ENHANCING AGILE WITH NATURAL LANGUAGE PROCESSING TECHNIQUES FOR ANCIENT GREEK INSCRIPTION LEMMATIZATION

Kavin Ramadoss
Sunset High School
MehtA+

Michael Wei
Ward Melville High School
MehtA+

Ari Philip
Monte Vista High School
MehtA+

July 27, 2023

ABSTRACT

Several lemmatizers have already been built for ancient Greek; however, none of them have dealt with inscription. AGILe is an ancient Greek inscription lemmatizer utilized by scholars across the globe; the steps taken to improve it is highlighted in this paper. Numerous techniques were employed to increase the accuracy, including Jaro-Winkler distance, normalization, and adjusting the method of edit distance to account for accents and capitalization. In the end, we achieved an increase in accuracy of 1.61 percent.

1 Introduction

In the constantly evolving, ChatGPT-led present day, where Large Language Models (LLMs) dominate the current landscape of machine learning, lemmatization is irreplaceable. With it, artificial intelligence can understand the context of and accurately respond to the millions of queries people ask daily. Due to its importance, lemmatization has been swiftly implemented into Natural Language Processing (NLP) models in most languages. The same is the case for Ancient Greek. Historians still uncover artifacts from thousands of years ago, but deciphering the ancient Greek written on these materials is often tedious. There have been great strides in ancient Greek lemmatization, but the vast majority of these models have been trained with data from literary text written in ancient Greek manuscripts. Therefore, there remains a gaping hole in the scholarly world for the lemmatization of Ancient Greek inscriptions. AGILe[1], a new ancient Greek inscription lemmatizer with an accuracy of 85.09 percent, intends to fill this hole. It is the most effective Ancient Greek inscription lemmatizer by a margin of 20 percent. A 'clean' dataset of literary texts is inadequate for inscriptions as the texts inscribed on materials such as ceramic and stone vary significantly from the dialects and morphology of the ancient Greek manuscripts. Hence, the dataset most appropriate for this project's purposes was *CGRN: A Collection of Greek Ritual Norms*[3]. Our goal was to improve the preexisting model of AGILe to increase its efficiency and accuracy, determining that AGILe had four significant issues.

- The edit distance function struggled as it was returning words unrelated to AGILe's predicted words. The edit distance function is how the model matches the predicted word with the most similar word from the lexicon through a numerical distance calculation.
- Like English, proper nouns in Ancient Greek are capitalized. Despite this, AGILe views lowercase and uppercase characters as completely different, often returning incorrect capitalization.
- In certain situations, AGILe fails to offer any prediction at all. In this case, nothing but a blank underscore is printed.
- Like the capitalization issue, AGILe lacks the ability to recognize the similarity between two characters that only differ by accents. We combat these issues with techniques and an original code of our own to enhance AGILe.

2 Related Work

The original authors of AGILe's innovative work opened the path for creating a powerful and practical NLP toolkit, enabling us to solve complex linguistic difficulties precisely. In addition, the mentorship offered by the AGILe team was critical in assisting us with the various techniques we investigated throughout our research. Their combined subject knowledge and expertise guided us toward novel techniques, resulting in well-developed additions to their code. Our effort to understand how different rules worked in Ancient Greek was greatly helped by James K. Tauber's[2] study on character encoding issues for classical languages. His thorough study provided solutions that allowed us to handle and analyze Ancient Greek material accurately and, in the end, strengthen our model with our knowledge.

3 Methodology

Many different approaches were made to enhance AGILe. However, many of them failed or required considerable additional time investment to yield substantial results.

3.1 Edit Distance

To improve the system of edit distance, we assigned weights to individual letters within words in hopes that producing the word's root would prove advantageous. For example, take the word 'hello': The edit distance would assign a weightage of 5 to the 'h,' a 4 to the 'e,' and a 3 to the first 'l,' as portrayed:

H	E	L	L	O
5	4	3	2	1

If 'hello' was being compared to 'jello' and 'hèllo,' 'jello' would have an error of 5, whereas 'hèllo' would have an error of 4 due to the location of the error.

Another method for edit distance involved a library called Jaro-Winkler[5] distance to replace the existing Levenshtein distance. The difference is that Jaro-Winkler focuses more on letter similarities than strings and order. Additionally, it prioritizes prefixes before suffixes, so it has the capability of lemmatizing more accurately. This means that, in theory, the Jaro-Winkler distance would be a more practical solution for similarity than the Levenshtein distance.

Gradient descent was also one of the techniques we attempted to implement. It is an algorithm that optimally finds the local minimum. This is valuable because it can find the best word to output efficiently.

3.2 Capitalization

As a result of error analysis on incorrectly predicted words from the project's source files, it became evident that the model regarded characters that only differed by capitalization as entirely different. One method implemented was training everything on a lowercase dataset. Afterward, we added the uppercase characters back when we evaluated them so that they would maintain their original capitalization. We also tried another strategy: removing all the capital letters entirely. This method was considered an excellent idea because it would eliminate the capital letters entirely, meaning there should be no room for error when it came to capitalization errors.

3.3 Character Encoding

Character encoding issues account for the most significant portion of the 14.91 percent of total incorrect predictions. The problem was that a letter with an accent is considered dramatically different by AGILe from the letter without the accent, making it impossible to distinguish between the two when performing lemmatization. For example, the letter έ is ε with an accent, yet to AGILe, they represent two different Unicode characters. One possible solution involved separating the words from their respective accents in the CGRN dataset, performing the required lemmatization, and reinserting said accents back into the words. As we discovered later, this was ineffective primarily because the accents on the lemmatized word could not be derived from the accents of the original word.

3.4 Missing Predictions

In the case of specific tokens, AGILe provides no prediction, returning nothing. We addressed this by checking if the predicted word was an empty string, and if so, the predicted variable became the original word. This way, a prediction would be offered for every token in our dataset, even if the output was not indeed a prediction made by AGILe.

4 Model

4.1 Data

Ancient Greek inscriptions are quite diverse in terms of spelling and dialect. The Collection of Greek Ritual Norms (CGRN) used, a collection of 225 Greek inscriptions on the topic of religious rituals spanning five centuries and all of ancient Greece, exemplifies these variances to decrease the bias involved with training models. It is quite comprehensive for the purposes of the lemmatizer. The dataset consists of 38,034 tokens; 25,229 were lemmatized by hand. 12,667 tokens were omitted as they were insignificant parts of speech, primarily articles and prepositions. This is the same data set initially used for AGILe. In our case, 25 new inscriptions were added to the training data prior to modifying AGILe.

4.2 Preprocessing

Preprocessing the data is crucial to improve the model's results. It is done by removing unnecessary characters and words in the data. For example, punctuation marks and special characters are removed in preprocessing. This aided the model in developing a more accurate prediction because some characters within the data hindered the model's accuracy, outputting a flawed prediction.

4.3 Stanza Library

Stanza[4], a natural language processing library, was used for the model. It is compatible with 70 languages, including ancient Greek. Stanza is run on a multi-layer deep learning neural network. Its functions include lemmatization, tokenization, and named entity recognition. The lemmatization capabilities of the model were used for this project. This natural language processing library was of the utmost importance to us as we were able to spend more time increasing accuracy as opposed to coding new lemmatization features.

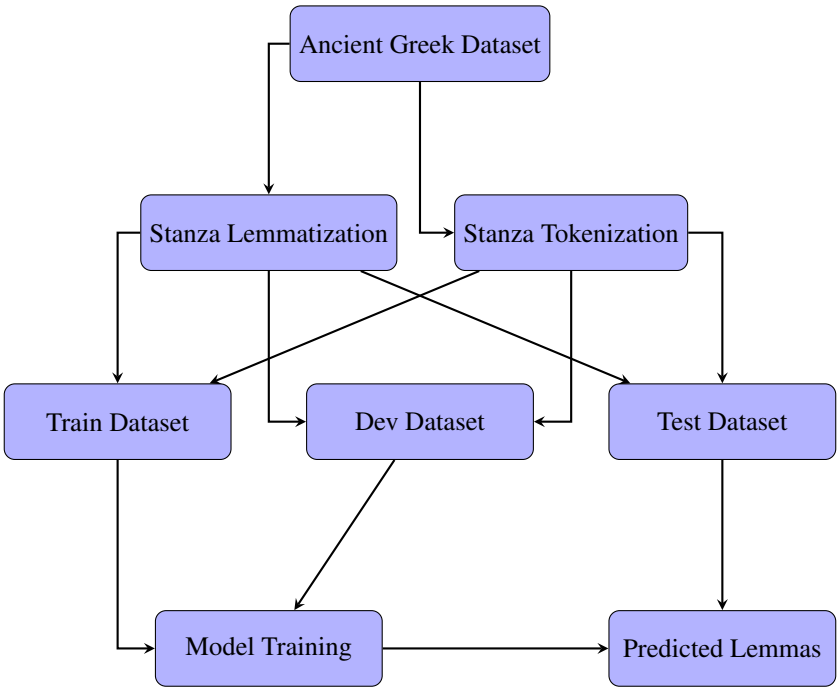

Figure 1: Illustrates how Stanza was used for our project

5 Results

We solved three of the four significant issues: character encoding, capitalization, and why no prediction was offered. The solutions for each involved careful corrections and debugging. For character encoding, the model was altered to recognize characters with different accents as the same character. For capitalization, we trained the model on a lowercase dataset. Once it was time to predict an output, the capitalized characters were added back. This caused a 0.3 percent increase, causing our accuracy to rise from 86.4 to around 86.7 percent. For the issue of an empty prediction, the original word was returned. However, we implemented the idea so that, in these instances, it would not be calculated as wrong or right. Another area for improvement with AGILe was that it occasionally failed to determine if a prediction was correct correctly. This issue was solved through normalization, as it is believed to be an error with the character encoding.

5.1 Diagrams

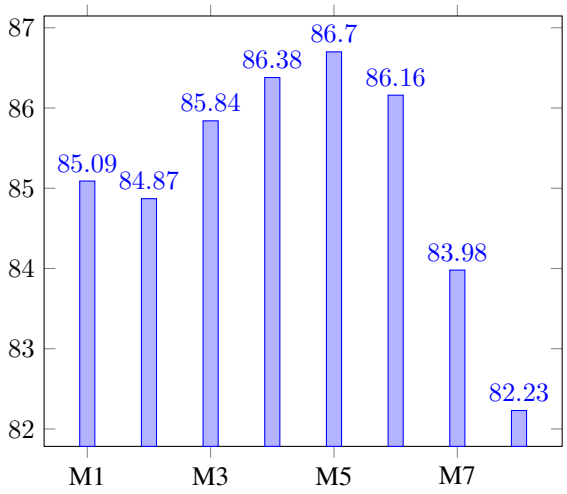

Figure 2: Accuracy vs. Methods

	Methods	Accuracy(%)
M1	Regular evaluate, regular agile	85.09
M2	Regular evaluate, accent removal	84.87
M3	Regular evaluate, accent removal, less speed	85.84
M4	Regular evaluate, scuffed accent-removal agile, less speed, capitalization solution	86.38
M5	Regular evaluate, accent removal, capitalization solution, and trained on lowercase dataset	86.70
M6	Regular evaluate, better accent-removal agile, less speed, capitalization solution	86.16
M7	Regular evaluate, weighted edit distance in agile	83.98
M8	regular evaluate, character encoding fix, Jaro-Winkler	82.23

Table 1: Methods and Accuracies

Within each of our diagrams is a depiction of each method attempted, as well as their corresponding accuracies. These graphical representations aid in identifying potential bottlenecks and assist us with improving on our thoughts and ideas.

6 Error Analysis

We ran into a multitude of errors while trying to run the code. These errors will be listed below for ease of access and inspiration. Strategies that appeared to be effective underachieved when implemented into AGILe. However, a few strategies did lead to the best solution achieving a high of 86.7 percent.

6.1 Edit Distance Issues

When we tried implementing the weighted edit distance, it reduced our accuracy by over 1 percent. This could have been executed more efficiently as the method seemed very viable on paper, but errors developed in the programming stage. Once we replaced the Levenshtein distance with the Jaro-Winkler distance, our accuracy decreased by over 1.5 percent. This was also rather perplexing because Jaro-Winkler is proven to be more useful at prioritizing a word's prefix, so in theory, it should have been more reasonable for lemmatization. However, issues likely arose in the programming stage as well. With more time, this implementation could be rediscovered. The last technique implemented for edit distance was gradient descent. This decreased the accuracy by around 0.5 percent. However, the idea was not pursued fully as it took around five hours to train every time, so time was spent solving other issues.

6.2 Capitalization Issues

One potential area for improvement with AGILe was that it viewed uppercase and lowercase characters as separate characters entirely, despite capitalization being the only difference. One approach that we tried failed, and that was removing capital letters entirely. It increased the accuracy by around 2 percent; however, after checking in with or graduate mentors, we were advised that removing the uppercase characters was not a solution because they were significant for distinguishing words.

6.3 Character Encoding Issues

We tried a few methods to fix character encoding issues. One seemingly viable method was removing the accents, which increased our accuracy by around 2 percent. However, we were informed that removing accents alters the meaning of the word. Another attempted method was to remove the accents altogether before lemmatization and add them back before the prediction was offered. However, it soon collapsed as there wasn't a simple solution to recovering accents from the words.

6.4 Windows Issues

Setting up AGILe came with numerous errors. It was likely a result of the Windows operating system as the team that originally made AGILe was on Linux. This generated many issues as Windows and Linux encode their characters differently. Therefore, the process of setting up the model took a while as these fixes were difficult to circumnavigate. Another major problem was that CLTK, the library used to lemmatize, only worked with Python versions 3.7,3.8, and 3.9. To combat this, we downgraded our Python versions and created environments. These issues should all be corrected in the new code.

7 Conclusion

In conclusion, enhancing AGILe has been both challenging and rewarding. We successfully tackled three pressing issues that were hampering the model's performance: character encoding issues, capitalization errors, and missing predictions. Through many hours of analysis and execution of different techniques, we corrected character encoding discrepancies, ensuring AGILe could adequately predict the differences between characters with and without accents. Ultimately, we accomplished the success of raising AGILe's accuracy by 1.61 percent. Although seemingly little, these implications are far-reaching, mainly when applied on a larger scale with abundant data. Scholars, historians, archaeologists, and linguists are now able to use AGILe more effectively. Our extensive time and effort of over 100 hours in 2 weeks dedicated to advancing AGILe have yielded significant results. AGILe is better than ever, and scholars from around the world stand to benefit.

8 Division of Labor

We divided the work as follows:

- During our time working with AGILe, Michael, Kavin, and Ari worked extensively on each of the four issues, all with varying success. Michael spent the majority of his time on capitalization and character encoding issues. Ari spent most of his time on character encoding and edit distance obstacles. Kavin spent his time on edit distance and missing prediction issues.
- All of the group members worked equally on the poster. Kavin and Michael made the flow chart and the tables. Ari and Kavin worked on picking out the color scheme. Ari spent most of his time writing the text.
- All three of us worked equally in generating the text in the paper.

9 Our code

Link to Google Drive

10 Acknowledgements

We want to acknowledge Ms. de Graaf and Ms. Stopponi for meeting with us and providing us with existing concerns as well as crucial advice. They highlighted various issues with AGILe's state for us to focus on. It has been a tremendous help to have them by our side over the course of this project, and we are very grateful to them for assisting us. We would also like to thank Ms. Haripriya for aiding us every step of the way. We would also like to thank Mr. Bhagirath for helping us make sense of the source code and pitching possible ideas for solutions when we needed them the most. Additionally, we want to thank Ms. Minnie and Mr. Isaac for helping us with constructive criticism and always being available for late-night office hours. Finally, much appreciation to Mr. Bos and Ms. Peels-Matthey for playing significant roles in the development of the original AGILe.

References

[1] Evelien de Graaf, Silvia Stopponi, Jasper K. Bos, Saskia Peels-Matthey, and Malvina Nissim. AGILe: The first lemmatizer for Ancient Greek inscriptions. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 5334–5344, Marseille, France, June 2022. European Language Resources Association.

[2] James K. Tauber. Character encoding of classical languages. In Monica Berti, editor, *Digital Classical Philology*, pages 137–158, Berlin, Boston, 2019. De Gruyter Saur.

[3] Jan-Mathiew Carbon, Saskia Peels-Matthey, and Vinciane Pirenne-Delforge. Collection of greek ritual norms (cgrn). 2017.

[4] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108, Online, July 2020. Association for Computational Linguistics.

[5] Srinivas Kulkarni. Jaro-winkler vs. levenshtein distance. In *Medium*, 2023.