Jacob Ballou

10/20/2020

Csci 390

Project Assignment 1

I intend to develop a script that, given a word, translates it into a foreign language with properly conjugated gender, tense, plurality, and so on, so as to preserve the word’s original meaning and context. Furthermore, one of the key features of this script will be its modularity. The intent of this project is to make it possible to create a “universal” conjugation/declension program, able to conjugate or decline any word in any language as long as it is provided with the proper ruleset. In order to accomplish this goal, I will need to perform three tasks. The first task is to write a function that, when provided with a lemmatized word and its plurality, tense, and number, will search through a provided ruleset and apply the needed conjugation or declension. The second task is to create a simple, standardized, easy-to-read format to store the rules of conjugation and declension for a given language. The third task is to create a dictionary containing the lemmatized forms of the language’s words.

Since the conjugation/declension script is supposed to be universal, each language will need its own dictionary and ruleset. Because of this, it is important to minimize each dictionary’s memory usage while maximizing the script’s accuracy whenever possible. If thousands of different language dictionaries were being stored, unnecessary memory usage would become a significant problem.

The most relevant article I could find was “A Study of Efficiency of Modern Inflection and Lemmatization Software”. The article discusses the efficiency of inflection software, which seems to be the same thing as the “conjugation/declension” concept previously detailed. The authors came to the conclusion that writing an efficient inflector is more difficult than writing an efficient lemmatizer. They compared several open-source inflectors to each other, and an inflector that they wrote themselves, coming to the conclusion that phpMorphy, an inflector tool, was the most efficient. (Sychev, Gurtovoy, & Penskoy,2017)

The article is relevant because it provides examples of several methods of inflection. These include the inefficient method of creating a dictionary containing a word’s various conjugations, combining the dictionary method with predicting the word form, and creating a set of rules the software can use for inflection. The first approach is the most accurate, but is unnecessarily time and resource intensive. The second approach is used by phpMorphy, and uses machine learning to predict the inflection form of a word not in the dictionary. The third approach is the least accurate, because it is difficult to deal with irregularly inflected words. On the other hand, the third approach might be the least resource intensive, because a dictionary containing all of a given word’s forms is not needed. Clearly, a hybrid between the first and third approaches would be an improvement.

I found one lemmatization article which might be useful for writing an inflector. The first article, “Open-Source Tools for Morphology, Lemmatization, POS Tagging, and Named Entity Recognition”, involves lemmatizing a language such as Czech, with many forms of a word per lemma. The authors detail a morphological dictionary and tagger that builds a specialized tree structure for each word, so that each character of a word corresponds to a node. All the forms of a given word may be found by choosing a path from the root node to a given leaf node. (Straková, 2014) This structure has the advantage of making a word’s lemma easy to detect: In this model, a lemma is the sequence of character nodes that form a straight, non-branching line and begin at the word’s root node.

This method of determining a lemma from the list of all of a word’s forms may prove useful for determining a word’s inflections, by taking each word node that is not part of the lemma. However, this would only work for languages with suffix-based conjugation, since the method works by assuming each form of a word starts with the same sequence of letters. It should be possible to modify the lemma detection algorithm so that it detects all “straight line” sequences of character nodes, not just the ones starting at the root node. Then, the lemma would be the largest such sequence. By finding the non-lemma sections of the character tree using this modified algorithm, it should be possible to not only determine a word’s conjugated forms, but to determine if the word has prefixes, suffixes, or both.

The article is useful primarily because it provides a new way of finding lemmas. The most useful feature of this tree-based method is the memory it would save. With the traditional, dictionary-based inflection method, entering a word into a dictionary would also mean entering all of its inflections, each of similar length. Storing a character tree, on the other hand, would take up only slightly more memory than storing a single inflection. This way of saving memory would be far less effective with some irregularly inflected words, since irregular words often have vastly different inflected forms. For example, the Spanish verb “ir” has “vaya” as one of its conjugations. The infinitive and its inflected form share no letters, so using a character tree to store both “ir” and “vaya” would take up just as much memory as storing them in the traditional way. The rest of the “ir” inflections resemble each other more closely: “vaya” resembles “voy” resembles “vas”, meaning that adding the other inflections to the word tree wouldn’t take up much more memory. However, the fact remains that using a character tree to store a word’s inflections is less efficient when the inflections are irregular. Since I plan on using rule-based inflection for regular words, use of the tree structure for only the irregular words would be inefficient. Ultimately, “Open-Source Tools for Morphology” is not useful within the project’s scope.

Another article, “Using the Levenshtein Edit Distance for Automatic Lemmatization: A Case Study for Modern Greek and English”, provides another way to determine a word’s lemma with high accuracy. It is highly language-independent, achieving 95% accuracy when lemmatizing Greek and 96% accuracy when lemmatizing English. (Lyras, Sgarbas and Fakotakis, 2007)This would make it useful for lemmatizing with multiple languages. The authors use a dictionary-based algorithm to automatically find the lemmatized form of regular or slightly irregular words. The algorithm uses string similarity and a database of the most common suffixes used during inflection of a word. However, as with the previous article, this one is not useful within the project’s scope. In addition, I was unable to do further research on this article because I was only able to see the abstract. The full paper was locked behind a paywall.

I plan on using the third, rule-based method detailed in “Open Source Tools for Morphology” for my script, but to make it more modular. Instead of hardwiring the rules into a separate script for each language, I will write a generic script that I can feed a given set of rules as a parameter. The motivation for this approach is to allow a more customizable way of inflecting languages, while also providing more simplified declension/conjugation rules for ease of use. I will separate the rules for declension and conjugation, so that I can provide the conjugation module when translating a verb and the declension module when translating other parts of speech.

I plan to handle irregularly inflected words by creating a second dictionary containing their forms. While this is similar to the hybrid approach used by phpMorphy, it is still more efficient because the majority of words are not irregularly inflected. As such, even though some words will need all their forms provided, most of them will not, reducing the resource requirements compared to the completely dictionary-based approach. I might be able to save further space by storing irregular words as character trees, in the way described in “Open-Source Tools for Morphology”. However, doing so is unlikely to provide any significant benefit, so I will use the ordinary dictionary format to store them.

Fig. 1

Complete Dictionary Inflection

New Approach

N

N

Regularly inflected word

Regularly inflected word

A single lemmatized version of the word

P different forms of the word

M

Irregularly inflected word

Irregularly inflected word

M

P different forms of the word

P different forms of the word

Fig. 1 illustrates the difference between complete dictionary inflection and the new approach (rule-based inflection supplemented by a full list of irregular inflections). Since inflected languages usually have more regular than irregular words, we can assume that N > M by a significant amount. Thus, the new approach creates dictionaries approximately N entries long, whereas complete dictionary inflection creates dictionaries approximately (N\*P) entries long. Even if M remains proportional to N, the new approach is still an improvement over complete dictionary inflection.

The fact that the new approach is memory-efficient compared to complete dictionary inflection makes it ideal for use in a modular approach to translating, where a new dictionary must be created for each language. If each dictionary is around N entries long rather than (N\*P) entries long, significant amounts of memory are saved.

In conclusion, “A Study of Efficiency of Modern Inflection and Lemmatization Software” was helpful because it enabled me to decide which method of inflection to use. “Open-Source Tools for Morphology” was helpful because it showed a space-saving alternative to the classic dictionary inflection method, even if the tree-based method was ultimately useless for my purposes. “Using the Levenshtein Edit Distance”, however, was not helpful due to the details being locked behind a paywall.

I will create a modular inflector, suitable for translating any inflected language if provided with a dictionary of lemmatized words. I plan on using the rule-based method of inflection, and to supplement it with dictionary inflection for the irregular words only. This should ensure that my inflector is as accurate as a completely dictionary-based inflector, while taking up much less memory, which is important for a modular inflector designed to translate many languages. Hopefully, it will be easy to code a rule-based inflector in a modular way. I remain cautiously optimistic with regards to the project.

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.