Universiteit Leiden

Master Computer Science

Scanning Latin poetry with machine learning

Name:	Luuk Nolden
Student ID:	1370898
Date:	15-07-2022
Specialisation:	Data Science
1st supervisor:	Suzan Verberne
2nd supervisor:	Matthew Payne

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science
(LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

This research addresses the scansion of Latin poetry using machine learning models, in particular neural networks. Having created training sets using the rule-based approaches by the Pedecerto and Anceps projects, we investigate the best way to scan dactylic verse. Subsequently, we investigate the generalisability of a model trained on dactylic meter to other scansion systems, like the iambic trimeter. We find that an LSTM with one-hot encoding outperforms CRF when scanning dactylic meter, with weighted F1-scores of 0.99 for the long, short and elision labels for the former and F1-scores of 0.90 for the latter. Additionally, the models have no problems scanning Latin from different authors, time periods and genres within the same metrical system. The model does require at least 3,000 lines of poetry as training material for a weighted F1-score of 0.98. Generalising the LSTM model trained on dactylic verse to iambic trimeter seems unfeasible, with F1-scores of $\sim$0.75 for the long label, $\sim$0.50 for short and $\sim$0.85 for elision. Using word embeddings trained on syllable level as input to the LSTM does not improve scores. Training both LSTM models on noisy trimeter data (with anceps labels instead of dedicated long/short labels) results in increased F1-scores for the one-hot model, but decreased performance for the embeddings LSTM. This seems to suggest that an LSTM model performs best when trained and tested on a specific metrical system, but that using word embeddings on syllable level does not help with determining syllable length information. When testing on additional meters like the anapest, we find that the CRF model generalises best to other meters with F1-scores of around 0.80. This in contrast to the LSTM models which reach F1-scores of $\sim$0.75 for the long and $\sim$0.50 for the short syllables.

Contents

1 Introduction

Rhythm and sound are essential parts of the content of poetry. Its perception can change by the way rhythmical patterns are presented, which is similar to experiencing different musical styles. Such differences in style are associated with different genres and different expectations about the content of the poem. To illustrate this, we as an audience expect different topics and storylines when hearing a limerick compared to hearing a ballad. This is as true for the modern as it was for the ancient world.

Within poetry, these rhythmical patterns could vary greatly, creating so-called meters, which group the natural sounds of words to create specific rhythms [43]. In Latin, this grouping is based on the quantity of a syllable, with some taking longer to pronounce and others shorter (see Section 2.1). Latin poetry features a wide range of different poetic meters, with each marking out a different genre. Some of these meters feature a simpler rhythm with few metrical variations, while others can have far more variation embedded in a much more complex rhythm. However, the core principles of how the sounds of poetry create rhythms remains the same.

Because a rhythmic meter can convey so much information about a text, the automated scansion (i.e. syllable labeling) of poetry by a computer has been a long-standing goal. Multiple attempts have been successful, like the Anceps [14] and Pedecerto [38] projects, which use a rule-based and a constraint-based approach respectively (see Section 2.3). However, in the case of Pedecerto, new rules have to be created per meter type, which is labour intensive and infeasible for more complex meters. In the case of Anceps, the program tries to force the labels of syllables according to the rules of the given meter, which might be inflexible and unnatural when scanning a line of an unknown meter.

We will therefore try a machine learning approach in this research and investigate its generalisability between meters. We train various machine learning models for sequence labeling on the dactylic meter, consisting of the hexameter and the pentameter, which are considered to be relatively simple rhythms with few variations. Additionally, in comparison to other meters, many dactylic lines are handed down to us, providing us a sizeable training set. We then use these models on more difficult meters and see how they perform. The idea behind this is to see to what extent the models can understand and learn the principles by which the lengths of syllables create rhythm. This resembles how students learn to scan poetry. First, they learn the basic principles and get a feeling for the relationship between these principles and how they work in practice in simpler meters. Later, this skill set is used to scan more difficult meters.

The central question in this paper is therefore as follows: *to what extent are machine learning models, in particular neural networks, able to scan more complex metrical systems when trained on simpler ones?* In other words, to what extent can a neural network learn and apply rhythm and syllable quantity principles?

If this is indeed possible, there are many interesting applications for research within Classics. One exciting possibility is the identification of metrical patterns of Latin fragments. These fragments are lines from Latin plays that do not survive to us in their complete

form, having only been preserved as quotations by other authors. These quotations are often syntactical entities and not metrical ones, making it difficult to identify the meter and therefore to learn from their content within the play. If the model could help us with scanning these fragments, it would teach us more about the fragments and the content of the plays. Another exciting possibility is to use the model to detect new fragments. As mentioned, fragments are handed down to us via quotations. However, scholars believe that many more quotations are hidden in prose texts without any attribution, as a citation from a popular play would be instantly recognisable for the audience. While it is difficult and rather time-consuming for a researcher to search for metrical patterns within the vast amount of prose texts, it would be trivial to let a model scan the texts and flag any passages that could contain metrical patterns. Lastly, if the model is successful in scanning other meters, it would be of great help to students, who would be able to check their scansions with a computer, which is as of the writing of this paper impossible for lesser used meters like the anapest and glyconeus.

Beyond the scope of Classics, a model trained on Latin texts could possibly be adapted to work in other languages. To illustrate, instead of labeling quantity as done in Latin, the model needs to label stress to allow it to function in Dutch or English poetry. And while a rule-based approach would require a rather intense rewrite of the code base, the machine learning approach could accommodate this migration more easily by retraining on a different data set. Additionally, a model that can find and learn rhythm in texts might perhaps be interesting within the field of text-to-speech, allowing for a more human-like speech synthesis in contrast to the often monotone sounding voices we have today.

Regarding the structure of this paper, Section 2 will take a look at the concept of scanning Latin, the application of models for sequence labeling and related work within the fusion of these two fields. Section 3 will relate about the data sets that were created to allow the training of the models related in Section 4. After this preliminary information, Section 5 will investigate the best approach to scanning dactylic meter and the subsequent generalisability towards the more difficult iambic trimeter. At the end of this chapter, the same generalisability will be tested for the anapest, tetrameter, dimeter, glyconeus and hendecasyllable. We will then move onto the discussion in Section 6 before closing with the conclusion in Section 7.

2 Background

2.1 Latin scansion

Before the written word, epic works like Homer's *Iliad* and *Odyssey* were handed down orally. To allow for easy remembering and colourful narrating, all syllables in a line of poetry were organised in patterns based on the length of their pronunciation [9]. Such a pattern is called a meter and gives each line a rhythm. Furthermore, the meter indicates the style of the poetry. For example, the dactylic hexameter was grouped under the generic label of 'epic', considered an apt meter for weighty and important matters[1] [24] [35].

In Latin poetry each syllable was either pronounced long, short or not at all, called elision. We denote long syllables with ¯ and short with ˘. Elisions are denoted by the character ⏑ following the syllable that is elided. In the dactylic hexameter, a verse consists of six feet, which are made up of dactyls or spondees [13, pp. 85–86]. Here a dactyl is a foot consisting of three syllables, the first being long, followed by two short syllables. The spondee is a foot created from two long syllables. The sixth foot always consists of a spondee or a trochee (a long syllable followed by a short one). Lastly, the fifth foot is most often a dactyl, though exceptions are possible.

It is then up to the poet to turn these building blocks into rhythmic and interesting poetry. One example of this employment is Vergil's *Aeneid* 8.596.

quādrŭpĕdāntĕ pŭtrēm sŏnĭtū quătĭt ūngŭlă cāmpum

This sentence, translated as *a hoof shakes the crumbling field with a galloping sound*, describes the movement of running horses. From a metrical standpoint, the line is made up of five dactyls and a closing trochee: the many dactyls and their short syllables imitate the quick movement made by the horses.

An opposite effect is also possible, like in line 452 of the same book. Here Vergil describes the forging of Aeneas' shield by the sons of Vulcan, who *lift their arms with great strength one to another*:

īlli⏑īntēr sēsē mūltā vī brācchĭă tōllunt

This time, the verse consists almost completely of spondees, except for the usual dactyl in the fifth foot, mimicking the slow and labour intensive pounding sound of the work.

Each verse of poetry therefore consists of long and short syllables. If the syllable contains a short vowel or a long vowel, its quantity is determined by its *nature*: in this case, the syllable is long or short because it contains a long or short vowel. To illustrate:

- A short syllable contains a short quantity vowel: *nŏvem, păter.*
- A long syllable contains a long quantity vowel: *vīta, māter.*

[1] Because of its association with epic, it is very probably that the dactylic hexameter came to be used for quite different kinds of poetry like satire, as we will see with the texts of Iuvenal and Persius in Section 3.

There are general rules to determine the nature of syllables. For example:

- A diphthong, *ae, au, ei, eu, oe, ui*, is always long. A vowel derived from a diphthong is also long: *exclūdō*, from *ex-claudō*.
- A final *-am*, *-em*, or *-um* is always short by nature.
- A final *-o*, *-i*, or *-u* is usually long by nature.
  - However, the final *-i* in *tibi* and *mihi* could be long or short according to the requirements of the meter.
- A final *-a* or *-is* is often short by nature.
  - However, *-a* in the first declension ablative singular is always long, as is *-is* in the first and second declensions dative and ablative plural.

The length of a syllable can also be determined by its position in the verse. A vowel is considered *long by position* when it is directly followed by two consonants. To illustrate, the *u* in *ŭrbs* is short by nature. However, it is long by position as it is followed by two consonants, and should be scanned as such in the verse. The consonants can also belong to the next word: in the case of *puēllā stat*, the *-a* of *puella* is short by nature (in nominative case). It is however long by position because of the consonants that follow. This is however not an absolute rule: if the first consonant is a mute (like *c* or *p*) followed by a liquid (like *l* or *r*), the preceding vowel will not automatically be long, but might be treated as such at the discretion of the poet.

Further points of interest are *-h-* being not considered a full consonant, the consonants *-x-* and *-z-* lengthening a preceding vowel, as they are double consonants (*-ks-* and *-ds-*, respectively), and *-i-* being sometimes read as the consonant *-j-*.

Regarding elision, there are two basic rules:

- If a word ends with a vowel, this vowel may be omitted if the next word starts with a vowel or an *h-*.
  - In *nocte egeris*, the final *-e* is elided, resulting in *noct⌣egeris*, pronounced as *noctegeris*.
- If a word ends with *-m*, the final syllable may be elided if the next word starts with a vowel or an *h-*.
  - *Pridem oportebat* would become *prid⌣oportebat*, or *pridoportebat*.

It is important to note that these elisions may or may not occur. Deliberate avoidance of elision is called hiatus, which is possible, although frowned upon[2].

[2]For more information on syllable quantity, including rules and exceptions, see [6, pp. 5–6],[13, pp. 12–14], [43] and `http://www.thelatinlibrary.com/satire/scansion.pdf`.

In addition to the hexameter, this thesis also investigates elegiac couplets and iambic trimeter. The former is a poetic meter which could be used for various themes of a smaller scale than epic [43, pp. 103–109]. For example, authors could write about their own love affairs, or feature as a character in their own stories. Other uses would be elegy (reflection and lamentation) and epigrams. Regarding its meter, elegiac couplets consist of sets of one dactylic hexameter followed by a dactylic pentameter. This pentameter is similar to a hexameter, having two patterns of two dactyls[3] followed by a *longum* (single long syllable) [13, pp. 109–115]. In general, the hexameter flows into the pentameter via an enjambement[4], forming a contrast from the often rising action of the first verse with a falling quality in the second. One great example to demonstrate this is in Catullus' 85$^{\text{th}}$ poem:

> *ōdi⏑ĕt ămō. Quāre⏑īd făcĭām, fōrtāssĕ rĕquīris?*
> *nēscĭŏ, sēd fĭĕrī sēntĭo⏑ĕt ēxcrŭcĭōr.*
>
> *I hate and I love. Why do I do this, perhaps you ask?*
> *I know not, but I feel it happen and am tormented.*

Lastly, this paper will treat the iambic trimeter, which was the most common meter to be used for spoken parts of tragedy, comedy, and satyr plays [13, pp. 136–138]. Each line consists of three iambic metra, with each metron consisting of the pattern *anceps, long, short, long*. Here, any syllable labeled as *anceps* could either be long or short, depending on its nature and position. Furthermore, the meter allows for resolutions, turning any long or anceps syllable into two short syllables if desirable. To illustrate, see the famous lines from Seneca's *Medea*, 170–171, in which the nurse pleads Medea not to kill her children:

> *mŏrĭērĕ – cŭpĭō – prŏfŭgĕ – paēnĭtŭīt fŭgae*
> *Mēdēă – fīām – mātĕr ēs – cuī sīm, vĭdēs*
>
> *You will die – I desire so – Flee! – I have repented of flight*
> *Medea! – I will become her – You are a mother! – You see by whom*

The first anceps has been resolved to two short syllables: *mŏrĭ*.

2.2 Models for sequence labeling

The previous section has demonstrated how every syllable in a verse has a length. This length can also be viewed as its label: *long*, *short* or *elision*. In natural language processing (NLP) terms, the task at hand is one of sequence labeling, where every syllable in the verse gets a label based on its natural or positional length.

Traditionally, sequence labeling has been using linear statistical models, such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF) [30] [37] [42]. Although

[3] Only the first two dactyls can be substituted by spondees.

[4] In practise, there is never a full stop within the hexameter: it flows semantically over into the pentameter.

achieving good results on various tasks, these methods rely heavily on hand-crafted features and task-specific resources. For example, part-of-speech taggers for the English language benefit from carefully designed and handcrafted word spelling features. These are however costly to develop and, as mentioned, task-specific [32], making these models difficult to adapt to new tasks or new domains [31].

In recent years, non-linear neural networks have been broadly applied to NLP problems with great success. To illustrate, Collobert *et al.* [11] proposed a simple but effective feed-forward neural network that independently classifies labels for each word by using contexts within a window of fixed size. More recently, recurrent neural networks (RNN) [19], together with its variants such as the long short-term memory (LSTM) [17] [22] and gated recurrent unit (GRU) [10] have shown great success in modeling sequential data.

In contrast to the features used by CRF models to label sequences, these non-linear neural networks capture latent syntactic and semantic similarities between input sequences to allow for successful labeling [3]. The input can, amongst others, be represented using one-hot (integer) encoding, character encoding or word embeddings. These embeddings, also known as word vectors, are trained on unlabeled data sets to capture the meaning of words in a text corpus, which can assist in learning and generalisation. According to Akbik *et al.*, three types of embeddings exist [3, p. 1638]:

- Classical word embeddings [33] [39]. These are embeddings that are pre-trained on large data sets to capture latent syntactic and semantic similarities over the entire corpus.
- Character-level features [28] [31]. These are trained at runtime on task data in order to capture task and text specific subword features.
- Contextualized word embeddings [12] [40] [41]. These capture word semantics within the context in order to detect and capture the polysemous and context-dependent nature of words.

It is important to note that although these neural networks often outperform their feature based counterparts, it is possible to synergise them, allowing augmentation rather than replacement. Think for example about using hand-crafted features such as word spelling and capitalisation patterns in combination with non-linear neural networks. For example, hybrid approaches for sequence labeling can use a bidirectional LSTM fed with aptly trained word embeddings and a subsequent CRF decoding layer [23] [31] for better performance.

2.3 Related Work

There is a small amount of prior work on the scansion of Latin, the first being Pedecerto, which is a rule-based Python program for automatic scansion of Latin hexameter and pentameter verses developed by the Università di Udine [38]. It is part of the MusisQue DeoQue (MQDQ) digital archive [36], which contains Latin poetry texts from the archaic period to the seventh century CE. From this archive, it has successfully scanned 247k of

the 263k dactylic verses available as of the time of this writing. These scansions allow scholars to find and research verses based on metrical features (like hiatus and synalepha) and metrical patterns.

Anceps is another computer-assisted tool by A. Fedchin [14] for scansion of Latin poetry, differing from Pedecerto by using meter as a constraint. It does this by considering all the possible ways the syllables in a verse might be labeled in order to be consistent with the meter. To achieve this, a dictionary of natural vowel quantities is used to have a ground truth about syllables in specific words. These dictionaries can be specified by the user to be period and author specific frequency-based dictionaries. For example, using the mentioned MusisQue DeoQue database, one can build such a dictionary from the golden age poets Horace, Ovid and Vergil. These authors usually scan the *a* in *patris*, the genitive of *pater*, as short, which Anceps will keep in mind when scanning new verses. Additionally, Anceps can assign confidence scores for each scanned verse using scansion frequencies retrieved from MQDQ. The program can furthermore be extended to scan other verses, although its main focus is with iambic trimeter. Lastly, the program will not label all ancipitia in a verse, leaving it up to the user to fill these in using their knowledge of the Latin language.

Regarding the use of neural networks for scansion, Haverals, Kestemont and Karsdorp use a recurrent neural network (RNN) in their 2019 paper *Rekenen op Ritme* to scan Dutch accent based iambic meter from the *Nederlandse Liederenbank*, which is a database of 50k songs from Dutch literature between the sixteenth and eighteenth centuries [21]. To disallow any absurd scansions, Haverals *et al.* use a semi-supervised method by utilising an automated syllabification tool in combination with a self written automatic lexical accentuation program. The idea being that the predicted accentuation must not be too different from the natural accentuation of a word. With these constraints, the RNN was trained and tested on 198 songs from various time periods and authors. Using binary decisions on which syllables are accentuated or not, the model reached an overall accuracy of 91,91% on syllable level. Similar projects to scan modern languages are *ZeuScansion* [1] and *Scandroid* [20] for English, *Metricalizer* for German [7], *Scansion Generator* [26] for Dutch and *Aoidos* [34] for Portuguese.

Methodologically related is the 2019 paper *Restoring ancient text using deep learning* by Assael *et al.* [5], which uses a bidirectional LSTM to recover missing characters from damaged Ancient Greek inscriptions. The model makes it possible to handle long-term context information and is efficient in dealing with missing or corrupted characters and/or words. To train the model, Assael *et al.* use the PHI database, the largest digital corpus of ancient Greek inscriptions [25]. From this database fully preserved texts are manually corrupted to allow for training and testing. Having provided a concatenation of word and character embeddings, the program, named Pythia, achieves a 30% character error rate, compared to the 57% of human epigraphists.

3 Data

3.1 Pedecerto dactylic meter data set

The first data set used for this thesis is created via the Pedecerto project as mentioned in Section 2.3. From the project website it is possible to download XML files of scanned dactylic poetry. We use the authors and texts seen in Table 1[5].

Author	Text	Verses	Meter
Iuvenal	*Saturae*	3,833	H
Lucretius	*De rerum natura*	7,365	H
Ovid	*Metamorphoses*	11,983	H
Persius	*Saturae*	649	H
Vergil	*Aeneid*	9,840	H
Propertius	*Elegiae*	3,998	E
Ovid	*elegiae*	18,832	E
Boethius	*De consolatione philosophiae*	76	E
Catullus	*Carmina*	795	E
Ennius	*Annalium fragmenta*	422	H
Horace	*Ars poetica*	476	H
Lucanus	*Pharsalia*	8,059	H
Statius	*Thebais*	9,739	H
Tibullus	*Elegiae*	1,420	E

Table 1: List of dactylic texts and their lengths in number of verses: *H* denotes hexameters, *E* elegiac couplets. Ovid's *elegiae* are all his texts using elegiac couplets.

The first five authors were selected to serve as training texts for our hexameter models, as these texts have various lengths, different genres and time periods, giving a representative yet not too temporally far apart sample of classical Latin poetry (see Table 5). The impact of these variabilities will be tested in Section 5. To test generalisability to elegiac couplets and train specialised elegiac models, the *elegiae* from Propertius and Ovid were selected. These texts are lengthy examples of elegiac verse and therefore great candidates to train a model on. The seven texts listed last are used as unseen test sets to evaluate the models we created (see Section 5.1.6). These are therefore again of various lengths, meters, genres and time periods.

To allow the sequence labeling models used in this paper to read the scansions encoded in the Pedecerto XML files, the texts were converted into a syllable-label list. In the XML, every line of poetry has entries for every word, with each one having a scansion. For example, the second word of Vergil's *Aeneid*, *virumque*, has the scansion `1c2A2b`. Here, integers denote the foot and letters indicate the position within this foot. Lastly, uppercase is used for long syllables and lowercase for short ones. To illustrate, the first syllable *vi* has the encoding `1c`, which means that it is part of the first foot (`1`, *ārmă vĭ*), of which it is the third syllable (`c`, the third letter of the alphabet). As the letter is

[5]Corrupted or problematic lines as indicated by Pedecerto were not included to allow for the smooth training of the models.

lowercased, we know that the syllable *vi* is short. Now, as we only need the length labels, integers are removed and the letters are converted into a length list based on their capitalisation. In the case of *virumque*, we will get the following list: [‘short’, ‘long’, ‘short’]. Using Pedecerto’s syllabifier, *virumque* is syllabified into [‘vi’, ‘rum’, ‘que’][6]. Next, the two lists are combined into a list of syllable-label tuples, which looks as follows: [(‘vi’, ‘short’), (‘rum’, ‘long’), (‘que’, ‘short’)]. This is then repeated for every word of every verse of a text, with the result being saved to disk for later use.

To illustrate, the first line of the *Aeneid* can be seen in Table 2. Spaces are included to keep track of word boundaries and are encoded as ‘-’ and labeled as *space*. Because this label is predicted with 100% accuracy by the models in this research, we do not include its scores in the results of Section 5.

Syllable	Label
ar	long
ma	short
-	space
vi	short
rum	long
que	short
-	space
ca	short
no	long
-	space
tro	long
iae	long
-	space
qui	long
-	space
pri	long
mus	short
-	space
ab	short
-	space
o	long
ris	long

Table 2: Syllable-label list of the first line of Vergil’s *Aeneid*.

3.2 Anceps iambic meter data set

The second data set used is created from the Anceps project [14] and contains verses in the iambic trimeter. More specifically, we will use the data set created and curated by

[6]To label elisions, we must notice that the elided word has one more syllable in text form than in the scanned form, since the scansion approximates oral delivery. For example, *ille* might have the scansion `2A`, meaning that *il* is long and that *le* is elided. The syllable *le* will therefore be labeled as *elision*.

Fedchin *et al.* for their 2022 paper *Senecan Trimeter and Humanist Tragedy* [15], which contains (amongst other passages) the proofread Anceps scansions of all ten plays attributed to Seneca Minor. As briefly mentioned in Section 2.3, the program Anceps does label some syllables as *anceps* instead of *long* or *short*. Since we do not want our models to predict this label, we consider this data set noisy[7].

Like the scansions made by Pedecerto, we converted those made available by Anceps to syllable-label lists. In contrast to Pedecerto, Anceps provides scansions as follows (Seneca's *Agamemnon* line 392):

```
de*lu_br(a) e^t a_ra*s c[ae]li^t(u)m e_t pa^tri^o_s la^re*s
```

In order to create the syllable-label list, the provided string was split on whitespace. We then extract special characters from every word and convert these to labels. To illustrate, we extract `*_()` from `de*lu_br(a)`, meaning ['anceps', 'long', 'elision'][8]. Next, the remaining word *delubra* is syllabified using the CLTK syllabifier[9]. The syllables and scansions are then combined into tuples, which form the syllable-label list of the entire text as described in Section 3.1. The texts seen in Table 3 were used to create the Anceps data set[10]. As these tragedies contain more than one meter type, only those lines with the iambic trimeter were selected.

Although it is possible to train models on this noisy data set, we cannot use it for testing. We want our models to predict whether a syllable is long or short: predicting the anceps label is not interesting for our use case. We have therefore manually created a dedicated test set containing around two hundred iambic trimeter lines of Seneca's *Agamemnon* (392–588)[11]. Taking the proofread data set by Fedchin *et al.*, we resolved the ancipitia using the dictionary and the rules of the iambic trimeter. This resulting noiseless data set will be used to test our models. To prevent overfitting, these two hundred lines were removed from the noisy *Agamemnon* training text.

[7] From a qualitative standpoint, this data set is not noisy at all. An anceps simply means that a syllable can be either long or short according to its length and the meter. One would have to consult a dictionary to find out whether the syllable is long or short by nature. For example, the *-us* in *tellus* is always long, so an anceps on this syllable would resolve to long if the meter allows it. However, from the standpoint of our machine learning models, the anceps is noise.

[8] Square brackets denote diphthongs, which are always long. See *caelitum*.

[9] The Classical Language Toolkit: see `http://cltk.org/`. This syllabifier differs ever so slightly from the one used in the Pedecerto project, which is an improvement upon the former, though not available for plain text. The trimeter test set (see last paragraph of this section) is syllabified by hand as the Pedecerto project would to prevent inconsistencies between data sets. Differences in syllabification between the Anceps and Pedecerto data sets include for example *troi-ae* versus *tro-iae* and should be inconsequential to the models.

[10] The proofread data set by Fedchin *et al.* only contains Seneca as a classical author. The other texts scanned are those by neo-Latin authors. These were not included in this thesis, as we focus on classical texts and authors.

[11] These lines were chosen because their continuity, being from one single messenger story, and their difficulty, containing names and various infrequently used nautical terms.

Author	Text	Verses	Meter
Seneca	*Agamemnon*	394	T
	Hercules Furens	913	T
	Hercules Oetaeus	1,253	T
	Medea	591	T
	Octavia	532	T
	Oedipus	637	T
	Phaedra	829	T
	Phoenissae	578	T
	Thyestes	656	T
	Troades	810	T

Table 3: Table of iambic trimeter texts and their lengths in number of verses.

4 Methods

As demonstrated in Section 2.1, we need to label a sequence of syllables. In this thesis we will experiment with three different machine learning methods to achieve this.

4.1 Conditional Random Fields

The first approach uses Conditional Random Fields (CRF), proposed by Lafferty *et al.* in 2001 as an improvement on their Maximum Entropy Markov Models (MEMM) [27]. As their paper states, *the critical difference between CRF and MEMM is that the latter uses per-state exponential models for the conditional probabilities of next states given the current state, whereas CRF uses a single exponential model to determine the joint probability of the entire sequence of labels, given the observation sequence.* This means that, in contrast to MEMM where the probability of the next state is computed given the current state and the observation, CRF is able to compute all state transitions globally in a single model. Additionally and in contrast to MEMM, the user can specify features, which, using their weights, can compete against each other in the various states.

For the scansion of Latin, CRF allows for a window to be specified. Sliding this window over every verse of every text will allow the model to get a sense of the rhythm in the text. We specify this window as follows:

$$\text{previous syllable} \leftrightarrow \text{current syllable} \leftrightarrow \text{next syllable}$$

This sequence should encapsulate most of the direct influences syllables have on each other in Latin poetry. For example, if the current syllable ends with a consonant and the next syllable starts with one, we know that the current syllable should be labeled as *long* (see Section 2.1). The CRF is further aided by hand-crafted features, which will be explored in Section 5.

4.2 One-hot LSTM

The second approach uses the long short-term memory (LSTM) model, which was introduced by Hochreiter and Schmidhuber in 1997 [22]. It is an improvement over recurrent neural networks (RNN) in the sense that it can remember which parts of a context are important for the long and the short term. This is an interesting feature for scansion, as some labels are purely dependent on the place of their syllable within the verse (see Section 2.1). Additionally, as mentioned in Section 2.2, an LSTM model captures latent syntactic and semantic similarities between input sequences to allow for successful labeling. This latent-based approach contrasts nicely to the feature-based approach of CRF.

Because an LSTM does not allow the input of character strings, we will first try a model where all syllables are one-hot encoded (resulting in a sparse matrix). To illustrate, the first part of Vergil's *Aeneid* would look as follows:

ar - ma - space - vi - rum - que - space - ca - no

10 - 12 - 3 - 26 - 18 - 19 - 3 - 11 - 15

Furthermore, as the LSTM requires all input to be of the same length, which verse clearly is not, post padding was used to make all lines of even length. Regarding the structure of the one-hot LSTM, an embedding layer with fifty output dimensions was used, followed by a dropout of 0.1 to prevent overfitting. Subsequently a bidirectional LSTM was implemented with a recurrent dropout of 0.1. The model was fitted via a softmax activation and the *rmsprop* optimiser, using a batch size of 32.

4.3 Embeddings based LSTM

The third approach uses an LSTM like described in the previous section, but uses embeddings as input instead of one-hot encoded integers. Intuitively, encoding syllables as integers loses information. For example, encoding *ar* as 10 and *ma* as 12 does not show the first syllable ending with a consonant and the second one starting with one, indicating that *ar* should be long. We therefore extend the previous approach by using an LSTM with word embeddings as input. To achieve this, we employ the Flair NLP framework by Zalando [4], which allows us to easily apply natural language processing models. For example, Flair supports named-entity recognition, part-of-speech tagging and sense disambiguation for various modern languages. Interesting for our use case is their text embedding interface, which allows us to test and combine different word embeddings and use these as input for an LSTM.

In contrast to using the less informative integers as input, word embeddings are dense vector representations of words, which are trained based on word usage in a text. This means in essence that similar words have similar vectors to represent their similar meaning. Furthermore, these vectors are of lower dimensionality than the sparse one-hot dictionary used in the previous section. This is beneficial for the majority of neural network architectures [18], as these dense vectors trained on large amounts of text are a richer representation of words than only the words themselves.

Regarding embeddings, the main question we address in this thesis is to find out whether these word embeddings and the generalisation they bring can also work for syllables. We might be able to know a word by the company it keeps [16], but does the same hold for syllables? In other words, does the company of syllables convey latent information about their lengths? We tackle this question in Section 5.2.2 and onwards.

4.3.1 Embedding types

As briefly mentioned in Section 2.2, multiple ways of training word embeddings exist. In this paper, three different embeddings are experimented with:

CharacterEmbeddings are character-level embeddings of words, which are able to capture low level morphological and orthographic information [28, pp. 260–261]. This sensitivity to lexical aspects within words derives from the use of characters as atomic units [29, p. 1528], which is interesting for our use case: the combination of characters conveys information about the length of the syllable. Another benefit is the compactness and simplicity of the model, as it stores only one vector per unique character. Its main

difference from traditional word representation models is therefore having only a single vector per character type, which are concatenated together to form word embeddings [29, pp. 1521–1522]. We use the implementation provided by Flair to create these embeddings.

FastTextEmbeddings are word embeddings with subword features [8]. Here each word is represented as a bag of character n-grams, with each character n-gram being represented by a vector (comparable to the Skip-gram model). Word embeddings are then represented as the sum of these representations. In essence, FastText learns word representations by taking subword information into account, which allows for learning the specific morphology of words. In addition to the subword feature detection, FastText can handle unknown words/syllables, as their embeddings are created from their substrings.

FlairEmbeddings are contextualized character-level embeddings [3]. To create these, a character language model is taken to model words as sequences of characters, without any explicit notions of the words themselves. Next, these character embeddings are put together and contextualised by their surrounding text, which allows for the same word to have different embeddings depending on its use in the context. Akbik *et al.* found that these embeddings captured syntactic-semantic word features well and could disambiguate words in context [3, p. 1647], which would be apt for our use case. A syllable is for example long by position based on its context.

In order to train these three types of embeddings, we create one large text from all the syllabified training texts (first seven texts from Table 1). Here, we represent syllable boundaries with a whitespace and word boundaries with a hyphen. For example, the first three lines of the *Aeneid* would look like this:

> ar ma - vi rum que - ca no - tro iae - qui - pri mus - ab - or is
> i ta li am - fa to - pro fu gus - la vi ni a que - ve nit
> li to ra - mul tum - il le - et - ter ris - iac ta tus - et - al to

5 Analysis and Results

This chapter will consist of three main parts. In the first part, a CRF and one-hot LSTM model will be trained and tested on dactylic verse to examine whether hexameters and pentameters can be scanned and whether different training size, author, genre and time period influence results. The second part will focus on scanning iambic trimeter using the one-hot LSTM and LSTMs with different combinations of word embeddings as input. This is done by training the models on dactylic and on iambic trimeter. The third part tests the best performing models on small test sets of five additional meters to get a feeling for the accuracy of the different models.

The quality of labels is given by their F1-scores. Here, we call a score of 0.85 or higher *satisfactory*, 0.95 or higher *good* and scores of 0.98 or higher *excellent*.

5.1 Scanning dactylic meter

Recalling the idea of this thesis as written in the Introduction, we will first train models on the dactylic hexameter. If a model is able to scan this meter well, we will investigate its ability to scan the similar dactylic pentameter. The best model will then be used to scan iambic trimeter in Section 5.2.

5.1.1 CRF and one-hot LSTM

The first question that needs answering is to what extent machine learning is capable of scanning Latin poetry. To find out, we picked Vergil's *Aeneid* as a testing ground, which is one of the longest continuous pieces of poetry from the classical Latin period. The text consists of around 10,000 hexameter lines with little over 150,000 syllables. On this text, we train and test a CRF and one-hot LSTM model.

Tabel 4 shows the F1-scores[12] for the CRF and one-hot LSTM models after k-fold cross-validation ($k = 5$). As is clearly visible, the CRF model achieves satisfactory results of 0.927, 0.880 and 0.951 for the *long*, *short* and *elision* labels respectively. It is however completely outclassed by the one-hot LSTM model[13], which has excellent F1-scores of 0.996, 0.993 and 0.990 for the aforementioned labels.

	Long	Short	Elision
CRF	0.9269	0.8800	0.9507
One-hot LSTM	**0.9956**	**0.9929**	**0.9895**

Table 4: Model quality (F1-scores) of the CRF and one-hot LSTM models on Vergil's *Aeneid* using k-fold cross-validation ($k = 5$).

[12]We show the F1-scores in this and following experiments, as the precision and recall scores were extremely similar for the one-hot LSTM on the dactylic meters. The CRF shows more variation: *Long*, precision: 0.9402, recall: 0.8924. *Short*, precision: 0.8416, recall: 0.9119. *Elision*, precision: 0.9335, recall: 0.8788.

[13]The LSTM model was trained for 25 epochs. Its structure is given in Section 4.2. Experiments with this structure such as additional layers or more nodes did not yield better results.

As mentioned in Section 4.1, a CRF model can be enhanced with features. In addition to solely providing the previous, current and next syllables, two enhancements were tested. First, the last and last two characters of the previous and current syllables were added as features, as well as the first and first two characters of the next syllable. These were specifically featured like this, because these two letters, either vowel or consonant, have the most influence on the label of any syllable. To illustrate, the feature dictionary of *qui* from the first line of the *Aeneid*, *arma virumque cano troiae qui primus ab oris*, looks as follows:

1. *qui*
    (a) last character current syllable: *i*
    (b) last two characters current syllable: *ui*
    (c) first character next syllable: *p*
    (d) first two characters next syllable: *pr*
    (e) last character previous syllable: *e*
    (f) last two characters previous syllable: *ae*

In this line, feature (d) results in *qui* being long by position (two consonants in the next syllable), which should be useful information for the CRF.

The second enhancement specifies whether a syllable contains a diphthong, which is always long in Latin poetry. Additionally, a boolean was specified whether the first or last syllable was a consonant. In all three cases, the first and last syllable of a sentence were labeled with the beginning-of-speech (BOS) and end-of-speech (EOS) tags respectively.

The first enhancement of featuring first and last characters of syllables outperformed the base model slightly, with around 0.01 to 0.02 improvement in F1-score for all labels. Interestingly, the addition of the diphthong and consonant features decreased quality slightly. Therefore, the best CRF model we could create contains the BOS and EOS tags, the entire syllables and the first enhancement as described.

Therefore, to conclude this subsection, the one-hot LSTM model clearly outperforms the CRF model in terms of F1-scores on Vergil's *Aeneid*. We will thus continue with the LSTM model in the next sections.

5.1.2 Effect of different genres and time periods

As seen in the previous section, the one-hot LSTM model is capable of scanning Vergil's epic from the first century CE. But what about other authors, genres and time periods? To evaluate this, four additional authors were selected and tested with the one-hot LSTM. Like before, the LSTM model was evaluated on the text of the given author using k-fold cross-validation ($k = 5$) and 25 epochs. From Table 5 we learn that the one-hot LSTM is able to achieve good to excellent F1-scores for all labels over all authors (excluding Persius), genres and time periods. To illustrate, the quality of the Iuvenal model does not differ too much from the one of Vergil, although being from a much later period and an entirely different genre. However, as mentioned, there is one outlier: Persius' model

Author	Title	Genre	Period	Verses	F1-score		
					Long	Short	Elision
Iuvenal	*Saturae*	Satire	2^{nd} CE	3,833	0.9812	0.9695	0.9490
Lucretius	*De rerum natura*	Epic	1^{st} BCE	7,365	0.9943	0.9910	0.9842
Ovid	*Metamorphoses*	Epic	1^{st} (B)CE	11,983	0.9963	0.9952	0.9829
Persius	*Saturae*	Satire	1^{st} CE	649	0.9135	0.8458	0.4382
Vergil	*Aeneid*	Epic	1^{st} BCE	9,840	0.9956	0.9929	0.9895

Table 5: One-hot LSTM weighted average F1-scores for different authors, genres and time periods.

performs much worse, especially on the elision label. Since his text has the shortest length, we will investigate the impact of training set size in the next section.

5.1.3 Effect of training set size

As seen in Table 5, a model trained on the text of Persius scores relatively low with F1-scores of 0.91, 0.85 and 0.44 for the three labels. This might be explained by the number of lines in the training set. Notice for example how the second lowest scores are those by the Iuvenal model, which is trained on the second smallest training set. A valid question is therefore what training size an LSTM needs for good results. In other words, how does the quality of a model develop with the training set size? To answer this, all texts were normalised to 3,600 randomly picked lines (except Persius of course). For every text a test set was created of 720 lines via an 80/20 split. Then, starting with a small training set of 100 lines, models were trained for 25 epochs and tested on the test set. When finished, the training sets were repeatedly increased with 100 lines and tested again, until all potential training lines were used.

As seen in Figure 1, a minimum of 1,000 lines is needed for an F1-score of 0.95. Furthermore, a training set of 3,000 lines seems to be the threshold for a quality of 0.98. The main takeaway is that more lines in the training set is advantageous for the model, as seen by the high scores of the larger texts in Table 5.

5.1.4 Cross author evaluation

In this section we address the question to what extent a model trained on one author or text is able to scan the lines of another author or text. In Figure 2 three plots are shown for each of the three labels. In each plot the F1-scores are given to show the model quality of the given predictor (y-axis) on the given predictee (x-axis). For example, if we train a model on the entire text of Iuvenal and let it predict the verses of Lucretius, we see F1-scores of 0.9774 for the long label and 0.9618 and 0.9267 for the short and elision labels respectively. For the diagonals the cross-validated results from Table 5 are used: training on Vergil's entire *Aeneid* and testing the model on that same text would after all return overfitted results.

Two things are worth discussing here. First, a model trained on a sufficiently large number of lines is capable of scanning not only the training author, but also other texts,

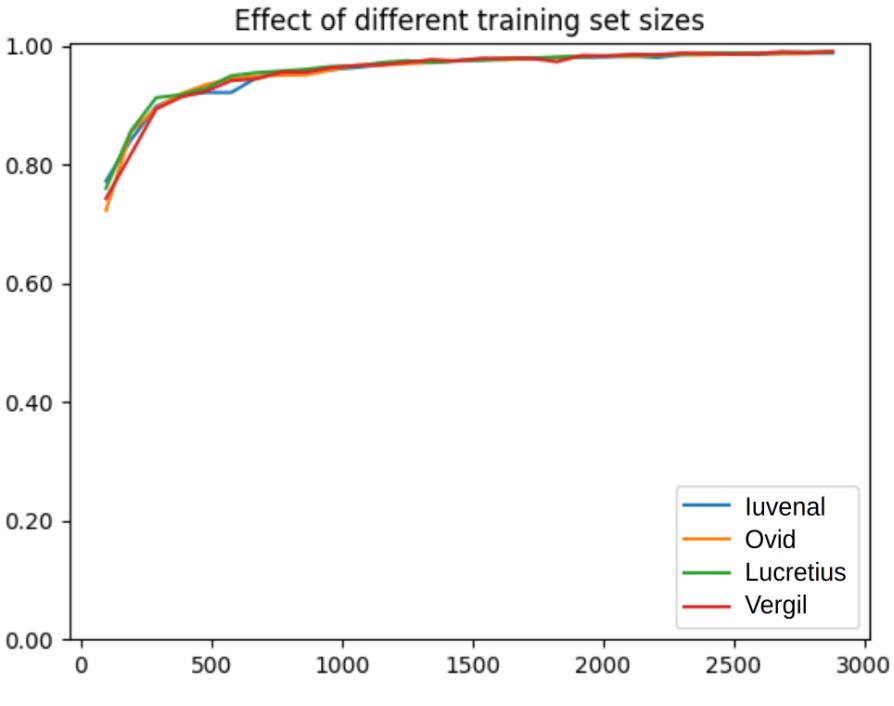


Figure 1: One-hot LSTM F1-scores for different training set sizes on test sets of 720 lines per author.

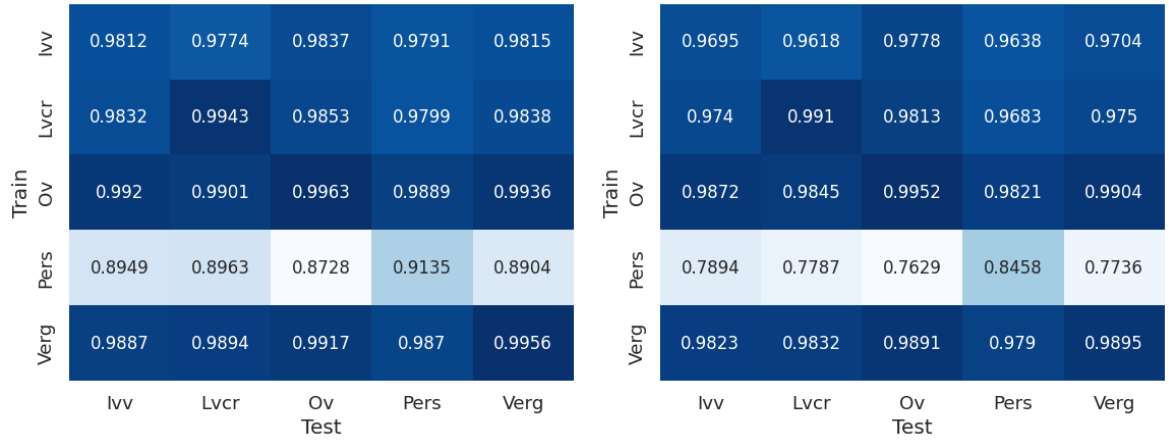
independent of genre and time period. Although the F1-scores are highest within the same author, the difference overall is not higher than 0.03. For example, Vergil predicts the long label for Ovid with a quality of 0.9917, which is almost as good as Ovid predicting itself (0.9963, see Table 5). Second, we noticed in the previous two sections that a model cannot be trained on the small number of lines from Persius. However, the other models are performing beautifully on Persius' text, with each model being able to scan all Persius' labels with a good quality of more than 0.95. This in contrast to the Persius model itself, which has great difficulty predicting the short and elision labels of other texts.

Therefore, in conclusion, we do not need a large quantity of text from a specific author in order to scan their text accurately: within the hexameter meter, we can use a model trained on another larger text, independent of time and genre.

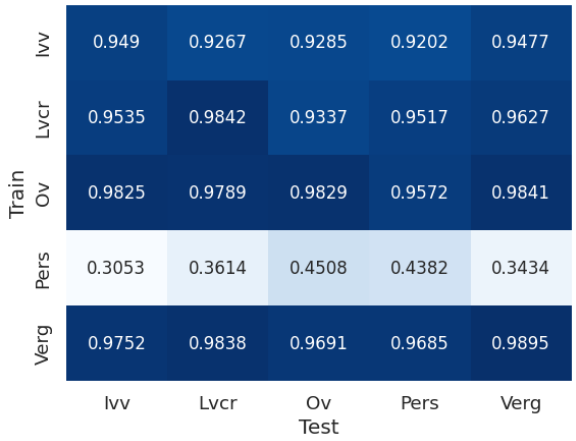
5.1.5 Generalisability to elegiac couplets

In the previous sections we successfully trained an LSTM to scan hexameters. To investigate the model's generalisability to other meters, we will now examine elegiac couplets, which consist, as related in Section 2.1, of pairs of a hexameter verse followed by a pentameter verse. We ask two questions. First, to what extent is the one-hot LSTM model able to scan an elegiac couplet when trained on this meter? And second, what is the extent of the generalisability between the two meters?

The first answer can be found in Table 6, showing good F1-scores for models trained on Ovid's and Propertius' elegiac couplets (5-fold cross-validation and 25 epochs were used



(a) long (b) short



(c) elision

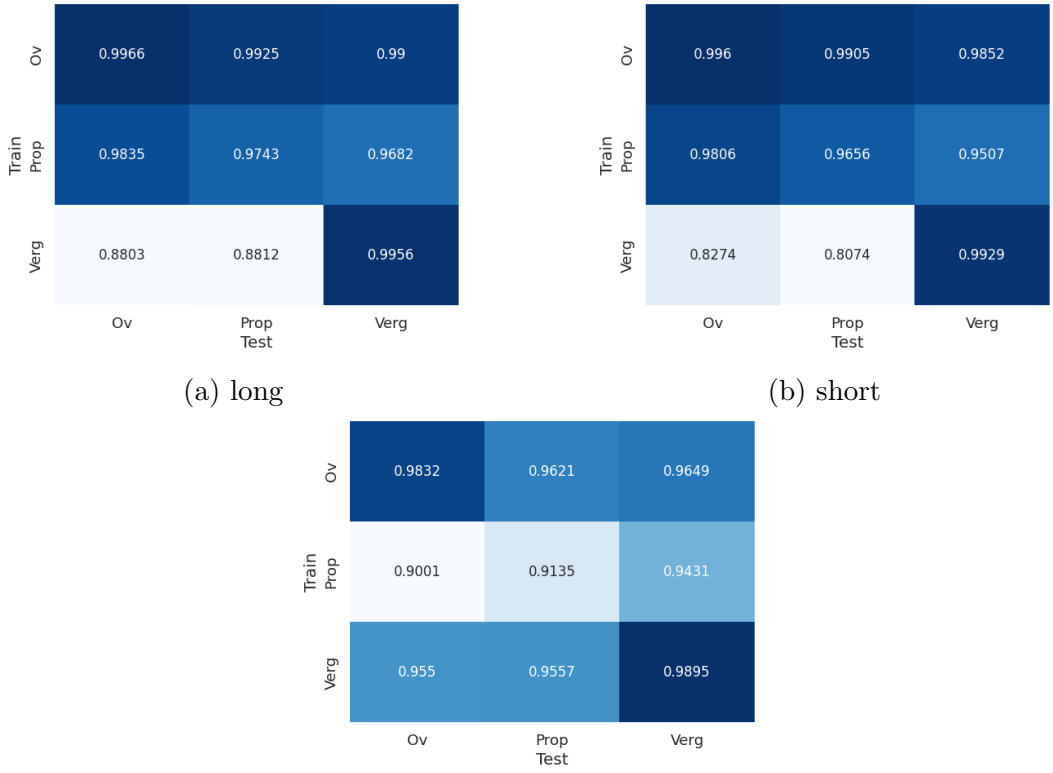
Figure 2: Cross author evaluation F1-scores for the one-hot LSTM model.

again, see Table 1 for more information about the texts). The one-hot LSTM does not seem to have a problem with scanning a combination of hexameters and pentameters, especially when given an adequate training set as is the case with Ovid's *elegiae* (18,832 lines).

	Long	Short	Elision
Ovid	0.9966	0.9960	0.9832
Propertius	0.9743	0.9656	0.9135

Table 6: Model quality (F1-scores) of the one-hot LSTM model on Ovid's and Propertius' *elegiae*.

To answer the second question, Vergil's *Aeneid* is added to represent a model trained on hexameters. Figure 3 shows that although Vergil is able to predict the labels from Ovid and Propertius well enough (0.88 for *long* and 0.80 for *short*), it is clearly outperformed by both elegiac models. Propertius, having only 3,998 verses to train on, outperforms Vergil with 9,840 training verses when testing on Ovid (long and short labels). In line



(a) long

(b) short

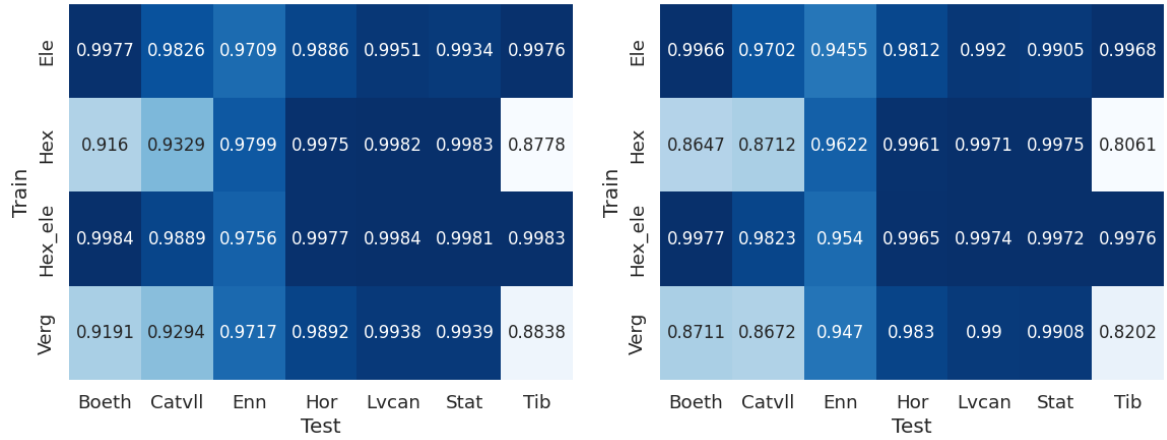
(c) elision

Figure 3: Cross author evaluation F1-scores for the one-hot LSTM model on elegiac couplets and Vergil's *Aeneid*.

with what we have seen before, the larger training set of Ovid directly results in a higher model quality when compared to Propertius. Lastly, we see that the Vergil model is still best in predicting Vergil (cf. Table 5). It is however interesting to see Ovid's *elegiae* performing this well on Vergil's hexameters. This indicates that a model trained on elegiac couplets, being hexameters and pentameters, is able to scan both hexameter and pentameter indifferently.

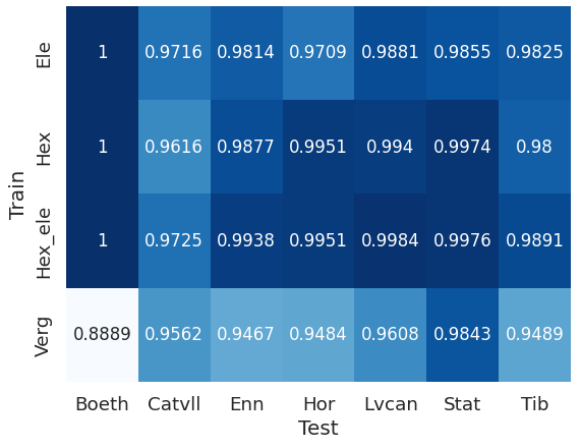
5.1.6 Best training set for scanning dactylic meter

The previous section suggested that it is possible to train a model on pentameters and hexameters to achieve a high quality on texts of one or both meters. It is therefore now worthwhile to find out which training set returns the best results for scanning both dactylic meters independently or simultaneously. We create four new models. The first one is only trained on Vergil's *Aeneid* called *Verg*. The second one is trained on the hexameter texts of Iuvenal, Lucretius, Ovid, Persius and Vergil, and is called *Hex*. Our third model is trained on the elegiac couplets of Ovid and Propertius, and is called *Ele*. The fourth and final model is trained on the texts of *Hex* and *Ele* combined, and is called *Hex_ele*. These four models are then tested on seven unseen texts to evaluate their performance (see Table 1). The texts of Boethius, Catullus and Tibullus feature both hexameter and pentameter verses, with Ennius, Horace, Lucanus and Statius having only hexameters.



(a) long

(b) short



(c) elision

Figure 4: One-hot LSTM model quality (F1-scores) when combining hexameter and elegiac texts. Models tested on unseen dactylic texts.

The results are shown in Figure 4. The first point of interest is how the elegiac model scores better on elegiac texts than on hexameter texts, whilst the hexameter model scores better on hexameter texts than on elegiac texts. Additionally, the *Verg* model is scoring similar or lower than the *Hex* model across labels on almost all authors[14]. Most noteworthy is the result of the *Hex_ele* model, which consistently outperforms the other models. Table 7 shows the average F1-scores for each label over the seven unseen authors, which clearly demonstrates the superiority of the *Hex_ele* model. This confirms again that more text is advantageous. Furthermore, texts with the dactylic meter, either hexameter or pentameter, can be scanned most accurately when a model is trained on both pentameters and hexameters.

[14]The exception being Tibullus' elegiac couplets, on which the hexameter models struggle as a whole for *long* and *short*.

	Long	Short	Elision
Ele	0.9894	0.9818	0.9829
Hex	0.9572	0.9278	0.9880
Hex_ele	**0.9936**	**0.9890**	**0.9924**
Verg	0.9544	0.9242	0.9477

Table 7: Averaged F1-scores from the created one-hot LSTM models on seven unseen dactylic texts.

5.2 Scanning iambic trimeter

In the first part of this chapter we have successfully taught a one-hot LSTM to scan hexameters and pentameters. The second part will investigate whether the LSTM learned the core principles and patterns of scansion, and whether it can now apply these to scan a completely different meter: the iambic trimeter.

5.2.1 One-hot generalisability to iambic trimeter

The previous sections demonstrated a rather seamless transition from scanning hexameters to scanning pentameters. Using the *Hex_ele* model, we will now investigate the generalisability to the iambic trimeter. We will do this by testing the model on the *Agamemnon* data set as described in Section 3.2. Table 8 shows the results. We add the weighted average F1-score, which is the mean of all per-class F1-scores while considering each class's support. This is interesting as the elision label has a high score, but low class support (± 17,000 elisions versus 500,000 long and 335,000 short labels in *Hex_ele*).

	Long	Short	Elision	Weighted average
One-hot LSTM	0.7463	0.5298	**0.9143**	0.6646
CRF	**0.8431**	**0.7488**	0.8085	**0.8046**

Table 8: One-hot LSTM and CRF F1-scores on the trimeter test set.

It is clear that our one-hot LSTM model, when trained on dactylic meter, is not very capable of scanning iambic trimeter. Alas, it is even outperformed by a CRF model (trained on the *Hex_ele* texts) on the long and short labels. Although the quality for *elision* remains satisfactory, it is clear that further investigation towards the scansion of trimeter is needed[15].

5.2.2 Trimeter and word embeddings

This section will improve the one-hot LSTM by changing its input from integers to word embeddings. As discussed in Sections 2.2 and 4.3, word embeddings convey more information than simple integers. In this section we will therefore investigate whether

[15]Training the LSTM for less epochs to possibly prevent overfitting on the dactylic meter did not yield better results. Epochs of five, ten, fifteen and twenty were tested.

embeddings contain phonological information and improve model quality and generalisability. This will be done with the Flair framework (see Section 4.3).

As related in Section 4.3.1, we will experiment with Character, FastText and Flair embeddings. Table 9 shows the F1-scores for the embeddings and the tried combinations of embeddings. As before, we train and test on the *Hex_ele* data set using k-fold cross-validation ($k = 5$). We notice that the highest score is reached when combining all three embeddings. We will therefore use this combination in the following experiments, calling this model the *embeddings LSTM*. If we compare the results with Table 7, we see that the embeddings LSTM outperforms its one-hot counterpart ever so slightly. The question now is whether embeddings as input proves useful on the iambic trimeter data set.

	Long	Short	Elision	Weighted average
Character	0.9711	0.9559	0.9013	0.9640
FastText	0.9931	0.9901	0.9782	0.9938
Flair	0.9834	0.9759	0.9159	0.9847
FastText + Flair	0.9955	0.9936	0.9905	0.9961
Character + Flair + FastText	**0.9965**	**0.9951**	**0.9909**	**0.9969**

Table 9: F1-scores of different word embeddings tested on the *Hex_ele* data set using k-fold cross-validation ($k = 5$).

First we train the embeddings LSTM on the *Hex_ele* data set using 25 epochs and a 70/20/10 split for training, testing and validating respectively. Table 10 shows the results of both LSTM models when testing on the trimeter data set. The embeddings LSTM clearly loses from the one-hot model, with much lower F1-scores for the short and elision labels. It therefore seems that using word embeddings as input does not improve the generalisability of the dactylic LSTM to the trimeter[16]. With regards to this generalisability, a more complex model seems to be counter productive.

	Long	Short	Elision	Weighted average
One-hot LSTM	0.7463	**0.5298**	**0.9143**	**0.6646**
Embeddings LSTM	0.7425	0.4572	0.8552	0.6319

Table 10: Dactylic one-hot and embeddings LSTM F1-scores on the trimeter test set.

Therefore, in summary, it does not seem that the tested LSTM models, when trained on dactylic meter, are easily generalisable to the iambic trimeter.

5.2.3 Weak supervision based on patterns

The previous section demonstrated that generalisability from dactylic meter to the trimeter does not achieve the high accuracy we aim for. This section will therefore test whether scanning trimeter is simply too difficult, or whether it can be done when training on a

[16]Training with different embedding combinations did not yield better results. Tested on the trimeter test set were CharacterEmbeddings, Flair, FastText and FastText + Flair.

trimeter data set. Using the noisy data set as described in Section 3.2, the one-hot and embeddings LSTMs, as well as the word embeddings themselves, are trained again. The only difference is that we do not allow the label *anceps* to be predicted by the models. For example, if the model is most confident that the syllable in question is an anceps, we check whether *long* or *short* has a higher confidence: we then go with the winner. The results can be seen in Table 11, which shows a large improvement in model quality for the one-hot LSTM when trained and tested on iambic trimeter. The embeddings model in contrast is performing worse with this data set, reaching only half the F1-score for the short label. Similar results were seen when training the word embeddings on the *Hex_ele* data set instead of on the 7,193 lines of the noisy Anceps data set. Explanations for this bad performance could be the rather small data set, or the embeddings model having trouble with the noise. For example, when asking the model on its second most confident label after *anceps*, it wants to predict *elision* and *space* rather than *long* and *short*. As we do not allow the former two labels to be predicted, we cannot put much confidence in its decision for either *long* or *short*.

	Long	Short	Elision	Weighted average
One-hot LSTM	**0.8643**	**0.8370**	**0.8966**	**0.8543**
Embeddings LSTM	0.7123	0.4081	0.8194	0.7040

Table 11: F1-scores for the one-hot and embeddings LSTM models trained on the noisy Anceps data set and tested on the trimeter test set.

5.3 Scanning other meters

The scansion of iambic trimeter is difficult for a dactylic model. But does this mean that the dactylic LSTM cannot scan any other meters, or that it is just unable to scan the trimeter? We therefore need to answer the question to what extent we can generalise the machine learning models to other meters. To do this, we selected five additional meters (see Table 12) and manually scanned around ten lines for each. Although this returns data sets that are far too small to draw reliable conclusions, it does give us an indication about the generalisability. Having tested the CRF, one-hot and embeddings LSTMs again on these texts, we see the results as shown in Table 13. As with the trimeter, the one-hot LSTM outperforms the embeddings LSTM, although both show low F1-scores for the short label. More importantly however is that both are outclassed by the CRF, which achieves similar results on the new meters as it did on the trimeter. It therefore seems that the simple CRF is the best model when generalising to other meters, with the LSTMs preferring to be trained and tested on the same metrical system, preferably without noise in the data set.

Author	Title	Meter	Lines	Syllables
Seneca	*Medea*	Anapest	787–796	102
Seneca	*Medea*	Glyconeus	75–92	144
Catullus	*Carmina*	Hendycasyllable	1.1–10	115
Seneca	*Medea*	Iambic dimeter	849-855	60
Seneca	*Medea*	Trochaic tetrameter	740–747	127

Table 12: Information on the five data sets with different meters.

Meter	CRF			One-hot LSTM			Embeddings LSTM		
	Long	Short	Elision	Long	Short	Elision	Long	Short	Elision
Anapest	**0.89**	**0.85**	1.00	0.81	0.62	1.00	0.82	0.66	1.00
Glyconeus	**0.87**	**0.75**	n/a	0.82	0.53	n/a	0.79	0.48	n/a
Hendycasyllable	**0.84**	**0.76**	0.89	0.76	0.51	0.89	0.75	0.45	0.89
Iambic dimeter	**0.83**	**0.70**	n/a	0.77	0.41	n/a	0.78	0.42	n/a
Trochaic tetrameter	**0.86**	**0.77**	1.00	0.78	0.59	1.00	0.78	0.53	1.00

Table 13: CRF, one-hot LSTM and embeddings LSTM F1-scores for different meters when trained on the *Hex_ele* data set.

6 Discussion

This thesis tried to answer the question to what extent machine learning models are able to scan more complex metrical systems when trained on simpler ones. The previous section showed that the LSTM models performed very well when trained and tested on the dactylic meter, and badly when trained on this meter and tested on another. By carrying out testing on larger data sets of verses in meters beyond the dactylic and iambic, this finding could be corroborated, but there is sufficient evidence from these results to suggest this trend. It is therefore worth discussing what improvements could be made in future research.

The first improvement relates to the syllabification of the Latin verses, which has been applied rather rigidly to the words. For example, in the spoken language, a word like *sacrum* would be syllabified as *să-crum*, with a short ă. However, a poet could use their poetic freedom to read *sacrum* as *sāc-rum*: here the first syllable would take longer to pronounce and would therefore be scanned as long. Additionally, the poet could require letters from following syllables to be taken with previous syllables. For example, the words *legīt autem* would, in the flow of the meter, be pronounced as *legĭ-tautem*, turning the long *ī* into a short one. This playfulness and nuance is lost in the rigid syllabification used in this thesis. As a solution one could remove all whitespaces and see the verse as a continuous string of sounds. This would mean that the model has to label specific parts of this continuous string, determining for itself which vowel needs which label. It would be extremely interesting to see whether we could provide the following single input:

armavirumquecanotroiaequiprimusaboris

and retrieve a list of fifteen labels from this line. In that case, the syllabification as a whole is unnecessary and nuances might be picked up more easily by the program. This could then also help the flexibility of the model when we train on the dactylic meter and test on others. However, these exceptions are far outnumbered by the regular instances which should experience no adverse effect from the used syllabification. The suggested model could thus result in a worse performing model as well.

The second improvement is rather small and has to do with the pre-processing of the texts, which lowercased all verses. This has been done because some text editions like to start a sentence with a capital, even though Romans only used capital letters, therefore having no distinction between the two cases. In essence, this means that the *Ar* from Vergil's first line would be the same entity as any other *ar* in the text. However, in the case of proper names, we lose accuracy, as these often have some uncommon scansions, especially if they are from Greek origin. Indeed, after a qualitative investigation of Vergil's *Aeneid*, many lines with Greek proper names were scanned incorrectly. Although these are also hard for students, it might be worthwhile to investigate any improvement when allowing capitalisation for proper names.

Regarding the experiments, normalisation improvements should be made in future research. For example, in Section 5.1.6, the texts of Ovid and Vergil are directly compared to each other. However, Ovid's model has around twice as many lines to be trained on as Vergil's. We allowed this because normalisation in combination with Propertius would

bring the number of lines down to four thousand lines, which would introduce the problem of having not enough training verses. Nevertheless, it does give Ovid's model an unfair advantage when compared one-to-one with Vergil. The same holds true when comparing the *Hex_ele* with the *Hex* and *Ele* data sets.

In the same vein, we compare results from the one-hot LSTM with the embeddings LSTM. Although compared when trained on the same texts, and both for 25 epochs, the structures of the models might differ, as Flair treats its LSTM implementation as a black box. To illustrate, our one-hot LSTM has an embeddings layer with fifty dimensions, followed by a dropout layer and one bidirectional layer with one hundred dimensions. For the embeddings LSTM we used the same number of neurons and the same dropout. However, the exact structure we could not replicate, as Flair does not allow tinkering with its LSTM structures without making substantial changes to the source code. Future comparisons should therefore be made with identical LSTM implementations to truly compare between integer and embedding inputs. Because Flair does not allow such customisation as of the moment of this writing, a custom implementation might be necessary.

Staying with the topic of embeddings, more interesting embedding types exist such as the Pooled Flair Embeddings [2] and ELMo Embeddings [40], which could result in better performance. Additionally, other machine learning methods that can extract syntactic and phonological information could be tried. Especially those that can work with character strings, like the CRF model, as these can directly read the important information about length embedded in the letters themselves. If staying with syllables as input, it would be interesting to add word features to each syllable. For example, the syllable *ma* could have the feature *word stem* being *mater*. The model should then learn that this syllable is always long.

Regarding the evaluation on the models, we started with hexameters, moved to elegiac couplets and the iambic trimeter before ending with five additional meters. It would be worthwhile to create test sets for multiple meters beforehand and evaluate models on all sets simultaneously. Now we disregarded the CRF early because of the better results from the LSTM on the hexameter. Regarding generalisability to other meters, the simple CRF clearly outperformed the more complex LSTMs, which became apparent only later because of the approach taken in this thesis. Future research could therefore benefit from improvements to the CRF model, for example increasing the window size or the addition of more complex features as mentioned in the previous paragraph.

Lastly, an in-depth qualitative analysis of the scansions made by the dactylic model on other meters should be performed. This could tell us where and why the model goes wrong, which could shed light on possible improvements and fixes. Most probably the increased variability between the hexameter and trimeter is at play. In dactylic hexameter there is a binary decision between dactyls and spondees, regardless of long and short syllables. In contrast, iambic trimeter has no underlying binary decision in the same way, because of resolutions (turning an anceps or long syllable into two short ones), which makes the distinction between elements of the meter fuzzier. The decision for resolution is simply more complex than choosing between a dactyl and a spondee, which likely results in the observed lower model quality when training on dactylic meter and testing on trimeter.

7 Conclusion

In this thesis we tried to answer the question *to what extent machine learning models, in particular neural networks, are able to scan more complex Latin metrical systems when trained on simpler ones?* Here the idea was to let models learn the principles of Latin scansion on the dactylic meter and to apply these to other meters. We find that both CRF and LSTM models are able to scan the hexameter meter, with the latter outperforming the former. We furthermore found that the LSTM is able to scan the dactylic meter indifferent of author, genre or time period, but requiring a minimum training set of 3,000 verses for F1-scores of 0.98 and higher. The generalisability between metric systems using an LSTM seems however harder, as the LSTM, having either one-hot encoding or word embeddings as input, did not produce satisfactory results when scanning the iambic trimeter and the other meters tried. We did see more successful results for the one-hot LSTM on the iambic trimeter when training on a noisy trimeter set of Senecan texts. This approach is however not feasible for other meters, as there are simply not enough verses handed down to us to reliably train an LSTM on. This thesis therefore found that the best way to scan different meters when trained on the dactylic meter is to use the CRF model with customly created features. However, the quality of the model needs improvement for it to be used in the field of Classics.

As mentioned in the discussion, future research could benefit from a rethinking of the input. The experiments showed that machine learning is capable of scansion and that it benefits from having the entire verse as input, as seen with the LSTM. However, the way the Latin words are syllabified to serve as input might be too rigid. Additionally, converting the syllables to integers or word embeddings might lose the phonological information embedded in the letters themselves. It would therefore be worthwhile to investigate a machine learning model that can read the letters of the entire verse as a continuous sequence of sounds and label these accordingly.

To facilitate future research and the reproduction of our experiments, we have released our code, data sets and experiments on Github: `https://github.com/Ycreak/Latin_scansion_with_neural_networks`.

References

[1] M. Agirrezabal et al. "ZeuScansion: A tool for scansion of English poetry". In: *Journal of Language Modelling* 4(1) (2016), pp. 3–28.

[2] A. Akbik, T. Bergmann, and R. Vollgraf. "Pooled Contextualized Embeddings for Named Entity Recognition". In: *Proceedings of ACL-2019*. 2019, pp. 724–728.

[3] A. Akbik, D. Blythe, and R. Vollgraf. "Contextual String Embeddings for Sequence Labeling". In: *Proceedings of CoNLL-2018* (2018), pp. 1638–1649.

[4] A. Akbik et al. "FLAIR: An easy-to-use framework for state-of-the-art NLP". In: *Proceedings of ACL-2019*. 2019, pp. 54–59.

[5] Y. Assael, T. Sommerschield, and B. Shillingford. "Restoring and attributing ancient texts using deep neural networks". In: *Nature* 603 (2022), pp. 280–283.

[6] M. Ayer. *Allen and Greenough's New Latin Grammar for Schools and Colleges*. Carlisle, PA, 2014.

[7] K. Bobenhausen. *Metricalizer*. `https://metricalizer.de/de/impressum/`. Online; accessed 2022-06-23. 2009.

[8] P. Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[9] C. Bozzone. "Technologies of Orality: Formularity, Meter, and Kunstsprache in Homer". In: *Rethinking Orality II: The Mechanisms of the Oral Communication System in the Case of the Archaic Epos*. Ed. by A. Ercolani and L. Lulli. 2022, p. 51.

[10] K. Cho et al. "On the properties of neural machine translation: Encoder–decoder approaches". In: *Syntax, Semantics and Structure in Statistical Translation* (2014), pp. 103–111.

[11] R. Collobert et al. "Natural language processing (almost) from scratch". In: *The Journal of Machine Learning Research* (2011), 12:2493–2537.

[12] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of ACL-2019* (2019), pp. 4171–4186.

[13] H. Drexler. *Einführung in die römische Metrik*. Darmstadt, 1967.

[14] A. Fedchin. *Anceps - a Latin scansion tool*. `https://github.com/Dargones/anceps`. Online; accessed 2022-05-22.

[15] A. Fedchin et al. "Senecan Trimeter and Humanist Tragedy". In: *American Journal of Philology* Forthcoming (2020).

[16] J. Firth. "A synopsis of linguistic theory 1930-1955". In: *Studies in Linguistic Analysis 1930-1955* (1962), p. 11.

[17] F.A. Gers, J. Schmidhuber, and F. Cummins. "Learning to forget: Continual prediction with lstm". In: *Neural computation* 12(10) (2000), pp. 2451–2471.

[18] Y. Goldberg. *Neural Network Methods in Natural Language Processing*. 2017.

[19] C. Goller and A. Kuchler. "Learning task-dependent distributed representations by backpropagation through structure". In: *Neural Networks* (1996), pp. 347–352.

[20] C.O. Hartman. *The Scandroid*. http://charlesohartman.com/verse/scandroid/index.php. Online; accessed 2022-06-23. 2005.

[21] W. Haverals, F.B. Karsdorp, and M. Kestemont. "Rekenen op ritme: Een datagedreven oplossing voor het automatisch scanderen van de historische lyriek in de DBNL". In: *Vooys* 37(3) (2019), pp. 6–23.

[22] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9(8) (1997), pp. 1735–1780.

[23] Z. Huang, W. Xu, and K. Yu. "Bidirectional LSTM-CRF Models for Sequence Tagging". In: *CoRR* (2015).

[24] G. Hutchinson. "Genre and super-genre." In: *Generic Interfaces in Latin Literature: Encounters, Interactions and Transformations*. Ed. by T.D. Papanghelis, S.J. Harrison, and S. Frangoulidis. 2013, pp. 19–34.

[25] The Packard Humanities Institute. *PHI: Searchable Greek Inscriptions*. https://inscriptions.packhum.org/. Online; accessed 2022-06-23.

[26] H. Koppelaar. *Scansion generator*. https://lab.kb.nl/tool/scansion-generator. Online; accessed 2022-06-23. 2013.

[27] J.D. Lafferty, A. McCallum, and F. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *ICML*. 2001.

[28] G. Lample et al. "Neural architectures for named entity recognition". In: *Proceedings of ACL-2016* (2016), pp. 260–270.

[29] W. Ling et al. "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation". In: *Proceedings EMNLP-2015* (2015), pp. 1520–1530.

[30] G. Luo et al. "Joint entity recognition and disambiguation". In: *Proceedings of EMNLP-2015* (2015), pp. 879–888.

[31] X. Ma and E. Hovy. "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF". In: *Proceedings of ACL-2016* (2016), pp. 1064–1074.

[32] X. Ma and F. Xia. "Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization". In: *Proceedings of ACL-2014* (2014), pp. 1337–1348.

[33] T. Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems* (2013), pp. 3111–3119.

[34] A. Mittmann, A. Von Wangenheim, and A.L. Dos Santos. "Aoidos: A System for the Automatic Scansion of Poetry Written in Portuguese". In: *CICLing*. 2016.

[35] L. Morgan. *The Dactylic Hexameter and its Detractors*. Oxford, 2010.

[36] Musisque Deoque. *A digital archive of Latin poetry*. https://mizar.unive.it/mqdq/public/. Online; accessed 2022-05-22.

[37] A. Passos, V. Kumar, and A. McCallum. "Lexicon infused phrase embeddings for named entity resolution". In: *Proceedings of CoNLL-2014* (2014), pp. 78–86.

[38] Pedecerto. *Automatic analysis of Latin verses*. `http://www.pedecerto.eu/public/`. Online; accessed 2022-05-22.

[39] J. Pennington, R. Socher, and C. Manning. "Glove: Global vectors for word representation". In: *Proceedings of EMNLP-2014* (2014), pp. 1532–1543.

[40] M. Peters et al. "Deep contextualized word representations". In: *Proceedings of ACL-2018* (2018), pp. 2227–2237.

[41] M. Peters et al. "Semi-supervised sequence tagging with bidirectional language models". In: *Proceedings of ACL-2017* 1 (2017), pp. 1756–1765.

[42] L. Ratinov and D. Roth. "Design Challenges and Misconceptions in Named Entity Recognition". In: *Proceedings of CoNLL-2009* (2009), pp. 147–155.

[43] D.S. Raven. *Latin Metre: an introduction*. London, 1965.