6-2016

Infixer: A Method for Segmenting Non-Concatenative Morphology in Tagalog

Steven R. Butler
Graduate Center, City University of New York

INFIXER: A METHOD FOR SEGMENTING NON-CONCATENATIVE MORPHOLOGY IN TAGALOG

by

STEVEN BUTLER

A masters thesis submitted to the Graduate Faculty in Linguistics in partial fulfillment of the requirements for the degree of Master of Arts, The City University of New York

2016

This manuscript has been read and accepted by the Graduate Faculty in Linguistics in satisfaction of the thesis requirement for the degree of Master of Arts.

Professor William Sakas

Date	Thesis Advisor

Professor Gita Martohardjono

Date	Executive Officer

THE CITY UNIVERSITY OF NEW YORK

Abstract

INFIXER: A METHOD FOR SEGMENTING NON-CONCATENATIVE MORPHOLOGY IN TAGALOG

by

STEVEN BUTLER

Adviser: Professor William Sakas

In this paper, I present a method for coercing a widely-used morphological segmentation algorithm, Morfessor (Creutz and Lagus 2005a), into accurately segmenting non-concatenative morphological patterns. The non-concatenative patterns targeted—infixation and partial-reduplication—present problems for many segmentation algorithms, and tools that can successfully identify and segment those patterns can improve a number of downstream natural language processing tasks, including keyword search and machine translation.

Included with this project is an implementation of the segmentation method described in the form of a Python library called `Infixer`. This approach involves a preprocessing step that restructures non-concatenative patterns into concatenative ones using regular expressions, which allows the algorithm to operate on the input data as it would any other data. The target language for this project is Tagalog, a major language of the Philippines that makes extensive use of non-concatenative morphology, and the training data is built from data from the IARPA Babel Program and the Tagalog Wikipedia.

The results for this test were promising, especially for the more straightforward cases of infixation tested. For the data tested, the Infixer implementation using affix regular expressions showed performance gains over those without, demonstrating an improved ability to segment data containing non-concatenative morphological forms. In the future it is hoped that this project can lead to the development of tools that can be used effectively on languages besides Tagalog and on a more diverse array of phenomena.

Acknowledgments

I would like to thank everyone who advised, assisted, and supported me through the completion of this project. That includes my advisors and instructors William Sakas, Andrew Rosenberg, Nava Shaked, Juliette Blevins, and all of the other professors I've had through my time at the Graduate Center. I would also like to thank many of my fellow students for the invaluable support and advice they have provided me, including Rebecca Lovering, Rachel Rakov, Maxwell Schwartz, Pablo Gonzalez, Michelle Morales, all of the other members of the computational linguistics reading group.

Finally, I have to thank my partner, Susan Heinick, for quite literally supporting me through every step of this process, as well as my family, Melanie Mendenhall, Robert Butler, Amanda Butler, Daphne Mendenhall, and everyone else.

This work was partially supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD / ARL) contract number W911NF-12-C-0012. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government. This effort uses the IARPA Babel Program Tagalog language collection release IARPA-babel106b-v0.2g.

Contents

List of Tables

1 Introduction

Morphology is the division of linguistics focused on morphemes, the smallest grammatical units in language, and morphological segmentation is an area of research in natural language processing (henceforth NLP) focused primarily on methods for splitting words into their component morphemes.

Accurate morphological segmentation can lead to dramatic improvements in many "downstream" NLP tasks, such as machine learning and keyword search, as segmentation can reduce the percent of out-of-vocabulary (OOV) tokens in a corpus, and can group words that share a lemma form but differ in inflection. When an NLP process only makes use of work tokens, morphologically-related forms that vary due to inflection or some other form of affixation will be treated as entirely distinct. To take an example from Indonesian, the verbs *mendukung* "support" (active voice) and *didukung* "support" (passive voice) are transparently derived from a single root, *dukung* "support". Without segmentation of the data, a keyword-search operation would fail to correctly return *didukung* on a query for *mendukung*, reducing the usefulness of the system overall.

For high-resource languages like English, Spanish, and French, a variety of well-supported and tailored morphological segmentation systems already exist, such as the tools found in the Stanford CoreNLP Toolkit (Manning et al. 2014). Tools like these can and do provide high quality results for their target languages, but the time and money needed to build a system of that quality are substantial, and preparing something similar for even a tiny fraction of the more than 7000 languages currently spoken across the globe would be prohibitively expensive. For the speakers of many of those languages, even those with tens of millions of speakers, access to tools that result from the development of advanced NLP systems are currently out of reach.

Instead, many researchers have turned to designing semi-supervised and unsupervised machine-learning algorithms that can be applied to many different languages, both high- and low-resource. One particular family of unsupervised algorithms, Morfessor, first proposed in Creutz and Lagus (2002), has seen a great deal of use for a number of NLP applications when the task involves

segmentation of less widely-used languages. While both the original algorithm and many of the variations on it have been shown to be quite effective on languages with rich *concatenative* morphological systems, the design of the algorithm prevents it from being effective for languages that make use of non-concatenative patterns, such as infixation and partial-reduplication. These patterns, which will be discussed in more detail in the following section, offer a unique challenge to most segmenters because they cause discontinuities in word roots and unanalyzable repetative segments. Because of this, techniques that work well for languages with largely concatenative morphological systems (like English or Japanese) fail for languages with significant non-concatenative patterns (like Tagalog and Arabic).

In this paper, I present investigate a method for coercing a recent Python implementation of Morfessor (Smit et al. 2014) into segmenting non-concatenative patterns, as well as a Python implementation of the method described, dubbed `Infixer`. The Python files, which can be used as a library interface, are included with this project. This approach involves a preprocessing step that restructures non-concatenative patterns into concatenative ones, allowing the algorithm to operate on the input data as it would any other dataset. The target language for this project, Tagalog, is a major language of the Philippines and makes extensive use of non-concatenative morphology. Though at this time the project falls short of providing a tool set that can be used on languages besides Tagalog, I hope to demonstrate that it is possible to modify already extant, well-documented tools in a way that expands the range of phenomena that can be investigated with them.

In section 2 I present background information on the linguistics of morphology, morphological segmentation with Morfessor, and Tagalog. In section 3, I discuss training data sources and corpus preparation. In section 4, I present the processing algorithm and the use of regular expressions. In section 5 the algorithm's implementation in detail. In section 6, I discuss the method's evaluation results and its strengths and weaknesses. Finally, in section 7 I discuss methods for improvement and potential future work on this question.

2 Background

2.1 Morpho Challenge and Morfessor

From 2005 to 2010, an annual MorphoChallenge event was organized by researchers at Aalto University in Espoo, Finland, aimed at encouraging the development of unsupervised morphological segmentation algorithms targeting raw text data in morphologically rich languages like Finnish, Turkish, and Arabic. Throughout the five years of the challenge, more than 50 segmentation algorithms were evaluated, refined, and collaborated on, and escalating challenges for algorithm designers were introduced, such as improvements focusing on information retrieval or statistical machine translation (Kurimo et al. 2010). During each of the Challenges, the competing algorithms were tested against each other and the baseline, aptly titled Morfessor Baseline. Morfessor Baseline is a open algorithm first presented in Creutz and Lagus (2002), and in most of the tests throughout the different years of the Challenges, it performs well above average in a variety of tasks. It was selected as the target morphological segmenter for this project because of its use within the morphological segmentation literature as a baseline of comparison, its wide-use in toolchains for research in other areas of NLP, and its easily accessible implementation in Python.

Morfessor Baseline is an unsupervised, probablistic generative model that uses a minimum-description length cost function to generate a lexicon and grammar that can describe the compounds found in a corpus. The algorithm is agnostic as to the type of data it is segmenting, and works effectively on a variety of tasks such as morphological segmentation (the task in this paper), the segmentation of Chinese sentences into words, and shallow parsing and chunking tasks. Its primary search units are *atoms*, *constructions*, and *compounds*, which for the purposes of this project are equivalent to letters, morphemes, and words. When provided with a corpus consisting of words from a single language to be segmented, the algorithm conducts a "greedy and local search" on each word. That is, it attempts to find both the optimal segmentation for a given word, as well the optimal segmentation of that word given the segmentations proposed for all other words in the corpus. A trained model can then be used to segment new data, and for that it uses a Viterbi

algorithm that finds optimal segmentations without altering the parameters of the model.

The first version of Morfessor as a piece of software was released in 2005 as a Perl command-line utility (Creutz and Lagus 2005b). The version being used in this project is Morfessor 2.0, an improved and expanded version of the tool re-implemented in Python, as described in Virpioja et al. (2013). This version includes full Unicode support, which makes it possible to use non-Latin data sets, and a library interface, which make it possible to integrate calls to the algorithm directly within other Python programs. This version of Morfessor was released under a BSD license, a type of very liberal open-source software license[1]. This fact makes it significantly easier to reuse and modify included code, a positive note for choosing this implementation as a base for this project over others.

2.2 Non-Concatenative Morphology

Non-concatenative morphology can be defined, broadly, by what it is not: morphological patterns that cannot be described by the simple concatenation of morphemes. It is contrasted, unsurprisingly, with concatenative morphology, where morphemes are concatenated linearly[2] (Haspelmath and Sims 2010). Because this topic is quite broad, I want to focus on two phenomenon, both present in the project's target language Tagalog, that present particular challenges to concatenative morphological segmenters: infixation and reduplication.

The phenomenon of infixation is a canonical example of a non-concatenative pattern. Though somewhat rare cross-linguistically, it is relatively common throughout the Austronesian language family, and appears in languages as different as Yurok (Algic, formerly spoke in California, USA) and Pingding Mandarin (Sino-Tibetan, spoken in Shanxi, China)[3]. Blevins (2012) defines an infix as "...a bound morpheme whose phonological form consists minimally of a single segment, is preceded and followed in at least some word-types by non-null segmental strings". This causes

[1] See `https://opensource.org/licenses/BSD-2-Clause` for details.

[2] For example, the word *words* is the concatenation of two morphemes, *word* and the bound morpheme and plural marker, *-s*.

[3] See Yu (2004) for details on the latter.

the base that receives the infix to become phonologically (and by extension, orthographically) discontinuous. For example, the Tagalog verbal root *sulat*, meaning to write, can take the infix *-um-* (discussed below), as follows:

(1) *sulat* + *-um-* → *s**um**ulat*

The root *sulat* is separated into two discontinuous parts, *s* and *ulat*, which bear no meaning of their own and are not valid morphemes in Tagalog. Understandably, this kind of pattern is difficult to detect for systems that only expect variation at the edges of words (i.e., prefixes and suffixes) rather than right in the middle of words.

Reduplication is a significantly different kind of phenomenon, but also arguably falls outside of the scope of pure concatenation. It is a pervasive phenomenon throughout the world's languages, and involves the copying of either part or all of a word base is copied or attached to that base (the former is generally referred to as "partial reduplication"). It is a different type of phenomenon than normal affixation because its actual form varies a great deal from word-to-word, even when the grammatical information it conveys is consistent. For example, using the same verb as above, *sulat*:

(2) *sulat* → ***su**sulat*

The partially-reduplicated segment is *su*; its presence indicates contemplative aspect on the verb, but its actual phonemic and orthographic forms do not correspond to that meaning in other verbs (Haspelmath and Sims 2010). Thus, for an algorithm like Morfessor, there is no way to group the particular surface forms together as instances of the same *type* of morpheme, as their orthographic forms bear no similarity to each other; the only constant is the relationship in form to the host morpheme or root.

2.3 Tagalog Grammar

Tagalog is an Austronesian language spoken primarily in the Philippines, with approximately 21.5 million native speakers as of 2000 according to Ethnologue. Additionally, it serves as the basis for the national language of the Philippines, Filipino, which is spoken by an additional 45 million people as of 2013 (Lewis, Simons, and Fennig 2016). The Filipino language is generally considered to be a standardized variety of Tagalog by most speakers and linguists, and will be considered synonymous for the purposes of the work in this paper.

Nearly all of the languages spoken in the Philippines are Austronesian languages of the Philippine sub-group, which includes Cebuano, Ilocano, Hiligaynon, and more than 120 others (Schachter and Reid 2009). The Austronesian language family itself is a large family of languages spoken throughout maritime southeast Asia and the south Pacific, with outliers in Madagascar and Hawaii. Besides Tagalog, other notable Austronesian languages include Malay, Indonesian (a standardized register of Malay used as the national language of Indonesia), Javanese, Malagasy (the official language of Madagascar), Hawaiian, and Maori. Morphologically, the languages of this family tend to use affixation to mark things like transitivity, voice, and focus, and virtually every one of them have productive use of reduplication. (Clark 2009)

Include below are descriptions of a few features of Tagalog grammar that are relevant to the project in this paper. They are drawn primarily from Schachter and Otanes (1972), one of the few comprehensive descriptions of Tagalog grammar available in English. Note that all of the features discussed below are exclusively verbal phenomena, and morphology for other parts of speech (nouns, adverbs, etc.) remains outside the focus of this paper. Additionally, the description below is extremely simplified, focusing only the aspects of Tagalog morphosyntax relevant to the topic at hand.

2.3.1 Focus System

An oft-discussed feature of Tagalog and many other Philippine language is what is generally referred to as its focus system. In contrast to many more widely studied European languages, where

verbal affixes and paradigms tend to relate to tense and the verb's subject, Tagalog verbs have a large number of affixes that modify the verb's overall argument structure, effectively putting different arguments of the verb in focus. This often has the side-effect of specifying the definiteness of one of the arguments, similar to the use of the articles "a" and "the" in English. An illustration of two common types of focus, Actor and Object focus (henceforth AF and OF) is demonstrated with the verb *hawak* "hold" (Schachter and Otanes 1972, p. 295):

(3) **Actor focus**: *Humawak siya nang libro.* "He held **a** book."

(4) **Object focus**: *Hinawakan niya ang libro.* "He held **the** book."

Variations in the morphology on the verb root (the addition of the infix *-um-* and addition of the infix *-in-* and suffix *-an*, respectively) along with case particles like *nang* and *ang* signal how the arguments interact with the verb.

2.3.2 *-um-*

The infix *-um-* is a common verbal affix that denotes, broadly, actor focus, and can appear with both transitive and intransitive verbs. In most cases, it is placed between the first consonant and first vowel of a given verb root, as in:

(5) **Verbs using <um>**

a. *kain* "eat" → *kumain* "eat (AF)"

b. *hiram* "borrowed" → *humiran* "borrow (AF)"

c. *dating* 'arrival' → *dumating* "come (AF)"

Exceptions to this placement include a ban on usage with bases beginning with *m* or *w*, as well as its usage as a prefix for vowel-initial roots, as in:

(6) *abot* "overtaken" → *umabot* "reach for (AF)"

2.3.3 *-in-*

The infix *-in-* is one way of marking object focus in verbs. Like *-um-*, it is generally placed between the first consonant and first vowel of a verb root, as long as the verb is not vowel initial:

(7) **Verbs using *-in-***

a. *kain* "eat" → *kinain* "ate (OF)"

b. *gupit* "cut with scissors" → *ginupit* "cut with scissors (OF)"

c. *bigay* "give" → *binigay* "gave (OF)"

2.3.4 Partial Reduplication and Aspect

Partial reduplication plays an interesting role in the Tagalog verbal system. Formally, it repeats the first full syllable of the root and marks the contemplative aspect:

(8) *basa* "read" → *babasa* "read (AF, CONT)"

(9) *tawa* "laugh" → *tatawa* "laugh (AF, CONT)"

When combined with either of the infixes above, it indicates imperfective aspect. Interesting, the infixes are apparently inserted after the reduplication takes place, as they are placed between the first consonant and vowel of the *reduplicated* form:

(10) *basa* "read" → *bumabasa* "read (AF, IPFV)"

(11) *basa* "read" → *binabasa* "read (OF, IPFV)"

This fact is the impetus for the multi-step affix search process outlined below in section 4.2.

2.3.5 Tagalog as a Test Language

Tagalog was selected as the target language for this project because of its pervasive, productive use of both infixation and reduplication and because of the relative availability of data sources to

some of the other potential test languages (see section 7.1). Sequential segmentation works well for many Tagalog verbs, but given the ubiquity of infixes like *-um-* in common verbal paradigms, non-concatenative patterns cannot be ignored.

3 Data

3.1 Corpus Preparation

Two Tagalog-language corpora were used in this project: a smaller corpus from the IARPA Babel Program, and one containing the full contents of the articles from the Tagalog Wikipedia. The two data sources were processed in such a way that the final output was a wordlist of tokens prepared for input into the Morfessor segmentation algorithm. Token frequency was preserved via repetition, while any sort of contextual information (i.e., *n*-grams) was not stored, given its irrelevance to the segmenter. Tokens were extracted from the texts using a combination of tools developed for this project as well as language-agnostic, whitespace-oriented tokenizers included in Python's Natural Language Toolkit, henceforth NLTK (Bird, Loper, and Klein 2009). After the initial token extraction, each set of tokens was put through a multi-step filtering process to ensure that the final dataset was composed with tokens relative to the task as possible. Those steps included the removal of:

1. Names included in the *Names Corpus* provided by NLTK (Kantrowitz and Ross 1991). This was especially relevant for the Tagalog Wikipedia corpus, which includes many articles about individuals. Though the corpus seems to be heavily geared toward names used in English-speaking countries, filtering using this resource seemed to be moderately effective.

2. Non-word tokens. In the Babel corpus, this primarily consisted of notation used to designate non-linguistic content like laughing and coughing, "foreign" words[4], and pauses. In the Wikipedia corpus, numerals were by far the largest category removed.

[4]It seems that this was left to the discretion of the transcriber, as many transparently English and Spanish loanwords were not considered foreign enough to be excluded.

3. Words and phrases written in non-Latin characters. This was primarily to remove obviously non-Tagalog words written in other character sets, such as the Greek, Chinese, and Cyrillic writing systems.

4. Words that never appeared uncapitalized[5]. Tagalog appears to broadly follow the same capitalization norms that are used in English[6], and proper nouns generally appear capitalized in texts like Wikipedia. Because the focus of this project is morphology present on verbs, nouns like names, places, corporations, etc., are not relevant and could affect the segmenter by leading to the segmentation of non-native patterns. While this makes the data significantly more unlike corpora that might be encountered in a non-experimental setting, because this project aims to be a proof-of-concept, this was taken as a simplifying measure.

5. The top 10,000 most frequent English and Spanish as measured by wordlists obtained from opensubtitles.org.[7] Tagalog makes extensive use of both Spanish and English loanwords, and the intention of this step was not to remove non-Austronesian derived vocabulary, but rather to remove tokens that exhibit no Tagalog inflectional or derivational morphology. This is particularly important for the Wikipedia corpus, for the reasons outlined in section 3.3. For instance, the token *gumraduate* "to have graduated" with an *-um-* infix would be retained, but the token *graduate* would not. Like in 4, this decreases the naturalness of the corpus with the goal of decreasing the likelihood of misidentifying foreign morphological patterns that are not productive in Tagalog as native Tagalog morphology.

3.2 IARPA Babel Program

The IARPA Babel Program, currently in its fourth generation, aims to encourage the development of tools that will aid in the analysis of low-resource languages of emerging importance to the

[5]This is not relevant to the Babel data, which is entirely uncapitalized.

[6]See *Microsoft Filipino Style Guide* (2011) for details.

[7]Following Baumann and Pierrehumbert (2014), I also used the compiled wordlists from `https://invokeit.wordpress.com/frequency-word-lists/`

US intelligence community[8]. As such, the program provides researchers with audio data in a variety of languages recorded in a variety of contexts, including in environments with a lot of noise and in conversations that vary in register and level of formality. The languages chosen for research each year come from a diverse array of language families and display a great deal of typological diversity; languages in the program's first year included Cantonese (Sino-Tibetan), Tagalog (Austronesian), and Pashto (Indo-Iranian).

While the thrust of the program is the development of tools that can perform *audio* keyword-search tasks with high accuracy across a variety of languages, transcripts are provided for a certain percentage of each of the corpora, corresponding to the training and development sets. Those transcripts have seen a great deal of use in the NLP community, given the relative rarity of high quality text transcriptions of some the languages in question. Researchers interested in problems related to morphological parsing have made a great deal of use of this data, and unsupervised morphological segmentation in particular seems to have become a particularly fruitful area of interest; see Baumann and Pierrehumbert (2014) and Rasooli et al. (2014) for interesting examples.

The audio transcript files for the Tagalog language data are divided into two sets labeled training (3052 files with an average of approximately 9500 words per file) and dev (21 files with an average of approximately 600 words per file)[9]. The dev files were originally intended for feature tuning, but in practice tend to be used as a test set by researchers, given that test sets are not provided to researchers except during evaluation by IARPA. The files contain extensive timestamps delimiting each utterance as well as time not containing dialog, as below:

```
[18.145]
<no-speech>
[19.425]
kumusta 'di mo ako nari-
[23.315]
<no-speech>
```

[8]See `https://www.iarpa.gov/index.php/research-programs/babel` for more information.

[9]These files are from IARPA Babel Program Tagalog language collection release IARPA-babel106b-v0.2g

```
[25.055]
dito lang sa dito lang sa bahay ng tropa namin
[28.165]
<no-speech>
```

The files are transcripts of both scripted and impromptu dialogs, recorded in a variety of settings; however, the dialogs are not labeled for their contents, so it has no bearing on the analysis. After the steps outlined in 3.1, this corpus consisted of 36,510 tokens of 7,728 types.

3.3 Wikipedia

The Wikimedia Foundation, which hosts Wikipedia, offers downloads of complete copies of the Wikipedia article database in each language that it hosts[10]. The downloads are *wikitext* source (the article text and associated context) and metadata embedded in XML files, and are updated approximately monthly. The Tagalog Wikipedia download used in this project is from the January 11th, 2016 update.

The article text was extracted from the XML using a Python script called `WikipediaExtractor` that is maintained by the Media Lab at the University of Pisa and made freely available on Github[11]. The extraction process removes everything but the titles, texts, document IDs, and URLs of each article, leaving text output that in this case totaled approximately 44 MB, 65,000 articles, and 6.7 million words.

As mentioned in the preparation steps outlined in 3.1, the removal of English text was especially important for the Wikipedia data, as English text appears throughout the corpus in metadata, captions, *stubs* (articles with little content, sometimes serving as a starting point for new articles or article translations), and article content. After the processing of the data as recorded above, the corpus consisted of 1,211,930 tokens of 139,922 types.

[10] `http://dumps.wikimedia.org`
[11] `https://github.com/attardi/wikiextractor`

4 Method

In this section, I will outline the process used and how it was implemented. In section 4.1 I discuss the algorithm I have developed in general terms, in 4.2 I discuss using regular expressions to model affix morphology, and in 5 I discuss the method's implementation using the Python programming language.

4.1 Overview of the Process

The method is a five-step process requiring the following input:

1. A corpus, which should consist of a text file containing each word in the corpus separated by newlines. Duplicate tokens should be preserved, but order need not be.

2. A set of affixes expected to be found in the corpus, formatted as regular expressions. These affixes should be in descending order of precedence, taking into account affixes that preclude the presence of another affix.

The steps are as follows:

1. Run the unmodified corpus data through one pass of the Morfessor algorithm and extract a set of releavant features from the resulting trained model's segmentation list. Store these features in a hash table, with a key for each type present in the corpus mapped to another has table containing the needed features, including the list of segments and the hypothesized word root.

2. Test each key in the hash table against the set of affixes in descending order. If a match is found, a modified form (the "test form") of the word is stored in its feature table (the modification process is discussed below in section 4.2).

3. Build a new wordlist consisting of the test forms from step 2 along with the remaining words in the table where no match was found. Rerun the Morfessor algorithm and store the same features extracted in step 1 in the feature set for the original form of the word.

4. For each word in the table, determine whether the modified or unmodified form should be used for training the final model. This involves testing whether the hypothesized word roots from the model in step 3 appears in a list of all of the hypothesized roots from the initial model. Store the final choice in the feature table.

5. Train the final Morfessor model using the final word forms determined in step 4.

From there, the trained model can be used for evaluation and segmentation tasks in a similar way to a standard Morfessor model.

4.2 Regular Expressions for Affixes

For the method defined in this paper, each affix under investigation needs to be translated into a regular expression that matches the largest number of instances of that affix as possible while minimizing false positives. A regular expressions is, at its core, a notation for characterizing sets of strings. Widely used in virtually all systems dealing with text data at some level, regular expressions allow patterns to be defined for search and match operations that can make use of a variety of wildcard characters (Jurafsky and Martin 2009). While the purest implementations of regular expressions are essentially implementations of finite state automata, some modern versions, including the one used in the Python programming language, have extra features such as named groups.

The design of the program, outlined below in 5, errs on the side of preferring regular expressions that lead to more false positives than false negatives, and provides a process for balancing out the results of that decision. For infixation, writing regular expressions that conform to these criteria turns out to be fairly straightforward, while partial-reduplication is a much more difficult problem to deal with, due to its reliance on the named groups feature mentioned before.

Included in each section are rewrite rules, which are special regular expressions that define the transformations that should take place on a word that is matched by one of the affix regular expressions. The purpose of these rewrite rules are to make what is a non-linear process (the non-

concatenative morphology discussed) into a linear one, thereby allowing Morfessor to process this data as it would any other.

4.2.1 Infixation

For the two infixes under investigation, *-in-* and *-um-*, the regular expressions needed to find them in offer few surprises. Though the two of them could have been easily combined into a single expression using using a disjunction group, keeping them as distinct expressions allowed for individual testing, reordering, and removal when necessary.

(12) ***-in-***: `/^(i?\w)(in)(\w*)/`

(13) ***-um-***: `/^(\w)(um)(\w*)/`

(14) **Rewrite**: `/<\g<2>>-\g<1>\g<3>/`

Both of the infixes will match any string where their respective two letter sequences appear after the first letter of a word. Additionally, the expression for <in> will match instances of the sequence *in* after the second letter of a word if the word begins with *i*, which is a common prefix with either a causative or benefactive meaning that can co-occur with the *-in-* infix.

The **Rewrite** regular expression is used in step 2 of the method outlined in section 4.1 above. It uses group notation to reorder the component groups matched by the regular expression. That expression reconstructs the full form the word base, moves the infix to a word initial position followed by a hyphen, a character that Morfessor splits words on by default. The infix is also surrounded with carets, which aids in correctly maintaining the moved segments as single units during testing. Some example transformations include:

(15) *binasa* 'read (PERF, object focus)' → `<in>-basa`

(16) *humálik* 'kiss (PERF, actor focus)' → `<um>-hálik`

(17) *ibinalik* 'bring back (PERF, object focus)' → `<in>-ibalik`[12]

The flexibility of the regular expressions above does lead to some to some false positives. Two notable examples include *hindî*[13], a very common word used to negate verbs, and *sinungaling*, a root word meaning "lie, liar":

(18) **hindi* 'no, not' → `<in>-hdi`

(19) **sinungaling* 'lie, liar' → `<in>-sungaling`

Because the output from the filtering leads to proposed roots like *hdi* that do not appear in the corpus (and do not conform to Tagalog phonotactics more generally), the proposed filtered form will not be accepted into the word list for training the final output model.

4.2.2 Infixation within Reduplication

For this project, two particular types of reduplication were targeted: partial reduplication with followed by the insertion of an infix, and partial reduplication of a word's initial syllable. As was noted in above in section 2.2, reduplication is pervasive in Tagalog, and targeting the myriad types found in the language would test the law of diminishing returns. The two types of reduplication chosen are both used by Tagalog's aspect-marking system (along with the bare infixes), meaning they are frequently used contrastively with the same word roots and could be tested on similar types of test data (Schachter and Otanes 1972).

The regular expressions and rewrite rules for reduplication with infixation are:

(20) **Reduplication with *-in-***: `/^(?P<CON>\w)(in)(?P<VOW>\w)((?P=CON)(?P=VOW)\w*)/`

(21) **Reduplication with *-um-***: `/^(?P<CON>\w)(um)(?P<VOW>\w)((?P=CON)(?P=VOW)\w*)/`

(22) **Rewrite**: `/<redup>-<\g<2>>-\g<4>/`

[12] Note that this is not yet correctly segmented, as the *i-* prefix is still attached to the base.
[13] Often spelled *hindi* in the data sets.

The `(?P<x>)...(?P=x)` syntax denotes a named group and a callback to that group in the regular expression syntax used by Python's standard `re` module, and is not necessarily compatible with other regular expression implementations. In (20) and (21), the named groups capture a pattern like that in the word *bumabasa* in its derivation from *basa* 'read':

(23) *basa* + reduplication → *babasa* + *-um-* → *bumabasa*

The first and second consonant of the base *basa* are reduplicated and appended to the beginning of the root, and then the <um> infix is inserted following its normal pattern. The named groups `(?P<CON>\w)` and `(?P<VOW>)` represent the consonant and vowel in the reduplicated initial syllable, allowing the two to be recalled later. Their rewrite rule leads to the following transformations:

(24) *hinahanap* "find (IPFV, object focus)" → `<redup>-<in>-hanap`

(25) *lumalabas* "go out (IPFV, actor focus)" → `<redup>-<um>-labas`

Like the infixation rewrite rule in 4.2.1, the infix is prepended to the beginning of the reconstructed base and surrounded by carets. Additionally, a reduplication marker `<redup>` is added to beginning to mark the presence of a reduplicated element. Marking reduplication with a tag rather than preserving its original form[14] has the advantage of allowing a downstream syntactic or semantic analysis to assign a function to the underlying process (reduplication) rather than a potentially unlimited set of heterogeneous orthographic forms (such as *ha-* and *lu-*, from the previous examples).

While the specificity of this set of expressions leaves little room for false positives, it does have a major source of false negatives: words beginning with digraphs. While infrequent, there are certain two letter combinations that represent a single phoneme in Tagalog, and those will not be found with the above expression:

(26) *ngiti* 'smile' + reduplication → *ngingiti* + *-in-* → *ngumingiti*

[14]For (24), preservation of the original form would lead to the less useful `<ha>-<in>-hanap`.

Words beginning with the sequence “ng”, representing the phoneme //, were infrequent enough in the data sets that it was not factored into the regular expression to ensure that it targeted a reasonable percent of potential instances of the affixes.

4.2.3 Reduplication Alone

The regular expression and rewrite rule used for initial-syllable reduplication is as follows:

(27) **Reduplication**: `/^(?P<REDUP>\w*)((?P=REDUP)\w*)/`

(28) **Rewrite**: `/<redup>-\g<2>/`

Unlike the regular expressions in 4.2.2, this expression has the potential to have quite a number of false positive matches, given that it will match any word that has an arbitrarily long sequence of letters repeated at its beginning. Some example matches include:

(29) *aalis* ‘withdraw from (CONT, actor focus)’ → `<redup>-alis`

(30) *lilipad* ‘fly (CONT, actor focus)’ → `<redup>-lipad`

(31) *ngingiti* ‘smile (CONT, actor focus)’ → `<redup>-ngiti`

Note both one-, two-, and three-letter initial reduplication is detected by this expressions, with one-letter reduplication generally applying to vowel initial word roots.

4.2.4 Ordering

When affix regular expressions are input into a model for training, they have to be ordered in such a way that the most appropriate regular expression is used first in order to ensure a match. In the implementation presented below, the list of affixes is searched until a match is found (if one is found at all), the loop breaks, and the search begins anew on the following word. For Tagalog, this is especially important: infixation within reduplication needs to be searched before infixation

alone, as the set of strings matched by the former is a subset of those matched by the latter. If the order of the two sets of affixes were reversed, there would be a large number of false positives for infixation alone, and theoretically very few hits for infixation within reduplication.

While the same set theoretic relationship does not hold between regular expressions for reduplication alone and either of the other sets, it was deemed necessary to order it last, as it had the highest probability of producing false positives on its own. The final ordering for all of the regular expressions was set as:

(32) **Final Regular Expression Ordering**

a. **Reduplication with *-in-***: `/^(?P<CON>\w)(in)(?P<VOW>\w)((?P=CON)(?P=VOW)\w*)/`

b. **Reduplication with *-um-***: `/^(?P<CON>\w)(um)(?P<VOW>\w)((?P=CON)(?P=VOW)\w*)/`

c. **Infix *-in-***: `/^(i?\w)(in)(\w*)/`

d. **Infix *-um-***: `/^(\w)(um)(\w*)/`

e. **Reduplication**: `/^(?P<REDUP>\w*)((?P=REDUP)\w*)/`

This relative ordering was held for all models tested, including those that only tested a subset of the available regular expressions.

5 Implementation

The process outlined in 4 is implemented as a Python module that attempts to replicate most of the functionality of the Morfessor 2.0 Python library, except for the command-line tools. It has been given the tentative name `Infixer`.

Infixer targets Python 3, and is compatible with both the CPython reference implementation and PyPy, an alternative Python implementation that enables large speed increases in code execution via Just-In-Time compilation[15]. It is currently not compatible with Python 2 due to its reliance on Python 3's native support for Unicode literals without needing to call special conversion libraries.

[15] `http://pypy.org/`

The writing systems for Tagalog and many other low-resource languages make use of characters not directly supported by encoding scheme used in Python 2; ensuring a stable character encoding scheme was deemed a priority for an algorithm that explicitely looks for patterns in a language's orthography to determine its segmentation.

The module's main external dependency is the `morfessor` library hosted on PyPI[16]. The tokenizers included in `tokenizers.py` also include a dependency on NLTK[17], but the rest of the library does not require the use of the tokenizers. Besides those, only Python standard library resources are used. Dependencies were kept to a minimum in order to ensure compatibility with both of the mentioned Python implementations.

5.1 Tokenizers

Infixer includes a file, `tokenizers.py`, that contains two classes for the tokenization and filtering process outlined in 3 (`BabelTokenier` and `WikipediaTokenizer`), as well as a superclass (`GeneralTokenizer`) that can be subclassed to deal with other data sets if needed. Assuming the API outlined in the file's documentation is maintained, any output file from these tokenizers should be compatible with both Morfessor and Infixer.

5.2 Filtering and Modeling

The file `model.py` includes the classes necessary for implementation of the algorithm outlined in 4.1, and also includes some calls to a file containing utility functions, `utilities.py`.

5.2.1 `AffixFilter`

The class `AffixFilter` is initialized with a list of regular expression affixes, as described in section 4.2. It then pairs each regular expression with one corresponding to the aforementioned rewrite rules, enabling the generation of transformed forms. As of this writing, the regular expressions

[16] `https://pypi.python.org/pypi/Morfessor`
[17] `https://pypi.python.org/pypi/nltk`

for rewrite rules are hard-coded and determined by checking a given regular expression's named groups; I intend to improve this part of the system at a future date.

The `filter_word` method is where the ordering described in section 4.2.4 is used to filter an input word with the first regular expression that matches it, if at all. It returns either the filtered word or the original word, if no filtering is needed.

The `filter_feature_dictionary` method is used by the `InfixerModel` class for the process outlined in section 4.1, step 2. It takes an `InfixerModel`'s feature dictionary as input and returns an updated version of it that includes added fields for words filtered by one of the input regular expressions.

5.2.2 `InfixerModel`

The `InfixerModel` class is the centerpiece of the module, and allows for the successive creation of the three Morfessor models needed for the process in section 4.1. It is initialized with a word list (as output by the tokenizers outlined in 5.1), a list of affixes for input into an `AffixFilter` class, and an optional frequency dampening parameter for the Morfessor models. The model as a whole is constructed around a feature dictionary that stores features of each word included in the initial word list, and is essentially a `dict` data structure containing `word:dict` key-value pairs.

The phases of the building of a model are labeled `INIT`, `TEST`, and `FINAL` internally. These correspond to the training of the initial Morfessor model and the extraction of initial features from it, the testing of each word against the affix regular expressions and the training of another Morfessor model using those transformed forms, and the process by which the appropriate final form of a word is selected and the final Morfessor model trained.

In the `INIT` phase, a Morfessor model is trained on the input word list and the relevant features from that model's output are stored in the feature dictionary. This and subsequent Morfessor model calls are currently done via a function in the `utilities.py` file that acts as a wrapper for a call to the command-line version of Morfessor; it is hoped that this interface can be improved in the future. Upon a call to the `build_test_model` method, the `TEST` phase begins and the feature

dictionary is updated by using the `AffixFilter` class's `filter_feature_dictionary` method, which updates the entry for each word that matches one of the supplied regular expressions with a transformed form following the rewrite rules discussed previously. The `FINAL` phase is initiated by calling the `build_final_model` method, which examines each entry that underwent a transformation in the previous model and determines whether that transformation should be used in the final model. It makes use of a `test_root_count` feature (described below) and a user-determined threshold value. The form of the word used in the `TEST` phase is used again as `fina_word_root` for any entry with a `test_root_count` greater than or equal to the threshold, while all other words revert to their original form.

The features stored by the feature dictionary are as follows, and their names generally reflect the phase at which they are added to the feature dictionary. Type information is included for each feature for clarity. Starred forms are present for every entry, unstarred forms are only present if relevant to that word:

1. * `count` (`int`): the number of times the word occurs in the corpus
2. * `init_word_base` (`str`): the original word
3. * `init_root` (`str`): the hypothesized root for the original word, corresponding to the longest morpheme in `init_segments` (see below)
4. * `init_segments` (`list`): the original word's segmentation from the first Morfessor model
5. * `test_root` (`str`): the hypothesized root for the word used in the second Morfessor model (either `test_transformed` or `init_word_base`)
6. * `test_segments` (`list`): the segments for the word used in the second Morfessor model
7. `test_transformed` (`str`): the rewritten form of the original word, if it was matched by one of the affix regular expressions
8. `test_has_redup` (`bool`): whether or not the transformed word was matched by a regular expression containing reduplication

9. `test_infix` (`str`): the infix found, if the word was matched by a regular expression containing an infix

10. `test_root_count` (`int`): the number of occurrences of the `test_root` in a list of the roots found by the first Morfessor model

11. `test_root_per` (`float`): the `test_root_count` as a percentage of all roots in the list of roots found by the first Morfessor model

12. `final_word_base` (`str`): the form of the word selected for use in training the final Morfessor model, equal to either `init_root_base` or `test_transformed`

13. `final_root` (`str`): the hypothesized root for the word used in the final Morfessor model

14. `final_segments` (`list`): the `final_word_base`'s segmentation from the final Morfessor model

The first two hypothesized root forms (`init_root` and `test_root`) are needed in order for the model to make predictions about whether or not a particular transformation because of a regular expression match is valid or not. Returning to the example in section 4.2.1, the regular expression that matches words with the infix `<in>` will match the word *hindi* "no, not" erroneously, producing the following feature dictionary entry:

(33) `hindi: { ... test_transformed: "<in>-hdi", test_root: "hdi", ...}`

Assuming the threshold value discussed above is greater than zero, this transformation will be ignored, and the original form *hindi* will be added to the word list for training the final Morfessor model.

The hypothesized roots are generated through a rather crude method at this time: the longest string in the list of segments that does not contain caret symbols (used to identify infix and reduplication morphemes) is selected as the root word. In the event that two or more morphemes of equal length are all the longest in the list, the last alphabetically is chosen. While manual inspection of

the data indicates that this process works a great deal of the time, it is still prone to obvious errors, especially given that some valid Tagalog affixes like *mang-* could reasonably be as long or longer than the verb root it is attached to. At a future date it is hoped that this process can be improved, but for the moment it is the best approximation available for an unsupervised system such as this one.

The final trained model and the feature dictionary, both of which are needed for segmentation and evaluation, can be returned as objects or written to `pickle` or `JSON` output files, respectively.

5.3 Segmentation and Evaluation

The final output from the previous section can be used to either segment an input list of words or can be evaluated using a gold standard segmentation file. The classes for performing those tasks, called `InfixerSegmenter` and `InfixerEvaluation` respectively, are available in in their own file, `eval_segment.py`. Both classes share a similar architecture for using the output from `InfixerModel`, which consists of a Morfessor model object and a feature dictionary (a nested `dict`).Both are initialized with those outputs and the list of affixes used to train that model[18].

The trained Morfessor model only has access to the word list it was trained with, meaning that any of the transformed forms used to train the final model cannot be connected to their original, non-transformed form. Instead, the feature dictionary associated with the model is used to check each input word and supply the correct transformed form, when applicable, to the Morfessor model for segmentation. If the word submitted for segmentation is out of vocabulary (OOV), then it is checked against the regular expression list and transformed in the manner described previously in section 5.2.1. This process has obvious downsides, namely that the possibility of false positive matches, and future versions of this program should include a mechanism for checking the validity of a transformation in a similar manner to the training process outlined in the previous section.

`InfixerSegmenter` has two public methods, called `segment_word` and `segment_file`, that

[18] At a future time, I hope to implement a way to serialize all of objects into a single loadable model. At this point in time, however, it made more sense to have the system working rather than focus on perfecting its API.

have uses that should be fairly self-evident. Also included is the method `segment_gold_standard`, which takes a gold standard segmentation file as input and returns that file with extra fields for each annotation: the word's OOV status, the form of the word sent to the model for segmentation (either the original or transformed form), and the segmentation output from the model. This file allows the user to inspect the entire segmentation or evaluation process, and has been invaluable for debugging this part of the program.

The `InfixerEvaluation` class's public methods are, in general, wrappers for the evaluation methods included with the Morfessor implementation, and can either be returned as a string or written to `stdout`. The Morfessor evaluation interface has settings for number of samples taken and sample size when evaluating a test set, and returns precision, recall, and F-score for a model given a particular gold standard annotation file.

5.4 Interface

A command-line interface for the functionality described above is in development, but all of the training and testing for section 6 below was done through scripting. The command-line interface, when complete, will mirror the Morfessor interface, with commands for training, segmenting, and evaluating models.

6 Evaluation

Each of the models generated by `Infixer` were evaluated using Morfessor 2.0's built-in evaluation tools, which provide precision, recall, and F-scores for a stable sample of a gold standard segmentation file (Smit et al. 2014). Virpioja et al. (2013) defines those evaluation metrics in the following ways:

(34) $$\text{precision} = \frac{\text{number of correct boundaries found}}{\text{total number of boundaries found}}$$

(35) $$\text{recall} = \frac{\text{number of correct boundaries found}}{\text{total number of correct boundaries}}$$

(36) $\text{F-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$

As the data below will demonstrate, these measures do not always perfectly capture the nuances of this particular system. Because of this, I have also included another simple metric with all of the evaluation results: percent of forms in a given test set that were segmented perfectly. I believe this metric is useful for demonstrating more clearly that the system described above is indeed able to correctly segment a greater portion of the vocabulary than an unmodified system would be.

6.1 Evaluation Data

Four different test sets were constructed to evaluate models trained with different combinations of the affixes discussed in section 4.2. The four sets were constructed separately in order to test the effectiveness of a particular affix or combination of affixes on a dataset that was guaranteed to contain relevant data.

All of the words in the test sets are verbs, as the affixes targeted in this project were all verbal affixes. The vast majority of the data used was gathered from SEAsite, a website maintained by the Center for Southeast Asian Studies at the University of Northern Illinois (Center for Southeast Asian Studies 2012). They provide a table containing hundreds of Tagalog verb roots with conjugations in both actor focus and object focus for three temporal aspects (perfective, imperfective, and contemplative). I segmented each of the verbs with reference to the various Tagalog affixes outlined in Schachter and Otanes (1972), a process that was greatly aided by the fact that each of the verb forms were grouped by affix class. Except for irregular forms, the vast majority of the forms segmented contained one of the morphemes under investigation, or a small number of concatenative morphemes commonly used with Tagalog verbs (*mag-*, *-in*, *-an*, etc.). The number of forms in each test set are presented in Table 1.

Each file contains one annotation per line in the format designated by Morfessor:

```
<compound> <constr1> <constr2>... <constrN>, <constr1>...<constrN>, ...
```

"Compound" corresponds to "word" and "constrN" to morpheme. Multiple segmentations are

Table 1: Count of Word Annotations per Test Set

Test Set	Affixes	Number of Word Annotations
1	infixation	449
2	infixation + reduplication	422
3	reduplication	428
4	none	443

allowed and are separated by a comma, though this feature was not used in the preparations of these sets.

6.2 Results by Test Set

The key for "Affixes" column in each of the tables in this section is as follows:

- *i*: infixation
- *d*: infixation within reduplication
- *r*: reduplication
- -: none

If more than one affix set was used to train a model, they are listed in the same order that they were input into the model; i.e., *dir* means infixation within reduplication, infixation alone, and then reduplication alone.

6.2.1 Test Set 1: Infixation

The results for the first test set, which consisted only of words showing infixation alone, are perhaps the most surprising and most difficult to explain. For both data sets, the best performing model when looking at F-score and precision was the model that only tested for reduplication (*r*). However, that same model reported 0% for percent correct. The same pattern holds true for the baseline model with no affixes. These results seem to indicate that the values returned on the other evaluation metrics are not especially strong indicators of model quality. My suspicion is that the

denominator in the precision formula (see (34), previously) is a factor: a trained model that tended toward marking fewer boundaries would have a smaller denominator and a larger score. Still, though, this result is curious, and brings into question the effectiveness of Morfessor's built-in evaluation tools

Other than that, models trained on the Babel corpus with the infixation affix set (*i*) tended to have stronger F-scores and recall scores, while on the Wikipedia corpus those results were less pronounced. In general, models trained on the Wikipedia corpus performed significantly better than those trained on the much smaller Babel corpus, though the latter's results are still promising.

Table 2: Results for Babel corpus trained models on test set 1

Affixes	**F-score**	**Precision**	**Recall**	**% Correct**
-	0.657	0.618	0.704	0%
d	0.549	0.508	0.599	0%
i	0.579	0.431	0.885	51.4%
r	**0.683**	**0.645**	0.727	0%
di	0.574	0.43	0.866	51.9%
dr	0.529	0.508	0.554	0%
ir	0.597	0.447	**0.897**	53.7%
dir	0.578	0.432	0.873	**54.3%**

Table 3: Results for Wikipedia corpus trained models on test set 1

Affixes	**F-score**	**Precision**	**Recall**	**% Correct**
-	0.744	0.795	0.7	0%
d	0.694	0.745	0.65	0%
i	0.632	0.502	**0.855**	74.2%
r	**0.758**	**0.822**	0.703	0%
di	0.622	0.491	0.849	72.4%
dr	0.716	0.771	0.669	0%
ir	0.622	0.49	0.853	**74.4%**
dir	0.62	0.487	**0.855**	74.2%

6.2.2 Test Set 2: Infixation within Reduplication

The results on the second test set, which consisted of words showing the infixation within reduplication pattern, were significantly weaker than the previous test set on most metrics. On both the

Babel and Wikipedia corpora (tables 4 and 5), models trained with the infixation within reduplication affixes fared better than those that were not; this is especially evident on the Wikipedia corpus. All of the models with affixes fared better than the baseline model with no affixes in nearly every category.

Table 4: Results for Babel corpus trained models on test set 2

Affixes	**F-score**	**Precision**	**Recall**	**% Correct**
-	0.0591	0.0596	0.059	0%
d	0.288	0.216	0.433	48.3%
i	0.202	0.163	0.265	0%
r	0.0656	0.065	0.0665	0%
di	0.271	0.202	0.413	49.3%
dr	**0.296**	**0.222**	**0.444**	49.8%
ir	0.238	0.197	0.3	0%
dir	0.277	0.208	0.417	**51.2%**

Table 5: Results for Wikipedia corpus trained models on test set 2

Affixes	**F-score**	**Precision**	**Recall**	**% Correct**
-	0.0471	0.0803	0.0338	0%
d	**0.306**	0.241	0.422	68.7%
i	0.113	0.0977	0.134	0%
r	0.0438	0.0742	0.0313	0%
di	0.288	0.223	0.408	68.2%
dr	0.311	**0.245**	**0.425**	**69.2%**
ir	0.183	0.161	0.213	0%
dir	0.295	0.231	0.412	68.2%

6.2.3 Test Set 3: Reduplication

The results for the third test set, which consisted of words containing reduplication of the initial syllable, showed a stark contrast in virtually all measures between models that were trained with reduplication affixes and those that were not. While this affix regular expression theoretically allows for a greater number of false positives than the others (see section 4.2), it seems that does not seem to be too much of a detriment to model quality. The baseline model's performance generally mirrored models trained with the *r* affixes.

Table 6: Results for Babel corpus trained models on test set 3

Affixes	F-score	Precision	Recall	% Correct
-	0.132	0.157	0.116	0%
d	0.105	0.12	0.0963	0%
i	0.129	0.176	0.108	0%
r	0.499	0.381	**0.725**	41.6%
di	0.113	0.114	0.112	0%
dr	**0.502**	**0.385**	0.724	43.7%
ir	0.483	0.368	0.7	44.9%
dir	0.488	0.38	0.685	**45.3%**

Table 7: Results for Wikipedia corpus trained models on test set 3

Affixes	F-score	Precision	Recall	% Correct
-	0.117	0.278	0.075	0%
d	0.112	0.255	0.0725	0%
i	0.115	0.314	0.0712	0%
r	0.511	0.41	0.679	62.6%
di	0.141	0.266	0.0988	0%
dr	0.515	0.41	**0.69**	62.4%
ir	**0.523**	**0.424**	0.681	**63.6%**
dir	0.517	0.415	0.686	**63.6%**

6.2.4 Test Set 4: Concatenative Morphology Only

The fourth test set, which only contains words that do not display any of the non-concatenative patterns being studied, was included to test whether or not the method outlined above would drastically affect the model's quality when segmenting other types of words and morphology.

On the Babel corpus, the baseline model performed better than all of the other models, though the differences were not substantial. On the Wikipedia corpus, the models with affixes started to perform better again. This seems to indicate that corpus size plays a factor, and a larger training corpus is needed to ensure that the trade-offs of the Infixer method do not cause a decrease in a model's performance. In general, there is little variation between the models within the two corpora, and the best performing models were rather similar to all of the others. For both corpora, inclusion of the *i* set of affixes seems to be somewhat of a predictor for slightly improved quality, but otherwise the results are broadly similar. It seems that these models are able to segment data

without any of the affixes present quite well.

Table 8: Results for Babel corpus trained models on test set 4

Affixes	F-score	Precision	Recall	% Correct
-	**0.721**	**0.685**	**0.761**	**51.9%**
d	0.658	0.619	0.705	48.8%
i	0.668	0.626	0.717	48.3%
r	0.644	0.598	0.701	49.0%
di	0.644	0.61	0.684	47.9%
dr	0.638	0.594	0.69	49.7%
ir	0.648	0.601	0.707	51.5%
dir	0.674	0.626	0.732	49.9%

Table 9: Results for Wikipedia corpus trained models on test set 4

Affixes	F-score	Precision	Recall	% Correct
-	0.722	0.73	0.716	69.3%
d	0.719	0.735	0.705	66.6%
i	**0.754**	**0.761**	**0.749**	**71.1%**
r	0.717	0.73	0.706	67.9%
di	0.747	0.75	0.745	70.7%
dr	0.69	0.697	0.684	64.1%
ir	0.721	0.723	0.72	67.9%
dir	0.727	0.735	0.72	69.1%

6.3 Discussion

Broadly, the results above seem to indicate that the modified Morfessor model outlined in this paper can have some success on test data containing non-concatenative morphological patterns. Because both corpora were highly idealized versions of a Tagalog data set (see section 3.1), it remains to be seen whether these same results can obtained on a "dirtier" training corpus, in particular one that better incorporates Tagalog speakers' tendency to intersperse a large amount of English and Spanish vocabulary.

The strongest differentiator for different models was training corpus size: models trained on the larger Wikipedia corpus outperformed models trained on the Babel corpus in every measure across every test set. This result unsurprising, as for many natural language processing tasks, a

relatively small corpus like Babel is always going to have relatively weak results. Other than that, most results seemed to closely track to whether the relevant affix was included in the training phase or not. Though individual F-scores varied widely, the percent correct scores stayed between 40-50% for the Babel corpus and between 60-75% on the Wikipedia corpus, excluding models that were not trained with an affix and scored 0% on those test sets.

One further type of evaluation not yet performed on this data would be a breakdown of how models performed on in-vocabulary words versus out-of-vocabulary. This could shed light on how much the multi-step training process outlined in section 4 actually affects performance on OOVs.

7 Future work

7.1 Testing on Other Languages

The most obvious place to test this system going forward would be on other languages that demonstrate similar types of morphological patterns, to ensure that the results obtained were not due to design choices that overfit the process to Tagalog. Good choices include other Austronesian languages, such as Cebuano/Visayan in the Philippines and Javanese in Indonesia. Both languages make use of infixation in a way similar to Tagalog (Bunye and Yap (1971), Horne (1961)), and both have datasets available from the IARPA Babel Program, which would allow straightforward comparison.

7.2 Generalization

The ultimate goal of the work outlined in this paper was to develop a tool that could be used for segmenting languages that exhibit morphological patterns that are rare in commonly researched, high-resource languages. In order to reach that goal, the software developed must be *generalizable* to other languages besides Tagalog while still maintaining a relatively good level of accuracy.

Key to that goal would be a defined language for affix and rewrite rule regular expressions.

A major shortcoming of the code as developed is that rewrite rules corresponding to a particular affix's regular expression are hard-coded into the `AffixFilter` class used to check each word for instances of an affix. A more general implementation could have a defined set of affix group naming patterns (like those 4.2) and use that to generate rewrite rules. For example, in the regular expressions for reduplication used in this project, a general rewrite rule such as "Move all groups where `P=REDUP` to beginning of string" could be used to help define a regular language for affix description and rewrite. For researchers in linguistics, a system that follows the system outlined in the Leipzig Glossing Rules would allow for quick translation of research glosses into a machine readable form that retains the nuance of the descriptive linguistics literature (Max Planck Institute for Evolutionary Anthropology 1982).

Furthermore, the process for determining whether a hypothesized written form of a word is valid needs to be improved significantly. As outlined in section 5.2.2, the system currently assigns the longest string in a list of segmented morpheme the role of "root". This process seems to be generally fine for Tagalog, as most of the verb roots in the gold standard segmentation sets are the longest morpheme in the verb. However, there is no reason why this should be the case cross-linguistically or across writing systems. Alphabetic writing systems, like the Latin and Cyrillic scripts, have a fairly-straightforward correspondence between individual phonemes and letters (or digraphs); the same does not hold for syllabic writing systems (e.g., Japanese hiragana, Devanagari[19]) or logographic systems (Chinese). A measurement like "string length" is imprecise for syllable-oriented systems like Devanagari, as vowel diacritics attached to consonants would not necessary register as lone character in all character encoding schemes, and it is not totally clear whether it even should. One possibility for dealing with this issue would be to define several different measures, and have them selectable at input time. Some examples might be: longest string, syllable count, or position number (if a root tended to be the first, second, last, etc., position in a word).

[19]The alphasyllabary used for languages like Hindi and Marathi.

7.3 Affix identification

Another future area of work in this area is quite far outside the scope of this project: the development a system that can, in an unsupervised manner, propose potential affixes.

One potential avenue for developing a system like this could follow the morphological chains model outlined in Narasimhan, Barzilay, and Jaakkola (2015). In that paper, the authors develop an unsupervised system for finding morphological chains, or sequences of full words that are created by the successive addition of other morphemes. For example, the sequence *nation* → *national* → *international* → *internationally* would be a morphological chain for the word *nation*. The key feature that differentiates their system from other segmenters is their use of semantic information by embedded word vectors generated using the program `word2vec` (Mikolov et al. 2013). The embedded word vectors aid in the word training process by helping to cluster semantically related words, which can then further be analyzed into chains.

If the information provided by semantic vectors works as well for a language like Tagalog as it does for some of their test languages (English and Turkish), a search process that uses a measure like minimum edit distance could potentially point out places where a non-concatenative morpheme has been inserted. However, their system has noted weaknesses in identifying morphological chains for Arabic when compared to the other languages tested. Much of Arabic's morphology is built around tri-literal roots with different vowel and affix patterns, making it highly non-concatenative. A system like this would need to be tested extensively to ensure viability.

References

Baumann, Peter and Janet Pierrehumbert (2014). "Using Resource-Rich Languages to Improve Morphological Analysis of Under-Resourced Languages." In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. Reykjavik, Iceland: European Language Resources Association, pp. 3355–3359.

Bird, Steven, Edward Loper, and Ewan Klein (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Blevins, Juliette (2012). "Infixation." In: *The Oxford Handbook of Derivation*. Ed. by S. Lieber and P. Stekauer. Oxford, pp. 1–39.

Bunye, Maria Victoria and Elsa Paula Yap (1971). *Cebuano for Beginners*. Honolulu, HI: University of Hawaii Press.

Center for Southeast Asian Studies (2012). *Tagalog Table of Verbs*. URL: http://www.seasite.niu.edu/tagalog/tagalog%7B%5C_%7Dverbs.htm (visited on 01/01/2016).

Clark, Ross (2009). "Austronesian Languages." In: *The World's Major Languages*. Ed. by Bernard Comrie. Second, pp. 781–790.

Creutz, Mathias and Krista Lagus (2002). "Unsupervised Discovery of Morphemes." In: *Proceedings of the ACL-02 workshop on ...* P. 10. DOI: 10.3115/1118647.1118650. arXiv: 0205057 [cs]. URL: http://dl.acm.org/citation.cfm?id=1118650$%5Cbackslash$nhttp://arxiv.org/abs/cs/0205057.

— (2005a). "Inducing the Morphological Lexicon of a Natural Language from Unannotated Text." In: *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, pp. 106–113.

— (2005b). "Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0." In: *Publications in Computer and Information Science, Report A81, Helsinki University of Technology*, pp. 1–27.

Haspelmath, Martin and Andrea D. Sims (2010). *Understanding Morphology*. 2nd ed. London, UK: Hodder Education. ISBN: 9780340950012.

Horne, Elinor Clark (1961). *Beginning Javanese*. New Haven, CT: Yale University Press.

Jurafsky, Daniel and James H Martin (2009). "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition." In: *Speech and Language Processing An Introduction to Natural Language Processing Computational Linguistics and Speech Recognition* 21, pp. –934. ISSN: 08912017. DOI: 10.1162/089120100750105975. URL: http://www.mitpressjournals.org/doi/pdf/10.1162/089120100750105975.

Kantrowitz, Mark and Bill Ross (1991). *Names Corpus, Version 1.3*. URL: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/.

Kurimo, Mikko et al. (2010). "Morpho Challenge 2005-2010: Evaluations and Results." In: *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology* July, pp. 87–95.

Lewis, M. Paul, Gary F. Simons, and Charles D. Fennig (2016). *Ethnologue: Languages of the World, Nineteenth edition*. Ed. by M. Paul Lewis, Gary F. Simons, and Charles D. Fennig. 19th ed. Dallas, Texas: SIL International. URL: http://www.ethnologue.com.

Manning, Christopher D et al. (2014). "The Stanford CoreNLP Natural Language Processing Toolkit." In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60. URL: http://aclweb.org/anthology/P14-5010.

Max Planck Institute for Evolutionary Anthropology (1982). *The Leipzig Glossing Rules: Conventions for interlinear morpheme-by-morpheme glosses*. URL: http://www.eva.mpg.de/lingua/resources/glossing-rules.php.

Microsoft Filipino Style Guide (2011). Tech. rep. Microsoft Corporation.

Mikolov, Tomas et al. (2013). "Distributed Representations of Words and Phrases and their Compositionality." In: *Advances in neural information processing systems*, pp. 3111–3119. arXiv: 1310.4546.

Narasimhan, Karthik, Regina Barzilay, and Tommi Jaakkola (2015). "An Unsupervised Method for Uncovering Morphological Chains." In: *Transactions of the Association for Computational Linguistics*. Vol. 3, pp. 157–167.

Rasooli, Mohammad Sadegh et al. (2014). "Unsupervised Morphology-Based Vocabulary Expansion." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Baltimore, MD: Association for Computational Linguistics, pp. 1349–1359.

Schachter, Paul and Fe T. Otanes (1972). *Tagalog Reference Grammar*. Berkeley, CA: University of California Press.

Schachter, Paul and Lawrence A. Reid (2009). "Tagalog." In: *The World's Major Languages*. Ed. by Bernard Comrie. Second, pp. 833–856.

Smit, P et al. (2014). "Morfessor 2.0: Toolkit for statistical morphological segmentation." In: *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden, pp. 21–24. ISBN: 9781937284756. URL: http://www.aclweb.org/anthology/E/E14/E14-2.pdf%7B%5C#%7Dpage=35.

Virpioja, Sami et al. (2013). *Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline*. Tech. rep., p. 38. URL: http://urn.fi/URN:ISBN:978-952-60-5501-5.

Yu, Alan C L (2004). "Infixing with a Vengeance: Pingding Mandarin infixation." In: *Journal of East Asian Linguistics* 13.1, pp. 39–58. ISSN: 09258558. DOI: 10.1023/B:JEAL.0000007241.87718.69.