Eliot Seo Shekhtman

LING 3324: Cayuga Language and Culture

December 10, 2020

Searching in a Polysynthetic Language

Introduction

Current search methods were created with western languages in mind, which tend to be fusional at most and primarily inflect the endings of words. This allows a first-to-last letter spelling of a word to reliably match with whatever word is desired in a dictionary - the first letter being the most important, with every additional letter narrowing down the options but being of less importance to the meaning of the word. In contrast, in Gayogo̱hó:nǫ' many words are the combination of a stem/root and a series of affixes attached to both the front and end of the word. This complicates the process of searching through conventional dictionaries and can hinder early learners of the language. In my paper, I will explore multiple possible solutions to this problem, and I will attempt a fair comparison between them for a final recommendation of methods, with a description of future studies that could be done on this topic. I will analyze two baseline exact match search algorithms, an edit distance algorithm to match with similar words but with not necessarily the same exact spellings, and a stemming algorithm to reduce both the input word and the words in the document to approximate stems for comparison.

In Gayogo̱hó:nǫ', words can be grouped into three distinct parts of speech - nouns, verbs, and particles. Nouns have a stem, a noun prefix, and a noun suffix (both of which are inflected on case and number); verbs have a pronominal prefix and a stem, the stem composed of a root with a possible incorporated noun or derivation suffix, with possible prepronominal prefixes. It's only particles that appear in a single form (Price 7-9). Each of these prefixes and suffixes gives

critical information about the word; however, inflections exponentially increase the amount of distinct space-delimited character strings upon which a new learner of Gayogo̱hó:nǫ' might inquire. In most Western languages, while varying levels of inflection exist, they typically modify the endings of words with the first few characters usually denoting the stem and giving the core meaning of the word, additional characters giving specifications - allowing that even in more inflected Western languages like Russian, not having all possible inflections in an online dictionary wouldn't prevent a learner from finding any word they desired by finding the stem and later deducing the ending based on a grammatical table. In contrast in Gayogo̱hó:nǫ', searching the first few characters of a verb might only give the grammatical gender of a noun it's being applied to, and searching the end of a verb isn't any more viable as the last few characters might only give the aspect of the verb.

Methods

Fair comparisons must be done between algorithms to ensure we don't report higher accuracies for one algorithm over another unfairly. As such, we attempt to standardize the method for finding accuracies. As a dictionary, we'll be using the first 99 pages of the Cayuga-English section of the English-Cayuga/Cayuga-English Dictionary (Froman 377-476). From here, 40 fully formed words and 10 stems will be sampled without replacement, removing each from the dictionary and adding them to a separate word bank, taking note of all related words present in the dictionary along with the stems of the 40 fully formed words. It must be noted that the PDF letter-recognition for Gayogo̱hó:nǫ' orthography is very imperfect, and many characters not found in English have other representations in the document's text - some examples being the character "ę" being transcribed as "~" or "t", and "ǫ" being transcribed as "q." After extensive examination, it was discovered that these patterns were semi-regular and as

such minor preprocessing of the data was done to mitigate any confounding effects. Each algorithm will score the entries of the dictionary based on their similarity to each of the words in the word bank, and output the top 10 closest entries. From there, the algorithms themselves will receive two general scores. The first will be a weighted score based on how many of the outputted words/stems are related to the original word by having the same or a similar stem out of the number of related words present in the dictionary, with examples with more than five other related words scoring full points if at least five relatives are found. Due to the unscientific nature of the scoring system, it is to serve more as a descriptive benchmark for how the algorithm is generally performing rather than a hard measurement, while more in-depth analyses delve more into the results and failings of each algorithm. This score will be referred to as the weighted score. The second scoring system will be whether a related word was found at all within the top 10, and will be called the one-found score.

As a baseline to illustrate the issue at hand, we'll use exact-match algorithms to emulate a typical search feature. The first algorithm will match by checking if the first letter of the input is equal to the first letter of an entry, then if the second letter of the input matches the second of the entry, etc until a word runs out of letters or a letter pair is found that doesn't match, returning the number of matched letters as the score. We'll call this the first-match algorithm. A second algorithm will match by checking letters one by one again; however, it may start searching for the input from any matching letter in the middle of the entry word and will output the highest number of letter matches of all possible starting positions. We'll call this the mid-match algorithm. This is done to simulate the act of typing in a word into a search bar letter by letter to see all the results that would be fetched regardless of location within each dictionary entry, and

also emulates the search function of other aspiring online dictionaries like the Saik'uz Carrier Dictionary (Poser).

In an attempt to improve upon these algorithms, we shall use two widely used methods to coerce similar words together: edit distance algorithms and automatic stemming algorithms. As described in *Speech and Language Processing* by Dan Jurafsky and James H. Martin, edit distance measures "how similar two strings are based on the number of edits (insertions, deletions, substitutions) it takes to change one string into the other" (Jurafsky 3). In so, we would hope that related words with the same stem but different affixes will require less "edits" to transform into one another than unrelated words, resulting in them having lower "distances" and being ranked as closer to one another. This algorithm may modify any part of the input word, and as such has the potential to relate words with completely different prefixes and suffixes, or even words where misspellings in the dictionary might have confounded the previous algorithms. These "edits" may be weighted as well - a fully optimized edit distance algorithm might have different costs associated with different substitutions from one letter to another letter, or from removing a letter that it knows is typically important. We'll use a standard edit distance algorithm called Levenshtein distance to compare words, acting as a proof-of-concept for a more in-depth edit distance algorithm. Levenshtein distance is calculated by adding a distance of 1 with every addition/deletion of a letter, and by adding 2 with every substitution (interpreted as an addition + deletion): as such, it's a very unbiased method of measuring distance, taking orthographic transcriptions at their raw value. Stemming algorithms use another method to compare words - instead of editing a word semi-naively letter by letter, they perform a prescribed set of conditional edits in an attempt to pull out the root of two words such that the supposed stems might be compared in the original words' stead (Jurafsky 3). Devising such an algorithm

requires a comprehensive understanding of Gayogo̲hó:nǫ' prefix and suffix structures; however, with some effort, it can produce great results, especially when tied together to another algorithm to compare the outputted stems.

Several challenges came with creating a concise stemmer. While the basic rules apply rather broadly and uniformly, subtle variations in glottal stop metathesis, vowel lengthening, and other such changes exist which confound a stemmer looking blindly for certain affixes, and there doesn't always exist full documentation on them. As such, some broad preprocessing had to be underdone before a stemmer could be made, removing most possible locations for this variation so that the stemmer could properly identify affixes despite irregularities. From the general category of nouns, there exists structural nouns, nouns with animate prefixes, unanalysable nouns, and verbal nouns. Unanalysable nouns don't require any stemming (they don't have discernable structure) and verbal nouns are structured as verbs - but structural nouns obey a small and strict set of rules. A structural noun will have a basic noun prefix of "o-" or "ga-" or just an "a" or a possessive noun prefix instead, and a noun suffix of "-a^{7}," "-e^{7}," "-e:," "-e," or "-ę^{7}" or a locative suffix instead (Price 11-19). Nouns with animate prefixes require some different parsing, having pronominal prefixes which varied on gender and number instead of possessive or locative forms for the most part; however, it is still rather regular, and not many examples of these words exist in the dictionary either way. Verbs pose a whole separate issue - there exists an enormous amount of prefixes and suffixes to take into account, there easily being over 50 pronominal prefixes alone, over a hundred distinct affixes being documented in total (Price 52-102). In the interest of keeping the stemmer concise only agent/patient pronominal prefixes, aspect suffixes, and modal prefixes were considered. A final issue was that a word shouldn't be affected by the stem modifications for a noun and a verb at the same time, but there

isn't a good way to distinguish accurately between a noun and a verb without increasing the complexity of the model. As such, a temporary fix was implemented, where verb-stemming rules simply weren't applied if a certain amount of modifications could be made by the noun-stemming rules already.

Experiments

As expected, the first-matching algorithm performs abysmally - out of the 50 selected words, it received an average weighted score of 9.6%, failing to fetch a single correctly related entry for 37/50 inputs (one-found score of 26%) including 9/10 of the stem inputs, due to the algorithm primarily matching confounding prefixes which don't hold the defining information content of each word, and due to the stems not appearing at the front of any of their related words. In addition, for only one word "gr̲ahe:t" (living tree) did the first-matching algorithm find the input's real stem, being "grahe'da" (tree). The input for which the algorithm returned the highest matching accuracy was "ohaha^{7}," meaning "a road" (Froman 458). Even though the root, "(h)ah(a)," was only found after a prefix, many of the provided related words appeared with the same "o-" prefix such as "oha:de^{7}" (an existing road) and "ohaha^{7}dihǫh" (the Milky Way). This frequency is due to "o-" being a neuter noun prefix (inanimate object), typically but not exclusively used with nouns that designate naturally-occurring things (Price 11). Indeed, it's only with input words and entry words both in their basic noun forms that there's a chance of related words being matched together using the first-match algorithm - something we want to improve on.

The mid-match algorithm demonstrated significant improvement but still experienced subpar scores - now receiving an average weighted score of 20.8%, and finding at least one correctly related word for 24/50 inputs, including 9 of the 10 input stems for a 48% total

one-found score. The same singular fully-formed input returned a related stem entry as for the first-matching algorithm. Once more, the issue lay in words not being presented in their basic noun forms, the complex prefixes foiling any attempts at matching since the algorithm tried to match the whole input word to each entry. This is further illustrated by the greater effectiveness of matching stem inputs rather than whole words, and by ohaha[7] still having the highest non-stem weighted score. The only stem to not find a related word was "dogę:," (be right, true, factual) and this was possibly due to a mistype in the dictionary as even though the stem doesn't betray a long o, each of the related words uses a long o in its place - an example being "gado:gę:" (a certain way; together; a certain thing S) (Froman 428). Overall, this method shows promise but it only reaches full effectiveness when solely the stem is inputted - or in other words, when the user already has baseline knowledge of Gayogo̱hó:nǫ'.

Levenshtein distance achieves an average weighted score of 36%; however, upon looking at the individual words, it may be seen that it has an average one-found score for fully formed words of 37/40, or 92.5%, while the average one-found score for stems is 2/10 or 20%. This is a remarkably higher one-found score for fully formed words than was observed for either baseline algorithm, with the superior mid-match algorithm only finding a related word for 15 of the 40 fully formed inputs, or 37.5%. On the other hand, its one-found score for stems was closer to that of first-match than the 90% of mid-match. From looking at the accuracy with fully formed words, we can first see that this de-emphasis on the first few letters of the input helps match fully formed input words much more accurately, just like the de-emphasis on the first few letters of the entries in the change from first-match to mid-match. Words with completely different prefixes now match to one another; however, as this process is naive, it doesn't discriminate between edits on the beginning and end of a word and edits in the middle. As a result, interestingly, the

previously high scoring ohaha[7] is one of the three fully formed words without a match found in their top 10 closest words according to the Levenshtein distance algorithm. Editing the stem of (h)ah(a) ends up producing many similar words according to Levenshtein distance which ends up drowning out the actually related words. A separate issue occurred when trying to match stem inputs to related entries - the process of adding letters added to the cost significantly, resulting in stems more easily matching with other stems or smaller unrelated words in general; however, 47.5% of fully formed inputs returned their respective stem in the top 10 closest entries, likely reflecting more variance in longer entries resulting in further distances between the fully formed inputs and similarly long unrelated entries than between the stem inputs and their similarly short unrelated entries. This shows evidence of promising results for edit distance algorithms, with room for improvement if a specialized algorithm taking advantage of Cayuga's unique phonological rules is created.

The stemming algorithm was tested with both Levenshtein edit distance as the ranking algorithm and with mid-match as the ranking algorithm, given mid match's previous effectiveness at matching known stems to related words and Levenshtein's previous effectiveness at matching fully formed words to their stems. Interestingly, Levenshtein edit distance applied to stemmed words had lower scores than Levenshtein edit distance applied to pre-stemmed words: it achieved an average weighted score of 36%, finding at least one related word for 33/40 fully formed input words and for 6/10 stems for a total average one-found score of 76%. This shows a drop in one-found score for fully-formed input words but an increase for input stems. The drop in one-found score for fully formed input words can be attributed to the same process by which Levenshtein edit distance only found related words for 2/10 input stems: editing the smaller stem produces many similar words according to Levenshtein distance which end up drowning out the

related words. On the other hand, the increase for input stems indicates that the stemmer is accurately getting the stems of the fully formed entries to be matched with the true stem. The average one-found score when solely matching fully-formed words to their true stems was 62.5%, once again showing the effectiveness of the stemmer. It was the mid-match algorithm that could fully make use of the stemming feature, exhibiting its ability to match stems once again. While its average weighted score was 19.6%, it found a related word for 90% of input words/stems. Examining the entries that each input connected to, most of the inputs only matched with their respective true stems; however, the one-found score of fully formed inputs when only checking for their respective true stem entries was 92.5%, showing evidence that this method is far more effective at matching to an input word's true stem than any other thus far. The input words for which mid-match couldn't find related entries were primarily verbs using affixes unattested in the stemming algorithm, indicating that further work could further increase the effectiveness of this approach.

Conclusions

Overall, this study explores and compares multiple methods for comparing Gayogo̱hó:nǫ' words and stems. Further work would have to be done to formalize these results and prove the statistical significance of these results; however, they show evidence for a series of conclusions about the effectiveness of word comparison algorithms for application in a searchable online Gayogo̱hó:nǫ' dictionary. First of all, it is shown that without prior knowledge of Gayogo̱hó:nǫ' word structure the first match and the mid-match methods of searching are ineffective at matching fully formed words to related words. Edit distance is shown to be a promising replacement for these algorithms in this respect, the naive Levenshtein distance algorithm finding related words for 92.5% of fully formed input words, with considerable room to improve

should a specialized algorithm be made. A basic stemmer is shown to boost mid-match performance to rival Levenshtein distance in finding related entries for fully formed inputs. Finally, it is shown that a basic stemmer combined with mid-match outpaces any other examined algorithm in identifying the true stems of fully formed input words. This stemmer also has considerable room for improvement, but it lies for future studies to show if an improved edit distance algorithm can work with an improved stemmer more effectively than the mid-match algorithm. This shows evidence that in an online searchable dictionary, a stemming algorithm combined with mid-match should be used to produce entries for fully formed input words, and a normal mid-match algorithm should be used to find related entries when a stem is inputted - the latter conclusion echoing the decision of the online *Saik'uz Carrier Dictionary*. Hopefully, this work may prove useful in future attempts to create such a searchable online dictionary to streamline the word-searching process for new learners of Gayogo̲hó:nǫ'.

References

Froman, F., & Dyck, C. J. (2014). English-Cayuga/Cayuga-English dictionary = Gayog_oho:nǫ/Hnyǫ ǫhneha: Wadęwęnaga:da:s Ohyad_ǫhsrǫ:dǫ. Toronto: Univ. of Toronto Press.

Jurafsky, D., & Martin, J. H. (2014). Regular Expressions, Text Normalization, and Edit Distance. In Speech and language processing (3rd ed., pp. 1-30). Harlow: Pearson.

Poser, B. (2020, September 20). Retrieved December 18, 2020, from https://www.billposer.org/SaikuzCarrierDictionary/

Price, C. (2011). The Ontario Curriculum, Grades 1 to 12: Native Languages: A Support Document for the Teaching of Language Patterns: Oneida, Cayuga and Mohawk: Resource Guide. Ontario: Ministry of Education.