Elaboration of a practical lemmatiser for Latin using Artificial Intelligence

Philippe Verkerk

Elaboration of a practical lemmatiser for Latin using Artificial Intelligence

Philippe Verkerk

CNRS, UMR 8523 – PhLAM
Laboratoire de Physique des Lasers Atomes et Molécules,
Université de Lille,
Lille, France

Philippe.Verkerk@univ-lille.fr

ABSTRACT: I present here a plug-and-play lemmatiser that converts a plain text into an annotated file in LASLA's standard (APN-file). It is a hybrid code that involves a rule-based decomposition of the forms and a AI-based disambiguation. The AI part derives from Latin-BERT, trained on purpose with the texts annotated at the LASLA. This lemmatiser has been tested on one of the Cicero's texts annotated by the LASLA and its overall accuracy is 97%, with sub-tasks' accuracies ranging from 98.5% to 99.4%. It will be soon available for on-line lemmatisation on the Hyperbase web-site.

RÉSUMÉ : Je présente ici un lemmatiseur prêt à l'emploi pour convertir un fichier texte en un fichier annoté aux standards du LASLA (fichier APN). Il s'agit d'un code hybrid qui associe une décomposition des formes suivant les régles de flexion et une désambiguïsation reposant sur l'intelligence artificielle. La partie IA dérive de Latin-BERT et a été entraînée spécifiquement avec les textes annotés au LASLA. Ce lemmatiseur a été testé sur l'un des textes de Cidéron annotés au LASLA et son efficacité globale est de 97%, avec une précision variant de 98,5% à 99,4% pour chacune des sous-tâches. Il devrait bientôt être disponible pour effectuer une lemmatisation en ligne sur le site web d'Hyperbase.

Introduction

The production of fully annotated corpora in Latin is an on-going task for philologists. It is a necessary step for several further tasks. Unfortunately, it is a time-consuming task and several standards are available which are not necessarily compatible. The use of a computer can help, but the result is not fully reliable, which implies that a philologist has to check and correct the computer's output[1]. To my knowledge, there is no evaluation of the accuracy that is really needed for the various tasks, so that the only goal that can be expressed is perfection. But, for instance, statistics performed on annotated texts introduce some error-bars by their own processes, so probably some errors in the annotated files would be tolerable. The consequence of this unreachable goal is that people organise evaluations of the available systems to stimulate an upward evolution[2] .

1 Philippe VERKERK, Yves OUVRARD, Margherita FANTOLI and Dominique LONGRÉE. « L.A.S.L.A. and Collatinus: A convergence in Lexica. » *Studi e Saggi Linguistici,* LVIII (1): 95-120 https://www.studiesaggilinguistici.it/index.php/ssl/article/view/275.

2 Rachele SPRUGNOLI, Marco PASSAROTTI, Flavio Massimiliano CECCHINI, Margherita FANTOLI, and Giovanni MORETTI. « Overview of the EvaLatin 2022 evaluation campaign. » In *Proceedings of second Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA 2022),* p. 183–188.

But these evaluations take sometimes the form of "schoolyard contests", putting aside the ultimate goal which is to alleviate the philologist's task.

Artificial intelligence (AI) has, nowadays, a leading position in NLP applications. One can find an interesting overview, for instance, in Sommerschield *et al.*[3]. In the field of Latin, the two candidates of EvaLatin 2022 use AI. In this contest, Wróbel and Nowak[4] have better results than Mercelis *et al.*[5]. It is not clear from these articles if the authors plan a future development of a complete system to produce annotated texts, which includes the tested sub-tasks together with the tokenisation which was absent of the contest. Latin-BERT[6] seems to be an interesting starting point to introduce AI in Latin studies. The original article and the GitHub repository are clear and their examples are appetising[7]. As pointed out by Sommerschield *et al.*[8], the availability of a pre-trained model which can be freely reused is a strong advantage for future developments. However, for a new comer, the installation of Latin-BERT is not so straightforward. The fact that the code works only with Python 3.7 or 3.8 (now at 3.12) is quite alarming for the entire Python ecosystem. Unfortunately, it was not possible to work with a Mac OS system[9], so one has to use Windows[10] instead, although with some difficulties.

The most challenging domain for text annotation is probably Medieval Latin. For instance, the corpus of the Acta Sanctorum in the Corpus Corporum[11] counts more than 52 million words, more than 25 times the size of the corpus of the LASLA[12]. Lemmatisation and annotation are the key for textometry, which can be useful for various tasks. The corpus of Medieval Latin texts is remarkable not only by its size, but also by the variability in the vocabulary, spelling and construction of the sentences. It cannot be ruled out that Medieval Latin needs specially developped tools to be treated properly. However, one has to begin somewhere and most of the annotated corpora deal with classical texts. The present lemmatiser is mainly based on Collatinus[13] which can handle the principal variation in medieval spelling. On the other hand, one can easily add new lemmata in its database. For the AI part, the training can be done only with existing annotated texts, mainly classical ones. One has to consider the possibility of iterations: running the present tagger on medieval texts, correcting the produced annotated corpus and injecting it into the training process. One can think also about fine-grained models: one for each genre, period, geographic area, etc. However, a balance has to be found between the improved efficiency and the increasing complexity of the task.

3 Thea SOMMERSCHIELD, Yannis ASSAEL, John PAVLOPOULOS, Vanessa STEFANAK, Andrew SENIOR, Chris DYER, John BODEL, Jonathan PRAG, Ion ANDROUTSOPOULOS and Nando de FREITAS. « Machine Learning for Ancient Languages: A Survey. » *Computational Linguistics* 49 (3) https://doi.org/10.1162/coli_a_00481

4 Krzysztof WRÓBEL and Krzysztof NOWAK. « Transformer-based Part-of-Speech Tagging and Lemmatization for Latin. » In *Proceedings of second Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA 2022),* p. 193–197.

5 Wouter MERCELIS and Alek KEERSMAEKERS. « An ELECTRA Model for Latin Token Tagging Tasks. » In *Proceedings of second Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA 2022),* p. 189–192.

6 David BAMMAN and Patrick J. BURNS. « Latin-BERT: A contextual language model for classical philology. » *arXiv preprint arXiv:2009.10053*.

7 https://github.com/dbamman/latin-bert

8 T. SOMMERSCHIELD *et al*. « Machine Learning » cit. n. 3

9 Somewhere a module has to be compiled for Mac OS from a Rust piece of code, which leads to an error referenced on the web for at least a decade but clearly still uncorrected.

10 Probably, Linux would be another possible choice.

11 Corpus n° 28 in https://mlat.uzh.ch/browser?path=cps28

12 The annotated files can be found in https://dataverse.uliege.be/dataverse/lasla, more specifically the Latin files in APN-format: https://doi.org/10.58119/ULG/QJJ0SA

13 Yves OUVRARD and Philippe VERKERK. « Collatinus, un outil polymorphe pour l'étude du latin » in *Archivum Latinitatis Medii Aevi*, 72, pp. 305-311.

In the first section, I shall present some preliminary work that provides a way to get acquainted with Latin-BERT and its vectors. Then I shall describe the structure and organisation of the program. The third section is devoted to the results. It ranges from the preparation of the annotated data to the final evaluation, passing through testing of each sub-task. Then I shall conclude hinting at some directions of future developments.

Preliminary exploration

In the original article about Latin-BERT[14], the case study about homonomy with *cum* seemed to me simple enough and is a good starting point to manipulate vectors. In a computer program, and especially in AI codes, words are represented by vectors which are an ordered collection of numbers (the so-called embedding[15]). These numbers are the coordinates of a point in a huge abstract space. In Latin-BERT, the authors choose to work with a space with 768 dimensions (768D in the following; and similarly, 1D, 2D and 3D for a space with one, two or three dimensions). This choice is probably dictated by conflicting arguments: on one hand, the calculation time increases with the dimensionality and, on the other hand, a low dimensionality will not grasp the subtleties of the natural language. The particular choice of $768 = 3 \times 256$ ($256 = 2^8$ is a "magical" number for computer scientists) is due to programming constraints. Individual words are first imbedded in the 768D space, and then mixed up with their neighbours in the sentence using matrices in several steps or layers. These matrices are the result of the "pre-training" performed on a corpus of 642.7 million of tokens and form the initial model.

The obvious question one asks after reading the section devoted to the distinction between the two homonyms of *cum* is: would it work for any other homonyms, and especially inflected ones as *populus* or *levis*? A quick look at LASLA's files shows that *levis* is a better candidate than *populus* as the ratio between the two homonyms is 1:10 instead of 1:117. Thus I gathered the 803 sentences in LASLA's corpus that contain any form of the two *levis*: 731 *levis*_1[16] (*lĕvis*, light) and 72 *levis*_2 (*lēvis*, smooth). Then, I made Latin-BERT vectorise these sentences and I picked up the vectors corresponding to the various forms of *levis*.

These vectors are just points in a huge 768D space which represent any form of one of the lemmata *levis*, taking into account the context (contextual language model). The distinction between the hononyms can be seen as a separation in two groups of points. The simplest idea is to measure the Euclidean distance[17] between two vectors of this set, keeping track of the homonyms. One can plot histograms of the distances within the two groups of the homonyms and between one group and the other. The histogram counts the number of values (i.e. distances between pairs of points) in each successive bin[18]: here, I have 100 bins with a 0.25 width[19]. The resulting plots are shown in Figure 1, and the curves are very similar, except for the vertical axis that depends on the size of the

14 D. BAMMAN and P. J. BURNS. « Latin-BERT » cit. n. 6

15 For a general discussion about words embedding in Natural Language Processing, one can refer for instance to Stuart J. RUSSELL and Peter NORVIG, « Artificial intelligence : a modern approach » 4th Edition; Pearson series in artificial intelligence, 2021 section 24.1 p. 856. For a mathematical background about vectors and matrices, see for instance Appendix A2, p. 1025 *ibid*.

16 For compact notations, I paste the homonym index to the lemma: *levis*_1 is thus the lemma *levis* with index 1.

17 In the usual 3D space (our world), the Euclidean distance between two points is the one measured with a ruler. It is given by a simple mathematical expression that can be generalised to any dimensionality.

18 In statistics, a bin is a range of numerical values: a pair of points will be counted in the bin associated with the distance 16 (the peak in the two first plots) if the distance between the two points is larger than 16 (or equal) and (strictly) smaller than 16.25.

19 The vectors being in an abstract space, distances have no unit but, if convenient, one can imagine they are given in meters or inches.

considered sample. Some noise appears on the histogram of *levis*_2 simply because of the reduced size of the sample (note that the vertical scale has been reduced by a factor 1,000 for the two leftmost plots). The average distance between two *levis*_1 is $<D_{11}>=15.83$ with a standard deviation of $\sigma_{11}=1.57$, while these values are $<D_{22}>=15.36$ and $\sigma_{22}=1.83$ for *levis*_2. The average distance between a *levis*_1 and a *levis*_2 is slightly larger $<D_{12}>=15.94$ and $\sigma_{12}=1.41$, but the difference is not significant as it is much smaller than any of the spreads, σ. The points associated with the two homonyms seem to sit in the same part of the 768D space. One interesting point is that in the *levis*_1 sub-set, two vectors are identical: looking closer to the texts, it turns out that Virgil wrote twice the same sentence.

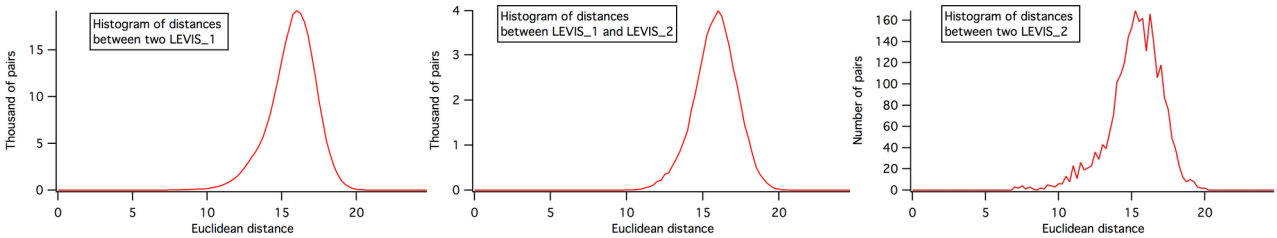

Figure 1: Histograms of the distances between two vectors representing a form of the lemmata *levis*. left: Both forms are from the *levis*_1 homonym; center: The two forms come from a different homonym; right: Both forms are from the *levis*_2 homonym. The maximum of the two leftmost histograms is reached for a distance of 16, while the noise shifts the maximum of the histogram on the right to a slightly smaller value, 15.25 (the resolution on the horizontal axis is 0.25).

Direct approach

In their study of the two *cum*, the authors of Latin-BERT[20] used an elaborate method based on a variation of Stochastic Neighbor Embedding, called t-SNE[21]. As piling one black-box on top of another black-box would not enlight the behaviour of any of them, I prefer a direct approach rather than the elaborate one used originally. If the distinction between the homonyms is possible, the vectors of each homonym should cluster in different parts of the space. Obviously, the abstract space in which the points are has 768 dimensions (which means that one needs 768 numbers to know where the point is) and is therefore more difficult to apprehend. However, the vision of a rubber ring to hold the pencils or of a line drawn on a sheet of paper with points on each side corresponding to the different homonyms still holds.

One can choose two points in space and measure the distances between these points and those associated to the forms. The hope is that one of the homonyms remains closer to the first point than to the second one and vice-versa for the other homonym. An intuitive choice for the two points is the average position for each of the homonyms[22]. This average position is often called Centre of Mass, later noted as CoM[23]. One then projects each vector on the line connecting the two points and the origin is chosen to be the CoM of the *levis*_2 homonym. The CoM of *levis*_1 is associated to a

20 D. Bamman and P. J. Burns. « Latin-BERT » cit. n. 6

21 Laurens van der Maaten and Geoffrey Hinton. « Visualizing data using t-SNE. » *Journal of machine learning research*, 9 (Nov): 2579–2605.

22 If one uses all the sample for the evaluation (which is what I did), one loses the predictive power of the method. However, one could play the usual trick to sacrifice 90% of the sample for training and to keep the remaining 10% for testing. One has to be careful about the size of the samples and the resulting statistical errors.

23 Also called centroid or barycentre, the notion of Centre of Mass is widely used in Physics, but is adopted in several other domains as soon as one needs to determine the weighted average of a set of points.

positive value. With these choices, the abscissas of *levis*_1 span from -0.93 to 6.34, while those of *levis*_2 span from -4.3 to 3.25, thus one notes a large overlap of the two groups. The overlapping region contains 292 points linked to *levis*_1 and 48 to *levis*_2.

As a 1D representation of the sample is not enough to distinguish the two groups, let us move to 2D. Our samples of vectors have to be projected into the sub-space orthogonal to the first direction that we have chosen. Then we calculate the two CoM of the homonyms that are in the grey zone, i.e. the overlap between to two groups in 1D. As a second coordinate in the 2D plane, we consider the projection of the vectors on the line that joins these two new CoM. Along that projection axis, the CoM of the *levis*_1 homonyms that were in the overlapping region is again associated to a positive value. The result is to push upwards (on average) the *levis*_1 points that are in the overlapping region, while the *levis*_2 are pushed downwards (on average). As shown on Figure 2a, a new tilted region of overlap appears, which is narrower than the original overlapping region (a vertical stripe going from the leftmost blue point to the rightmost red point). We define a tilted ribbon and choose its angle to minimise the number of points it contains. The optimal ribbon contains 131 *levis*_1 and 37 *levis*_2, much less than the original grey zone. One can rotate the reference frame so that this new overlap zone becomes vertical again and forget about the vertical coordinate. Then we can iterate the process in a sub-space orthogonal to the directions already treated (i.e. 766D for the second iteration). One computes the position of CoM of each sub-sample and projects each vector onto the line that joins these new CoM. Figure 2b shows the resulting distribution of points after an second iteration: 133 points remain in the region of overlap.

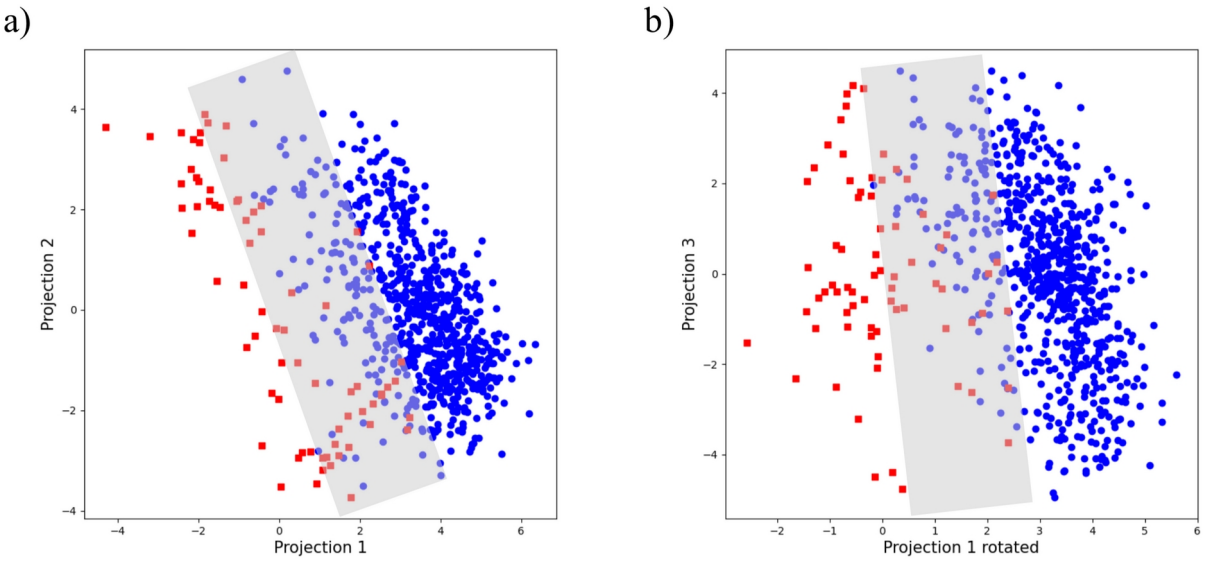

Figure 2: 2D projections of the 768D vectors representing the forms of the two homonyms *levis*_1 and *levis*_2. The two homonyms overlap in the grey zone. The directions onto which the projections are made are obtained by iteration (see text): a) corresponds to the first iteration and b) to the second one. Note that the projection axes are different in the two plots.

PRINCIPAL COMPONENT ANALYSIS

Principal component analysis (PCA) is a well established method to reduce the dimensionality while keeping the major part of the variations. It consists in the diagonalisation of the correlation matrix. The eigenvalues are ordered in descending order and the first corresponding eigenvectors give the directions along which the points spread mostly. Usually, one uses the first two components to have a simple representation in a plane or the three first components to have a richer 3D plot. In Figure 3a, one clearly sees at least three groups of points, but they are not linked to the homonyms

which are colored in red for *levis*_2 and in blue for *levis*_1. The red points are evenly distributed between the three groups.

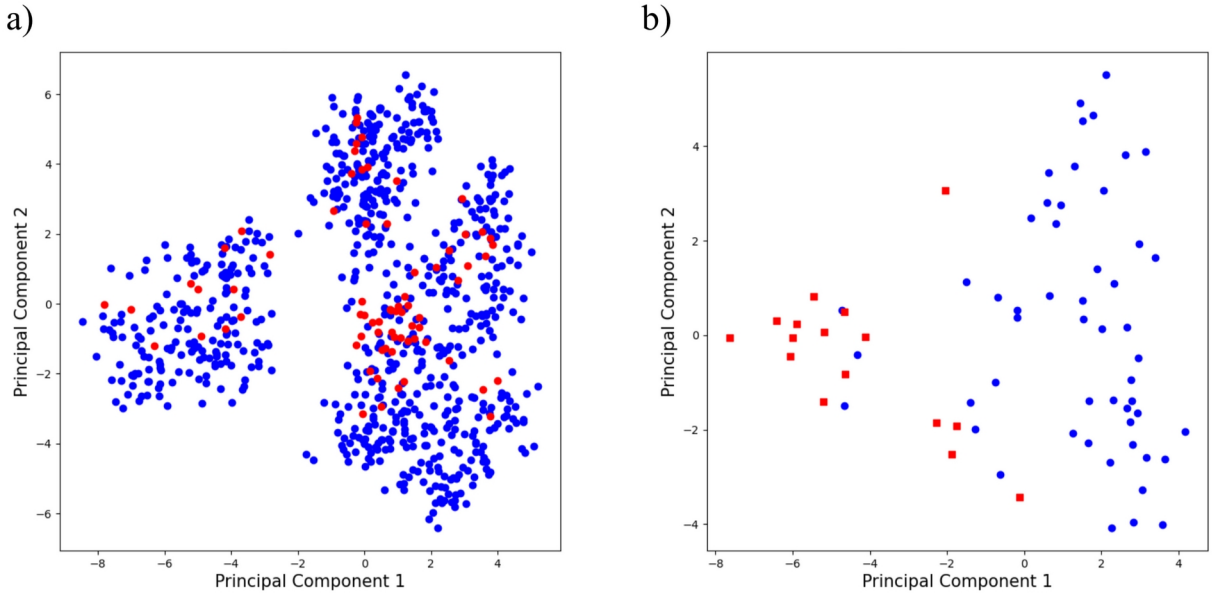

Figure 3: Two component analysis of the *levis* homonyms. The horizontal axis gives the projection of the vectors along the first eigenvector (corresponding to the highest eigenvalue) and the vertical axis is the projection along the second eigenvector. The blue circles are for *levis*_1 and the red squares for *levis*_2. In a) we consider all the sample, while in b) we restrict the analysis to the single form *levia*.

So what are these groups? As a matter of fact, they depend on the inflected form: the group on the left contains the forms *leve*, *levem* and *leves*, while the upper-right group is the form *levis* itself and the last group, bottom-right, contains all other forms. It is not so surprising that the resulting vectors keep track of the word-ending, as Latin-BERT contains a "sub-word-tokeniser" that splits apart these word-endings. I have not checked in the code of this sub-word-tokeniser if the observed result is consistent with the actual word-splitting.

If PCA is strongly sensitive to the inflected form, one can extract from the sample the vectors corresponding to the *levia* form. We have 68 points, of which 52 belong to *levis*_1 and 16 to *levis*_2. The choice of this form is suggested by the size of the sub-sample and the ratio between the two homonyms. For this sub-sample of vectors, we compute again the first two principal components and show them in Figure 3b. Note that the points in this figure do not correspond to those of the form *levia* in the previous plot: the PCA of a sub-set is not the corresponding sub-set of the PCA of the original ensemble. The two homonyms form almost two distinct groups except for three blue points that seem to fall into the red group and five red points that are too close to the blue group.

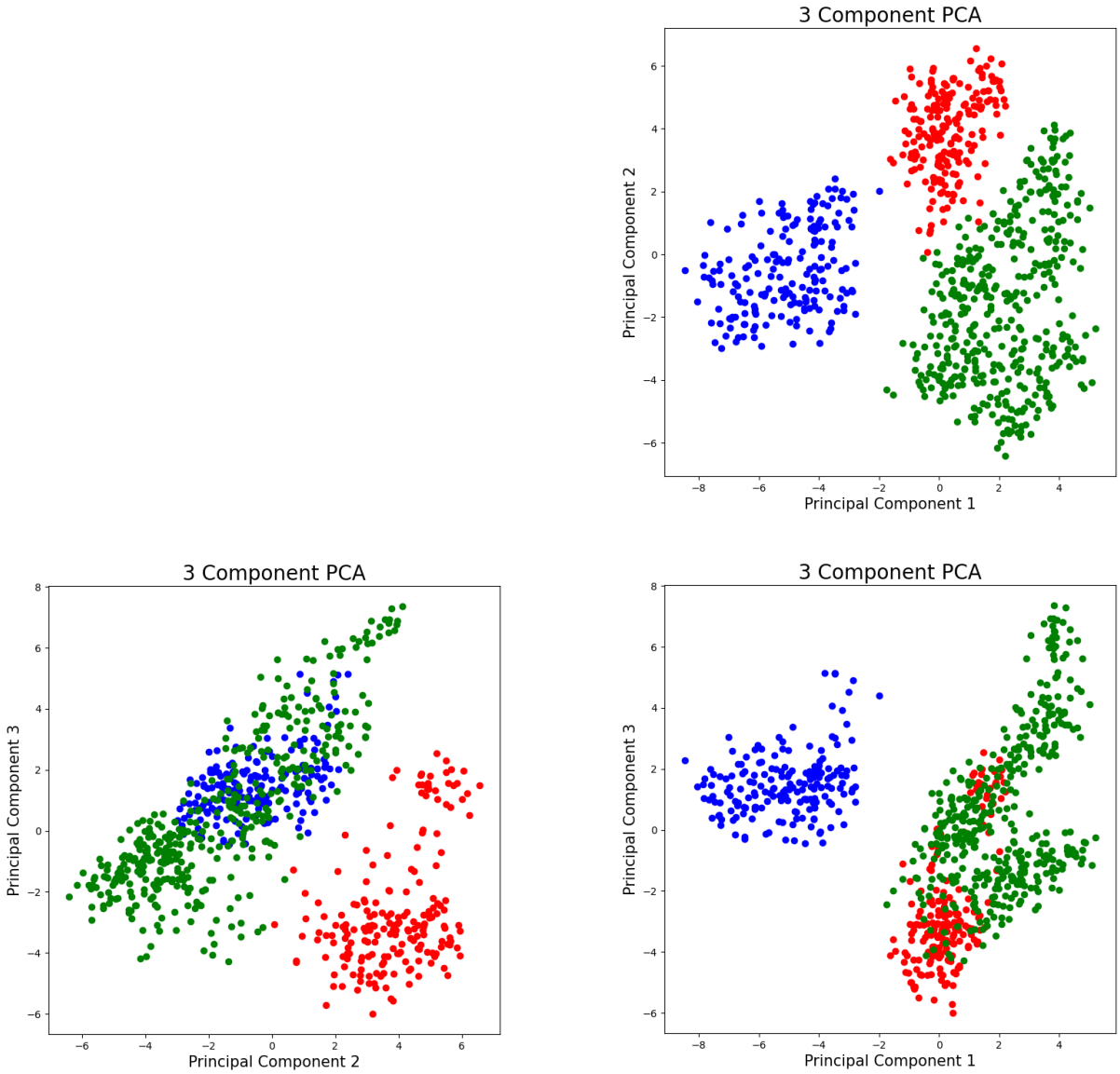

Figure 4: Three component analysis of the inflected forms of *levis*. The three forms *leve*, *levem* and *leves* are in blue, *levis* itself is in red and all the other forms are in green. The three plots correspond to the projections on the sides of a cube surrounding the 3D space. The top-right plot is in the plane of the first two components and is identical to the one of Figure 3a, except for the colours.

The three 2D projections of a three components analysis (see Fig. 4) show more details and we see some structures in the groups. The colours are now associated to the forms: *levis* in red, *leve** in blue and the rest in green. The introduction of the third component clearly splits the green group into at least two parts. More surprising is the small cloud of red points that seems to fly away from the rest of the group along the third component. Almost all these points come from Caesar. Three of them are from Tacitus. This is also true for the topmost part (along the third component) of the

green group. But the reverse is not true: some points from Caesar are present elsewhere, although most of them are associated with a high value of the third component in the PCA.

One can conclude that PCA does show some things, but the homonymy does not appear in the first components, except to some extent when the sample is limited to a single form, *levia* in the present example. The first direct method, described above, seems to converge slowly, if it does converge at all. Some extra work would be needed to draw a definite conclusion. But the direct examination of the vectors representing a lemma (in its various inflected forms) is not the main goal of this article. In conclusion, the example given in the original article[24] with *cum* seems hard to reproduce with a homonym that has inflected forms.

Description of the lemmatiser

A missing word

Usually, "lemmatisation" is, for the philologist, the operation that attributes "the" lemma to the form found in some text. However, thinking more deeply about it, "lemmatisation" in its usual meaning can be decomposed into two steps: 1) the analysis of the form that leads to the collection of all the possible lemmata that could explain this form (for instance, the form *amicae* is associated with the lemmata *amica* and *amicus*_2, which is the adjective[25]), and 2) the disambiguation that selects the correct solution in the wider set of possible lemmata (in the previous example, the Part of Speech would determine the correct solution). The first step is technically a lemmatisation, but in a more general sense than the one usually accepted. In the following, I will address this operation as "lemmatisation", with the warning that its result is an set of possible lemmata and not a single one.

Would it make sense to ask AI to lemmatise a form? I will not try to answer this delicate question. I will just comment saying that a student, with some patience, can do the job with a dictionary and a set of rules for inflection. Several pieces of software can do that, among which Collatinus[26]. Indeed, it has already been integrated in a probabilistic tagger, presented elsewhere[27]. In this tagger, Collatinus is used in conjunction with another lemmatiser (based on the list of forms found in LASLA's corpus) in order to associate to each form of the text all the known lemmata and analyses. It seems thus natural to reuse this existing piece of software[28].

Disambiguation

The aforementioned tagger disambiguates the possible analyses with a third order hidden Markov model (HMM3); for an introduction to HMM3 see for instance[29]. It uses a tag deduced from LASLA's code for the morpho-syntactical analysis. The tag contains the part of speech (PoS), together with the case and the number for substantives and adjectives and the mood for verbs. In most cases, the tag associated with a form and a lemma allows to recover LASLA's code. The probabilistic tagger chooses the tag for each word using the statistics about sequences of three tags in LASLA's corpus. The resulting accuracy was acceptable, but lower than the results of the PoS-

24 D. Bamman and P. J. Burns. « Latin-BERT » cit. n. 6

25 The index of homonymy is arbitrary, and I stick to the choice made by the LASLA. The lemma itself results from a choice: here also, I stick to the conventions and I will not use the lemma *amare* for the verb *amo*, as some dictionaries of Medieval Latin do.

26 Y. Ouvrard and P. Verkerk « Collatinus » cit. n. 13.

27 P. Verkerk *et al.* « L.A.S.L.A. and Collatinus » cit. n. 1.

28 The code and data are available on GitHub: https://github.com/PhVerkerk/LASLA_tagger

29 Lawrence R. Rabiner, « A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition » in Proceedings of the IEEE, 77, 257-285 (1989)

tagging achieved by Latin-BERT[30]. It seems thus justified to replace HMM3 with an AI-based selection of the tag.

Another weakness of the probabilistic tagger is that it is unable to make any prediction about homonyms. In case of homonyms (with the same tag), the probabilistic tagger chooses the most frequent solution, which means that *populus*_2 (poplar) will never appear as the lemma associated to any corresponding form in any text. Although the preliminary explorations were not very conclusive on that matter, the index used to distinguish homonyms is not so different from a PoS or a tag. AI seems to be sensitive to the meaning, so it may also succeed in the determination of the correct homonym (when they exist). A third ingredient which was not taken into account previously is the subordination code. This code is associated to the syntactic analysis of the sentence and was previously considered out of reach. With Latin-BERT, it is just a matter of building a model on the corpus with an emphasis on this piece of information. As it is straight forward and similar to what is done for tags and indices, it will not be detailed here.

Reconciliation

After the proper training, the AI-code will predict independently the tag and the index for homonyms, the starting point being always the sequence of words (or tokens) in the sentence. The independence in the two predictions can lead to conflicts. Just consider a very naive and simple case: the form *amicus* can be associated to the tag "A11" (noun nominative singular) or to the tag "C11" (adjective nominative singular[31]). In LASLA's convention, the lemma *amicus*_1 is the noun, while *amicus*_2 is the adjective. The independent predictions might give "A11" for the tag and "2" for the index (or "C11" and "1") which are incompatible. In a fully automatic workflow, the program has to trust one of the predictions, throwing away the other. The tag being the prediction carrying more information[32], I chose to trust the prediction of the tag. But, obviously, in some instances it will be the wrong choice. In a supervised mode, the program has to warn the philologist that the predictions are conflicting.

More generally than for the conflicting predictions, the AI-code can propose a tag (or an index) that is not available in the set of possible lemmatisations and analyses. The program then warns the philologist who will have to decide what to do. Meanwhile, the program sticks to the choice made by the probabilistic tagger, which always exists. A third category of warnings sent to the philologist contains unknown forms. Although Collatinus has collected most of the entries of the main dictionaries (Lewis & Short, Gaffiot etc.), several proper names are missing and, if leaving the classical period of Latin, one will find new words or forms in the texts. One strategy could be to extend what has been done with the classical dictionaries to those in Medieval Latin or Neo-Latin. The other option being to rely on the knowledge of the philologist.

Workflow

As a summary, we can draw the workflow for the entire process to go from the plain text to the final annotated text. Three different programs are used as depicted in Figure 5. In a pure UNIX-style, these programs could have been piped (the output of one being used as the input of the next one), but I have prefered to produce intermediate files. In particular, this allows the third program to use the outputs of the first two programs. In Fig. 5, the initial file has a generic name, "myText.txt", and the names of the produced files are automatically derived from that name by changing the extension and/or adding explicit segments to the name.

30 D. Bamman and P. J. Burns. « Latin-BERT » cit. n. 6

31 gender is not present in the APN-format.

32 In the case of the form *amicus*, the tag and the index carry the same amount of information. But if one considers the form *amico*, one has four possible tags and two indices (if one forgets the hapax verb *amico, are*). The tag carries the information about the case. The index does not. So it is better (more informative) to keep the tag.

The first piece of code derives from the probabilistic tagger presented by Verkerk *et al.*[33]. It is written in C++ with the Qt libraries. It accepts as input any plain text file and proceeds to its tokenisation and lemmatisation. As pointed out in Sprugnoli et al.[34], tokenisation itself depends on choices and can lead to misalignment in the output. Strictly speaking, tokenisation and lemmatisation are not independent sub-tasks. This is true in particular with forms carrying an enclitics and with lemmata which can be written as two words, e.g. *res publica*. Our choice is that if the word exists as a whole, we shall not split appart the enclitic, e.g. *quoque* will never be tokenised as *quo* + *que*, which sometimes occurs. On the other hand, the APN-format allows multi-word tokens and we stick to that convention. This program produces three output files. The first two are the annotated files obtained by the lemmatiser and the probabilistic tagger (based on HMM3) and are in LASLA's APN-format. The interest of keeping these files goes beyond historical reasons, as we'll see later that they can be used to improve the accuracy of the process by comparison of two different predictions (see the Prospect section below). The third file is a CSV-file (some would call it TSV as the tab is the separator) which will contain all the possible lemmatisations and analyses of each word. This file serves also as an input file for the prediction part of Latin-BERT with the form in the second field of a normal line, all the extra information hidden in commented lines (beginning with a #) and a blank line to end the sentences. As a matter of fact, this file contains more commented lines than normal ones. This CSV-file will also be used in the post-correction phase when a GUI will be available for the program (see below).

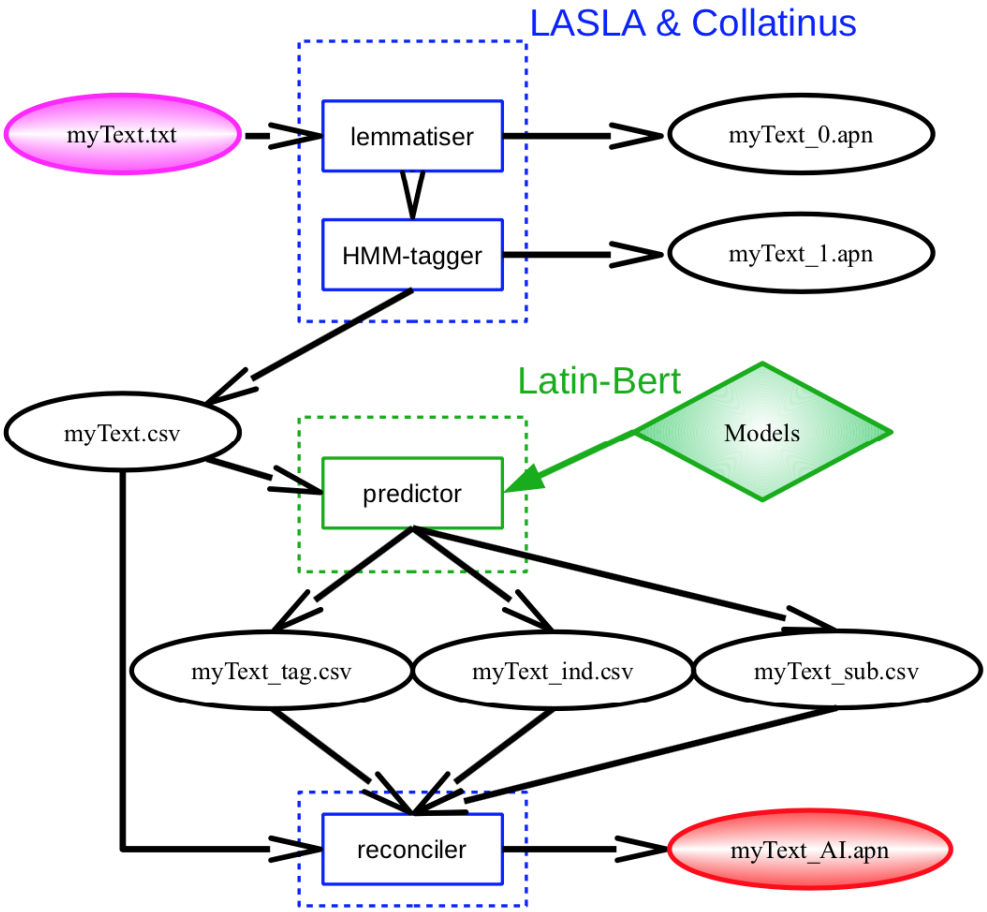

Figure 5: Sketch of the workflow to transform a plain text file "myText.txt" to a fully anotated APN-file "myText_AI.apn". It presently involves two programs written in C++ with the Qt environment (blue rectangles) and one

33 P. Verkerk *et al.* « L.A.S.L.A. and Collatinus » cit. n. 1.

34 R. SPRUGNOLI et al. « Overview of the EvaLatin 2022 » cit. n. 2

program in Python (green rectangle). Some intermediate files are produced and stored. These files, together with the initial and final ones, are shown in ellipses. The initial file is highlighted in magenta, while the final one is in red. The green rhombus represents the models needed by the Python AI-code. They are described in the next section.

The second program is a simplification of the script in Python proposed for PoS-tagging by Bamman and Burns[35], retaining just the "prediction part". As shown in Fig. 5, it takes as input the CSV-file generated by the first program and the models created elsewhere (see next section), and produces output files with the predictions (made independently for tags and indices as discussed above). For users who are uneasy with Python and the tons of modules to install, I have built an EXE file for Windows using PyInstaller that should run on any computer with Windows.

The last program, again written with Qt, takes the output files from the first and second programs to produce the final APN-file. In the nearest future, the first and last programs will merge in a graphical environment to allow the philologist to edit and correct the APN-file. The execution of the intermediate program will be performed by an external call within the Qt code. The warnings issued during the reconciliation task would pinpoint the lemmatisations and analyses to check first, because they are the results of conflicting predictions.

Results

Corpus preparation

The case study devoted to PoS-tagging with Latin-BERT uses files in the CoNNL-U format where each line corresponds to a token. The relevant parts of the lines are the second field (the form) and the fourth field (the PoS), the fields being separated by a tab. An empty line is inserted between the sentences, and lines beginning with a hash ("#") are considered as comments and are ignored. Instead of sticking to the CoNNL-U format and changing the code in Python, I prefer to generate three CSV-files for LASLA's texts, containing each either the tag (which is a short version of LASLA's morphology code), the index of homonymy or the subordination code.

On the other hand, the APN-format has fields with fixed lengths. As I am not interested in the production of a CoNNL-U-like file, I just had to pick up the proper fields and insert them into the second and fourth fields of a pseudo-CoNNL-U CSV-file. The numbering of the sentences in the APN-file allows to introduce a blank line at the end of the sentence. Some care has to be taken for the multi-word tokens that are present in LASLA's files. In the original corpus (154 annotated texts) of the LASLA that I used to extract statistical data for the HMM3 tagger, some texts are very short (fragments of mostly lost texts, down to 8 tokens) and I have excluded them from any training corpora. Meanwhile, new texts have been made available and I have added them to the corpus. I kept four texts for testing[36]. These texts have been converted into plain text. As usual in Latin, the enclitics are pasted at the end of the preceding word: tokenisation has to be done together with lemmatisation, as mentionned above. A full stop was added at the end of the sentence (the APN-files did not contain the original punctuation of the text). Line breaks are also restored using the references given in the APN-files.

35 D. Bamman and P. J. Burns. « Latin-BERT » cit. n. 6

36 CAESAR: De Bello Hispaniensi [apocryphum, 6,165 tokens]; CICERO: Divinatio in Q. Caecilium [5,858 tokens]; LIVY: Ab Vrbe Condita [VI, 13,688 tokens]; VITRVVIVS: De Architectura [part, 6,145 tokens]

In summary, I have 146 texts converted to CSV-files with 1,903,104 tokens[37] for training or to evaluate the models, while 4 other texts with 31,856 tokens are kept for testing. These 4 texts have been converted to CSV-files to test the sub-tasks, but also to plain text files for testing the entire process.

Training and evaluation

In their article[38], the authors present an example of PoS tagging. Respect to the pre-trained Latin-BERT model, an extra layer is added on the top of the 12 already implemented. This layer, composed of a linear transformation and a soft-max operation, will try to find the correct tag for the vectors obtained with the previously introduced contextual embedding. At the beginning of the training, the matrix needed for the linear transformation is filled with random numbers. The training is then an iterative process. At each step, the model is applied to the training file and its predictions are compared to the expected ones (those that are in the file). The number of errors is counted and this figure is called the loss function. Then the parameters of the model are tuned (i.e. changed in a clever way) and the loss function is evaluated again. The goal of the training phase is to minimise the loss (less errors means more correct results): knowing the initial and final values of the loss function and the changes in the parameters, the program determines how to change them for the next trial, and it goes on again and again. The number of parameters to tune is so large[39] that a single iteration is almost meaningless and they are grouped in epochs[40]: a bunch of iterations at the end of which the program interacts with the external world (i.e. informs the operator of the achieved value of the loss function and, in the supervised mode, the accuracy measured on the ancillary file). Depending on the size of the training file, the number of iterations to complete an epoch varies: I saw 660 iterations (for the smallest training file) up to 3,210 (for the largest).

In order to start with some small tests, I trained a first model on a single text: Caesar's *Commentarii Belli Gallici* which counts 45,729 tokens. It took half an hour for a training of 10 epochs on my laptop without Graphics Processing Unit (GPU) and 1h25min for 25 epochs[41]. Then I tested these two models (which differ by the duration of the training) on two test-texts, obtaining accuracies of 94.3% and 94.7% on Ps-Caesar's test-text[42] and 90.9% and 92.9% on Cicero's test-text, for the tags. The execution time is reasonable and the results are behaving as expected: the accuracy increases with the duration of the training and is better for the text similar to the one used for training. I repeated the same operations (training with 10 and 25 epochs, followed by the testing) for the homonym index leading to slightly higher accuracies (95.5 and 96.4% for Ps-Caesar and 94.8 and 95.8% for Cicero). The choice being simpler for the index (10 possible values) than for the tag (185 values), the higher accuracies are expected.

37 Note that the number of tokens in the CoNNL-U-like files differs from the original number in LASLA's files, because the full-dot at the end of the sentences is counted as a token in the Python code. The figures I give are the tokens that will appear as a line in the APN-files.

38 D. Bamman and P. J. Burns. « Latin-BERT » cit. n. 6

39 At the end of the training process, the program produces a model that is a huge file: its size is about 445 MByte.

40 The choice of this word in the AI jargon can be related to its use in geology (e.g. the Pliocene epoch) where an epoch groups several million years or to its etymology *ἐποχή* = stoppage, pause.

41 To be fully informative, I should provide the characteristics of my computer, but my point is not to give exact figures, just rough estimates. The important point is that if one needs a result for the group meeting on Monday at 2pm, he/she has to anticipate the move. It may be that with a GPU, the code will run 10 times faster, but I have not checked.

42 Although *De Bello Hispaniensi* has been attributed to Caesar (a long time ago), I shall refer to it using "Ps-Caesar".

With these encouraging first results, I moved to a spare computer[43] in the lab to go on with larger training corpora. Having no scientific arguments to set-up a larger training corpus, I just joined eight of the largest files, with a total of 435,008 tokens[44]. The training corpus being tenfold larger, the execution time increases accordingly: 35 minutes per epoch[45]. The model is obtained with 25 epochs and leads to an accuracy of 96.1% for Ps-Caesar and 97.1% for Cicero. Now, the training corpus contains texts of different authors correcting the bias due to the similarities between the training text (of Caesar) and Ps-Caesar's test-text in the first attempt (similarities due to genre, style, argument etc.). As expected, a larger training corpus leads to higher accuracies. Going on with 21 texts collected in a corpus of 929,615 tokens, the training time increases to 1h20 per epoch. I ran the program for 25 epochs giving accuracies of 96.4% for Ps-Caesar and 98% for Cicero, confirming the trend that Cicero is "easier" to tag than Ps-Caesar. I ran it also with 60 epochs and the accuracies dropped to 96.3% and 97.5%, respectively. Clearly the training has reached a limit and overfitting degrades the results. The next move being the full corpus which is more than twice as large as the previous one, one has to think about this phenomenon.

An interesting question arises as soon as one has different models at hand. For instance, the difference between an accuracy of 96.4% and 96.1% may be just about 20 more errors. But is it actually true? I switched to the "predict" mode and gathered the CSV-files with the predictions into a single spreadsheet, together with the original. For these predictions, I used the three models described above: the model obtained with the corpus of 435,008 tokens and 25 epochs and the two others with the corpus of 929,615 tokens and, respectively, 25 and 60 epochs. In this sheet, I collected all the lines where at least one result is wrong. I have obtained 392 lines, many more than the 254 errors of the first model or than the 234 errors of the best one. Among these 392 lines, 142 have wrong predictions for all the models. About 40% of the errors are related to "weak predictions" where these three models do not agree on the tag to choose. I am not sure that a "majority vote" would increase accuracy, but the fact that different models err on different tokens could help the philologist in checking the resulting predictions. Obviously, it would imply a post-correction step (see the Prospect section below).

Putting together the 146 files from LASLA's corpus gives a third training file which leads, after 25 epochs, to an accuracy of 96.7% and 98.2%, respectively on Ps-Caesar and Cicero. However, as overfitting showed up, it is possible that a shorter training gives better results. The Python code includes a supervised mode, where an ancillary file (called in the jargon "dev file", a name kept in the following) is used to measure the accuracy epoch by epoch. The model is saved only if the achieved accuracy exceeds the previously obtained maximum value. A first attempt was done keeping four texts[46] as a "dev file" with 43,880 tokens and the 142 other texts in the training file. A maximum accuracy of 97.3% was achieved on this dev file at the end of the third epoch. This seemed surprisingly short to me, but it may be attributed to the reduced size of the dev file (2.3% of the whole corpus). For the tags, the accuracy on the usual texts of Ps-Caesar and Cicero were respectively 96.70% and 98.34%, always higher for Cicero. They were slightly higher for the homonym index: respectively, 97.90% and 99.39%.

---

43 Without GPU either, slightly slower than my laptop, because older. Once again, its exact characteristics are not relevant. The fact that it is a "spare" computer is more important, in the sense that it has to be dedicated to that task for a long period of time (several days), without interruption.

44 CAESAR: Commentarii Belli Gallici [45,729 tokens]; CICERO: De Officiis [34,594 tokens]; LUCRETIVS: De Rerum Natura [50,056 tokens]; PLINY: Epistulae [67,070 tokens]; QVINTVS CURTIVS: Historiarum Alexandri Magni Libri [72,494 tokens]; SENECA: De Beneficiis [45,572 tokens]; TACITVS: Historiae [52,401 tokens]; VIRGIL: Aeneis [67,092 tokens]

45 I do not pretend to have shown that the execution time "is" linear with the size of the training set, but it looks like.

46 CAESAR: De Bello Alexandrino [apocryphum, 10,666 tokens]; CICERO: Pro Murena [10,526 tokens]; LIVY: Ab Vrbe Condita [book 8, 13,225 tokens]; PLAVTVS: Amphitruo [9,463 tokens]

The random choice of four texts for the dev file has the defect of focusing on four authors. The sample was also a bit limited, compared to the usual 10% or 5% of the corpus that is used for supervision. I then adopted a new method to build the two train/dev files: I chose one sentence out of every ten and placed it in the dev file, the nine other sentences being copied to the training file. Having no arguments to choose the first (modulo 10) or the tenth sentences, I built two train/dev sets[47] with the fourth (modulo 10) and the seventh sentences. As the length of the sentences is not constant, the size of the dev files is roughly (but not exactly) 10% of the corpus. The program provides the values of the accuracy and of the loss at the end of each epoch (about 2h40 per epoch). During the training phase, the program tunes the parameters of the model (the coefficients in the matrices) trying to minimise the loss function which is the number of errors obtained with the model in each step of optimisation. Within a normalisation coefficient, the loss function is the complement of the accuracy (based on the number of errors instead of the number of good predictions) except that it is evaluated on the training file and not on the dev file. I have plotted the loss and the accuracy in figure 6. A logarithmic scale was used for a better display of the loss function because it varies by more than one order of magnitude (a factor 10). The maximum is reached very quickly both for the tag and the index. Four or five epochs are enough to reach this maximum and afterwards the accuracy decreases slowly. Note that the values of the accuracy for each train/dev set should not be compared. These accuracies depend obviously on the obtained model, but also on the sample used for the evaluation which is not the same in the two cases.

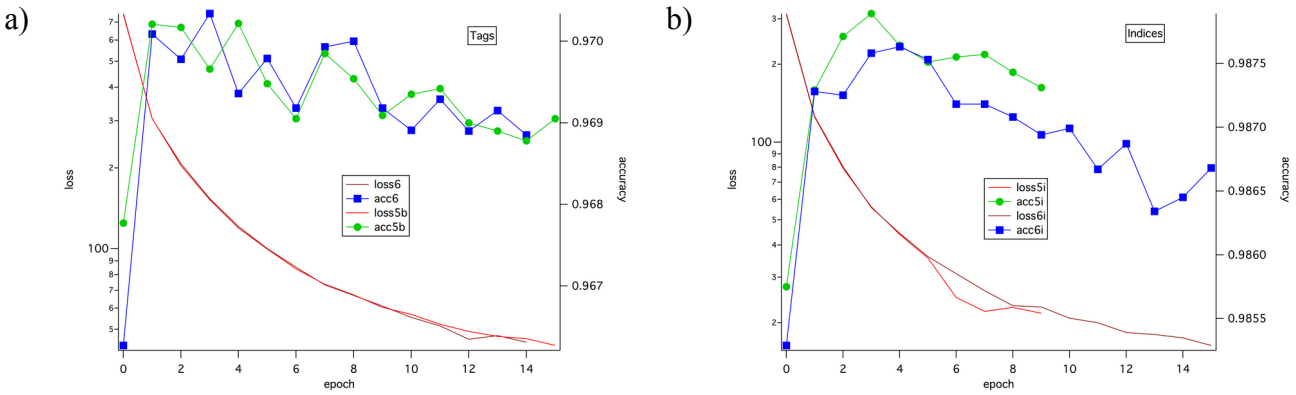

Figure 6: Plots of the loss function (solid lines, left axis, log scale) and the accuracy (line and markers, right axis, linear scale) for the two train/dev sets. On the left, for the tag and, on the right, for the homonyms' index.

One is usually happy when a program runs smoothly and quickly, but it seems to me that the maximum was reached too quickly. I cannot give any scientific reason for that, but my intuition, fed by a long experience in numerical simulations, tells me to be careful[48]. The subsequent decrease in the accuracy, although limited to about 0.1%, is also a bad point. In Latin-BERT's code, one finds a variable "fine_tuning_learning_rate" with the value 5 10^{-5} which was probably optimised for the case study on Proiel or Perseus data. I reduced it by a factor of 2 and ran again the last two supervised trainings. The losses and accuracies are plotted as previously in Figure 7. The accuracy at the end of the first epoch is slightly lower than previously (2‰ lower), but its maximum value is slightly higher: 97.211% instead of 97.034% for the tag in the last set[49]; 98.825% vs 98.789% for

47 I could have built ten sets, but it seemed unnecessary.

48 A hint for that is given by the default values in the original code: a variable for the maximum number of epochs has the value 25 and another variable called "patience" has the value 10. These values suggest that 7 or 8 epochs would be a adequate goal to reach the maximum accuracy.

49 It makes sense to give the accuracy with 5 digits as the sample used to measure it counts about 200,000 tokens. In some cases, an integer representing the number of errors would have been a better choice.

the index in the first set. The maximum is also reached a bit later (much later in one of the curves). Although a 2‰ difference in accuracy may lead to winning or losing a contest, it is not the most important point here. In my opinion, the flatter curve after the maximum is a hint of a better convergence and, hopefully, of a less marked overfitting.

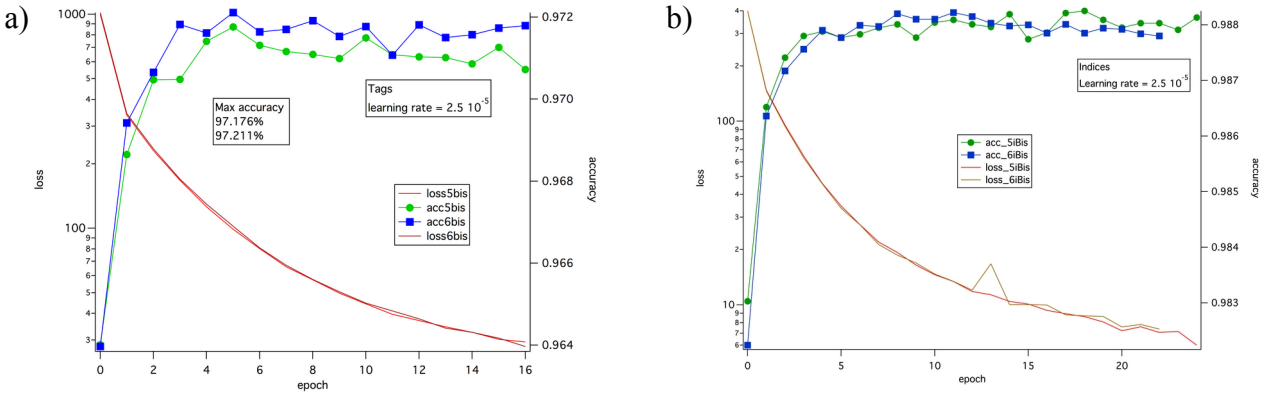

Figure 7: Same as figure 6, but the "fine_tuning_learning_rate" has been reduced by a factor of two. The maximum in accuracy is reached later and its decrease is less prononced.

One more test can be done: the influence of the random numbers that are used to start the learning process. I have changed the seeds used by the random number generators[50] and I repeated the training on the first train/dev set for the tags. The training reached an accuracy of 97.201%, which is marginally higher than the value obtained with the first run 97.176%. These two values can be compared as they are evaluated on the same sample. The difference in accuracy reveals an extra 50 errors on a total number of errors around 6,000.

However, the larger the training corpus, the better the accuracy. It is thus tempting to use the whole available corpus (except for the four texts I set apart from the beginning) for a blind training. If enough space is available for storage, one can save the model at the end of every epoch and run the program for 25 epochs: for the tags only, it required more than 11 GB of storage (and 3 days of calculations). A simpler (safer) alternative is to repeat the same exploration with 5% of the corpus in the dev file (one sentence each twenty). The maximum accuracy becomes now 97.247% for the tag. But, as pointed out previously, the evaluations of the accuracy during the training process is performed on different samples. A final test on the same texts that were excluded from the training corpora is needed to really check the models.

CHECKING THE MODELS

In the following tables, I provide the accuracy of the various models for the tags (Table 1) and for the indices (Table 2) when applied to the four texts. The last column is a weighted average of the results, taking into account the length of each of text. These models were obtained with a training in the supervised mode. The first part of the name of the models has an obvious meaning "tag" for the tags and "ind" for the indices. The digit in the name of the model represents the train/dev set: the dev files for 5 and 6 are obtained with one sentence each ten (respectively the fourth and the seventh, modulo 10), while for 7 it is obtained with one sentence each twenty. The ending "bis" means that the fine tuning learning rate was reduced by a factor of 2 respect to the original one. The "tag5ter" is the trial with a non zero seed for the random number generators, also with a reduced

50 For some obscure reasons, two random generators seem to be used. They had both the seed set to 0. I kept them equal but set them to 100. By mistake, I also ran twice the program with the same seeds and the same train/dev sets and the results were identical (also in the predictions). This is not a proof of the repeatability of the program, but the contrary would have been alarming.

learning rate. The “ind5bis_12” and “ind6bis_5” correspond to a first relative maximum in the evaluated accuracy: they are not the final models proposed by the Python code. In a few cases, the first, partial, model gives better results than the final one.

One notices that the accuracy depends strongly on the text or the author. Always excellent for Cicero, it is much lower for Vitruvius. The average values are slightly lower (by a few ‰) than the accuracy evaluated during the training. Another quite surprizing point is that the best accuracy for the different texts is obtained with different models. For instance, the model tag5ter is the best for the chosen texts of Ps-Caesar and Cicero, but tag7bis is better than the other models for Livy and Vitruvius. However, when dealing with a new text, one does not know which model would reach the best result. One will probably have to choose the model with the best accuracy on average.

Model	Ps-Caesar	Cicero	Livius	Vitruvius	average
tag5	96.675	98.407	97.614	93.233	96.765
tag6	96.690	98.407	97.480	92.769	96.622
tag5bis	96.767	98.506	97.586	93.233	96.789
tag5ter	96.767	98.555	97.509	93.434	96.802
tag6bis	96.660	98.374	97.572	93.295	96.749
tag7bis	96.705	98.489	97.663	93.682	96.892

Table 1: Accuracy of the various models for the tags evaluated on the four test texts. The last column is the weighted average. See text for the explanation of the name of the models.

Model	Ps-Caesar	Cicero	Livius	Vitruvius	average
ind5	98.008	99.343	99.162	96.268	98.437
ind6	97.992	99.425	99.148	96.515	98.490
ind5bis_12	98.038	99.359	99.191	96.407	98.484
ind5bis	98.130	99.359	99.205	96.159	98.461
ind6bis_5	97.640	99.409	99.141	96.113	98.342
ind6bis	98.069	99.442	99.099	96.500	98.483

Table 2: Accuracy of the various models for the indices evaluated on the four test texts. The last column is the weighted average. See text for the naming of the models.

In Figure 8, I have plotted the accuracy of the models obtained at the end of every epoch during the blind training on the whole corpus. They are measured for each text. The plot is a graphical representation of data which is very similar to the content of the previous tables. It makes sense to draw these curves as the models are trained on the same corpus and are obtained in successive calculations. The order of the points is determined by the running epochs. In the tables, I have ordered the models with arguments that are arbitrary. Another choice of the order is possible that would have changed the graphical representation: this is why I did not plot the data, preferring the form of tables. The various texts/authors reached very different accuracies in agreement with the supervised models: Cicero has the best result and Vitruvius the worse. As for the previous tables, one can observe that the position of the maximum is different on each curve. Livy’s curve reaches its maximum on the fourth point, Ps-Caesar’s on the sixth, Cicero’s on the seventh and Vitruvius’ on the eighth. However, this would not help and one has to choose the model that performs better on average. It is the fourth one which reaches an accuracy slightly higher than the best supervised model (96.919% vs 96.892% for tag7bis) but this is not really significative (a difference of 9 errors to be compared to a total of a thousand errors).

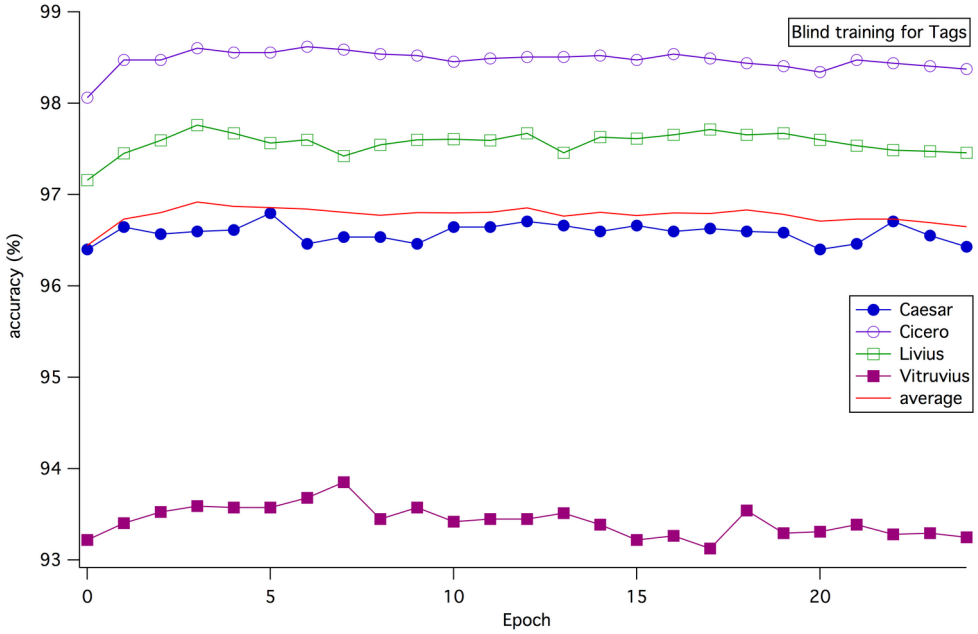

Figure 8: Accuracy of the models successively obtained during training on the entire corpus. The markers are associated to the different texts, while the red solid line is the weighted average of the four accuracies.

PUTTING THINGS TOGETHER

Now that the models are built and characterised, let us use them. As mentionned earlier, the texts have been reconstructed from the APN-files and these text files are then processed, giving APN-files. The original and final files having the same format (line-oriented) the diff tool in Linux can be used[51]. Provided that the files do not have a "<" sign or a ">" sign at the beginning of its lines (which is the case with APN-files), it is easy to know how many lines of the original or final files are different: just counting the number of "<" or ">" signs at the beginning of a line (not the total number of these signs as the APN-files can contain these symbols). The results are summarised in Table 3 for the three texts where the comparison is feasible[52]. The numbers of tokens (lines) in the original and final files are different because of divergence in the tokenisation process. For instance, a Roman number such as "XXII" is lemmatised as two words (with the lemmata *viginti* and *duo*) while the tagger considers that this is just a number. Ps-Caesar's text being rich in figures obtains a huge difference in the number of tokens. In Vitrivius' text, the number of unknown forms is quite high (48 occurrences) because he used some transliterated Greek words and also some single letters associated to drawnings[53].

51 The diff tool compares two files line by line and outputs only the lines that do not coincide. It distinguishes the additions and deletions from a change within the line. It also groups, as a single edit, successive lines that contain a change. But, in any case, a line coming from the original file has a ">" sign prepended, while a line coming from the final file has a "<" sign at its beginning.

52 Livy's original text is in the BPN format which means that two extra characters have been added at the end of the lines. It does not change much for the reconstruction and the processing, but the resulting APN-file cannot be compared to the original BPN file with the diff tool.

53 For instance, *Tunc perducendae sunt diametroe ab G ad L et ab H ad M*.

Number of	Ps-Caesar	Cicero	Vitruvius	Sum
lines orig.	6165	5858	6145	18168
lines final	6147	5854	6142	18143
diff. orig.	455	176	688	1319
diff. final	437	172	685	1294
error rate	7.2%	3%	11.2%	7.3%

Table 3: Number of lines and number of differences in the original and final files for the texts. Due to the tokenisation process, the number of lines may differ in the original and reconstructed files (see text for details about Cicero's text). The last line provides the error rate, i.e. the number of differences divided by the number of lines.

As the detailed comparison between the original APN-file and the final one is done mainly by hand, I shall concentrate on Cicero's text which gave the best results on the sub-tasks. Cicero's text has 5,858 tokens (lines) in the original LASLA file. The resulting file has 5,854 lines. It gives 152 groups of differences, some of them with several successive lines: 176 lines come from the original file and 172 lines from the AI generated file (see the discussion above and note 51). The error rate is thus 3% (=176/5858). The four missing lines are due to one *itaque* that was split into *ita* + *que* and three pairs of words lemmatised either as two tokens or as a single one: *eius modi* and *illius modi* were analysed originally as *is* + *modus* (twice) and *ille* + *modus*, but became lemmatised as the adverbs *eiusmodi* and *illiusmodi*. The opposite case (two lines in the AI-generated file for a single one in the original file) does not occur, as the tokeniser prioritises groups of words. These are "absolute" errors, while for the other 168 lines, one can look at the details (or sub-tasks). The relevant categories are the lemma with its index of homonymy, the code in 9 characters (PoS and morpho-syntactic analysis), the subordination code and the token. The counting is quite tedious, but the numbers of errors are 58 for the lemma, 38 for the token, 89 for the code in 9 characters and 54 for the subordination code. A few errors for the lemma and the form could have been avoided if the first letter in upper case were taken into account to distinguish the proper name from the substantive (which is not the case in Latin-BERT). Probably a few errors are open to discussion. For instance, the LASLA has the lemmata *Appuleius*_N[54] (with 2 "p") and *Apuleius*_N (with 1 "p"[55]) for the same form *Appuleius*[56].

Prospect

The comparison of the result of the process with the gold standard is satisfactory. But in real life, it would help the philologist to know where the errors are. Obviously, if one knows where the errors are, they would not be there. However, some hints telling where one should look first are welcome.

Incompatible predictions

A first hint can come from the incompatibilities between the predictions. The rule-based tokenisation-lemmatisation gives all the known solutions, with its set of possible tags and indices. The AI-script predicts independently the tag and the index. On one hand, these two predictions can be incompatible. On the other hand, the chosen tag and index may not be available in the known lemmatisations. The reconciliation code which puts together the predictions issues "warnings" when an incompatibility is detected. Although they are not used at present, the processing of Cicero's text studied above gave 29 "warnings". 13 of them correspond to errors in the final file. The gain is

54 For proper nouns and their derived adjective, the homonym index are "N" and "A" (sometimes "O" when several proper names share the same form for the lemma, e.g. *Pallas, adis,* f. and *Pallas, antis,* m.).

55 This form with a single "p" is preferred in Gaffiot, while Lewis & Short prefer the double "p".

56 The lemma with a single "p" is present only once and may be a misprint in one of LASLA's files.

small (2‰), but the cost is small too: the philologist has to read 29 sentences (with a focus on a single word) to find 13 errors.

External comparison

Another way to know where to look is to compare two predictions given by different processes. In the present case, the result of the HMM tagger can be compared with the predictions of AI. In the case of Ps-Caesar's text, the diff between the two resulting APN-files emphasises 457 lines. 65 of these lines point to errors in the predictions of the AI. This means that having a glance at 8% of the words in the text, the philologist can reduce the error rate from 3% to 2%.

On the other hand, it occurs quite frequently that discrepancies in the predictions are located in the vicinity of errors, often on each side of it: for instance, there are two errors on lines 100 and 101, while there are discrepancies in lines 98, 99 and 103. It is important to note that this comparison does not allow to detect tokenisation errors (as the tokens are the same) nor the errors in the subordination code (the only prediction comes from AI). However, it can be considered that one has to read the whole sentence to figure out the lemma and the analysis of a single word. In this case, the gain is reduced as the discrepancies are distributed across 167 sentences out of the 231 in Cicero's text (72% of the text).

Internal comparison, weak predictions

During the training and testing of the models, it was noticed that, even if the number of errors in a sub-task is almost constant (see Figure 8 or Table 1), the errors are not always on the same tokens. One can thus imagine to predict the tags, indices and subordination codes with different models, even if they are trained on the same corpus. In this way, one is able to construct a "confidence level" for each prediction, as for a weather forecast. Obviously, most of the predictions coincide. Ideally, one can dream of three "orthogonal" models, always making errors on different tokens so that a majority vote would lead to an exact result. But this is quite unlikely. For instance, one can compare, for Cicero's text, the predictions of the tag5bis and tag6bis models, which are obtained on very similar train/dev files. The number of erroneous tags is 91 for the first model and 99 for the second one. Does it mean that the tag6bis model just made 8 errors more than the tag5bis model? Not at all: the predictions of the two models differ in 54 places. These discrepancies in the predictions can be considered as weak predictions (in other words, with a low confidence level). Note that when both predictions are correct, they are identical. Reciprocally, when they differ, at least one of them is wrong (but they can be both wrong). When comparing these predictions with the gold standard for Cicero's text, one finds out that there are 118 cases where at least one prediction is wrong. However, one of the two predictions is correct in 46 of these cases. Obviously, when dealing with a new text, one does not have the gold standard to compare with, but the examination of the discrepancies in the two predictions would allow to trace back more than one half of the errors. Cicero's text taken here as an example is probably a favourable case, but the 56 discrepancies in the predictions represent less than 1% of the text and their examination by a philologist would allow to correct them, raising the accuracy to better than 99.4%.

A more elegant approach would be that the AI code associates directly a confidence level to the predictions. In their article[57], Bamman and Burns obtain the PoS-tagging prediction « by adding a linear transformation and softmax operation on top of the pre-trained Latin-BERT model, and allowing all of the parameters within the model to be fine-tuned during training ». If I understood correctly, the "softmax" picks up the best choice for the prediction. It would be nice to extract the magic value that is associated with the predicted tag. Even better, it would be more convincing to know also the "second choice" and the two numerical values that have justified the choice. It is not needed that they are real probabilities (i.e. all positive and normalised to unity), but the comparison between the two values would certainly be informative. If the first one is much larger than the

57 D. Bamman and P. J. Burns. « Latin-BERT » cit. n. 6

second one, one can consider that the confidence level is high enough. If the two values are almost the same, then the predictions should be considered weak and the philologist will have to choose the best solution. My knowledge of Python and of the AI modules does not allow me to modify the program to that point.

GUI AND EDITION

Depending whether perfection is required or not, it may be useful that a philologist checks the result of the automatic annotation process. The APN-files are just text files so one can use any text editor to modify them. However, the format is not so easy to edit as the various fields have a fixed length (a CSV-format would be more convenient). The initial version of the tagger, as described by Verkerk *et al.*[58], includes a dedicated editor which can be adapted to the new program. The editor mainly allows the philologist to choose a solution, from the set of possible solutions, which differs from the one selected by the AI. A form as *itaque* can be split into two tokens, if needed. Conversely, the two words *eius modi* which are considered by the tagger as a single token (associated to the lemma *eiusmodi*) can become two tokens with the lemmata *is* and *modus* (and the proper analyses). Renumbering the sentences or changing the references can be also done with the editor.

The GUI also allows a supervision by a philologist when a new text is treated. When an unknown form is found in a new text, the computer asks for its lemmatisation and analysis. This step can be skipped, as it is done with the present automatic program, leaving unknown lemmata which can be identified later. The warnings produced during reconciliation and the discrepancies between the predictions of the tagger and those of AI (or between the predictions of different models) could be highlighted to help the philologist in his/her quest for possible errors. In short, the editor and the GUI would make things easier for the philologist who wants to produce new annotated texts with no errors.

Conclusion

In this article, I mainly present the development of a lemmatiser for Latin texts. It is a hybrid program that uses rule-based lemmatisation and AI to disambiguate the possible solutions. The AI part derives directly from Latin-BERT[59]. The original article about Latin-BERT shows an example where the vector representation of the words allows the distinction between the two homonyms of *cum*. I have tried to repeat the experiment with the inflected forms of the two homonyms of *levis*. The direct manual approaches were not conclusive. However, the use of the PoS-tagging script of Bamman and Burns allows the determination of the index of homonymy with a very high precision, showing that the distinction between the two *levis* is possible although not immediately visible on the vectors.

The error rate of the whole process from a plain text file to the final APN-file is quite low. It goes down to 3% on a text by Cicero which was excluded from the training corpus. If the error rate is not low enough, the planned future developments will include a GUI with an editor to allow the philologist to correct the predictions of the program. On the other hand, the whole process is open for a change in the AI-engine: any AI that can predict the tag, the homonymy index and the code of subordination can be used instead of Latin-BERT, provided that it complies with the file format. For instance, Philipp Roelli[60] is considering to use a variant of the present lemmatiser using LatinCy[61] instead of Latin-BERT.

58 P. Verkerk *et al.* « L.A.S.L.A. and Collatinus » cit. n. 1.

59 D. BAMMAN and P. J. BURNS. « Latin-BERT » cit. n. 6.

60 I thank Philipp Roelli for sharing his article prior to publication in ALMA XXXX.

61 Patrick J. BURNS. « LatinCy: Synthetic trained pipelines for Latin NLP, » https://arxiv.org/abs/2305.04365v1

Acknowledgements

The author wants to thank especially Yves Ouvrard who gave live to Collatinus. Collatinus is a wonderful tool which is included in the tagger. Many thanks also to Dominique Longrée and Margherita Fantoli who introduced me to the field of annotated texts. All the collaborators of the LASLA are also acknowledged for their work that was used here to train the models. A special thank to David Bamman and Patrick J. Burns who developped Latin-BERT and gave a free acces to the code.