Statistical morpheme segmentation in Inuktitut

Jessica Huynh

December 19, 2016

1 Introduction

Inuktitut is an Inuit language spoken mostly in Canada, and like many other Eskimo-Aleut languages, it is highly agglutinative, although not very fusional.[6]

2 Background

The highly regular nature of Inuktitut morphology means that one can write a rule-based morpheme segmenter with relatively high accuracy. Indeed, the National Research Council of Canada has already built one, claiming over 95% accuracy on the Nunavut Hansard corpus and similar accuracy on Inuktitut web pages, composing a list of thousands of roots and suffixes to do so.[5]

3 Experiment

3.1 Corpora

For the main corpus, I was originally going to use the Inuktitut language portions of the Nunavut Hansard.[4] The data in the hansard is formatted as such:

```
**************
qinuvvigijumavakka pigiaqtitsiqqaujuq aippiiqqaujurlu pigiaqtitauqqaujumut tasiqsiqullugit
uqaqtimik uqaqtiup iksivautangannut.
-----
I would ask that the mover and the seconder of the motion escort the Honourable Speaker to
the Speaker's chair.
**************
```

As such, it was a relatively simple task to extract just the Inuktitut portions. I opted to only take the first 25,000 lines or so, to keep the size manageable. However, the model still took too long to build, so I switched to using a much smaller corpus instead. In the future, if I had more time, I would try to use this corpus.

For the secondary corpus, I used portions of the Inuktitut Bible, which was completely translated by the Canadian Bible Society in 2012.[7] The text is in syllabics, so to make it easier to work with, I used the National Research Council of Canada's Inuktitut transcoder to get a Latin alphabet transcription. For those characters it did not successfully transliterate (namely, those oriented facing upwards and with a dot for a long vowel), I used a syllabary resource[6] and did the substitution in a text editor myself. The syllabics that needed manual substitution are in `syllabics.txt`.

3.2 Tools

3.2.1 Morfessor

Morfessor[8] is a package of machine learning methods designed for segmentation tasks, provided as both command-line and a Python package.

Morfessor considers morphemes separate from *morphs*, which could be considered like the surface representation of a morpheme, being the word segments in the data. At its core, Morfessor treats each word as a Hidden Markov Model, with the morphs as a sequence of observations. The hidden states are categories of morphs.[2]

3.2.2 Morphological analyzer

The National Research Council of Canada (NRC) has the Uqailaut project, a rule-based morphological analyzer for Inuktitut.[5] It is provided as a Java .jar file to download and in this project provides the 'ground truth' for the test sets, as well as the annotated data used to bootstrap some of the models.

While there are supposed to be some Java methods packaged within the .jar that can be incorporated into other programs, in practice I couldn't get this to work and instead just grabbed the output of the .jar for each word I wanted the morpheme breakdown for.[1]

3.2.3 Transcoder

The NRC also has an Inuktitut transcoder, to transcode between Inuktitut syllabics and the Roman alphabet.[3]

3.3 Method

The GitHub repository for this project is at `https://www.github.com/jessicahuynh/inuktitut-morpheme-segment`. Broadly, there is a pipeline script, `pipeline.sh`, that prepares the data and models and runs the different experiments before evaluating them.

3.3.1 Creating test and train datasets

The file `scripts/Corpus.java` contains all the code to take in the various corpora, put them into the correct format to be read by Morfessor, and split them into training and test sets at about a 4:1 ratio.

3.3.2 Modeling

This part of the pipeline is relatively straightforward. Morfessor provides command line tools to take in a corpus and build a binary to be read in later for the model.

3.3.3 Training, decoding, and evaluation

The script `scripts/run_hmm.py` does the training, decoding, and evaluation, all using Morfessor's built-in methods.

3.4 Approaches

Morfessor's primary mode of operation is to do unsupervised training, but I also opted to use its built-in methods to do some semi-supervised training with an additional annotated corpus (that is, one where the morpheme boundaries are already given) and do some manual adjusting of the weighting of the annotated corpus. The weights balance the costs of the corpora.

I kept the training and test set, along with the annotations, consistent throughout all of my runs.

When building the model, it took 10-11 epochs depending on the run, with training done in under 15 seconds usually. The training was all batch training with the recursive algorithm. Decoding was Viterbi.

Because of the lack of data, I opted for the evaluation to do its sampling on the test set with ten samples of twenty words each.

4 Results

File name	F-measure	Precision	Recall
Unsupervised	0.328	**0.768**	0.216
Semisupervised weight 0.0	0.350	0.695	0.237
Semisupervised weight 0.1	0.381	0.663	0.28
Semisupervised weight 0.25	0.380	0.528	0.31
Semisupervised weight 0.5	**0.447**	0.427	0.486
Semisupervised weight 0.75	0.414	0.358	0.504
Semisupervised weight 0.95	0.411	0.354	0.493
Semisupervised weight 1.0	0.387	0.329	0.482
Semisupervised weight 2.0	0.413	0.350	0.515
Semisupervised weight 2.5	0.429	0.355	0.551
Semisupervised weight 5.0	0.403	0.323	0.551
Semisupervised weight 7.5	0.426	0.351	**0.555**
Semisupervised weight 10.0	0.381	0.315	0.491

None of the results from the different experiments were especially great.

The unsupervised approach had by far the best results when it came to precision, meaning it had a higher percentage of relevant morpheme segments among the morpheme segments it picked out.

Recall was not very good for any of the results, meaning that not many relevant morpheme segments were produced. The best was the semisupervised experiment with a weight of 7.5, which still only had a recall of 0.555.

The F-measure, taking both precision and recall into account, puts the experiment with the best results as semisupervised with a weight of 0.5.

5 Discussion

In general, it looks like factoring in already annotated segmentatinos improves the number of relevant morpheme segments at the cost of the ratio of actually relevant segmentations, although none of the results are fantastic.

Part of the reason for this is the small dataset I ended up using, parts of the first two books of the Inuktitut Bible instead of the Nunavut Hansard, purely for the sake of time. With more data, the algorithm would be able to segment better. As is, the data is really sparse. There is just not much overlap between the training and test sets token-wise due to the nature of the language.

In the `results/` directory are the hypothesized segmentations for each experiment. A lot of the more common morphemes were segmented out, such as *-(i)llu*, *-up*, and *-lu*. Many of the words, however, aren't even segmented and sometimes those that are are segmented over-enthusiastically, such as *tama+k+k+uni+nga* for *tamakkuninga* in the semisupervised weighted 7.5 results.

If I had additional time, I would also see what results online and online+batch training would give, in addition to training with Viterbi.

6 Conclusion

My attempts at statistical morpheme segmentation of Inuktitut with HMMs (through the Morfessor package) and a relative lack of data produced bad-to-mediocre results. Increasing the amount of training data would help quite a bit. As is, between a statistical segmentation with a small dataset and a rule-based one, for a language with as regular a morphology as Inuktitut, I would take the rule-based segmenter.

References

[1] java runtime.getruntime() getting output from executing a command line program. http://stackoverflow.com/questions/5711084/java-runtime-getruntime-getting-output-from-executing-a-command-line-program.

[2] Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning.

[3] National Research Council of Canada. Inuktitut transcoder. http://www.inuktitutcomputing.ca/Transcoder/index.php?la

[4] National Research Council of Canada. The nunavut hansard: Inuktitut-english parallel corpus. http://www.inuktitutcomputing.ca/NunavutHansard/info.php?lang=en, 2008.

[5] National Research Council of Canada. The uqailaut project: Inuktitut morphological analyzer. http://www.inuktitutcomputing.ca/Uqailaut/info.php, 2012.

[6] Omniglot. Inuktitut @ omniglot. http://www.omniglot.com/writing/inuktitut.htm.

[7] Canadian Bible Society. Eastern arctic inuktitut bible. https://www.bible.com/versions/455-eaib-aclaait-ityorngnaaittot.

[8] Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. *Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline.*