DAAL 2024 @ Paratexts in Pre-Modern Writing Cultures

The First Workshop on Data-driven Approaches to Ancient Languages

(DAAL 2024)

Proceedings of the Workshop

Colin Swaelens, Maxime Deforche, Ilse De Vos,
Els Lefever (eds.)

June 27, 2024
Ghent, Belgium

Order copies of this proceedings from:

Language & Translation Technology Team (LT3)
Groot-Brittanniëlaan 45
9000 Ghent
Belgium
Tel: +32-9-33-11-941
https://lt3.ugent.be

ISBN 9789078848127

Development of Linguistic Annotation Toolkit for Classical Armenian in SpaCy, Stanza, and UDPipe

Lilit Kharatyan, Petr Kocharov
Julius-Maximilians-Universität Würzburg
Oswald-Külpe-Weg 84, D-97074 Würzburg
{lilit.kharatyan, petr.kocharov}@uni-wuerzburg.de

Abstract

This paper presents pioneering pipelines for the UD annotation of Classical Armenian developed within the three leading linguistic annotation frameworks - UDPipe and Stanza, and SpaCy. Classical Armenian is a low resourced ancient Indo-European language, and the development of an efficient open-access NLP toolkit for it constitutes a challenging and long awaited task. The presented pipelines are trained on the Armenian Gospels with a morphological and syntactic annotation following the Universal Dependencies guidelines. The pipelines are compared to each other for the accuracy of tokenisation, POS, morphological, and syntactic parsing. Through rigorous testing, Stanza emerges as the standout system, demonstrating superior overall performance, particularly for dependency parsing and morphological tagging. In contrast, while UDPipe shows strong potential with considerable improvements in its second iteration, it does not quite reach the benchmark set by Stanza. SpaCy, despite its wide usage in NLP applications, lags behind in this comparative study, highlighting areas for potential enhancement. The paper addresses major challenges of adjusting the three frameworks to the complexities of Classical Armenian. The findings stress the importance of evaluating multiple systems in order to identify the best available solution for assisted linguistic annotation.

Keywords: Classical Armenian, automatic linguistic annotation, Universal Dependencies

1. Introduction

Classical Armenian is the earliest attested variety of the Armenian language first put to record in the early 5th century after the invention of the Armenian alphabet. The Classical Armenian literature, spanning over fifteen centuries, is of paramount importance for many aspects of culture and history, and yet it remains under-resourced regarding linguistically annotated text corpora and NLP tools. In particular, up to now, there is no open access general-purpose toolkit for the linguistic annotation of Classical Armenian. The paper presents the comparative analysis of three pipelines developed for the SpaCy, Stanza, and UDPipe frameworks and trained on a Universal Dependencies treebank of the Classical Armenian Gospels as part of the CAVaL: Classical Armenian Valency Lexicon project[1]. Although each of the models has limitations related primarily to the small size of training dataset, they constitute an important step towards developing effective tools for the assisted morphological and syntactic annotation of larger corpora of Classical Armenian texts.

1.1. Existing NLP recourses

Important resources for the lemmatization and morphological annotation of Classical Armenian have been developed within the GRE*g*ORI[2] and Calfa[3] projects. The rule-based POS-tagger and morphologizer, based on the morphological dictionary of the GRE*g*ORI project, generate annotation with multiple possible analyses for a word form in context (Kindt and Kepeklian, 2022). The analyser accurately annotates morphologically unambiguous forms found in the dictionary but the output requires disambiguation of ambiguous forms and does not annotate unregistered ones. The morphological tags rely on the unique annotation scheme of the GRE*g*ORI project (Coulie et al., 2022).

A statistical POS-tagger and lemmatizer trained on a subcorpus of the GRE*g*ORI texts (67.039 tokens) has been developed in the Calfa project (Vidal-Gorène and Kindt, 2020). The training corpus includes post-classical texts, which may depart from the Classical Armenian grammar of the 5th century and include post-classical lexical items. The following results were obtained by the Calfa team on approximately 40% of the dataset, on which the models presented in this paper were trained (see 2.1): lemmatizer f1: 94.59%; POS-tagger f1: 93.27%; morphologizer f1: 97.94% (Vidal-Gorène, p.c.). This pipeline inherits the annotation scheme of the GRE*g*ORI project and, to the best of our knowledge, is currently not in open access.

Up to now no model for the syntactic annotation

[1] https://gepris.dfg.de/gepris/projekt/518003859

[2] https://www.gregoriproject.com

[3] https://calfa.fr

of Classical Armenian has been released.

The current situation with the NLP of Classical Armenian encourages to develop a general-purpose open-access pipeline with comparable or better performance than that of the aforementioned solutions, which would provide a standardized morphological and syntactic annotation of Classical Armenian literary texts.

2. Method

2.1. Dataset

The linguistic annotation pipelines discussed in the present paper have been trained on a treebank with the morphological and syntactic annotation of the Classical Armenian translation of the Gospels (ca 82K tokens) stored in the CoNLL-U format. The treebank results from a semi-automatic conversion of the PROIEL treebank of the Classical Armenian Gospels[4] (Haug and Jøhndal, 2008) to the Universal Dependencies (UD) annotation scheme and is released as part of Universal Dependencies, v2.14.[5] Although the annotation scheme of the Classical Armenian treebank takes into account the tagsets applied to the UD treebanks in Modern Eastern and Western Armenian (Yavrumyan and Danielyan, 2020), it diverges from them in many respects insofar as significant differences of grammar require. In that regard, models trained on the Modern Armenian treebanks are not suitable for the linguistic annotation of Classical Armenian.

The treebank is split as follows: training data: 62831 tokens; development data: 8658 tokens (Matthew 4, 5; Mark 4, 5; Luke 3, 4, 5; John 3, 4, 5); test data: 10509 tokens (Matthew 6, 7, 8; Mark 6, 7; Luke 6, 7, 8; John 6, 7). The data split follows the guideline of UD, which requires that corresponding sentences in datasets that constitute a multi-lingual parallel treebank (in this case, the Gospels) end up in the same part of the dataset (training/dev/test). The split is aligned with the Ancient Greek text of the Gospels.

Training the pipelines on the UD morphosyntactic tagsets allows to obtain tools for a standardized linguistic annotation compatible with a growing number of linguistic corpora of typologically diverse languages.

2.2. Model Architecture

To effectively establish automatic linguistic annotation pipelines for Classical Armenian, a comprehensive exploration of three principal methodologies has been undertaken: UDPipe[6], Stanza[7], and SpaCy[8]. This section is dedicated to elucidating the processes involved in training, deploying, and evaluating these models. By developing several pipelines, we have pursued a practical task of empirically identifying the best solution for the assisted UD annotation of Classical Armenian.

The parameter configuration proposed for the presented models results from an approach aimed at achieving optimal performance in processing Classical Armenian data, and adhering to the standards of transparency and replicability in research. The hyper-parameters for the models were determined through a combination of empirical tuning and systematic approaches. Initially, manual tuning was performed, starting with the parameters recommended by each of the systems (Stanza, SpaCy, UDPipe). The parameters such as the number of epochs, learning rate, batch size, number of layers, units per layer, and dropout rate have been iteratively adjusted based on observed improvements in validation metrics. This process was further informed by using a random search technique, which involved sampling random combinations from the parameter space to identify promising configurations. The models have been made available online[9].

2.2.1. UDPipe

While developing annotation pipelines for Classical Armenian, UDPipe was selected as the primary tool because of its proven efficacy in processing the UD annotation stored in CoNLL-U format, including tokenization, lemmatization, POS and morphological tagging, and dependency parsing. The pipelines were trained for two versions of UDPipe, 1 (Straka and Straková, 2017) and 2 (Straka et al., 2021).

2.2.2. UDPipe 1

Tokenizer: The tokenizer component of UDPipe 1 is built around a bi-directional LSTM (Long Short-Term Memory) neural network (Sepp and Jürgen, (1997). This architecture is employed to accurately delineate both token and sentence boundaries within a given text. The operational mechanism involves the classification of each character into one of three distinct categories: 'token boundary', 'sentence boundary follows', or 'no boundary'. Additionally, the tokenizer incorporates the *SpaceAfter=No* attribute from the MISC field of the

[4] https://github.com/proiel/proiel-treebank

[5] https://github.com/UniversalDependencies/UD_Classical_Armenian-CAVaL

[6] https://ufal.mff.cuni.cz/udpipe

[7] https://stanfordnlp.github.io/stanza/

[8] https://spacy.io/

[9] https://github.com/caval-repository/xcl_nlp

CoNLL-U dataset file. This feature allows to determine space characters and their function in the context of tokenization.

POS/Morhological Tagger and Lemmatizer: The tagger and lemmatizer for Classical Armenian are characterized by distinct but complementary functionalities. The tagger utilizes a guesser to generate various triplets of values from the fields for the universal part-of-speech tags (UPOS), language-specific part-of-speech tags (XPOS), morphological features (FEATS) in the CoNNL-U treebank, for each word. This guesser is supported by an averaged perceptron tagger, which is equipped with a fixed set of features for the disambiguation of the generated tags. Similarly, the lemmatizer operates with a guesser, producing (lemma rule, UPOS) pairs, where lemma rules are designed to modify prefixes and suffixes of a word for accurate lemma generation. This process is enhanced by considering both the suffix and prefix of words. Disambiguation in the lemmatizer, akin to the tagger, is executed by an averaged perceptron tagger.

Dependency Parser: The dependency parser integrates Parsito - a neural network-based, transition-oriented parser. Offering a suite of transition systems like the projective arc-standard, partially non-projective link2, and a fully non-projective swap system, it adeptly caters to varied syntactic structures. Key training features include embeddings for the FORM, UPOS, UFeats, and DEPREL fields of CoNNL-U. In our model, the training parameters for the aforementioned components, focusing on aspects such as guesser suffix rules and dictionary enrichment, have been meticulously configured to optimize their efficiency and precision in handling the linguistic nuances of Classical Armenian. For the UDPipe 1 model, training was specifically tailored to enhance its performance on tokenization, tagging, and parsing. Training of a tokenizer was conducted over 100 epochs at a batch size of 50, a learning rate of 0.005, and a dropout rate of 0.1. The tagging component featured two models, each configured to improve morphological analysis through guesser rules and dictionary enrichment, focusing on values of the LEMMA, XPOS, and FEAT fields. Parsing leveraged a projective transition system with embeddings for UPOS, FEAT, and FORM, executed over 40 iterations with a hidden layer of 200 and a batch size of 10. The learning rate started at 0.02, decreasing to 0.001, with L2 regularization set at 0.5 to ensure the model's generalizability.

2.2.3. UDPipe 2

Tokenizer : In the development of our UDPipe 2 pipeline, tokenization and sentence segmentation are handled using the methodology established by the baseline UDPipe 1 configuration. Specifically, a tokenizer is trained following the methodology outlined in section 2.2.2, and is integrated into the UDPipe 2 pipeline at the point of deployment. The primary distinction between the two iterations of the model resides in the adjustment of input segment size for the bi-directional GRU (Cho et al., 2014); whereas previously, the segment size was capped at 50 characters, it has been expanded to 200 characters. The selection of the optimal model is subsequently based on its performance metrics on the development dataset.

POS/Morhological Tagger and Lemmatizer: During the POS tagging phase, word embeddings undergo processing through a layered bi-directional LSTM architecture to derive contextualized embeddings. When multiple recurrent neural network (RNN) layers are employed, residual connections are implemented for layers beyond the initial one. For tag categories UPOS, XPOS and UFeats, a comprehensive dictionary is compiled, aggregating every distinct tag identified within the training corpus. However, it is important to note that our dataset does not include the XPOS field. Subsequently, a softmax classifier is employed to process these contextualized embeddings, assigning each to an appropriate class based on the pre-established tag dictionary. Given the inherent limitations of a single-layer softmax classifier, an additional dense layer equipped with tanh activation and a residual connection is introduced prior to the softmax classification stage, enhancing the model's ability to perform more complex non-linear transformations. Lemmatization is approached through their classification into specific lemma generation rules, regarded as an additional type of tag. Hence it is introduced as a fourth tag category alongside the ones mentioned above, employing a similar architectural framework for its processing.

Dependency Parser: The dependency parsing framework is predicated on a graph-based bi-affine attention parser architecture (Dozat et al., 2017). Initially, contextualized embeddings are generated by bi-directional RNNs, augmented with an artificial ROOT word at the sentence's outset. These embeddings undergo a non-linear transformation into arc-head and arc-dep representations, which are subsequently integrated through bi-affine attention to yield a distribution for each word. This distribution signifies the likelihood of every other word serving as its dependency head. An arborescence, or directed spanning tree with maximal probability, is derived utilizing the Chu-Liu/Edmonds algorithm (Chu and Liu, 1965; Jack, 1967). For the labelling of dependency arcs, a parallel process is enacted: contextualized embeddings are non-linearly mapped into rel-head and rel-dep representations and merged via bi-affine attention. This merger produces a probability distribution over potential

dependency labels for each dependency edge.

The training parameters for UDPipe 2 were defined to optimize the model for the unique linguistic features of Classical Armenian. Parameters included a batch size of 32, LSTM with a cell dimension of 512 across two RNN layers, and a specific dropout rate of 0.5 to prevent overfitting. The training employed an adaptive learning rate strategy, starting at 1e-3 for the initial 40 epochs and reducing to 1e-4 for the subsequent 20 epochs, coupled with a word dropout of 0.2 to enhance generalization.

Significantly, the model has been adapted to the absence of mBERT's (Devlin et al., 2019) precomputed contextualized embeddings, which are a default expectation in UDPipe 2. This adjustment, made to accommodate the absence of support for Classical Armenian in mBERT, led to a modification in the deployment script by UDPipe developers to bypass the computation of contextualized embeddings. While this may slightly compromise accuracy, it also enhances the model's speed, presenting an advantageous trade-off. This nuanced approach to model training and deployment reflects a tailored adaptation to the challenges posed by Classical Armenian, ensuring efficient and effective linguistic processing within the constraints of available resources. Moreover, this solution paves the way for other languages lacking mBERT support, as the updated script now provides an easily replicable model for bypassing contextualized embeddings. The only requirement for others facing similar deployment constraints is to exclude the embedding from the options of the trained model.

It is important to note that, UDPipe 2, designed exclusively for Python environments and currently supported only on Linux, positions itself as a tool for research purposes rather than a direct, user-friendly successor to UDPipe 1.

2.2.4. Stanza

Tokenizer: The tokenizer employed in this pipeline exemplifies a sophisticated approach to parsing and understanding text through a unified sequence tagging model. This model, designed to accurately identify token ends, sentence boundaries, and multi-word tokens (MWTs), employs a combination of bi-directional LSTMs (BiLSTMs) and 1-D convolutional networks (CNN) for processing text at the unit level, where units are defined as single characters or syllables in accordance with language-specific orthography. This intricate setup facilitates the hierarchical classification of text segments into one of five categories: end of tokens (EOT), end of sentences (EOS), multi-word tokens (MWT), multi-word ends of sentences (MWS), and others (OTHER), through the use of binary classifiers and a gating mechanism to effectively integrate token-level information. The integration of CNNs alongside BiLSTMs aims to enhance the model's capacity for capturing local unit patterns, akin to the function of a residual connection, thereby improving the precision of the tokenizer in distinguishing between complex linguistic structures.

Lemmatizer: Stanza's lemmatizer model employs a nuanced and layered approach to the process of lemmatization, integrating both dictionary-based and neural network methodologies to address the varied and complex nature of linguistic structures it encounters. At its core, the model utilizes a dual-dictionary strategy, where the primary dictionary operates based on a combination of a word and its UPOS tag to derive lemmas, taking advantage of the predictive power of UPOS tags to enhance lemmatization accuracy while maintaining case sensitivity. In instances where the primary dictionary does not yield results, the model resorts to a secondary, word-only dictionary, providing a robust fallback mechanism. For inputs that elude the coverage of these dictionaries, the model activates its neural component, which is designed to tackle more complex lemmatization challenges that dictionaries alone cannot resolve.

This neural mechanism is intricately designed, integrating an edit classifier and a sequence-to-sequence model to handle the nuanced adjustments required for accurate lemmatization. The edit classifier is engineered to manage rare or unusually long words efficiently. It leverages the concatenated final states of an encoder, processed through a dense layer with rectified linear unit (ReLU) activation, to categorize lemmas into three distinct types: those identical to the input, those that are simply lowercased versions of the input, and those that necessitate intricate adjustments via the sequence-to-sequence model. This classification process, determined during training, allows the system to judiciously decide when to engage the more computationally intensive sequence decoder during runtime, based on the guidance from the classifier.

POS/Morhological Tagger: The POS and the morphological tagging component of the pipeline employ a sophisticated architecture centred again around a highway BiLSTM network. This network processes input that combines three distinct types of embeddings: pre-trained word embeddings, trainable frequent word embeddings for terms appearing more than seven times in the training set, and, in general applications, character-level embeddings derived from a unidirectional LSTM over each word's characters. However, for this specific implementation, character-level embeddings were not utilized, primarily for computational efficiency purposes. To compensate for the lack of character-level embeddings we implemented word2vec vec-

tors trained on a comprehensive dataset of Classical Armenian texts[10,11] (81763 unique tokens), which significantly exceeds the pipeline training dataset. The model was trained using the Skip-gram algorithm with a context window of 5 words, a vector size of 100, and a minimum count threshold of 5, over 10 epochs.

Assigning POS tags is achieved by transforming the BiLSTM output for each word through a fully connected layer, followed by the application of an affine classifier to predict the POS tag. For XPOS tags and UFeats tags, a similar strategy is employed, but with a nuanced addition of a bi-affine classifier for XPOS, which incorporates both the state of the word's XPOS and an embedding for its UPOS tag, ensuring a harmonious relationship between the tagsets. This model is fine-tuned to minimize cross-entropy loss, aimed at capturing the diverse grammatical nuances.

Dependency Parser: The dependency parser employs a neural network architecture that integrates a highway BiLSTM to process inputs comprising pre-trained word embeddings, embeddings for frequent words and lemmas, character-level word embeddings (where available), as well as summed embeddings for XPOS/UPOS and UFeats tags. To predict unlabeled attachments, the parser utilizes a bi-affine transformation to score potential relationships between words and their heads, incorporating both edge-dependent and edge-head representations derived from the BiLSTM outputs. This method, while not explicitly accounting for the relative positions of heads and dependents, enables the model to implicitly learn such spatial relationships.

The parser also introduces mechanisms to explicitly consider the linear order and distance between words and their potential heads. By factoring in the sign and absolute difference in positions, and applying Bayes' rule under the assumption of conditional independence, it calculates the probability of a word's dependency on another, adjusting for language-specific syntactic tendencies. This calculation is further refined through deep bi-affine scorers for both linear order and distance, integrating the Cauchy distribution to model the likelihood of discrepancies in predicted arc lengths. This approach allows the parser to discourage inaccurately long or short predictions for the distance between words, enhancing its precision. This use of separate scorers for attachment and relational probabilities, alongside specialized training for each component, ensures that it not only predicts the presence of an edge but also its nature, thereby trying to achieve a more detailed and accurate parsing outcome.

For the initial iteration of the Stanza models for Classical Armenian, the training was executed using the default parameters provided by the Stanza framework[12]. This decision was made after observing that the results obtained were more than satisfactory for the scope of this project. Specifically, for tokenization, no external resources such as dictionaries were utilized, aligning with the approach to leverage innate model capabilities for linguistic processing. In contrast, word2vec vectors supplied for POS/morphological tagging were also used in dependency parsing, as required by Stanza, to enhance model performance, taking advantage of additional linguistic information embedded in these pre-trained vectors.

The dataset underwent thorough preprocessing to ensure its compatibility with the training requirements of each model component. This preparation included adjustments for multi-word tokens and sentence segmentation anomalies, tailored to the specific characteristics of Classical Armenian. The training process incorporated an early stopping mechanism to prevent overfitting, ensuring that each model component achieved optimal performance without unnecessary computational expenditure.

2.2.5. SpaCy

To explore the most effective solutions for the linguistic annotation of Classical Armenian, a decision was made to extend the pipeline investigation beyond UDpipe and Stanza, employing SpaCy for its renowned robustness and user-friendly interface.

For the SpaCy pipeline, a trainable lemmatizer, tagger, morphologizer, and parser have been selected. The models were subject to both combined and individual training and deployment. This strategy proved crucial in identifying the most effective and efficient means of processing Classical Armenian. It was observed during deployment that certain components, especially the parser and lemmatizer, demonstrated enhanced performance when operated independently. This observation underscored the necessity to take into account interactions among constituents of a pipeline with respect to the annotation tasks at hand. Additionally, for the training of the models, the word2vec vectors mentioned in the section 2.2.4 have been used.

Lemmatizer: The trainable lemmatizer of the SpaCy pipeline is intricately configured for optimal performance. It includes a Tok2Vec component for token vectorization, utilizing a MultiHashEmbed layer and a MaxoutWindowEncoder. The lemmatizer itself is structured to back off to orthographic forms, with a neural component for complex cases.

[10] https://bible.armeniancathedral.org/

[11] https://historians.armeniancathedral.org/

[12] https://github.com/stanfordnlp/stanza

Key training elements involve an Adam optimizer and a dynamic batching strategy using a compounding schedule.

POS/Morhological Tagger: Among the models tested in our SpaCy pipeline, the morphologizer and tagger demonstrated the most notable performance. The configuration of the tagger and morphologizer models has been aligned for efficient linguistic analysis. Both models utilize the *spacy.Tagger.v2* architecture to ensure consistency in their operation. They are integrated with the *spacy.Tok2VecListener.v1*, which allows them to utilize the vector representations from the shared Tok2Vec component. The Tok2Vec component averages token vectors, providing the necessary input for these models. For both the tagger and morphologizer, a label smoothing technique is incorporated to help in generalization and mitigate the risk of overfitting. Optimization for both components is managed using the Adam optimizer. Training parameters include dropout regularization and a dynamic batching strategy with a compounding schedule in order to optimize the learning process.

Dependency Parser: The parser component of the SpaCy pipeline is configured using the *spacy.TransitionBasedParser.v2* architecture. It is linked with the *spacy.Tok2VecListener.v1*, similar to other components in the pipeline, to utilize the token vectors generated by the shared Tok2Vec component. The parser model includes key parameters such as a hidden width of 128 and maxout pieces set to 3, for capturing complex syntactic relations. Similar, to the previous models, for efficient and effective training, Adam optimizer, with specified beta values and L2 regularization, has been employed. The dropout rate of 0.1 and a dynamic batching strategy, following a compounding schedule, are tailored to optimize the learning process.

3. Results

3.1. UDpipe

The UDPipe models demonstrates strong capabilities in tokenization and tagging, with especially high accuracy in identifying and classifying individual word tokens and their grammatical features. However, it encounters more challenges in multiword token recognition and sentence boundary detection, areas that could benefit from further refinement. In contrast to lemmatization, the accuracy of annotating syntactic dependencies is relatively low. The low performance of the dependency parser can be attributed to the moderate size of the dataset, which detains the model from capturing the intricacies of less frequent syntactic patterns.

The evaluation results of the trained UDPipe models 1 and 2 are presented in Table 1.

Tokenizer: The models achieve commendable results in tokenization and word segmentation. The uniformity in performance of this task across both models is attributed to the identical implementation of the tokenizer, as mentioned in the section 2.2.3. In our previous iterations, leveraging a smaller dataset (ca. 24.500 tokens) necessitated the implementation of rule-based pre-processing to augment tokenization precision and effectiveness in subsequent tasks. Remarkably, the subsequent models, trained on bigger datasets exceeding 50K tokens (including the currently best model trained on 62.831 tokens), exhibit the capability to autonomously perform this task. However, sentence segmentation still presents a challenge, evidenced by its comparatively modest results. This can indeed be linked to the peculiarities of the dataset, where the boundaries of sentences, segmented on syntactic principles, are not always formally marked. Despite these challenges, it is noteworthy that UDPipe iterations outperform other models in sentence segmentation, underscoring its relative strength in this domain.

POS-tagger, morphologizer, lemmatizer: The comparative analysis of the POS tagging, morphological tagging, and lemmatization performance between UDPipe 1 and UDPipe 2 reveals noteworthy distinctions in their efficacy. UDPipe 2 demonstrates a consistent improvement across all metrics, indicating a refined understanding and processing of grammatical features. This enhancement is particularly significant in the realm of UPOS tagging and lemmatization. The results for these two tasks are presumably superior also to those reported for the RNN pipeline of the Calfa project mentioned in Section 1.1 above. However, it is essential to note that direct comparison may not be entirely fair due to the disparity in training datasets.

The incremental advancements in UFeats tagging and the composite metric of AllTags further underscore the sophistication of UDPipe 2 in handling complex linguistic patterns. The aforementioned morphology results of the Calfa project (97.94%) appear to outperform both UDPipe iterations. Despite the observable improvements, the differences between UDPipe 2 and UDPipe 1, while statistically significant, do not overwhelmingly favour one model over the other across all tagging and lemmatization tasks. The choice between the two versions may thus hinge on specific use case requirements, computational constraints, or the need for backward compatibility. Training UDPipe 2 demands significantly more computational power and a higher level of technical skillset, making it a more resource-demanding option. Conversely, UDPipe 1 is easy to use and has a more straightforward setup process. However, for applications demanding the utmost accuracy, UDPipe 2 offers a tangible advantage.

Metric	UDPipe 1			UDPipe 2			Stanza
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	F1 Score
Tokens	98.82%	98.52%	98.67%	98.82%	98.52%	98.67%	**99.06%**
Sentences	90.89%	95.40%	**93.09%**	90.89%	95.40%	**93.09%**	79.51%
Words	98.83%	98.53%	98.68%	98.83%	98.53%	98.68%	**99.07%**
UPOS	96.10%	95.81%	95.95%	97.01%	96.71%	96.86%	**98.25%**
UFeats	92.69%	92.41%	92.55%	94.46%	94.18%	94.32%	**95.72%**
AllTags	91.45%	91.18%	91.31%	93.61%	93.32%	93.47%	**95.02%**
Lemmas	95.89%	95.59%	95.74%	97.26%	96.96%	97.11%	**98.67%**
UAS	82.03%	81.78%	81.90%	87.23%	86.96%	87.09%	**90.97%**
LAS	78.13%	77.89%	78.01%	83.79%	83.54%	83.66%	**87.73%**
CLAS	71.91%	71.16%	71.54%	79.74%	79.03%	79.38%	**83.20%**
MLAS	64.14%	63.48%	63.81%	72.29%	71.64%	71.96%	**80.84%**
BLEX	69.24%	68.53%	68.88%	77.79%	77.09%	77.44%	**83.20%**

Table 1: Evaluation Results of UDPipe 1, UDPipe 2, and Stanza Models

Dependency Parser: The analysis of dependency parsing results from UDPipe 1 and UDPipe 2 offers a clear illustration of the advancements made in parsing capabilities between the two versions. UDPipe 2 shows a substantial improvement across all metrics of dependency parsing, reflecting a deeper and more accurate understanding of syntactic relationships within sentences.

The parsing scores, while improved, remain lower compared to other evaluated categories. The result is clearly influenced by the limited size or diversity of the training dataset, which may not encompass the full range of syntactic constructions of Classical Armenian with sufficient frequency. However, this first open-access syntactic parser of Classical Armenian is an important step forward in the development of NLP tools for that language.

3.2. Stanza

The performance of the Stanza toolkit, as indicated by the evaluation results, is commendably strong, positioning it well against UDPipe. Stanza exhibits exceptional proficiency in handling a broad spectrum of the tasks at hand, from basic tokenization to the more complex layers of parsing and lemmatization.

Tokenizer and Lematizer: Stanza's performance in tokenization and word accuracy stands out with subtle yet notable distinctions. Unlike UDPipe 1 and 2, which maintain a consistent and high level of accuracy across tokens and words, Stanza edges forward with marginally superior token and word recognition capabilities. However, its performance in sentence segmentation significantly trails behind UDPipe (almost 13.58%), marking a distinct area for improvement.

Similar to the tokenizer, Stanza's lemmatizer significantly enhances its utility in linguistic processing. With a lemmatization accuracy of 98.67%, Stanza surpasses both versions of the UDPipe models. The synergy between Stanza's tokenization and lemmatization capabilities suggests that its advanced handling of tokens directly contributes to its exceptional performance in deriving lemmas. This interplay highlights the importance of robust tokenization as a foundation for effective lemmatization, reinforcing Stanza's superiority in addressing complex linguistic tasks.

POS/Morphological Tagger: Stanza stands out in the domain of POS and morphological tagging as well, offering superior performance in all positions. While Stanza exhibits commendable results, showing a deep understanding of linguistic nuances, its performance, although impressive, does not significantly surpass that of UDPipe 2. The gap between Stanza and UDPipe in this aspect is very narrow. While Stanza showcases slightly advanced capabilities in this task, its computational efficiency presents a consideration worth noting. Unlike UDPipe, which trains all components concurrently, Stanza adopts a sequential approach, dedicating extensive computational resources and time. Consequently, UDPipe might offer a more pragmatic choice despite its slightly lower performance metrics.

Dependency Parsing: In the realm of dependency parsing, Stanza sets a new benchmark for precision and depth in linguistic modeling. With UAS and LAS towering at 90.97% and 87.73% respectively, Stanza not only eclipses the performance of both UDPipe iterations but also significantly distances itself from SpaCy, showing its capability to discern and accurately label syntactic relationships within the text. More than the numbers, Stanza's mastery of CLAS, MLAS, and BLEX underscores its profound understanding of the complex interplay between morphological features and their syntactic functions, a testament to its advanced parsing algorithms that intricately connect contextual cues and linguistic rules.

While UDPipe provides robust baseline models,

Tokenizer & Lemmatizer		Tagger & Morphologizer		Dependency Parser	
Token Acc	97.98%	**Morphology Acc**	73.57%	**UAS**	62.80%
Token P	97.96%	**Morph micro_P**	91.07%	**LAS**	51.94%
Token R	81.84%	**Morph micro_R**	84.66%	**Sent P**	66.47%
Token F1	89.13%	**Morph micro_F1**	87.75%	**Sent R**	83.27%
Lemma Acc	92.42%	**POS Acc**	81.86%	**Sent F**	73.74%

Table 2: Evaluation Results of SpaCy Models

with its latest iteration showing commendable improvements, this stark disparity in performance between Stanza and its counterparts - particularly the sophisticated handling of complex syntactic structures and linguistic phenomena - suggests an underlying architecture that prioritizes depth of linguistic analysis over mere surface-level parsing. This comparative evaluation suggests that while Stanza demands more in terms of computational resources, its accuracy in parsing justifies this investment for cases where linguistic precision is paramount.

3.3. SpaCy

Tokenizer and Lemmatizer: In analyzing the performance of the SpaCy tokenizer and lemmatizer (Table 2) compared to that of UDPipe, both trained and tested on the same dataset, several key observations emerge.

The SpaCy tokenizer demonstrates a particular balance in precision and recall, highlighting its effectiveness in accurately identifying token boundaries while also maintaining a reasonable coverage over the entire dataset. However, when juxtaposed with tokenizers of UDPipe and Stanza, which exhibits notably higher accuracy, it becomes evident that SpaCy's tokenizer may not be as finely tuned for the specific linguistic characteristics of the dataset.

In a focused analysis of the lemmatization results, the SpaCy lemmatizer, as evidenced by its performance metrics, demonstrates a moderate level of proficiency. Given the moderate performance of this component, it is pertinent to consider the use of a lookup file for lemmatization.

It is important to note the advantage of SpaCy in offering a lookup-based lemmatizer. This approach, which relies on a pre-compiled dictionary of word forms aligned with the training dataset, is expected to yield near-optimal accuracy in lemmatization tasks. Neenless to say, the efficiency of this solution entirely depends on the quality of the dictionary.

Tagger and Morphologizer: The SpaCy morphologizer's performance, marked by a POS accuracy of 73.18% and a morphological accuracy of 75.79%, indicates a reasonable capability in identifying both POS tags and morphological features. However, a critical comparison with UDPipe 1, 2 and Stanza, which achieve higher accuracy in both UPOS and FEATS, suggests that SpaCy's model, while functional, has a lower performance in these specific areas.

The SpaCy morphologizer exhibits high precision in accurately identifying morphological features when detections are made, but its significantly lower recall suggests a challenge in consistently recognizing all pertinent morphological features in the data, leading to a precision-over-recall imbalance in its performance.

Dependency parser: The performance of the parser, as indicated by its UAS and LAS metrics, suggests a notable gap in its ability to consistently and accurately handle dependency parsing. While the parser is relatively adept at identifying syntactic dependencies, it struggles more with accurately tagging these dependencies, especially for less frequent types or types attested in complex syntactic constructions. This uneven performance across dependency types suggests that the model might benefit from more diverse training data.

4. Conclusions

The present paper evaluates three pipelines of automatic linguistic annotation developed for Classical Armenian within the UDPipe, Stanza, and SpaCy frameworks. Even though the compared models were trained on a rather limited corpus (ca. 63K tokens) they show good results and potential for further improvement by increasing the size and genre diversity of the training dataset.

The comparative study of these models demonstrates the potential for significant advancements in linguistic annotation. It highlights the critical role of dataset size, the strategic use of embeddings (or effectively bypassing them for languages with constraint training datasets lacking mBERT support in the case of UDPipe 2), and the nuanced decision-making required in selecting the most suitable framework for specific linguistic tasks. Evaluation of the results shows that the UDPipe 2 and Stanza models by far outperform the SpaCy model, and are superior or comparable to the previously developed morphological analyzer of Classical Armenian in coping with various annotation tasks. With that Stanza shows overall better performance than UDPipe 2. These results are achieved by a meticulous empirical study on training parameters

for Classical Armenian, and required customization of training and deployment in order to adjust the frameworks to an under-resourced language.

5. Acknowledgments

The described annotation models have been developed as part of the "CAVaL: Classical Armenian Valency Lexicon" project funded by the Deutsche Forschungsgemeinschaft (DFG, 518003859). We thank the developers of Stanza, SpaCy and UDPipe for their indispensable help with the customization of the models, and to Chahan Vidal-Gorène for his valuable comments on the earlier version of the paper and comparative data on the efficiency of the linguistic annotator developed on the Calfa project. We acknowledge the permission of the Arak29 Charitable Foundation (`https://arak29.org`) for non-commercial use of their digital editions of Classical Armenian texts for training of the word2vec vectors used in the presented Stanza and SpaCy annotators.

6. Bibliographical References

References

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Bernard Coulie, Bastien Kindt, Gabriel Kepeklian, and Emmanuel Van Elverdinghe. 2022. Étiquettes morphosyntaxiques et flexionnelles pour le traitement automatique de l'arménien ancien. *Le Muséon*, 135(1-2):209–241.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.

Dag T.T. Haug and Marius Jøhndal. 2008. Creating a parallel treebank of the old indo-european bible translations. In *Proceedings of the second workshop on language technology for cultural heritage data (LaTeCH 2008)*, pages 27–34.

Edmonds et al. Jack. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.

Bastien Kindt and Gabriel Kepeklian. 2022. Analyse automatique de l'ancien arménien. évaluation d'une méthode hybride «dictionnaire» et «réseau de neurones» sur un extrait de l'adversus haereses d'irénée de lyon. In *Proceedings of the Workshop on Processing Language Variation: Digital Armenian (DigitAm) within the 13th Language Resources and Evaluation Conference*, pages 13–20.

Hochreiter Sepp and Schmidhuber Jürgen. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Milan Straka, Jakub Náplava, and Jana Straková. 2021. Character transformations for non-autoregressive GEC tagging. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 417–422, Online. Association for Computational Linguistics.

Milan Straka and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Chahan Vidal-Gorène and Bastien Kindt. 2020. Lemmatization and pos-tagging process by using joint learning approach. experimental results on classical armenian, old georgian, and syriac. In *Proceedings of LT4HALA 2020-1st Workshop on Language Technologies for Historical and Ancient Languages*, pages 22–27.

Marat Yavrumyan and Anna Danielyan. 2020. Universal dependencies and the armenian treebank. *Herald of the Social Sciences*, 2:231–244.