ARL-TN-0923 ● Oct 2018

US Army Research Laboratory

Provenance and Processing of an Inuktitut-English Parallel Corpus Part 1: Inuktitut Data Preparation and Factored Data Format

by Jeffrey C Micher

ARL-TN-0923 ● OCT 2018

US Army Research Laboratory

Provenance and Processing of an Inuktitut-English Parallel Corpus Part 1: Inuktitut Data Preparation and Factored Data Format

by Jeffrey C Micher
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE					*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**					
1. REPORT DATE *(DD-MM-YYYY)* October 2018	**2. REPORT TYPE** Technical Note				**3. DATES COVERED (From - To)**
4. TITLE AND SUBTITLE Provenance and Processing of an Inuktitut-English Parallel Corpus Part 1: Inuktitut Data Preparation and Factored Data Format					**5a. CONTRACT NUMBER**
					5b. GRANT NUMBER
					5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Jeffrey C Micher					**5d. PROJECT NUMBER**
					5e. TASK NUMBER
					5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-CII-T 2800 Powder Mill Road Adelphi, MD 20783-1138					**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-0923
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)					**10. SPONSOR/MONITOR'S ACRONYM(S)**
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT We describe the Nunavut Hansard, a parallel English–Inuktitut corpus derived from Nunavut legislative proceedings, and we describe the processing that was carried out to prepare the data for use in morphological analysis and downstream machine translation experiments. We provide all of the scripts and code used to process the data.					
15. SUBJECT TERMS Polysynthetic languages, parliamentary proceedings, Inuit languages, Inuktitut, morphology, machine translation, data processing					
16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Jeffrey C Micher
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	76	**19b. TELEPHONE NUMBER (Include area code)** (301) 394-0361

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Tables

1. Introduction

The purpose of this technical note is to describe the provenance and processing of the Inuktitut–English parallel corpus of data that has been used in a variety of experiments in natural language processing, morphological analysis, and machine translation at the US Army Research Laboratory (ARL). First, the origin of the data set is discussed. Second, the steps that were followed to "preprocess" the data set for experiments are described. Finally, in the Appendixes, the scripts used to do the data processing are provided.

2. Origin of the Inuktitut–English Corpus

The Inuktitut–English corpus originated during the ACL 2003 Workshop entitled "Building and Using Parallel Texts: Data-driven Machine Translation and Beyond" (HLT-NAACL 2003) and was made available during this workshop. The data were subsequently used for a shared task on alignment that took place in the same-named workshop in 2005 (UMD 2005). Participants were asked to develop methods of word alignment for this data set, which, at the time, was the only parallel data set containing English and some polysynthetic language, presenting a challenge to the state of the art in word alignment. The data set was assembled and sentence-aligned, and is described in Martin et al. (2003). The data set is available from http://www.inuktitut computing.ca/NunavutHansard/info.php and was downloaded from there. See this web page for an exact description of which Nunavut Hansard proceedings dates are contained in the data set.

From this site, version 1.0, 1.1, and 2.0 are available. The data set that was downloaded and used in experiments at ARL was the version 1.1. Note, the version 1.1 data set is one file containing a line of Inuktitut, a separator line, a line of English, and another separator line. This data set was subsequently processed for use in the second workshop mentioned and provided in the form of three zip files, one containing a "training" set, one a "trial" set, and one a "test" set[*]. The trial and test sets contained data held out from the training set and were used to develop and test the word alignment algorithms. The data in the training set, however, contained two parallel English and Inuktitut files, and it was these files that were used as the starting point for subsequent preprocessing. The files were called "SentenceAligned.v1_1.txt.e" and "SentenceAligned.v1_1.txt.i," containing English text and Inuktitut text, respectively. These two data files contain 340,526 lines each. The English file contains 3,992,298 tokens, with 27,127 types, and the

[*] These files are no longer available. The link to them is broken. However, they can be reconstituted from the original version 1.1 text file.

Inuktitut file contains 2,153,034 tokens, with 417,406 types (Table 1). The type-token ratios of the two data sets are dramatically different: 0.0067 for English versus 0.1938 for Inuktitut. The percentage of singletons is also dramatically different, with 32.41% in English versus 80.93% in Inuktitut. The average word length in characters is: 4.26 in English and 9.31 in Inuktitut. The average line length (number of words in line) is 11.72 in English and 6.22 in Inuktitut.

Table 1 English and Inuktitut parallel data compared

	English	**Inuktitut**
Tokens	3,992,298	2,153,034
Types	27,127	417,406
Type-token ratio	0.0067	0.1938
Percentage of singletons	32.41%	80.93%
Average word length in characters	4.26	9.31
Average line length in words	11.72	6.22

3. Preprocessing Steps

3.1 Inuktitut Morphological Analysis

Since the current research work revolves around what to do with morphologically complex languages like Inuktitut, a crucial step in working with such data is to perform morphological analysis. The following sections describes an available morphological analyzer for Inuktitut and how it was used to carry out the processing.

3.1.1 Morphological Analyzer

An Inuktitut morphological analyzer was developed in 2009 at the Institute for Information Technology within the National Research Council of Canada. It is currently available at http://www.inuktitutcomputing.ca/Uqailaut/ for downloading. The analyzer was used as downloaded, with no alterations to the source code whatsoever.

From the README file contained in the download, the following information has been extracted:

> *The Inuktitut Morphological Analyzer is written in Java 1.6.*
>
> *Its purpose is to decompose an inuktitut [sic] word into its constituent parts (or morphemes). The analyzer may return several decompositions for a given word. In general, the right analysis can be found in the top three ones.*

The analyzer takes an Inuktitut word as input and returns a morphological analysis or multiple morphological analyses if the word is ambiguous. When multiple analyses are returned, they are returned in multiple lines. Each analysis consists of a string of morphemes and analysis detail, enclosed in curly braces {}. Each morpheme and unit of analysis detail contain the surface form of the morpheme, followed by a colon, followed by the deep (dictionary) form of the morpheme, followed by a forward slash, followed by the morphological analysis detail. The following is a model of the structure and an example.

```
{<surface form>:<deep form>/<morphological analysis detail>}{..}{..}..etc.
```

The analysis of the word "saqqitaujuq" would return:

```
{saqqi:saqqik/1v}{ta:jaq/1vn}{u:u/1nv}{juq:juq/1vn}

{saqqi:saqqik/1v}{ta:jaq/1vn}{u:u/1nv}{juq:juq/tv-ger-3s}
```

See the Readme files contained in the download for further explanation of the information contained in the morphological analysis, since the details of this are not in the scope of this technical note.

3.1.2 Running the Morphological Analyzer

As the analyzer was written in Java, it can be run anywhere. Upon initial investigation of the speed of the analyzer running it on a standalone laptop, it was determined that certain strategies should be applied to minimize the time spent running the analyzer, since each word analyzed could take anywhere from less than a second to minutes to run. Since the analyzer does not rely on context, every unique "type" in the Inuktitut corpus was collected up, rather than running the analyzer on each and every "token": there are a total of 2,153,034 tokens, in the corpus, represented by 417,406 unique types. A database (in multiple file format) of the analyses provided for each word type was created and used in later processing steps to assign the appropriate analysis to each token in the corpus.

3.1.2.1 Collecting up the Types from the Inuktitut Half of the Corpus

To collect up the types in the Inuktitut corpus, the script corpus_stats.pl (see Appendix A) was used with the –f option, which provides a list of the types from a corpus of text, sorted by frequency, with the frequency count separated by two spaces. The input file was the Inuktitut half of the corpus, the "SentenceAligned. v1_1.txt.i" file. The output was named "SentenceAligned.v1_1.txt.freq.i". This file was then fed to a simple Perl command line invocation to remove the frequency numbers:

```
perl -ne '/^([^\s]+)\s/; print "$1\n";'
```

which says to match one or more non-space characters anchored at the beginning of the line before a single-space character, capture them, and print them out. The resulting file was named "SentenceAligned.v1_1.txt.freq.list.i". From there, since we only wanted to try to analyze words rather than words containing typos or processing errors containing punctuation characters, another line of Perl was used:

```
perl -ne 'if (/^[a-zA-Z\&]+$/) { print; }'
```

such that "if you match only any alphabetic characters (including the ampersand, which is used as an alphabetic character in this corpus) anchored at the beginning and end of the string, print them out." In addition, the inverse of this script was used to output those types that were not matched, with

```
perl -ne 'unless (/^[a-zA-Z\&]+$/) { print; }'
```

The respective output files from these two processes were named "SentenceAligned.v1_1.txt.freq.list.wordsonly.i" and "SentenceAligned.v1_1.txt. freq.list.whatsleft.i". The former contained 413,533 lines corresponding to as many types, and the latter 3,853 lines, corresponding to that many erroneous types.

3.1.2.2 Wrapping the Analyzer to Catch Processing Errors

When first running the analyzer, it was noticed that the analyzer sometimes crashed for various reasons: either it threw a NullPointerException or Java ran out of memory, or there was just no response for a long period of time. To address these various errors, a wrapper was built around the analyzer code to catch them, provide some type of output message, stop processing when no result was provided after a certain time limit, and continue running the analyzer. The wrapper InukMorph.java was written in Java and the code is contained in Appendix B.

InukMorph.java makes use of a variety of classes, and all of the code is provided in Appendix B. InukMorph.java creates an Analyzer object for each word in an input file. The Analyzer object wraps the Uqailaut.jar process and provides an analyze method. The analyze method collects NullPointerExceptions and OutOfMemory exceptions that the underlying process throws, and returns a code for each (either NP or OM). The Analyzer object uses code that had been previously written and modified somewhat to manage underlying processes. An MTModule object, which inherits from MTModuleBasic, creates a Runtime system process and a way to access the output and error streams from those processes after a certain wait time has been exceeded. It does this by creating StreamConsumer objects, which fire off Worker threads to check the streams. Once the consumers have completed their job, or have been running for 5 min without completing, the output from the streams is collected. If there were error messages in the error stream or if 5 min had elapsed with no output, the error message is returned (with "exceeded time limit" as the error message in the latter case); otherwise, the output from the output stream is returned.

3.1.2.3 Running the Wrapper in Batch Mode on an External Cluster

In the best-case scenario, with 413,533 types to process, each taking from 1 s to 5 min, would have required up to 4.78 days straight, or in the worst case, 1,435.87 days (i.e., 3.93 years) to process. The 1,000 most frequent types were processed through the analyzer, and the time per type was measured. On average for this set, each type took 1.299 s to process. That still came out to 6.21 days of straight processing when processing one type at a time. To alleviate this time burden, the analyzer process was sent off in batch jobs to a computer cluster.[†] First, the data were divided into 414 sets of 1,000 types each, with the last set containing 533 types. This was done using the script get_subsets.pl (Appendix C), creating files with the base name of "inuk" and extension numbers. The extension numbers were two digits from 01 to 99, then three digits from 100 to 414. The files were saved in an "in" directory. The input file to the get_subsets.pl script was the "SentenceAligned.v1_1.txt.freq.list.wordsonly.i", copied over to a file named "inuk" to provide a unique base file name for the script. Then, using these files as input, a bash script, qsubrun, (Appendix D) was used to create processing jobs to send to the cluster. The qsubrun uses the PBS job processing scripting language, fires off a job, and indicates where the output and error streams are written for the job when completed. Another layer written in Perl, was wrapped around this bash script, run_morph.pl (Appendix E), which fired off as many sets as desired.

It is worth dissecting the line of code inside the run_morph.pl script since it is quite complex. The following is the line:

[†] workhorse.lti.cs.cmu.edu, a cluster maintained by the Language Technologies Institute at Carnegie Mellon University.

```
system("qsubrun \"/home/jmicher/bin/java/bin/java -jar
/home/jmicher/myscripts/inuktitut_scripts/InukMorph.jar
inuk.".$i." \'/home/jmicher/bin/java/bin/java -jar
/home/jmicher/inuktitut/Uqailaut.jar\'\" inuk.$i");
```

The first thing to note is that the whole line is a Perl system call, “system(….).” Inside the parentheses is one long string that represents this system call. The string can be divided into three pieces, using the whitespace character (after resolving the variable $i and concatenating the string):

```
qsubrun
```

```
\"/home/jmicher/bin/java/bin/java -jar
/home/jmicher/myscripts/inuktitut_scripts/InukMorph.jar
inuk.".$i." \'/home/jmicher/bin/java/bin/java -jar
/home/jmicher/inuktitut/Uqailaut.jar\'\"
```

```
inuk.$i
```

What this indicates is that qsubrun is called with two arguments. The first argument is the command line that qsubrun fires off as a job to the cluster, and the second argument is a string that qsubrun uses to identify the job. It uses this string in two ways: 1) it creates a PBS script in a temp location using this string as the base, with a random 3-character extension, and 2) it uses this string to create a filename base for writing the error and output streams from the process it runs. This base consists of this string, concatenated with a “.” and the process ID of the qsubrun process (“$$”), concatenated with the letter “f.”

The command line that qsubrun fires off is dissected as follows:

```
\"/home/jmicher/bin/java/bin/java
```

```
-jar /home/jmicher/myscripts/inuktitut_scripts/InukMorph.jar
```

```
inuk.".$i."
```

```
\'/home/jmicher/bin/java/bin/java -jar
/home/jmicher/inuktitut/Uqailaut.jar\'\"
```

This is a java process, the code of which is stored in the executable jar file named “InukMorph.jar”, which takes two arguments: 1) a filename, (again, once the $i is resolved and concatenated into the string) and 2) a string (in single quotes, since the outer string is double quoted) representing the process that *IT* wraps, which happens to be another java -jar process, specifying the Uqailaut.jar as the executable jar file.

The outermost run_morph.pl script loops through as many jobs as are specified as arguments to it, in the form of lower and upper file extension numbers. Since there were quotas in place for how many jobs an individual could run at a time, run_morph.pl was fired off to process in batches of 10.

The output from running the analyzer was collected for each type and saved in corresponding .out files in an /out directory.

3.1.2.4 Results of Running the Morphological Analyzer

The output was then examined, and there were several possible outcomes for each word type: 1) the type was analyzed and an analysis or multiple analyses were provided; 2) the analyzer determined that it couldn't process the type and returned nothing; for these outputs, the InukMorph.jar wrapper provided an NA (no analysis) as the output; 3) the wrapper waited for 5 min and when it didn't get an analysis, supplying a "time exceeded" message and moving to the next type; 4) the wrapper caught a NullPointerException and provided this exception as the output and moved on; and 5) the wrapper did not capture anything in the output stream of the process it was wrapping for some reason. In these instances, the InukMorph process provided an "output holder empty" message. These messages and their types were counted up from the .out files using the script "count_analyses.pl" (Appendix F). Out of 413,553 types, the following were observed:

Analyzed types:	285,594
No analysis:	123,671
Null Pointer Exception:	305
Out of Memory:	18
Exceeded Time Limit:	2,372
Output missing:	1,593
Total:	413,553

3.1.2.5 Re-Processing Types That Caused Errors

The types with "missing output" or that "exceeded time limit" were subsequently reanalyzed on the same cluster. It is not clear why some types initially exceeded the time limit or produced missing output. In the case of missing output, it was observed that due to the nature of how the output stream was collected, occasionally, for the same type, the output stream could be lost. So after rerunning the analyzer on the 1,593 types that had an unretrievable output stream, 1,075 types resulted in a good analysis and 518 produced an NA (no analysis). These types are located in a file named reanalyzed_untretrievable_output.out.

The types that originally produced an "exceeded time limit" error were also subsequently reanalyzed. Out of 2,372 of these types, 1,189 types returned a good analysis, with 1,183 continuing to use more than 5 min of time for processing (and subsequently passed over.) These types are located in a file named exceeded_ analyzed.out.

Finally, the new analyses from the reanalyzed_unretrievable_output.out file and the exceeded_analyzed.out file were incorporated into the analyses database. First, a new analyses database directory was created called newout. All of the .out files from the original database were copied over using "cp ../out/*.out ." Then, a script was used, update_inuk_data.pl (Appendix G), to take in the new analysis from the two reanalyzed files and rewrite the .out files. Initially, this script created a .bak file for each .out file in case there were errors and the original files needed to be retrieved. But once this script was run and the counts were verified with count_analyses.pl, the .bak files were removed from the database (rm *.bak).

Final counts for the analyzed types are the following:

Analyzed types:	287,858
No analysis:	124,189
Null Pointer Exception:	305
Out of Memory:	18
Exceeded Time Limit:	1,183
Output missing:	0
Total:	413,553

Future work will include running the morphological analyzer over the remaining 1,183 "exceeded time limit" types with a longer time delay to see if simply giving the analyzer more time to work will produce some output for these remaining unanalyzed types.

Nicholson et al. (2012) report that the analyzer is able to provide at least a single analysis for approximately 218k Inuktitut types (65%) from the Nunavut Hansard corpus. Their 218k number may be an error, since they report the number of types to be 416k. Nonetheless, their finding that the analyzer does not process each and every type is in line with the current work, with approximately 30% of the types from the corpus not having an analysis.

3.2 Creating the Factored Files

3.2.1 Creating the Inuktitut Factored File

3.2.1.1 The Structure of the Inuktitut Factors

Factored files for various experiments in factored machine translation were created from the corpus data plus the morphologically analyzed tokens. The factors included in the Inuktitut factored format were 1) surface form of the morpheme (true surface form), 2) surface form as returned by the analyzer[‡], 3) deep form of the morpheme, 4) the morphological analysis (what the analyzer outputs for that morpheme), 5) morpheme category, and 6) morpheme subcategory. The first two factors are self-explanatory. The morphological analysis (factor #4) is given according to Table 2. This table was taken from the README file provided with the morphological analyzer.

[‡] The analyzer regularizes certain spellings, which was throwing off the length of words, so the script was edited to provide the original surface form, split according to where the analyzer indicated splits. For example, *itsivangngurama* is analyzed and returned by the analyzer *as {itsiva:iksiva/1v}{nngu:nnguk/1vv}{rama:gama/tv-caus-1s}*, which indicates that its surface form is *itsivanngurama*, missing a “g”.

Table 2 Morpheme grammatical information

<unique id>	<name>/<nb><type>
<nb>	integer
<type>:	
v	verb root
n	noun root
a	adverb
c	conjunction
q	tail suffix
nn	noun-to-noun suffix
nv	noun-to-verb suffix
vn	verb-to-noun suffix
vv	verb-to-verb suffix
tv-<mode>-<subject's person & number>[-<object's person & number>-[prespas|fut]]	verb ending
tn-<case>-<number>[-<possessor's person & number>]	noun ending
tad-<case>	demonstrative adverb ending
tpd-<case>-<number>	demonstrative pronoun ending
pd-<location>-<number>	demonstrative pronoun
ad-<location>	demonstrative adverb
rad	demonstrative adverb root
rpd	demonstrative pronoun root
<mode>: dec	declarative
ger	gerundive
int	interrogative
imp	imperative
caus	causative
cond	conditional
freq	frequentative
dub	dubitative
part	participal
<case>: nom	nominative
acc	accusative
gen	genitive
dat	dative
abl	ablative
loc	locative
sim	similaris
via	vialis
<number> : s	singular
d	dual
p	plural
<location> : sc	static or short
ml	moving or long

The following are some examples of analyzer output to demonstrate the morphological information codes:

- quja:quja/1v: first dictionary entry for this root; it is a verbal root.
- va:vaq/1nv: first dictionary entry for this lexical postbase; it converts a noun to a verb.
- jait:jait/tv-ger-2s-3s: verbal suffix, which is gerundive and indicates second-person singular subject with third-person singular object.
- manna:manna/pd-ml-s demonstrative pronoun, moving/long, singular.

The morpheme category (factor #5) is derived from the grammatical information given by the analyzer (factor #4) and includes ROOT, LEX, GRAM, CL, ADV, CONJ, EXCLAM, PR, AD, PD, RAD, RPD, TAD, and TPD. See Appendix H: POS.txt for an explanation of the labels. The morpheme subcategory (factor #6) is also derived from the grammatical information and serves to distinguish the various root and lexical postbase types.

The factors for words that have no analysis from the analyzer are 1) surface form (true), 2) surface form (analyzer), 3) surface form (because there is no analysis), 4) NA, 5) NA, and 6) NA.

3.2.1.2 Scripts Used for Creating the Inuktitut Factored File

Two scripts are used to produce the Inuktitut factored file, tok_punc.pl (Appendix I) and preprocess_inuk.pl (Appendix J), which uses the InukProcessing.pm perl module (Appendix K). First, all punctuation characters are tokenized and multiple dots (greater than three) are standardized to three dot ellipses, using the tok_punc.pl script. Then, the preprocess_inuk.pl script uses the results from running the morphological analyzer, stored in the out/*.out files, along with a punc.out file (added to these files to label punctuation characters) to create the Inuktitut factored file. The punc.out file (Appendix L) is formatted just as the .out files are formatted, so that punctuation can be labeled correctly in the factored file. The input file to this process was "SentenceAligned.v1_1.txt.i" and the output file was named "SentenceAligned.v1_1.txt.morphednfactored.i".

3.2.2 Creating the English Factored File

As we do not report on using the English data in a factored format, we do not include details here about how an English factored file was constructed. We simply used the English half of the corpus, tokenized with the tokenizer script provided by Moses.

3.3 Creating the Data Sets for Various Experiments

The “divides” are a set of index files that can be used to create a train, tune, and test set from the preprocessed Inuktitut–English corpus. There are 100 different index sets, each having a different randomly sampled tune and test set, leaving the remaining data for training, with no duplicates between any set. With the ability to select the data this way, 100 different experiments can be performed using the same data set to ensure some degree of statistical significance. In addition, different treatments of the base data set can be compared against others by using the same index set for the experiments being conducted.

First, the factored Inuktitut–English parallel corpus was copied and renamed to allow working with the clean-corpus-n.perl script from the current Moses (2018) distribution, which requires that the filenames of each half of the corpus have the same base, with language extension codes for each language. The new names for these files became “EI.factored.final.en” and “EI.factored.final.in”. From there, Inuktitut words were rejoined so that the index files would be relevant for baseline systems that used the Inuktitut data before the words had been split into morphemes. The subsequent names became “EI.fullword.en” and “EI.fullword.in” (again, conforming to the requirement of the clean-corpus-n.perl script, even though no additional processing had taken place on the English half of the corpus.) The extract_full_inuk_word.pl script (Appendix M) was used to extract the full Inuktitut word from the factored Inuktitut file. Since the filtering script worked on word counts, it was not necessary to extract out the English words from the factored English file, so this file was simply renamed, keeping in mind that when using it later, surface forms (factor 0) for the English words would have to be extracted from the factored file. The English file, however, was lowercased using the Moses script lowercase.perl (which, incidentally, lowercased the part-of-speech tags in factor #2, but this was unimportant.) and the resulting file was given a “.low” extension. The Inuktitut file, however, did not need to be lowercased, and, in fact, should not have been lowercased since the romanized Inuktitut form used in the corpus makes use of an uppercase “H”, which is distinct from a lowercase “h”. However, a copy of the Inuktitut file was made with a “.low” extension to match the name of the English lowercased file for filtering purposes. Finally, the clean-corpus-n.perl script from Moses was used to filter the corpus, with 80 as the max number of words in each line. The index numbers were retained in a “linesretained.txt” file. The output files were named “EI.fullword.low.filtered.en” and “EI.fullword.low.filtered.in”.

Although we used the English factored file when creating the “divides” index files, we reverted back to using the original English corpus, processed as described in

Section 3.2.2. This file, along with the Inuktitut factored file, were the bases for machine translation experiments.

4. Conclusion and Future work

We described the processing of the Inuktitut half of the Nunavut Hansard corpus and provided all the code for doing the processing. In future work, we will continue experiments with Inuktitut morphological analysis and machine translation, and describe the processing steps and code for various envisioned neural network experiments.

5. References

[HLT-NAACL] Building and using parallel texts: data driven machine translation and beyond. HLT-NAACL 2003 Workshop; 2003 May 27–June 1; Edmonton, Canada [accessed 2018]. http://web.eecs.umich.edu/~mihalcea/wpt/.

Martin J, Johnson H, Farley B, Maclachlan A. Aligning and using an English-Inuktitut parallel corpus. In: Mihalcea R, Pedersen T, editors. Proceedings of the HLT-NAACL 2003 Workshop on Building and Using Parallel Texts: Data Driven Machine Translation and Beyond–Volume 3; 2003 May 31; Edmonton, Alberta (Canada). Stroudsburg (PA): Association for Computational Linguistics; c2003. p. 115–118.

Moses statistical machine translation system. 2018 June 24 [accessed 2018]. http://statmt.org/moses/.

Nicholson J, Cohn T, Baldwin T. Evaluating a morphological analyser of Inuktitut. Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2012 June 3–8; Montreal (Canada). Stroudsburg (PA): Association for Computational Linguistics; c2012. p. 372–376.

[UMD] Data-driven machine translation and beyond. ACL 2005 Workshop on Building and Using Parallel Texts; 2005 June 29–30; College Park, MD. University of Maryland Institute for Advanced Computer Studies [accessed 2018]. http://www.statmt.org/wpt05/.

Appendix A. corpus_stats.pl

```
#!/usr/bin/perl

#################################################################################
#
# Name: corpus_stats.pl
# Date Created: Sometime in the Fall of 2013
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
################################################################################
#
# Description: This script will calculate various statistics over a corpus, and
# output various frequency lists.  See usage for more info.
#
################################################################################
#
# Change Log:
#
# 29 Nov 13: Added header from header.txt template.
#
# 20 Nov 13: Initial version, checked in to repository.
#
################################################################################

my $usage = "Usage: \n";
$usage .= "-f                sort by frequency\n";
$usage .= "-ft[0-9]+         sort by frequency, displaying top n items\n";
$usage .= "-ff[0-9]+           sort by frequency, displaying items with frequency
greater than n\n";
$usage .= "-w                sort by word length, display all\n";
$usage .= "-wt[0-9]+         sort by word length, display top n items\n";
$usage .= "-wl[0-9]+           sort by word length, displaying items with length
greater than n\n";
$usage .= "-s                sort substrings by frequency\n";
$usage .= "-st[0-9]+        sort substrings by frequency, displaying top n items\n";
$usage .= "-c[0-9]+          output type-token curve data, at intervals of n\n";
$usage .= "-r                reverse sort frequency\n";
$usage .= "-p                search for a pattern\n";
$usage .= "-toc              token count\n";
$usage .= "-tyc              type count\n";
$usage .= "-tsc              symbol type count\n";
$usage .= "-y                sort symbols by frequency\n";
$usage .= "-e                sort last n characters by frequency, default is 6\n";
$usage .= "-l                sort by line frequency";

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

my $linecount = 0;
$longest = "1";
my $data = [];
my $word_freq = {};
my $seg_len = [];
my $token_count = 0;
my $type_count = 0;
my $type_tok_ratio = 0;
my $hapax_count = 0;
my $line_len_freq = {};
my $type_tok_curve = [];
my $symbol_freq = {};
my $symbol_tok_count = 0;
my $end_freq = {};
my $line_freq = {};

if ($ARGV[0] eq "-h") {
    print STDERR "$usage\n";
    exit(0);
}


while (<STDIN>) {

    chop;
    $line = $_;
    $line_freq->{$line}++;
    # symbol counts
    # keep spaces intact
    for ($i = 0; $i < length($line); $i++) {
       $symbol_freq->{substr($line, $i, 1)}++;
       $symbol_tok_count++;
    }
    $line =~ s/^\s+//; $line =~ s/\s+$//; $line =~ s/\s+/ /g;
    $word_count = 0;
    $words = [];
    @$words = split(/ /, $line);
    foreach $w (@$words) {
       $word_count++;
       unless (exists($word_freq->{$w})) {
           $type_count++;
       }
       $word_freq->{$w}++;
       $token_count++;
       push(@$type_tok_curve, "$token_count\t$type_count");
       $ave_word_len += length($w);
    }
    $ave_line_len += $word_count;
    $data->[$linecount] = $words;
    $linecount++;
}
$ave_word_len = $ave_word_len/$token_count;
# type count
@types = keys (%$word_freq);
$type_count = @types;
$type_tok_ratio = $type_count / $token_count;
$ave_line_len = $ave_line_len / $linecount;
foreach $i (keys %$word_freq) {
    if ($word_freq->{$i} == 1) { $singletons++; }
}

print STDERR "TTR: $type_tok_ratio\nAVE LINE LENGTH: $ave_line_len\nAVE WORD LEN:
$ave_word_len\n";
print STDERR "SINGLETONS: $singletons\n";
$s_percent = $singletons/$type_count*100;
print STDERR "SINGLETON PERCENT: $s_percent\n";

# symbol set types
@symtypes = keys (%$symbol_freq);
$symbol_count = @symtypes;

print STDERR "Done reading corpus\n";
if ($ARGV[0] eq "-e") {

    foreach $w (@types) {
       for ($i = 1; $i < length($w) && $i < 7 ; $i++) {
           $end = substr($w,$i*-1);
           $end_freq->{$end} += $word_freq->{$w};
       }
    }
    foreach $end (sort {$end_freq->{$b} <=> $end_freq->{$a}} keys (%$end_freq)) {
       print STDOUT "$end\t$end_freq->{$end}\n";
    }
}

if ($ARGV[0] eq "-y") {
    print STDERR "symbol count (tokens): $symbol_tok_count\n";
    print_symbol_freq_by_top();
```

```
}

if ($ARGV[0] eq "-toc") {
    # token count
    print STDOUT "$token_count tokens\n";
}
if ($ARGV[0] eq "-tyc") {
    # type count
    print STDOUT "$type_count types\n";
}
if ($ARGV[0] eq "-tsc") {
    # symbol type count
    print STDOUT "$symbol_count symbols\n";
}

if ($ARGV[0] eq "-p") {

    $count = 0;
    ($c, $f) = count_search($data, $ARGV[1]);
    print STDERR "pattern: $ARGV[1] count: $c\n";
    foreach $key (sort {$f->{$b} <=> $f->{$a}} keys (%$f)) {
       print STDERR "$key\t$f->{$key}\n";
       if ($count > 100) { last; }
       $count++;
    }
}

if ($ARGV[0] =~ /^-f/) {
    #sort by frequency

    if ($ARGV[0] =~ /^-ff(\d+)$/) {
        #display until d items are printed
       print_word_freq_by_freq_bound($1);

    } elsif ($ARGV[0] =~ /^-ft(\d+)$/) {
       #display top d items
       print_word_freq_by_top($1);

    } else {
       #display all
       print_word_freq_by_top();
    }
}

if ($ARGV[0] =~ /^-s/) {
    #sort substrings by frequency

    if ($ARGV[0] =~ /^-sf(\d+)$/) {
        #display until d items are printed
       #print_word_freq_by_freq_bound($1);

    } elsif ($ARGV[0] =~ /^-st(\d+)$/) {
       #display top d items
       print_substring_freq_by_top($1);

    } else {
       #display all
       print_substring_freq_by_top();
    }
}

if ($ARGV[0] =~ /^-c(\d+)$/) {

    print_type_token_curve_data($1);
}

if ($ARGV[0] eq "-w") {
    #sort by length
    if ($ARGV[0] =~ /\d+$/) {
       print_types_by_length($&);
    }
```

```
}

if ($ARGV[0] eq "-l") {
    #sort by line frequency
    line_freq();
}
if ($ARGV[0] eq "-r") {
    reverse_sort_types();
}

##############################################################################
##
##  subroutines
##

sub line_freq {

    foreach  $line  (sort  {    $line_freq->{$b}    <=>    $line_freq->{$a}  }
keys(%$line_freq)) {
        print "$line_freq->{$line}\t$line\n";
    }
}

sub print_ave_line_length {
    print STDERR "AVE LINE LEN: $ave_line_len\n";
}
sub print_linecount {
    print STDERR "lines: $linecount\n";
}

sub print_tok_count {
    print STDERR "tokens: $token_count\n";
}
sub print_type_count {
    print STDERR "types: $type_count\n";
}
sub print_type_tok_ratio {
    print STDERR "type-token ratio: $type_tok_ratio\n";
}

sub ave_word_length {
    print STDERR "ave word len: $ave_word_len\n";
}

$num = 100;

sub print_type_token_curve_data {

    my ($interval) = @_;

    for (my $c = 0; $c < @$type_tok_curve; $c++) {
        if ($c % $interval == 0) {
            print "$type_tok_curve->[$c]\n";
        }
    }
}

sub print_types_by_length {

    my $bound = $_[0];
    foreach $w (sort {length($b) <=> length($a) || $word_freq->{$b} <=> $word_freq-
>{$a}} keys(%$word_freq)) {
        $len = length($w);
        if ($len > $bound) {
            print STDOUT "$w\t$len\t$word_freq->{$w}\n";
        }

    }
}
sub print_word_freq_by_freq_bound {
    my $bound = $_[0];
    foreach $w (sort {$word_freq->{$b} <=> $word_freq->{$a} || $a cmp $b} keys
(%$word_freq)) {
       if ($word_freq->{$w} > $bound) {
           print STDOUT "$w  $word_freq->{$w}\n";
       }
    }
}
sub print_word_freq_by_top {

    my $top = $_[0];
    my $count = 0;
    foreach $w (sort {$word_freq->{$b} <=> $word_freq->{$a} || $a cmp $b} keys
(%$word_freq)) {
       print STDOUT "$w  $word_freq->{$w}\n";
       if ($top ne "") { last if ($count > $top-2); }
       $count++;
    }
}

sub print_symbol_freq_by_top {

    my $top = $_[0];
    my $count = 0;
    foreach $w (sort {$symbol_freq->{$b} <=> $symbol_freq->{$a} || $a cmp $b} keys
(%$symbol_freq)) {
       print STDOUT "$w  $symbol_freq->{$w}\n";
       if ($top ne "") { last if ($count > $top-2); }
       $count++;
    }
}

sub print_substring_freq_by_top() {
    my $top = $_[0];
    my $count = 0;
    my $substring_freq = {};
    my $freq;
    my $len;

    # first create substring freq hash from word freq hash
    foreach $w (keys (%$word_freq)) {
       $freq = $word_freq->{$w};
       #include word boundaries
       $w = "#".$w."#";
       $len = length($w);
       for (my $winsize = 2; $winsize < $len; $winsize++) {
           # index
           for (my $i = 0; $i <= $len-$winsize; $i++) {
              $substring_freq->{substr($w, $i, $winsize)} += $freq;
           }
       }
       $count++;
       if ($count % 1000 == 0) {
           print STDERR "$count types processed in substring subroutine\n";
       }
    }
    print STDERR "Done calculating substring frequency\n";
    $count = 0;
    foreach $substring (sort {$substring_freq->{$b} <=> $substring_freq->{$a} || $a
cmp $b} keys (%$substring_freq)) {
       print STDOUT "$substring  $substring_freq->{$substring}\n";
       if ($top ne "") { last if ($count > $top-2); }
       $count++;
    }
}

sub reverse_sort_types {
    foreach $w (sort {reverse($a) cmp reverse($b)} keys (%$word_freq)) {
       if ($word_freq->{$w} > $bound) {
           print STDOUT "$w  $word_freq->{$w}\n";
       }
    }
}

sub print_singleton_count {
    my $count = 0;
    print STDERR "$count non-hapax types\n";
}

sub print_non_hapax_percent {
    #$non_hapax_percent = $count/$type_count*100;
    #print STDERR "Non-hapax percentage: $non_hapax_percent\n";
}

sub print_hapax_percent {

    #$hapax_percent = 100 - $non_hapax_percent;
    #print STDERR "So hapax percentage: $hapax_percent\n";
}

sub count_search {

    # this subroutine will count the number of tokens that contain a subpattern
    my ($data_ref, $pattern) = @_;
    my $count = 0;
    my $line_count = @$data_ref;
    my $ref;
    my $patt_freq = {};

    print STDERR "LINECOUNT: $line_count\n";
    for (my $line = 0; $line < $line_count; $line++) {
       $ref = $data_ref->[$line];
       my $line_length = @$ref;
       for (my $word = 0; $word < $line_length; $word++) {
           if ($ref->[$word] =~ /$pattern/) {
              $patt_freq->{$ref->[$word]}++;
              $count++;
           }
       }
    }
    return ($count, $patt_freq);
}
```

Appendix B. Java Classes for Wrapping the Uqailaut.jar process InukMorph.java

```
package inukmorph;

import java.io.BufferedReader;
import java.io.StringReader;
import java.io.FileReader;
import java.util.StringTokenizer;

/**
 * analyzes a whole file of Inuktitut words.  Uses an Analyzer object which creates
 * an MTModule object (inherits from MTModuleBasic) which wraps the morphological
 * analyzer process which is the main method running when invoking Uqailaut.jar.
 * This set up allows controlling for how much time to wait (default is set to 5
 * minutes) after which the process is killed and an "exceeded wait time
 * message is returned.  Also, it captures NullPointerExceptions and OutOfMemory
 * exceptionsthrown by the main process in Uqailaut.jar which result in killing
 * that process.
 *
 * @author jmicher
 */
public class InukMorph {

    private static BufferedReader br;

    /**
     * does the main processing
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        System.err.println("Processing " + args[0] + "...");
        Analyzer a = new Analyzer(args[1]);
        String line = "";
        StringTokenizer tok = null;
        int linecount = 1;
        BufferedReader linebreaker = null;
        try {
            br = new BufferedReader(new FileReader(args[0]));
            String currToken = "";
            while (br.ready()) {
                    line = br.readLine();
                    currToken = line;
                    String result = "";
                    System.out.print(currToken + "\t");
                    try {
                        result = a.analyze(currToken);
                        if (!result.equals("")) {
                            linebreaker       =       new       BufferedReader(new
StringReader(result));
                            String part = "";
                            String temp = "";
                            while (linebreaker.ready()) {
                                part = linebreaker.readLine();
                                if (part == null) {break;}
                                    temp += part + "|";
                                }
                                result = temp;
                            } else {
                                result = "NA";
                            }
                        } catch (Exception e) {
                            System.err.println(e.toString());
                        }
                        System.err.println(linecount++ + " " + currToken);
                        System.out.println(result);
            }
        } catch (Exception e) {
            e.printStackTrace(); }}}
```

Analyzer.java

```
package inukmorph;

import mtoutpost.*;
import java.io.*;

/**
 * This class wraps around the Uqailaut.jar process and will catch Null Pointers
 * and OutOfMemory exceptions that it may throw, while continuing to analyze words.
 * Provides a main method as well as the ability to create an object and use the
 * analyze method from it.
 *
 * @author jmicher
 */
public class Analyzer {

////////////////////////////////////////////////////////////////////////////////
//
// Fields

    private String codeCall;

////////////////////////////////////////////////////////////////////////////////
//
// Constructors

    /**
     * creates an Analzyer object, setting the underlying process to codeCall
     *
     * @param codeCall the command line string of the underlying process
     */
    public Analyzer(String codeCall) { this.codeCall = codeCall; }

////////////////////////////////////////////////////////////////////////////////
//
// Services

    /**
     * runs the Analyzer by calling the codeCall with its params
     * @param params the parameters of the underlying process
     * @return the output from the underlying process call
     */
    public String analyze(String params) {

        MTModule module = new MTModule("analyzer", "analyzes", codeCall, params);
        String out = new String();
        try {
            out = module.process("");
        } catch (Exception e) {
            out = e.toString();
        }
        if (out.startsWith("java.lang.NullPointerException")) {
            out = "NP";
        } else if (out.contains("OutOfMemoryError")) {
            out = "OM";
        }
        return out;
    }

    /**
     * main method, runs the Analyzer
     * @param args the commandline string of the underlying process
     */
    public static void main(String[] args) {

        Analyzer a = new Analyzer(args[0]);
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String input = "";
        String out = "";
        try {
            while((input=br.readLine())!=null) {
                System.err.println("analyzing " + input);
                System.out.println(a.analyze(input));
            }
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

MTModuleBasic.java

```
package mtoutpost;

import java.io.*;
import java.util.*;

/**
 * <CODE>class</CODE> MTModuleBasic encapulates the basic functionality of a
 * processing module.  This class is the superclass of {@link MTModule}
 * and {@link MTSubModule}, although currently, {@link MTSubModule}
 * is not being used.  MTModuleBasic provides the process()
 * method which is the main workhorse of a module.
 * @author Jeffrey C. Micher, Army Research Lab, Adelphi, MD
 */
public class MTModuleBasic {

////////////////////////////////////////////////////////////////////////////////
//
// Fields

    private String moduleName = null;
    private String description = null;
    private String codeCall = null;
    private String params = null;
    private boolean socketServer = false;
    private int port = -1;
    private boolean daemon = false;
    private int pause = -1;
    private ModuleSocketClient client = null;
    private ServerThread server = null;
    private Process p = null;
    private int waittime = 300000; // default 5 min

////////////////////////////////////////////////////////////////////////////////
//
// Constructors

    /**
     * no parameter constructor
     */
    public MTModuleBasic() {}

    /**
     * creates an MTModuleBasic object around a commandline
     * executable which does not need to load large data sets
     * @param moduleName the name of the module
     * @param description a description of the module
     * @param codeCall the commandline executable for this modules.  This needs
     *                 to be the whole commandline call, not just the location
     *                 of a file.  For example, "perl c:/mycode/fool.pl"
     * @param params parameters to the commandline executable
     */
    public MTModuleBasic(String moduleName, String description, String codeCall,
            String params) {

        initBasic(moduleName, description, codeCall, params);
    }

    /**
     * creates an MTModuleBasic object around a commandline
     * execuatable which needs to load a large data set and thus runs as a socket
     * server and waits in the background for connections
     * @param moduleName the name of this module
     * @param description a description for this module
     * @param codeCall the commandline executable for this modules.  This needs
     *                 to be the whole commandline call, not just the location
     *                 of a file.  For example, "perl c:/mycode/fool.pl"
     * @param params parameters for the commandline executable
     * @param port port that this executable runs on
     * @param isDaemon true if this module is to run as a daemon, false otherwise
     * @param pause amount of time in milliseconds that this executable needs to
     *              load its data
     */
    public MTModuleBasic(String moduleName, String description, String codeCall,
            String params,
            int port, boolean isDaemon, int pause) {

        initBasic(moduleName, description, codeCall, params);
        initSocket(port, isDaemon, pause);
    }

    /* initializes a module that is not a socket server */
    private void initBasic(String moduleName, String description, String codeCall,
            String params) {
        setModuleName(moduleName);
        setDescription(description);
        setCodeCall(codeCall);
        setParams(params);
    }

    /* initializes a module that is a socket server */
    private void initSocket(int port, boolean isDaemon, int pause) {

        socketServer = true;
        setPort(port);
        setClient(new ModuleSocketClient("localhost", port));
        setDaemon(isDaemon);
        setPause(pause);
        startServer(isDaemon, pause);
    }

////////////////////////////////////////////////////////////////////////////////
//
// Getters and Setters

    /**
     * sets the commandline executable for this MTModuleBasic
     * @param codeCall the commandline executable for this module to set
     */
    public void setCodeCall(String codeCall) { this.codeCall = codeCall; }

    /**
     * gets the commandline string of this module
     * @return the commandline string of this module
     */
    public String getCodeCall() { return codeCall; }

    /**
     * sets the name of this MTModuleBasic
     * @param moduleName the name to set for this module
     */
    public void setModuleName(String moduleName) { this.moduleName = moduleName; }

    /**
     * gets the name of this MTModuleBasic
     * @return the name of this <CODE>MTModuleBasic</CODE>
     */
    public String getModuleName() { return moduleName; }

    /**
     * sets the description of this MTModuleBasic
     * @param description the description of this module to set
     */
    public void setDescription(String description) { this.description = description;
}

    /**
     * gets the description of this MTModuleBasic
     * @return the description of this module
     */
    public String getDescription() { return description; }

    /**
     * sets the parameters for this MTModuleBasic
     * @param params the parameters to set for this module
     */
    public void setParams(String params) { this.params = params; }

    /**
     * gets the parameters of this MTModuleBasic
     * @return the parameters of this module
     */
    public String getParams() { return params; }

    /**
     * gets the socket server status of this MTModuleBasic
     * @return true if this module is a socket server, false if otherwise
     */
    public boolean isSocketServer() { return socketServer; }

    /**
     * sets the socket server status for this MTModuleBasic
     * @param socketServer the socket server status to set for this module
     */
    public  void  setSocketServer(boolean  socketServer)  {  this.socketServer  =
socketServer; }

    /**
     * sets the port for this MTModuleBasicto run its socket server
     * on
     * @param port the port to set for this module
     */
    public void setPort(int port) { this.port = port; }

    /**
     * gets the port that the socket server of this MTModuleBasic
     * runs on
     * @return the port that this module is set to run on
     */
    public int getPort() { return port; }

    /**
     * sets the client that this MTModuleBasic uses to test whether
     * the underlying socket server is running or not
     * @param client the client that this module uses
     */
    public void setClient(ModuleSocketClient client) { this.client = client; }

    /**
     * gets the client that is used to test whether the socket server of this
     * MTModuleBasic is running
     * @return the client
     */
    public ModuleSocketClient getClient() { return client; }

    /**
     * gets the Process object that is created when this MTModuleBasic wraps
     * a commandline executable
     * @return the Process that this MTModuleBasic creates
     */
    public Process getProcess() { return p; }

    /**
     * gets the underlying socket server object of this MTModuleBasic
     * @return the server of this module
     */
    public ServerThread getServer() { return server; }

    /**
     * sets the pause of this MTModuleBasic, which is the amount of time that
     * the module should pause to allow loading of data
     * @param pause the interval of time in milliseconds to set
     */
    public void setPause(int pause) { this.pause = pause; }

    /**
     * gets the pause of this MTModuleBasic
     * @return the pause of this module
     */
    public int getPause() { return pause; }

    /**
     * sets the daemon status of this MTModuleBasic
     * @param daemon the daemon status to set
     */
    public void setDaemon(boolean daemon) { this.daemon = daemon; }

    /**
     * returns the daemon status of this MTModuleBasic
     * @return the daemon status
     */
    public boolean isDaemon() { return daemon; }

////////////////////////////////////////////////////////////////////////////////
//
//  Utilities

    /* starts the socket server */
    private void startServer(boolean isDaemon, int pause) {
        //try to connect, if no connection then start new server
        try {
            System.err.println("testing connection");
            getClient().process("test");
        } catch (Exception ioe) {
            System.err.println("no connection, so starting server");
            // no connection, so start server
            server = new ServerThread(getModuleName() + "-server",
                getCodeCall() + " " + getPort() + " " +  getParams(), isDaemon);
            server.start();
            try {
                //pause to allow loading data for some modules
                Thread.sleep(pause);
            } catch (Exception e) {
                System.err.println(e);
            }
            // keep a reference to the process provided through the thread to be
            // able to destroy it when the module gets destroyed.  Otherwise,
            // the process continues to run in the background even when the
            // module has been destroyed.
            p = server.getProcess();
            return;
        }
    }

    /**
     * creates an xml String representation of this MTModuleBasic
     * @return the xml String
     */
    public String toXMLString() {
        StringBuffer temp = new StringBuffer();
        temp.append("    <name>" + getModuleName() + "</name>\n");
        temp.append("    <description>" + getDescription() + "</description>\n");
        temp.append("    <code_call>" + getCodeCall() + "</code_call>\n");
        temp.append("    <parameters>" + getParams() + "</parameters>\n");
        if (isSocketServer()) {
            temp.append("    <socket>\n");
            temp.append("      <port>" + getPort() + "</port>\n");
            temp.append("      <daemon>" + isDaemon() + "</daemon>\n");
            temp.append("      <pause>" + getPause() + "</pause>\n");
            temp.append("    </socket>\n");
        }
        return temp.toString();
    }

////////////////////////////////////////////////////////////////////////////////
//
// Services


    public String process(String input, int waittime) throws Exception {
        this.waittime = waittime;
        return process(input);
    }

    /**
     * processes the input String and produces an output String.
     * @param input the String to input to the underlying process
     * @throws java.lang.Exception exceptions from Runtime and Process that this
     * class uses
     * @return the output from the underlying process
     */
    public String process(String input) throws Exception {
        String returnValue = new String();
        StringBuffer buffer = new StringBuffer();
        if (socketServer == true) {
            returnValue = client.process(input);
            ServerThread current = getServer();
            if (current != null) {
                if (!current.isDaemon()) {
                    current.getProcess().destroy();
                }
                current.interrupt();
            }
        } else {// not a socket server
            Runtime rt = Runtime.getRuntime();
            String output = "";
            String errors = "";
            try {
                p = rt.exec(getCodeCall() + " " + getParams());
                PrintStream processInputStream = new PrintStream(
                        p.getOutputStream(), true, "utf-8");
                processInputStream.println(input + "\n\u0003");//put in garbage to
kill Bikel's loop
                StreamConsumer errorConsumer =
                        new StreamConsumer(p.getErrorStream(), "error", 1);
                StreamConsumer outputConsumer =
                        new StreamConsumer(p.getInputStream(), "output", 10);
                outputConsumer.start();
                errorConsumer.start();
                int countloops = 0;
                int time = 0;
                String message = "";
                while (!outputConsumer.isComplete()) {
                    //wait
                    Thread.sleep(100);
                    countloops++;
                    time += 100;//one tenth of a second
                    if (time > waittime) {// just wait 5 minutes
                        message  =  "exceeded  waittime  of  "  +  waittime  +  "
milliseconds";
                        break;
                    }
                }
                errors = errorConsumer.getOutput();
                output = outputConsumer.getOutput();
                if (!message.equals("")) {
                    errors = message;
                }
            } catch (IOException ioe) {
                System.err.println("Module error: " + getModuleName());
                ioe.printStackTrace();
                System.exit(0);
            }
            p.destroy();
            if (errors.equals("")) {
                returnValue = output;
            } else {
                returnValue = errors;
            }
        }
        return returnValue;
    }
}
```

MTModule.java

```
package mtoutpost;

import java.io.*;

/**
 * <CODE>class</CODE> MTModule encapulates a processing module.  It
 * is a subclass of {@link MTModuleBasic}.
 * @author Jeffrey C. Micher, Army Research Lab, Adelphi, MD
 */
public class MTModule extends MTModuleBasic {

////////////////////////////////////////////////////////////////////////////////
//
// Fields

    private MTSubModule preprocess = null;
    private MTSubModule postprocess = null;

////////////////////////////////////////////////////////////////////////////////
//
// Constructors

    // MAKE SURE THE codeCall IS ACTUALLY AN INVOCATION OF A PROCESS, NOT JUST THE
    // PATH TO THE CODE EG:  "perl c:\script.pl" not "c:\script"
    /**
     * no parameter, default constructor
     */
    public MTModule() { super(); }

    /**
     * creates an <CODE>MTModuleBasic</CODE> object around a commandline
     * executable which does not need to load large data sets
     * @param moduleName the name of this module
     * @param description a description of this module
     * @param codeCall the commandline executable for this module.  This needs
     * to be the whole commandline call, not just the location
     * of a file.  For example, "perl c:/mycode/fool.pl"
     * @param params the params for the underlying commandline code call of this
module
     */
    public MTModule(String moduleName, String description, String codeCall, String
params) {
        super(moduleName, description, codeCall, params);
    }

    /**
     * creates an <CODE>MTModuleBasic</CODE> object around a commandline
     * execuatable which needs to load a large data set and thus runs as a socket
     * server and waits in the background for connections
     * @param moduleName the name of this module
     * @param description a description of this module
     * @param codeCall the commandline executable for this module.  This needs
     * to be the whole commandline call, not just the location
     * of a file.  For example, "perl c:/mycode/fool.pl"
     * @param params the parameters for the underlying commandline executable
     * @param port the port that the server of this module runs on
     * @param isDaemon true if this module is a daemon, false otherwise
     * @param pause the time, in milliseconds, that this module
     * needs to pause to load data
     */
    public MTModule(String moduleName, String description, String codeCall, String
params,
        int port, boolean isDaemon, int pause) {
        super(moduleName, description, codeCall, params, port, isDaemon, pause);
    }

    /**
     * creates an MTModule that has pre and post processing modules
     * @param moduleName the name of this module
     * @param description a description of this module
     * @param codeCall the commandline executable for this module.  This needs
     * to be the whole commandline call, not just the location
     * of a file.  For example, "perl c:/mycode/fool.pl"
     * @param params the parameters for the commandline excutable of this module
     * @param preprocess the preprocessing submodule of this module
     * @param postprocess the postprocesing submodule of this module
     */
    public MTModule(String moduleName, String description, String codeCall, String
params,
        MTSubModule preprocess, MTSubModule postprocess) {
        super(moduleName, description, codeCall, params);
        initSubProcesses(preprocess, postprocess);
    }

    /**
     * creates an <CODE>MTModuleBasic</CODE> object around a commandline
     * execuatable which needs to load a large data set and thus runs as a socket
     * server and waits in the background for connections.  Also
     * this module has pre and post processing submodules
     * @param moduleName the name of this module
     * @param description the description of this module
     * @param codeCall the commandline executable for this module.  This needs
     * to be the whole commandline call, not just the location
     * of a file.  For example, "perl c:/mycode/fool.pl"
     * @param params the parameters for the commandline excutable of this module
     * @param port the port the the server of this module runs on
     * @param isDaemon the daemon status of this module
     * @param pause the time, in milliseconds, that this module
     * pauses to load data
     * @param preprocess the preprocessing submodule of the module
     * @param postprocess the postprocessing submodule of this module
     */
    public MTModule(String moduleName, String description, String codeCall, String
params,
        int port, boolean isDaemon, int pause,
        MTSubModule preprocess, MTSubModule postprocess) {
        super(moduleName, description, codeCall, params, port, isDaemon, pause);
        initSubProcesses(preprocess, postprocess);
    }

    /* initializes the subprocesses */
    private void initSubProcesses(MTSubModule preprocess, MTSubModule postprocess)
{
        setPreprocess(preprocess);
        setPostprocess(postprocess);
    }

////////////////////////////////////////////////////////////////////////////////
//
// Getters and Setters

    /**
     * sets the preprocessing submodule of this MTModule
     * @param preprocess the preprocessing submodule to set
     */
    public  void  setPreprocess(MTSubModule  preprocess)  {  this.preprocess  =
preprocess; }

    /**
     * gets the preprocessing submodule of this MTModule
     * @return the preprocessing submodule of this MTModule
     */
    public MTSubModule getPreprocess() { return preprocess; }

    /**
     * sets the postprocessing submodule of this MTModule
     * @param postprocess the postprocessing submodule to set
     */
    public  void  setPostprocess(MTSubModule  postprocess)  {  this.postprocess  =
postprocess; }

    /**
     * gets the postprocessing submodule of this MTModule
     * @return the postprocessing submodule of this MTModule
     */
    public MTSubModule getPostprocess() { return postprocess; }

////////////////////////////////////////////////////////////////////////////////
//
//  Utilities

    /**
     * creates an XML string representation of this
     * <CODE>MTModule</CODE>
     * @return the xml string representation of this
     * <CODE>MTModule</CODE>
     */
    public String toXMLString() {
        StringBuffer temp = new StringBuffer();
        temp.append("  <module>\n");
        temp.append(super.toXMLString());
        if (getPreprocess() != null) {
            temp.append("    <pre>\n");
            temp.append(getPreprocess().toXMLString());
            temp.append("    </pre>\n");
        }
        if (getPostprocess() != null) {
            temp.append("    <post>\n");
            temp.append(getPostprocess().toXMLString());
            temp.append("    </post>\n");
        }
        temp.append("  </module>\n");
        return temp.toString();
    }

////////////////////////////////////////////////////////////////////////////////
//
// Services

    /**
     * processes the input string to the underlying process
     * of this <CODE>MTModule</CODE>.  Overrides process()
     * in {@link MTModuleBasic}.
     * @param input the input String to the underlying
     * process
     * @throws java.lang.Exception exceptions from Runtime and Process
     * @return the output of the underlying process
     */
    public String process(String input) throws Exception {
        String returnValue = input;
        if (getPreprocess() != null) {
            returnValue = getPreprocess().process(returnValue);
        }
        returnValue = super.process(returnValue);
        if (getPostprocess() != null) {
            returnValue = getPostprocess().process(returnValue);
        }
        return returnValue;
    }
}
```

StreamConsumer.java

```
package mtoutpost;

import java.io.*;

/**
 * <CODE>class</CODE> StreamConsumer consumes a stream in a separate Thread
 * @author Jeffrey C. Micher, Army Research Lab, Adelphi, MD
 */
public class StreamConsumer {

////////////////////////////////////////////////////////////////////////////////
//
//  Private Inner Classes

    private class Worker extends Thread {

        /////// Fields
        InputStream is;
        String type;

        /////// Constructor
        Worker(InputStream is, String type, int priority) {
            this.is = is;
            this.type = type;
            setPriority(priority);
        }

        /////// Services
        public void run() {
            BufferedReader br;
            try {
                br = new BufferedReader(new InputStreamReader(is, "utf-8"));
                String line = null;
                line = br.readLine();
                output = "";
                while(line != null) {
                    output += line + "\n";
                    line = br.readLine();
                }
                complete = true;
            } catch (Exception e) {
                output = e.toString();
                complete = true;
            }
        }
    }

////////////////////////////////////////////////////////////////////////////////
//
// Fields

    private String output;
    private Worker worker;
    private String type;
    private boolean complete;

////////////////////////////////////////////////////////////////////////////////
//
// Constructors

    /**
     * creates a StreamConsumer with the input stream that this consumer consumes
     * @param is the input stream that this consumer consumes
     */
    public StreamConsumer(InputStream is, String type, int priority) {
        output = new String("output holder");
        worker = new Worker(is, type, priority);
        this.type = type;
        complete = false;
    }

////////////////////////////////////////////////////////////////////////////////
//
// Getters and Setters

    /**
     * gets the output that is read by this StreamConsumer
     * @return the output that this StreamConsumer has read
     */
    public String getOutput() { return output; }

    /**
     * gets the type of this StreamConsumer, either "output" or "error"
     * @return the type of this StreamConsumer
     */
    public String getType() { return type; }

////////////////////////////////////////////////////////////////////////////////
//
// Services

    /**
     * starts the thread
     */
    public void start() { worker.start(); }

////////////////////////////////////////////////////////////////////////////////
//
//  Utilities

    /**
     * checks whether this StreamConsumer is complete or not
     * @return true if this StreamConsumer is complete, false otherwise
     */
    public boolean isComplete() { return complete; }
}
```

Appendix C. get_subsets.pl

```
#!/usr/bin/perl

#################################################################################
#
# Name: get_subsets.pl
# Date Created: before 23 Jan 2013
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
################################################################################
#
# Description: This script will take a list of numbers which indicate the start
# and end line numbers for deriving a subset of a corpus.  It will then create
# all of the subset files based on the line numbers.
#
# Specify two files on the command line: 1) line-number-file 2) file to divide up
#
# The line number file lists eachs sequence of lines, separated by a tab.  For
# example:
#
# 1<tab>10
# 11<tab>20
# 21<tab>30
#
# will create 3 files labelled <inputfile>.01, <inputfile>.02, <inputfile>.03
#
################################################################################
#
# Change Log:
#
# 15 March 2016: cleaned and updated, added header, ready to check in.
#
################################################################################

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

my $usage = "Usage: $0 <line-number-file> <corpus>\n";

@ARGV == 2 || die "$usage";
$corpusfile = $ARGV[1];
open (IN, $ARGV[0]) || die "Could not open $ARGV[0]\n";
open (IN2, $corpusfile) || die "Could not open $corpusfile\n";

while (<IN>) {
    chop;
    ($start,$end) = split(/\t/,$_);
    push(@ends, $end);
}
$previousend = 0;

while (@ends) {
    $end = shift(@ends);
    $filecount++;
    $newfilename = $corpusfile.".";
    if ($filecount < 10) { $newfilename .= "0".$filecount; } else { $newfilename .=
$filecount; }
    open(OUT, ">$newfilename");
    for ($i = $previousend; $i < $end; $i++) {
       $line = <IN2>;
       print OUT "$line";
    }
    close(OUT);
    $previousend = $end;
}
```

Appendix D. qsubrun

```
#!/bin/bash

source ~/.bashrc

CWD=`pwd`;
FNAME="";
ME=`whoami`;

# write to specified file name
if [ $# -eq 2 ]
    then
    FNAME=`mktemp /home/jmicher/tmp/$2.XXX`
    TMPNAME=$2.$$f
else
    FNAME=`mktemp /home/jmicher/tmp/pbs.XXX`
    TMPNAME=$$f
fi

echo "#!/bin/bash" > $FNAME;
echo "#PBS -l nodes=1" >> $FNAME;
echo "#PBS -q long" >> $FNAME;
echo "#PBS -l walltime=3:00:00" >> $FNAME;
echo "#PBS -l mem=4GB" >> $FNAME;
echo "#PBS -M jeffrey.c.micher.civ@mail.mil" >> $FNAME;
echo "#PBS -m bae" >> $FNAME;
echo "#PBS -e /home/jmicher/tmp/$TMPNAME.ER" >> $FNAME;
echo "#PBS -o /home/jmicher/tmp/$TMPNAME.OU" >> $FNAME;

# do use -k only in urgent cases with small error output, otherwise the disk quota
is easily exceeded!
#echo "#PBS -k e" >> $FNAME;

echo "source ~/.bashrc" >> $FNAME;
echo -e "cd $CWD" >> $FNAME;
echo -e "$1" >> $FNAME;

# put it all together
JOB=`qsub -V $FNAME`;

# run it
echo $JOB
```

Appendix E. run_morph.pl

```
#!/usr/bin/perl

#################################################################################
#
# Name: run_morph.pl
# Date Created: ~October 2013
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
################################################################################
#
# Description: This script is used to batch process the Inuktitut morphological
# analyzer, sending off batches in individual jobs to a computing cluster.  Makes
# use of qsubrun and InukMorph.jar and the analyzer, Uqailaut.jar.
#
# Specify lower and upper file extension numbers for the batch.  For example,
# specifiying 01 20 on the command line will fire off a process for the inuk.01
# file through inuk.20. The filename base where the word types are located is
# currently hardcoded into the script.
#
################################################################################
#
# Change Log:
#
# 16 March 2016: added header, cleaned up.
#
################################################################################

($lower, $upper) = @ARGV;

for ($i = $lower; $i < $upper+1; $i++) {

    system("qsubrun             \"/home/jmicher/bin/java/bin/java             -jar
/home/jmicher/myscripts/inuktitut_scripts/InukMorph.jar                inuk.".$i."
\'/home/jmicher/bin/java/bin/java   -jar   /home/jmicher/inuktitut/Uqailaut.jar\'\"
inuk.$i");
}
```

Appendix F. count_analyses.pl

```
#!/usr/bin/perl

##################################################################################
#
# Name: count_analyses.pl
# Date Created: ~October 2013
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
#################################################################################
#
# Description: This script is used to tally up the output from running the
# Inuktitut morphological analyzer, Uqailaut.jar over all of the 417K Inuktitut
# types.  It will count how many words were analyzed, as well as those types which
# produced an error message.  It outputs the types with errors, sorted by error
# type.  Input is all of the inuk.*.out files that were produced during the
# morphological analysis stage.
#
#################################################################################
#
# Change Log:
#
# 21 June 2016: added options to print the types having the various errors
# 22 March 2016: recreated to disallow repeats (to allow updates with new analyses)
# 15 March 2016: Added header.  Cleaned.
#
#################################################################################

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

### Specify options

use Getopt::Long;
my $n_opt = '';
my $p_opt = '';
my $m_opt = '';
my $M_opt = '';
my $x_opt = '';
my $o_opt = '';

# options will be: to print out the types according to the various categories of
errors
# -n: no analysis
# -p: null pointer
# -m: Out of memory
# -M: OM out of memory
# -x: exceeded time limit
# -o: output holder not retrieved

GetOptions ('n' => \$n_opt, 'p' => \$p_opt, 'm' => \$m_opt, 'M' => \$M_opt, 'x' =>
\$x_opt, 'o' => \$o_opt);

my $database = {};

while (<STDIN>) {
    chop;
    # check if already counted
    $_ =~ /^([^\t]+)\t+([^\t]+)$/;
    $type = $1;
    $analysis = $2;

    if (exists($database->{$type})) {
        # is there a new analysis?
        if ($analysis ne $database->{$type}) {
            $database->{$type} = $analysis; # will update pre-existing analysis with
new analysis
        }
    } else {
       # new type
       $database->{$type} = $analysis;
    }
}

foreach $k (sort (keys (%$database))) {
    my $anal = $database->{$k};
    if ($anal eq "NA") {
       push(@na, $k);
       $na_count++;
    } elsif (substr($anal,0,2) eq "NP") {
       push(@np, $k);
       $np_count++;
    } elsif (substr($anal,0,2) eq "OM") {
       push(@om, $k);
       $om_count++;
    } elsif ($anal =~ /^exceed/) {
       push(@ex, $k);
       $ex_count++;
    } elsif ($anal =~ /^output/) {
       push(@out, $k);
       $out_count++;
    } elsif ($anal =~ /^Out/) {
       push (@Out, $k);
       $Out_count++;
    } elsif ($anal eq "") {
       $blank++;
    } elsif ($anal =~ /^\{/) {
       push(@na, $k);
       $an_count++;
    } else {
       print STDERR "$k:$anal not accounted for\n";
    }
    $total_count++;
}
print                                                                        STDERR
"analyzed\t$an_count\nNA\t$na_count\nNP\t$np_count\nOM\t$om_count\nEX\t$ex_count\n
";
print STDERR "output\t$out_count\nOut\t$Out_count\nBlankline\t$blank\n";
print STDERR "total\t$total_count\n";

############
## print output according to options selected

if ($n_opt == 1) {
    print "\nNo analysis\n";
    foreach $w (@na) {
       print "$w\n";
    }
}

if ($p_opt == 1) {
    print "\nNull Pointer\n";
    foreach $w (@np) {
       print "$w\n";
    }
}

if ($m_opt == 1) {
    print "Out of Memory\n";
    foreach $w (@Out) {
       print "$w\n";
    }
}

if ($M_opt == 1) {
    print "\nOM (Out of Memory)\n";
    foreach $w (@om) {
       print "$w\n";
    }
```

```
}

if ($x_opt == 1) {
    print "\nExceeded wait time (5 min)\n";
    foreach $w (@ex) {
        print "$w\n";
    }
}

if ($o_opt == 1) {
    print "\nCouldn't retrieve output\n";
    foreach $w (@out) {
        print "$w\n";
    }
}
```

Appendix G. update_inuk.data.pl

```
#!/usr/bin/perl

##################################################################################
#
# Name: update_inuk_data.pl
# Date Created: June 20, 2016
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
#################################################################################
#
# Description: This script will update the inuk.*.out files with the new analyses
# from reanalyzed_unretrievable_output.out and exceeded_analyzed.out.  The script
# is hard-coded to work from inside the directory containing the .out files.  The
# reanalyzed_unretrievable_output.out and exceeded_analyzed.out are fed into the
# script on STDIN.
#
#################################################################################
#
# Change Log:
#
# 20 June 2016: created
#
#################################################################################

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

my $data = {};

# load new analyses
while (<STDIN>) {
    $count++;
    chop;
    ($type, $anal) = split(/\t/, $_);
    if (exists($data->{$type})) {
        print STDERR "skipping $type, type already accounted for\n";
    } else {
        $data->{$type} = $anal;
    }
}
print STDERR "$count items loaded\n";

# open old files, rewrite and replace them
for ($i = 1; $i <= 414; $i++) {
    if ($i < 10) {
        $ext = "0".$i;
    } else {
        $ext = $i;
    }

    $filename = "inuk.$ext.out";

    # saving old files, just in case
    system("cp $filename $filename.bak");

    print STDERR "Processing $filename\n";
    open(IN, "<:encoding(utf8)", $filename) || die "Could not open $filename\n";
    $lines = [];
    while(<IN>) {
        chop;
        $line = $_;
        ($type,$anal) = split(/\t/, $_);
        if (exists($data->{$type})) {
            print STDERR "replacing $_\n";
            $replaced_count++;
            $line = "$type\t$data->{$type}";
        }
        push(@$lines, $line);
    }
    close(IN);
    open(OUT, ">:encoding(utf8)", $filename) || die "Could not open $filename for
writing\n";
    foreach $line (@$lines) {
        print OUT "$line\n";
    }
    close(OUT);
}
print STDERR "$replaced_count items replaced\n";
```

Appendix H. POS.txt

Inuktitut morpheme labels derived from the grammatical information provided by the analyzer. This list contains the labels, a description of the category, and the regular expressions used to convert the information to the label.

label	description	regular expression
ROOT	root (noun or verb)	^\d[nv]$
LEX	lexical postbase	^\d[nv][nv]$
GRAM	grammatical suffix	^t[nv].+
CL	clitic	^\d?q$
ADV	adverb	^\d?a$
CONJ	conjunction	^\d?c$
EXCLAM	exclamation	^\d?e$
PR	pronoun	^\d?pr$ ^\d?rp$ ^\d?rpr$
AD	demonstrative adverb	^ad-(..)$
PD	demonstrative pronoun	^pd-(.+)$
RAD	demonstrative adverb root	^rad-(..)$
RPD	demonstrative pronoun root	^rpd-(.+)$
TAD	demonstrative adverb suffix	^tad
TPD	demonstrative pronoun suffix	^tpd

Notes:

1) The grammatical code for pronouns is given inconsistently by the morphological analyzer, hence the 3 regular expressions needed to convert these to a single, unified label.
2) When demonstrative adverbs and pronouns have suffixes, they are analyzed as having a root and a suffix. Otherwise, they are simply analyzed as a single unit (with no indication of root).

Appendix I. tok_punc.pl

```
#!/usr/bin/perl

#################################################################################
#
# Name: tok_punc.pl
# Date Created: before 01 Jan 2014
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
################################################################################
#
# Description: This script will tokenize the punctuation in the Inuktitut half
# of the parallel Inuktitut-English corpus.  Specifically, it adds whitespace
# around the following punctuation characters:
#
# \x{21}  !
# \x{22}  "
# \x{23}  #
# \x{24}  $
# \x{25}  %
# \x{26}  &
# \x{27}  '
# \x{28}  (
# \x{29}  )
# \x{2a}  *
# \x{2b}  +
# \x{2c}  ,
# \x{2d}  -
# \x{2e}  .
# \x{2f}  /
# \x{3a}  :
# \x{3b}  ;
# \x{3c}  <
# \x{3d}  =
# \x{3e}  >
# \x{3f}  ?
# \x{40}  @
# \x{5b}  [
# \x{5c}  \
# \x{5d}  ]
# \x{5e}  ^
# \x{5f}  _
# \x{60}  `
# \x{7b}  {
# \x{7e}  ~
#
# and then fixes the ellipses (three dots) and double right-angle brackets,
# which get separated).
#
################################################################################
#
# Change Log:
#
# 08 August 2016: added header info, fixed broken ellipses
#
################################################################################

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

while (<STDIN>) {
    $line = $_;
    $line =~ s/\.\.\.\.+/\.\.\./g;
    while         ($line         =~          s/([\x{21}-\x{2f}\x{3a}-\x{40}\x{5b}-
\x60}\x{7b}\x{7e}])([\x{21}-\x{2f}\x{3a}-\x{40}\x{5b}-\x60}\x{7b}\x{7e}])/$1 $2/g)
{
       $c++;
```

Approved for public release; distribution is unlimited.

```
    }
    # now fix the ellipses
    $line =~ s/\. \. \./\.\.\./g;
    # double right-angle brackets
    $line =~ s/> >/>>/g;
    # double left-angle brackets
    $line =~ s/< </<</g;
    print "$line";
}
```

Appendix J. preprocess_inuk.pl

```
#!/usr/bin/perl

################################################################################
#
# Name: preprocess_inuk.pl
# Date Created: Fall 2013
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
###############################################################################
#
# Description:
#
# This script will preprocess a train, test, or dev file of Inuktitut by
# using the pre-determined morphological analyses.  Initially, it will
# use the first analysis, if there are any.  Later it may be expanded
# to use all analyses.
#
# The underlying morph will be used as the token.
# When there is no analysis for a word, choose a backoff strategy
#
# Backoff strategies can be:
#  1) Just use the unanalyzed token
#  2) Use the head, if it can be identified, else use unanalyzed token
#  3) Use the head, the middle and the grammatical ending, if they can
#     be determined, else use unanalyzed token
#  4) Use an unsupervised analysis, try pymorphy
#
# As of Feb 2016, only the first backoff strategy has been implemented
#
# The script uses the Inuktitut morphological analyses files, found in:
# type_subsets/out/.  Pass all of these files in as arguments on the command
# line:
# ~: preprocess_inuk.pl type_subsets/out/*.out < input-file > output-file
#
# Input format: Inuktitut text file, single space between words
#
# Example: pivigaarjugumavunga ilitarijumallugit innait inuusuttuillu
# tavvaniiqatigijattinnik ullumiujuq .
#
# Output format: *****UPDATED SEPT 13,2016**********Inuktitut words,
# separated into morphemes, with 6 factors for each morpheme. 1) true
# surface morpheme (what comes from the imput) 2) surface morpheme (what
# the analyzer returns as the surface form), 3) deep morpheme,
# 4) morphological analysis (what the analyzer outputs for that morpheme),
# 5) morpheme category (see POS.txt), 6) morpheme sub-category (see
# README for explanation of subcategories)
#
#         Example:         pivi|pivi|pivik|1v|ROOT|v         ga|ga|gaq|2vv|LEX|vv
arju|arju|arjuk|2vv|LEX|vv   guma|guma|juma|1vv|LEX|vv   vunga|vunga|vunga|tv-dec-
1s|GRAM|vt                 ilita|ilita|ilitaq|1v|ROOT|v        ri|ri|gi|4vv|LEX|vv
juma|juma|juma|1vv|LEX|vv        llugit|llugit|lugit|tv-part-1s-3p-prespas|GRAM|vt
inna|inna|ingnaq|1n|ROOT|n   it|it|it|tn-nom-p|GRAM|nt     inuu|inuu|inuu|1v|ROOT|v
sut|sut|suq|1vv|LEX|vv        tu|tu|juq|1vn|LEX|vn        il|il|it|tn-nom-p|GRAM|nt
lu|lu|lu|1q|CL|q        tavv|tavv|tagv|rad-sc|RAD|sc    ani|ani|ani|tad-loc|TAD|loc
i|i|it|1nv|LEX|nv          qati|qati|qati|1vn|LEX|vn           gi|gi|gi|1nv|LEX|nv
jat|jat|jaq|2vv|LEX|vv    ti|ti|ji|1vn|LEX|vn    nnik|nnik|nnik|tn-acc-s-1s|GRAM|nt
ullumi|ullumi|ullumi|1a|ADV|a       u|u|u|1nv|LEX|nv       juq|juq|juq|1vn|LEX|vn
.|.|period|PUNC|PUNC|PUNC
#
# Updated output format: 6 factors, first factor is now the "true" surface form
# which is not necessarily returned by the analyzer.
#
# Note that the output preserves the word boundaries, indicated with a double
# space character.
#
```

```
# THIS IS AN OLD NOTE, TO BE ADDRESSED AT SOME POINT?
# -> POS for punctuation needs to be fixed
#
################################################################################
#
# Change Log:
#
# 05 Apr 2017: added ability to specify as first arg the number of analyses to
# output.  Default is 1 if first argument does not consist of all digits.
#
# 05 Apr 2017: Moved all Inuktitut processing to a perl module, InukProcessing.pm
# to allow using the functions by other scripts.
#
# 11 Feb 2017: Updated to collapse RP, PR and RPR which are all pronouns
#
# 14 Sept 2016: Updated to remove final space which was messing up downstream
# processes.
#
# 13 Sept 2016: Updated to output 6 factors.  The first factor will be the "true"
# surface form, which is not always provided by the analyzer (which happens to
# normalize some of the variation, but not all.
#
# 17 Feb 2016: Updated to label demonstrative adverb and pronoun endings
# as TAD and TPD, to mirror the labels for the roots (RAD, RPD) and to
# distinguish these from standard GRAMmatical endings (on nouns and verbs)
#
# 10 Feb 2016: Renamed the factors so they would make more sense
# surface|deep|analysis|morph-cat|morph-subcat
#
# 03 Feb 2014: Now it creates a factored output with
# surface|lemma|morph-cat|morph-subcat|lex-cat
#
################################################################################

use lib "/home/jmicher/myscripts/inuktitut_scripts";
use InukProcessing;
use strict;

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

my $analyses = {};
my $data_count = 0;
my $words_per_line = 0;
my $words_analyzed = 0;
my $linecount = 0;
my $analyses_per_line = 0;
my $blanklines = 0;
my $num_analyses = 1;

if ($ARGV[0] =~ /^-?\d+$/) {
    $num_analyses = shift(@ARGV);
}

$analyses = InukProcessing::load_analyses_data(@ARGV);

while (<STDIN>) { # process each line
    chop;
    if ($_ ne "") {
    $linecount++;
    $words_analyzed = 0;
    my $result = "";
    my $line = $_;
    $line =~ s/\s+/ /g;
    $line =~ s/^\s//;
    $line =~ s/\s$//;
    my @words = split(/\s/,$line);
    $words_per_line = @words;
    for (my $w = 0; $w < @words; $w++) { # process each word
        $result .= InukProcessing::analyze($words[$w],$analyses,$num_analyses);
        if ($w < @words - 1) {
            $result .= "  "; # add double space at word boundary
        }
    }
    $analyses_per_line += $words_analyzed/$words_per_line;
    print "$result\n";
    } else {
        $blanklines++;
        print "\n";
    }
}
```

Appendix K. InukProcessing.pm

```
#!/usr/bin/perl

##################################################################################
#
# Name: InukProcessing.pm
# Date Created: 03 April 2017
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
################################################################################
#
# Description: This is a perl module providing access to useful processing of
# Inuktitut morphological analyses.  When using, provide the following in the
# script:
#
# use InukProcessing;
# use strict;
#
# to use the functions without namespace qualification as in:
# InukProcessing::analyze($w,$a,2);
#
# make sure to write this:
# use InukProcessing qw(analyze);
#
################################################################################
#
# Change Log:
#
################################################################################

package InukProcessing;

use strict;
use Exporter;
use vars qw($VERSION @ISA @EXPORT @EXPORT_OK); # pacifies strict
$VERSION     = 1.00; # set a version number
@ISA         = qw(Exporter); # make Exporter part of this module using the @ISA
@EXPORT       = (); # a list of functions that we export by default. Generally the
less you export by default using @EXPORT the better.
@EXPORT_OK       =  qw(load_analyses_data  reconstruct_surface  get_morpheme_parts
get_morpheme_strings analyze); # list of functions exported by asking


######## analyze(word, data_hash, num_analyses)
#
# takes a single word and produces up to however many analyses specified
# default is all analyses
# input: single word, ref to data hash, optional num
# output:
sub analyze {
    my $temp_analyses = {};
    my $num_analyses = -1;
    my ($word, $data_ref, $n) = @_;
    my $result = "undef|undef|undef|undef|undef|undef"; # default to undef
    unless ($n eq "") {$num_analyses = $n;}
    if (exists($data_ref->{$word})) {
       $result = "";
       my $ref = $data_ref->{$word};
       my $top;
       if ($num_analyses == -1) { # output all analyses
           $top = @$ref;
       } else {#output up to $num_analyses analyses
           if ($num_analyses >= @$ref) { #truncate number to output
               $top = @$ref;
           } else {
               $top = $num_analyses;
```

```
            }
        }
        $result = "";
        for (my $i = 0; $i < $top; $i++) {
            $result .= morphs2factors($ref->[$i]);
            if ($i < $top-1) {
                $result .= "\n";
            }
        }
    } else { # exceptional cases, not in the analysis data
        # These can't be stored in the .out files since they are key characters
        # in the morphological analyzer output.
        $result = "";
        if ($word eq ":") { # colon
            $result = ":|:|colon|PUNC|PUNC|PUNC";
        } elsif ($word eq "/") { # forward slash
            $result = "/|/|forward_slash|PUNC|PUNC|PUNC";
        } elsif ($word =~ /^\d+$/) { # any number composed of only digits
            # maybe I should expand this to cover more number formats?
            $result = "$word|$word|$word|NUM|NUM|NUM";
        } else { # not analyzable
            $result = $word."|".$word."|".$word."|NA|NA|NA";# later maybe make guess
about Root type?
        }
    }
    return $result;
}

######## morphs2factors(morpheme_array)
#
# converts morpheme array to factored form
# input: morpheme array
# output: factored string
sub morphs2factors {
    my $morph_array = $_[0];
    my $ret_str = "";
    for (my $i = 0; $i < @$morph_array; $i++) {
        $ret_str .= parts2factors($morph_array->[$i]);
        if ($i < @$morph_array - 1) {
            $ret_str .= " ";
        }
    }
    return $ret_str;
}

######## parts2factors(morpheme_parts_array)
#
# converts morpheme parts to factored form
# input: morpheme parts array
# output: factored format
sub parts2factors {
    my ($parts_array) = @_;
    my $ret_str = "";
    for (my $i = 0; $i < @$parts_array; $i++) {
        $ret_str .= $parts_array->[$i];
        if ($i < @$parts_array -1) {
            $ret_str .= "|";
        }
    }
    return $ret_str;
}

####### load_analyses_data(array_of_file_names)
#
# input: array of filenames in which analysis data is found
# output: reference to hash of types with analysis data
sub load_analyses_data {

    my $data_count = 0;
    my $analyses_data = {};
    foreach my $arg (@_) { # this loads the whole set of type analyses
```

```
        open(IN, $arg) || die "Could not open $_[0]\n";
        print STDERR "LOADING: $arg \r";
        while(<IN>) {
            chop;
            (my $word, my $analyses) = split(/\t/, $_);
            if (exists($analyses_data->{$word})) {
                print STDERR "skipping duplicate: $word\n";
            } elsif ($analyses =~ /^\{/) { ## skip NA
###### 03 April 2017
#### storing all analyses data, not just the first
                my  $analyses_string  =  substr($analyses,0,length($analyses)-1);  #
remove last |
                my $analyses_array = [];
                foreach my $analysis_string (split(/\|/,$analyses_string)) {
                    my $morpheme_array = [];
                    my $morpheme_strings = get_morpheme_strings($analysis_string); #
reference to array of morpheme strings
                    foreach my $morph_str (@$morpheme_strings) {
                        my $morph_parts_array = [];
                        my $morph_parts = get_morpheme_parts($morph_str);
                        foreach my $p (@$morph_parts) {
                            push(@$morph_parts_array, $p);
                        }
                        push(@$morpheme_array, $morph_parts_array);
                    }
                    push(@$analyses_array, $morpheme_array);
                    # each analysis must be tested to recover true surface form
                    my $test = reconstruct_surface($analysis_string);
                    if ($word ne $test) {
                        #print STDERR "mismatch: $word $test\n";

                        my  $true_surf  =  find_true_surf($word,$morpheme_array);  #
reference to array of true surface strings
                        # here we have access to $morpheme_array
                        # what are we passing in? key plus $morpheme array
                        for (my $j = 0; $j < @$morpheme_array; $j++) {
                            my $parts_ref = $morpheme_array->[$j];
                            unshift(@$parts_ref,$true_surf->[$j]);
                        }
                    } else {
                        for (my $j = 0; $j < @$morpheme_array; $j++) {
                            my $parts_ref = $morpheme_array->[$j];
                            unshift(@$parts_ref,$parts_ref->[0]);
                        }
                    }
                }
                $analyses_data->{$word} = $analyses_array;
                $data_count++;
            }
        }
    }
    print STDERR "$data_count analyses loaded\n";
    return $analyses_data;
}

######## find_true_surf(true_surface_string, morpheme_array)
#
# finds the true surface form
# input: $true_surf string (the input word before analysis)
# input: a reference to a $morpheme_array
# output: a reference to an array of true morpheme surface strings
sub find_true_surf {

    my $ret_ar = [];
    my ($true_surf,$morpheme_array) = @_;
    my $start = 0;
    for (my $i = 0; $i < @$morpheme_array; $i++) {
        # try to match each analyzer surface form
        my $len = length($morpheme_array->[$i][0]);
        my $test = substr($true_surf,$start,$len);
        if ($morpheme_array->[$i][0] ne $test) {#doesn't mach, do something
```

```
            # find start of next one, hope for a match.
            # starting from start_len, try to match next one
            if ($i+1 != @$morpheme_array) { #not at last asr
                my $next_start = $start+1; # there was a problem when $start+$len
went beyond where the next one started.  This seems to have worked
                my      $index      =      index($true_surf,      $morpheme_array-
>[$i+1][0],$next_start);
                if ($index == -1) {
                    ### really complain
                    print STDERR "Oh, that sucks...didn't work for $true_surf!\n";
                } else {
                    my $tmp = substr($true_surf, $start, $index-$start);
                    push(@$ret_ar, $tmp);
                    $start = $index;
                }
            } else {# when at last asr
                # just spit out what's left
                push(@$ret_ar, substr($true_surf,$start));
            }
        } else {
            push(@$ret_ar, $test);
            $start += $len;
        }
    }
    return $ret_ar;
}

######## reconstruct_surface(analysis_string)
#
# returns a surface form extracted from the analysis data
# input: analysis string {morph1}{morph2}...{morphN}
# output: the surface form according to the analysis data
sub reconstruct_surface {
    my ($form) = @_;
    $form = substr($form, 1, length($form)-2);
    my @morphs = split(/\}\{/, $form);
    my $result = "";
    foreach my $m (@morphs) {
       $m =~ /^([^:]+):/;
       $result .= $1;
    }
    return $result;
}

######## get_morpheme_parts(morpheme_string)
#
# input: full morpheme string, ex. saqqi:saqqik/1v, juma:juma/1vv, ta:ta/tv-imp-1p,
lu:lu/1q
# output: reference to array of morpheme parts, where [surface_form, deep_form,
gram_info,CAT,SUBCAT]
sub get_morpheme_parts {
    # uses the full morpheme string, so saqqi:saqqik/1v, juma:juma/1vv, ta:ta/tv-
imp-1p, lu:lu/1q
    # returns morpheme category and subcategory
    my $result = [];
    my $q;
    my $a;
    $_[0] =~ /^([^\:]+)\:([^\/]+)\/(.+)$/;
    push(@$result, $1,$2,$3);
    my $string = $3;
    if ($3 =~ /^\d([nv][nv])$/) { push(@$result, "LEX",$1); }
    elsif ($3 =~ /^\d([nv])$/) { push (@$result, "ROOT",$1); }
    elsif ($3 =~ /^t([nv]).+/) { push(@$result, "GRAM",$1."t"); }
    elsif ($3 =~ /^\d?q$/) { push(@$result, "CL","q"); $q++;}
    elsif ($3 =~ /^\d?a$/) { push(@$result, "ADV","a"); $a++;}
    elsif ($3 =~ /^\d?c$/) { push(@$result, "CONJ","c"); $q++;}
    elsif ($3 =~ /^\d?e$/) { push(@$result, "EXCLAM","e"); $q++;}
    elsif ($3 =~ /^\d?i$/) { push(@$result, "I","i"); $q++;}
######### COLLAPSE THESE THREE CATEGORIES SINCE THEY ARE ALL THE SAME PRONOUN
## DONE 11 FEB 2017
    elsif ($3 =~ /^\d?pr$/) { push(@$result, "PR","pr"); $q++;}
    elsif ($3 =~ /^\d?rp$/) { push(@$result, "PR","rp"); $q++;}
    elsif ($3 =~ /^\d?rpr$/) { push(@$result, "PR","rpr"); $q++;}
#########
    elsif ($3 =~ /^ad-(..)$/) { push(@$result, "AD",$1); $q++;}
    elsif ($3 =~ /^pd-(.+)$/) { push(@$result, "PD",$1); $q++;}
    elsif ($3 =~ /^rad-(..)$/) { push(@$result, "RAD",$1); $q++;}
    elsif ($3 =~ /^rpd-(.+)$/) { push(@$result, "RPD",$1); $q++;}
########## NEW STUFF 2/17/2016
    elsif ($3 =~ /^tad-(.+)$/) { push(@$result, "TAD",$1); }
    elsif ($3 =~ /^tpd-(.+)$/) { push(@$result, "TPD",$1); }
########## END NEW STUFF 2/17/2016
    else { # prob garbage
       push(@$result, $3,$3);
    }
    return $result;
}

######## get_morpheme_strings(analysis_string)
#
# input: analysis string {morpheme1}{morpheme2}{morpheme3}...{morphemeN}
# output: reference to an array of morpheme strings
sub get_morpheme_strings {
    my ($analysis_string) = @_;
    my $morph_string = "";
    $analysis_string = substr($analysis_string, 1, length($analysis_string)-2); #
remove outer braces
    my $array = [];
    push(@$array, split(/\}\{/, $analysis_string));
    return $array;
}

1;
```

Appendix L. punc.out

```
.       {.:period/PUNC}|
,       {,:comma/PUNC}|
(       {(:left_paren/PUNC}|
)       {):right_paren/PUNC}|
-       {-:dash/PUNC}|
?       {?:question_mark/PUNC}|
$       {$:dollar_sign/PUNC}|
>>      {>>:double_right_angle_brackets/PUNC}|
;       {;:semicolon/PUNC}|
"       {":double_quote/PUNC}|
'       {':single_quote/PUNC}|
--      {--:double_dash/PUNC}|
#       {#:octothorpe/PUNC}|
=       {=:equals_sign/PUNC}|
*       {*:asterisk/PUNC}|
%       {%:percent_sign/PUNC}|
\       {\:back_slash/PUNC}|
...     {...:ellipsis/PUNC}|
{       {{:left_curly_brace/PUNC}|
!       {!:exclamation_point/PUNC}|
+       {+:plus_sign/PUNC}|
_       {_:underscore/PUNC}|
}       {}:right_curly_brace/PUNC}|
@       {@:at_sign/PUNC}|
&       {&:ampersand/PUNC}|
>       {>:right_angle_bracket/PUNC}|
<       {<:left_angle_bracket/PUNC}|
```

Appendix M. extract_full_inuk_word.pl

```
#!/usr/bin/perl

################################################################################
#
# Name: extract_full_inuk_word.pl
# Date Created: July 15, 2014
# Author: Jeffrey C. Micher
# Copyright US Army Research Lab
# This file and all of the data contained herein are property of the
# United States Government.
#
################################################################################
#
# Description: # This script will extract the full inuktitut words from the
# fully factored file, SentenceAligned.v1_1.txt.tokpunc.morphednfactored.i
#                                     which                                    has
true_surface|returned_surface|deep|morph_analysis|morph_type|morph_sub_type
#
################################################################################
#
# Change Log:
#
################################################################################

binmode STDOUT, ":utf8";
binmode STDIN, ":utf8";

while (<STDIN>) {
    chop;
    @words = split(/\s\s/,$_);
    $count = @words;
    for ($i = 0; $i < $count; $i++) {
       $w = "";
       my @morphs = split(/\s/,$words[$i]);
       foreach $m (@morphs) {
           $w .= get_part($m,0);
       }
       print "$w";
       if ($i < $count -1) { print " "; }
    }
    print "\n";
}

sub get_part {
    my ($in, $index) = @_;
    my @parts = split(/\|/,$in);
    return $parts[$index];
}
```

1	DEFENSE TECHNICAL
(PDF)	INFORMATION CTR
	DTIC OCA
2	DIR ARL
(PDF)	IMAL HRA
	RECORDS MGMT
	RDRL DCL
	TECH LIB
1	GOVT PRINTG OFC
(PDF)	A MALHOTRA
9	ARL
(PDF)	RDRL CII T
	T R HOBBS
	C VOSS
	S LAROCCA
	J MICHER
	C BONIAL
	S TRATZ
	M VANNI
	RDRL CII
	S YOUNG
	RDRL CII
	J KLAVANS
3	CARNEGIE MELLON UNIVERSITY
(PDF)	LANGUAGE TECHNOLOGIES INSTITUTE,
	DR LORI LEVIN
	DR GRAHAM NEUBIG
	DR YULIA TSVETKO
1	UNIVERSITY OF ILLINOIS
(PDF)	DEPARTMENT OF LINGUISTICS
	DR LANE SCHWARTZ