Jacob Ballou

11/15/2020

Csci 390

Project Assignment 2

I am developing a universal inflection software that, when given a dictionary of lemmatized words and a set of rules for conjugation and declension, will correctly apply the proper rule to the proper word, conjugating or declining it as needed. In order to accomplish this goal, I needed to perform three tasks. I developed a model for the structure of each language’s ruleset, using nested dictionaries in order to use memory efficiently. I would rate the completion as 3/5. I wrote a function that, when provided with a lemmatized noun and its plurality, case, and gender, searches through the provided ruleset and retrieves the appropriate rule. I would rate the completion as 4.5/5. Finally, I began work on a function to take a given rule and apply it to a word, although I ran into some trouble getting it to work. I would rate the completion as 2/5

In “A Study of Efficiency of Modern Inflection and Lemmatization Software”, the authors came to the conclusion that writing an efficient inflector is more difficult than writing an efficient lemmatizer (Sychev, Gurtovoy, & Penskoy,2017). My experience writing the noun inflector seems to bear this out.

I decided to implement the ruleset for each language as a multidimensional dictionary- that is, a structure analogous to an array. Unlike arrays, multidimensional dictionaries are not implicitly supported by Python. I plan on implementing them by creating a normal dictionary and giving it other dictionaries as values.

This might seem inefficient compared to the alternative (using an array with each tense corresponding to an indices’ value). However, by describing a language’s ruleset with a dictionary, it becomes possible to omit cases and tenses that other languages might have. This is useful because languages often differ in the number and nature of their tenses, genders, rules for plurality, and so on. For example, English is a non-gendered language. Spanish has three genders, Masculine, Feminine, and Neutral. If I were to use the same sized array for every language’s ruleset, many languages would have a lot of empty space in their ruleset.

Figure 1

|  |  |  |  |
| --- | --- | --- | --- |
| Spanish | Masculine | Feminine | Neutral |
| Past |  |  |  |
| Present |  |  |  |
| Future |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| English | Masculine | Feminine | Neutral |
| Past |  |  |  |
| Present |  |  |  |
| Future |  |  |  |

Pictured above, in Figure 1: Two simplified inflection tables, implemented using arrays rather than nested dictionaries. Gray cells contain rules, but white cells do not. The English ruleset is mostly empty, taking up unnecessary space in memory. One could, of course, duplicate the rules across every cell in the same row. However, this would be needlessly redundant and it would thus still waste memory.

The problem becomes even worse when you consider that some languages have even more genders, corresponding to concepts such as animate vs inanimate. It would be a waste of space to create an array where the index has more values than the language has genders. It would also be impossible to tell what the value of the index corresponded to, unless a separate dictionary were created to translate between the index value and the corresponding gender, case, tense, etc. Since one such dictionary would be needed for each axis of the array, using arrays would be less efficient than just creating a nested dictionary, which can be directly accessed by just using the gender, tense, case, etc. as keys.

On a scale of one to five, I would say that I’m at a three with respect to creating the structure of a language ruleset. I have the nested dictionary structure I plan to use for each ruleset mapped out. However, I have not created an actual ruleset to test the code on, because I ran into a snag during preliminary testing, which has to be fixed in order to test a ruleset in the first place.

I am well on my way to finishing my modified version of the rule-based inflection detailed in “Open Source Tools for Morphology” (Straková, 2014). I created a notation for each rule in a ruleset that is relatively simple to understand. In this context, “rule” refers to a string in the bottommost layer of the nested “ruleset” dictionary. A given rule may have more than one line, allowing for complex operations. Each line begins with one of three keywords: “prefix”, “suffix”, or “replace”. The keyword is followed by an equal sign, and then a string or set of strings. The prefix and suffix keywords are fairly straightforward- the string following the equal sign is appended to the front or back of the word being inflected. The replace keyword is more complicated. The format is as follows:

replace=[substring 1],[substring 2],[integer or "all"]

The first substring, between the equal sign and the comma, represents a substring that’s being replaced with something else. The second substring replaces the first. The third substring, which may be an integer character/characters or “all”, represents the position of the substring being replaced. If the third substring is “3”, for example, the third instance of substring 1 is replaced with substring 2. If the third substring is “all”, all instances of substring 1 are replaced. The format supports backwards indexing- if the third substring is “-2”, the second-from-last instance of substring 1 is replaced. The format also supports removing substrings- leave substring 2 blank to remove an instance of substring 1.

While the article “Using the Levenshtein Edit Distance for Automatic Lemmatization: A Case Study for Modern Greek and English”, (Lyras, Sgarbas and Fakotakis, 2007) did not have a direct relation to this project, I did exploit one of the same things the article does: Inflecting a word is equivalent to applying Levenschtein edits. The rules I developed bear a strong resemblance to the rules of editing used in the Levenschtein edit distance. However, the prefix and suffix keywords both correspond to adding a letter, while the replace keyword corresponds to both replacing a letter and removing it. In addition, my rules work with substrings rather than single characters. As such, applying them would be faster than applying the Levenschtein rules, because the rule-applying function would not have to change one character at a time.

On a scale of one to five, I would say that I’m at a four and a half with respect to rule notation. I believe that I have succeeded in writing an easy-to-understand rule format, but I am having trouble with the function that interprets the rules, and this might be due to the rule format.

I wrote code to search through the rule dictionary and retrieve the appropriate rule for the provided noun case, gender, and number. However, the function that applies a rule to a noun does not seem to be working as intended. I am currently working on pinpointing the cause, but progress is slow and difficult. In main, I call the function on its own (with appropriate parameters) to test it. Normally, it would be called by the inflection function, which passes it the required rule from the language’s rule dictionary. When I test the rule-application function, it doesn’t return the inflected word the way it’s supposed to, or any word at all for that matter.

In respect to writing the function to apply a rule to a provided word, I’d say I’m at a two out of five. I wrote the function, but am having trouble making it work.

In order to test the basic concept of a universal, module-based Inflector, I am using JRR Tolkien’s Quenya, a detailed conlang (constructed language) with its own grammatical rules. It is an ideal conlang to use because it is highly inflectional and detailed, with ten noun cases and four grammatical numbers (most languages only have two numbers, singular and plural). Tolkien detailed the correct declension of several Quenya words in a letter to Dick Plotz, allowing for an easy test of the inflection software’s accuracy. Quenya resembles Finnish, Latin, and other highly inflectional languages. As such, two articles that should prove helpful are “Declension of Czech Noun Phrases”, by Zuzana Nevěřilová, and “Finnish Noun Inflection”, by Paul Kiparsky. The first details an algorithm to apply a grammatical case to a Czech noun- like this project, but more limited in scope. (Nevěřilová, Z. 2012) The second article is of more general use, because it demonstrates the rules that determine how inflection occurs- the “why” of inflection rather than the “how”. In Finnish, such rules include required patterns of stressed and unstressed syllables. (Kiparsky, P. 2003) Another article, “Predicting Declension Class from Form and Meaning”, is useful because of the information in section 2, Declension Classes in Language. In that section, the authors discuss the ways in which phonetic or phonological factors influence the written form of a declension. For example, most English words have the suffix -s when plural, but some have the suffix -es due to the difficulty of pronouncing several consonants in a row. (Williams, A., Pimentel, T., McCarthy, A. D., Blix, H., Chodroff, E., & Cotterell, R. 2020)

Modeling the problem seems straightforward- The rule retrieval takes place through a nested dictionary structure analogous to an array. Rule interpretation seems easy as well- I wrote a function to interpret the first part of a “rule” string as the type of change applied to a word, and the second part as the letters of the prefix, suffix, or substitution.

In order to determine my inflector’s accuracy, I will provide it with the nouns Tolkien used in the Plotz letter, then compare its output to the correct declensions Tolkien provided. I plan to use the cosine similarity, word similarity, and residual sum of squares to provide a meaningful way of determining the accuracy of my code. I will get the cosine-based word similarity between my program’s output and the correct declension to determine similarity for each word. If I then subtract each such similarity from the number one, I can use the results, which will measure word dissimilarity, like the error values in linear regression. I will then calculate the RSS using the word dissimilarities, and it would provide a measure of how accurate my program actually is.

Where is the inflected noun created by my program and is the inflected noun Tolkien used in the Plotz letter.

I began testing by trying the rule interpretation function on its own. I provided both the noun and a rule to apply to the noun. However, the rule interpretation function did not return anything, nor did it run at all. Due to the problem with the rule interpretation function, I was unable to retrieve any data from my program. As soon as I fix the problem, I will begin work on plotting the dissimilarity between Tolkien’s declensions and the ones created by my inflector, as well as calculating the RSS.

Works Cited:

Sychev, O., Gurtovoy, V., & Penskoy, N. (2017). A Study of Efficiency of Modern Inflection and Lemmatization Software. *Proceedings of the IV International Research Conference "Information Technologies in Science, Management, Social Sphere and Medicine"(ITSMSSM 2017)*. doi:10.2991/itsmssm-17.2017.30

Straková, J., Straka, M., &amp; Hajič, J. (2014). Open-Source Tools for Morphology, Lemmatization, POS ... Retrieved October 20, 2020, from <https://core.ac.uk/download/pdf/48906950.pdf>

D. P. Lyras, K. N. Sgarbas and N. D. Fakotakis, "Using the Levenshtein Edit Distance for Automatic Lemmatization: A Case Study for Modern Greek and English," 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), Patras, 2007, pp. 428-435, doi: 10.1109/ICTAI.2007.41.

Williams, A., Pimentel, T., McCarthy, A. D., Blix, H., Chodroff, E., & Cotterell, R. (2020). *Predicting Declension Class from Form and Meaning* (Unpublished doctoral dissertation). Cornell University. Retrieved November 15, 2020, from https://arxiv.org/abs/2005.00626

Nevěřilová, Z. (2012). *Declension of Czech Noun Phrases* (Unpublished doctoral dissertation). Masaryk University. Retrieved November 15, 2020, from https://www.researchgate.net/profile/Zuzana\_Neverilova/publication/301887499\_Declension\_of\_Czech\_Noun\_Phrases/links/572aee9d08aef5d48d30d55d/Declension-of-Czech-Noun-Phrases.pdf

Kiparsky, P. (2003). Finnish Noun Inflection. Retrieved November 15, 2020, from https://www.researchgate.net/profile/Paul\_Kiparsky/publication/265756560\_Finnish\_Noun\_Inflection/links/56bfaa1408ae44da37fa7053/Finnish-Noun-Inflection.pdf.