corresponds exactly with the NL translations. Only the NL lines will be written to a new file, with a suiting filename (instead of xxxxxxxxxx\_en-nl 🡪 xxxxxxxxxx\_NL\_output).

The original bilingual files are stored for each MT system in the subfolder “00Bilingual”

the script to generate the clean NL files is called “Dutch\_output\_only.py “ and can be found in the directory “master”. The NL clean files are saved in the directory “01NL\_output”.

* Result clean files containing only the NL Machine Translation output

Attention : the NL\_output files are in fact the input of this particular project (output refers to the result of the translation done by the machine).

#### **Step 2: preprocess clean NL files**

Problem: the txt files are read as whole strings.  
solution: POS tagging of NL files, making use of LeTs: preprocessing tool of LT3

<http://lt3serv.ugent.be/wiki/index.php/LeTsPreprocess>)

LeTsPreprocess is installed on Lt3Babel (/opt/lt3\_preprocessor\_2.0) and Thoth.

LeTsPreprocess requires two arguments, viz. the path to the directory where the input files are located and the language code (nl, fr, en, de), e.g. 'prepro2.0 ~/test nl' or 'pos2.0 ./testdir fr'

The complete preprocessing pipeline can be called with the command prepro2.0:

* prepro2.0: entire preprocessing chain (charNorm, sbd, tok, pos, lem, chunk & ner)
  + Input: \*.txt
  + Output: \*.utf8, \*.sbd, \*.tok, \*.pos, \*.lem, \*.cnk, \*.ner

The different subcomponents can also be called separately:

* charNorm2.0: character normalization and encoding conversion to UTF8
  + Input: \*.txt
  + Output: \*.utf8
* sbd2.0: sentence boundary detection
  + Input: \*.utf8
  + Output: \*.sbd
* tok2.0: tokenisation
  + Input: \*.sbd
  + Output: \*.tok
* pos2.0: part-of-speech tagging
  + Input: \*.tok
  + Output: \*.pos
* lem2.0: lemmatization
  + Input: \*.pos
  + Output: \*.lem
* chunk2.0: chunking (shallow parsing)
  + Input: \*.pos & \*.lem
  + Output: \*.cnk
* ner2.0: Named Entity Recognition
  + Input: \*.pos & \*.lem
  + Output: \*.ner

🡪 via command line : prepro /directory name nl (en|fr)

I manually placed this generated pos files in the subfolder “03POS” of the MT systems.

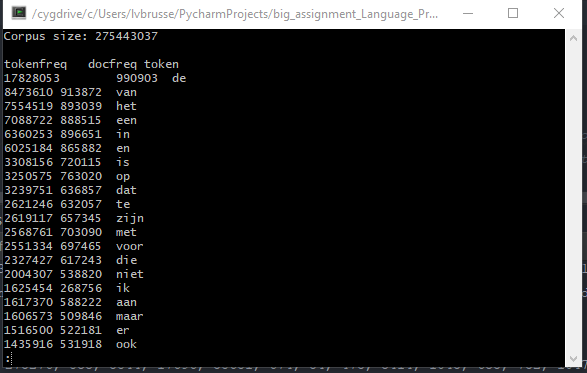
(folder 02tokenized was meant for the storage of the tokenized files, but since I opted to work directly with the pos files, this folder is empty.

For my Phd however, someday I might want to have and check the tokenized files of the MT output, therefor I kept this subfolder.)

#### **Step 3: get frequency list**

Sonar500 frequency list : txt file to compare the words of the MT output with.

NOTE: this frequency list is far too large to just open with notepad or something🡪 to have a look: best to open via command line and only show first 100 lines or search for specifics rows using Linux commands (less or more)



* readlines 🡪 returns a list of every line  
  from this list we only need the index (= position).
* tokens start on line 4  
  file

#### **Step 4: get frequency position**

Problem: Function words will disguise my hypothesis  
Solution: Use the POS tagged files to calculate type token ratio only taking into account the content words

Make an embedded dictionary

***Rough idea:***

dict = { }

For word in NL output:  
If word in dict:

occurences+= 1  
frequency = get value of word in SoNaR frequency list

If word not in dict

occurences=1

dict [word]=[occurences, value SoNaR]

NOTE: value= actually the position/ ranking of the word in the SoNaR frequenty list

RESULT :This will generate a dictionary with all the number of occurences and frequency ranking

#### **Step 5: analysis and visualization**

to investigate the frequency of the used content words in the MT output, it is not necessary to take into account the number of occurences.

therefore I will generate a list for each MT system, containing only the frequency positions for every content word

if we take the sum of these list divided by the number of content words, we can compare the systems and see which system uses most frequent words (and has the lowest sum).

For the whole dictionary (Occurrence \* ranking score)

Make a visualization of results of this list  
find a nice way to compare the results of the 3 MT systems.  
make use of matplotlib.

#### **Step 6 Calculate the Type Token Ratio**

Use the POS tagged files to calculate type token ratio only taking into account the content words

Input files = POS tagged files: the modified script:

RESULT :This will generate a list/ dictionary with all the number of occurrences and frequency ranking for the content words in the Dutch machine translated texts.

* + 1. Calculate Types:

Types= the set of unique words in the NL output

= number of keys in dict\_frequency\_output\_content

* + 1. Calculate Tokens:

Tokens = the total number of tokens. (all instances of the types)

sum of all keys multiplied by their occurrences

* + 1. Calculate Type Token Ratio (TTR):

Types/ tokens

Look for a nice visualization manner to compare the performance of all three systems

# Retrospective remarks:

NMT has a slightly higher TTR, but to proof the hypothesis that NMT has a higher variance, we need to have a bigger corpus of output probably.

With respect to the frequency of the content words, the average frequency score of NMT is indeed higher than SMT and RBMT. But one weakness is that we used the SoNaR frequency list which was built with text from the Dutch Parallel Corpus DPC, while our source files also stem from the DPC. This is a circular move, maybe it would be interesting to see what happens if we compare the output with another frequency list. The only problem might be that not much adequate frequency lists might be a better candidate….

A simple division of classes in the frequency list, based on the total number/ number of classes doesn’t represent the data well. cf. Zipfs law: the frequency of a word is inversely proportional to its rank in the frequency table. The most frequent words (articles mostly) will occur twice as often as the second most frequent word, three times as often as the third most frequent word.

Before making bins (for plotting the data), this should be taken into account to visualize the data.

-