

# Designing and Comparing G2P-Type Lemmatizers for a Morphology-Rich Language

Steffen Eger<sup>(✉)</sup>

Text Technology Lab, Goethe University, Frankfurt am Main, Germany  
steeger@em.uni-frankfurt.de

**Abstract.** We consider the statistical lemmatization problem in which lemmatizers are trained on (word form, lemma) pairs. In particular, we consider this problem for ancient Latin, a language with high degree of morphological variability. We investigate whether general purpose string-to-string transduction models are suitable for this task, and find that they typically perform (much) better than more restricted lemmatization techniques/heuristics based on suffix transformations. We also experimentally test whether string transduction systems that perform well on one string-to-string translation task (here, G2P) perform well on another (here, lemmatization) and vice versa, and find that a joint  $n$ -gram modeling performs better on G2P than a discriminative model of our own making but that this relationship is reversed for lemmatization. Finally, we investigate how the learned lemmatizers can complement lexicon-based systems, e.g., by tackling the OOV and/or the disambiguation problem.

## 1 Introduction

Lemmatization can be defined as the normalization task of mapping the inflected forms of lexical words to their canonical form, i.e., their *lemma* (cf. [5,6,12]). A related problem is *stemming*, in which the variability of word forms is reduced by mapping different variants to a common root or *stem*, which may be a crude abstraction that does not need to correspond to any valid linguistic unit.<sup>1</sup> Lemmatization and stemming are important preprocessing steps in information retrieval, text mining, and knowledge discovery.

In this work, we view the lemmatization problem within the general string-to-string translation setup of mapping arbitrary strings  $\mathbf{x} \in \Sigma^*$  to arbitrary other strings  $\mathbf{y} \in \Gamma^*$ , where  $\Sigma$  and  $\Gamma$  are arbitrary alphabets (finite sets). While this setup would also include other natural language processing (NLP) tasks such as

- *grapheme-to-phoneme conversion (G2P)* [2,11], in which  $\mathbf{x}$  is a letter-string and  $\mathbf{y}$  is a string of phonemes,
- *transliteration* [21], in which  $\mathbf{x}$  is a word form in one script (e.g., Cyrillic, Hebrew, Latin, etc.) and  $\mathbf{y}$  is a corresponding form in another script, or
- *spelling error correction* [4], in which  $\mathbf{x}$  is a wrongly spelled word form and  $\mathbf{y}$  its desired correction,

<sup>1</sup> In stemming, all that typically matters is that related words map to the same (linguistic or even non-linguistic) object.

our focus is, as indicated, on the lemmatization task in which  $\mathbf{x}$  is a word form and  $\mathbf{y}$  its lemma. Our statistical problem is to *learn* mappings  $\mathbf{x} \mapsto \mathbf{y}$  from pairs of strings  $\{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, 2, 3, \dots\}$ .

As [12] point out, the difficulty of the lemmatization problem heavily depends on the types of natural languages involved. While lemmatization is considered relatively easy in highly analytical languages such as English, the problem becomes considerably more difficult in languages that exhibit sufficient morphological variability, such as the Slavic languages or ancient Greek and Latin. In these, “stems can combine with many different suffixes, and the selection of appropriate ending and its combination with the stem depends on morphological, phonological and semantic factors” [12]. In the present work, our focus is on lemmatization in ancient Latin, because, on the one hand, ancient Latin is a prime exemplar of a language with rich morphology in which more than hundred distinct forms may be associated with a single (e.g., verb) lemma. On the other hand, we are currently actively developing several NLP tools for ancient Latin,<sup>2</sup> of which a lemmatizer (as well as, in subsequent steps, a POS tagger and a parser) is an integral part.

Arguably, the most well-researched domain within the field of string-to-string translation is G2P and it is tempting to simply apply one of the existing G2P toolkits to the problem. Our approach in this work is indeed to evaluate how standard G2P models perform on the lemmatization task and how their performance relates to standard statistical lemmatizers. As one of our results, we will show that two of the general G2P models that we review, a standard joint  $n$ -gram model and a discriminative model of our own making, not only perform orders of magnitude better than two off-the-shelf lemmatizers on the G2P problem, but also considerably better on the lemmatization task. We also show that ordering of performance relationships is not necessarily preserved across string transduction problems. More precisely, we show that the joint  $n$ -gram modeling that we test is considerably better on G2P conversion than our own discriminative model, but that this relationship is reversed on the lemmatization task. After reviewing the models in Sect. 2, we briefly outline our data base in Sect. 3, and conduct performance comparisons in Sect. 4. In this section, we also investigate how our learned lemmatizers may complement lexicon-based lemmatizers, e.g., by tackling the out-of-vocabulary (OOV) problem and/or by disambiguation. We conclude in Sect. 5.

Concerning related work, Porter stemmer [19] is a rule-based heuristic for solving the stemming problem in English. The approaches that we survey in the current work are much more closely related to modern machine learning approaches for string transduction. For instance, Dreyer and Eisner [6] present a discriminative log-linear model learning latent classes and apply it to lemmatization. Gesmundo and Samardzic [9] reformulate lemmatization as a tagging problem in a setting where they assume that lemmas are derived from word forms by prefix and suffix transformations and the tag label encodes the substitution patterns. Toutanova and Cherry [23] show that considering lemmatization

<sup>2</sup> See, e.g., <https://prepro.hucompute.org/>.

and part-of-speech tagging *jointly* may be mutually beneficial. Their character-based lemmatization module is similar to the G2P-type lemmatizers we consider below but only considers one-to-one character transformations plus a heuristic for dealing with suffixes.

## 2 Models

In this study, we rely on the following software/models for learning the lemmatization problem.

- *Phonetisaurus* [18] implements a joint  $n$ -gram model [2] in a weighted finite state transducer setup, and has originally been designed for G2P conversion. Like our own modeling, *Phonetisaurus* can be used in a more general setting, however, to learn to transduce arbitrary input sequences into arbitrary output sequences. *Phonetisaurus* seems to perform on par or better than competitors on the G2P problem and trains and decodes orders of magnitudes faster [18].<sup>3</sup>
- *LemmaGen* [12, 13] is a lemmatizer that learns ‘if-then’ rules from  $(\mathbf{x}, \mathbf{y})$  pairs as shown in Table 4. To transduce/lemmatize a new input form, rules (and their exceptions) are ordered, and the first condition that is satisfied fires the corresponding rule [12]. Importantly, *LemmaGen* learns to transduce word form *suffixes* into lemma *suffixes*, so it might be prone to committing errors, e.g., when initial or middle parts of word forms need to be adjusted to generate the correct lemma.
- *Mate* [3] provides a full pipeline of lemmatization, tagging, morphological tagging, and dependency parsing. It is trainable on appropriate input format. In our context, we only train the lemmatizer module of the pipeline.

**Our own modeling** implements a two-stage tagging procedure to translate input strings into output strings. In the first stage, an input word is *segmented* into parts using a sequence labeler (tagger) that maps input character sequences to ones or zeros, depending on whether a split occurs at the given character position. In the second stage, each part of the segmented input string is tagged with an output string subsequence. Table 1 illustrates. The training data for both sequence labelers is taken from *monotone many-to-many aligned* input strings as in Table 2.<sup>4</sup> The second stage tagger directly trains on the aligned data, while the first stage tagger learns sequence segmentations from the segmented  $\mathbf{x}$  sequences in the alignments using a binary coding scheme (cf. [1, 7]). Table 3 illustrates this—note that we encode a split as a ‘1’ and a non-split (continuation) as a ‘0’.

As a **tagging model**, we use linear chain conditional random fields (CRF) [14].<sup>5</sup> This allows us to include arbitrary features in the tagging process. We use the following:

<sup>3</sup> In our experiments below, we choose an  $n$ -gram order of size 6 for *Phonetisaurus*. Increasing  $n$ -gram order size did not lead to better performance in preliminary tests.

<sup>4</sup> We use the alignments produced by the *Phonetisaurus* toolkit.

<sup>5</sup> Although CRFs are rather old and typically not always the best-performing sequence labeling models [17], we use them here mainly for practical reasons. In particular, the CRF package we are using, available from <https://code.google.com/p/crfpp/>, provides a very convenient interface to modeling sequence labeling.

**Table 1.** Sample decoding phase.

præformet $\rightsquigarrow$ p-r-æ-f-o-r-m-et	passvrum $\rightsquigarrow$ p-a-ss-vra-rum
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	↓ ↓ ↓ ↓ ↓ ↓
p-r-æ-f-o-r-m-o	p-a-ti- o - r

**Table 2.** Sample aligned input string pairs  $(\mathbf{x}, \mathbf{y})$ .

d-i-s-s-o-n-verat	d-i-s-s-o-n-o
c-o-n-r-e-ct-v-s	c-o-n-r-i-g-e-o
j-m-p-e-d-i-o	i-m-p-e-d-i-o
c-o-m-p-u-t-arīs	c-o-m-p-u-t-o
t-e-r-r-e-batvr	t-e-r-r-e-o
a-d-i-u-t-o-rivm	a-d-i-u-t-o-r
p-r-a-e-p-e-d-i-m-e-n-t-a	p-r-a-e-p-e-d-i-m-e-n-t-um
u-n-d-e-c-i-m-am	u-n-d-e-c-i-m-a
d-u-l-c-i-l-o-c-u-t-i-ssimarum	d-u-l-c-i-l-o-c-u-t-u-s

**Table 3.** Word forms  $\mathbf{x}$  from Table 2, corresponding segmentations and binary encodings of the segmentations.

Word form	Segmentation	Binary encoding
dissonverat	d-i-s-s-o-n-verat	01111110000
conrectvs	c-o-n-r-e-ct-v-s	011111011
jmpedio	j-m-p-e-d-i-o	0111111
computaris	c-o-m-p-u-t-arīs	0111111000
terrebatvr	t-e-r-r-e-batvr	0111110000
adiutorivm	a-d-i-u-t-o-rivm	0111111000
praepedimenta	p-r-a-e-p-e-d-i-m-e-n-t-a	011111111111
undecimam	u-n-d-e-c-i-m-am	011111110
dulcilocutissimarum	d-u-l-c-i-l-o-c-u-t-i-ssimarum	011111111110000000

- (1) *Contextual features*: for each input symbol (a character or a subsequence of characters), we include all character subsequence  $m$ -grams (unigrams, bigrams, trigrams, etc.) that fit inside a window of size 6 around the current input symbol position.<sup>6</sup> To illustrate, when the second stage tagger views the current input position  $e$  in the input word form *c-o-n-r-e-ct-v-s*, it sees that the next position contains *ct*, the previous position contains *r*, etc.; the next two positions contain  $(ct, v)$ , etc.

<sup>6</sup> Increasing window size typically does not lead to better performance, as we verified in preliminary experiments.

- (2) *Linear-chain and transition features*: These include features between subsequent output symbol characters (character subsequences) as is the defining property of linear chain conditional random fields.
- (3) *Intra-subsequence-character features*: We also include features that allow the second stage tagger to not only see the character subsequence at the current input position but also the characters that constitute it. For example, rather than merely knowing that the last input position of *d-i-s-s-o-n-verat* is *verat*, the tagger would also know that *verat* is made up of the characters *v*, *e*, *r*, *a*, *t*. This may help the classifier in case longer character subsequences are sparse. Since additional features increase computational costs, we only include unigram intra-subsequence-character features.

We call the system that includes features (1) and (2) *AliSeTra* (for align-segment-translate), while we refer to the system that in addition includes features (3) as *AliSeTra++*. We also note that our design of phrasing string transduction as a (two-stage) sequence labeling approach is, per se, nothing novel — it is one of the standard paradigms in G2P (cf., e.g., [10, 11]).

### 3 Data

Our data base is a huge Latin lexicon containing almost 10 million distinct word forms subsumed under more than 100,000 lemmas [8, 15]. This lexicon has been semi-automatically created from several freely available Web resources and via lexical expanders and subsequent manual correction (where necessary). Of these forms, almost 80 % belong to the open word classes nouns, verbs, and adjectives, on which we will focus in the remainder of this work.<sup>7</sup> Our task will be to learn to lemmatize Latin word forms falling under the named word classes from pairs of examples as shown in Table 4.

**Table 4.** Example string pairs in the data base. (Potential) substitution patterns highlighted for clarity of exposition.

Verbs		Nouns		Adjectives	
ingemu <b>istis</b>	ingem <b>isco</b>	princip <b>ibvs</b>	princ <b>ps</b>	denuntiati <b>ssimam</b>	denuntiati <b>us</b>
exmactau <b>issetis</b>	exmact <b>o</b>	frag <b>i</b>	frag <b>um</b>	perrectas	perrect <b>us</b>
conrect <b>vs</b>	conrige <b>o</b>	chyrogrill <b>io</b>	chyrogrill <b>ius</b>	dedolati <b>ores</b>	dedolatu <b>s</b>
emundatar <b>um</b>	emundo <b>o</b>	adversat <b>vm</b>	adversatu <b>s</b>	praestanti <b>oribvs</b>	praestans
superintex <b>ere</b>	superintego <b>o</b>	eruptur <b>us</b>	eruptur <b>us</b>	infortunati <b>ssimvs</b>	infortunatu <b>s</b>
dispute <b>bant</b>	dispute <b>o</b>	sciotheric <b>orvm</b>	sciotheric <b>um</b>	resoni <b>ores</b>	resonu <b>s</b>

<sup>7</sup> Typically, word forms in other word classes are also not inflectional, so that the learning problem would be trivial.

## 4 Evaluation

We now evaluate the four described systems. We start with an evaluation on a G2P conversion task, in order to see how the systems perform in a string-to-string translation setting different from lemmatization. We train and evaluate all systems on the General American (GA) variant of the Combilex [20] G2P data set. Throughout our evaluation settings, we use *word accuracy* as our evaluation measure, defined as the fraction of correctly transduced forms in the test set, i.e., it is defined as the number of strings  $\mathbf{x}_i$  in the test set that satisfy  $\hat{\mathbf{y}}_i = \mathbf{y}_i$ , divided by the size of the test set. Here  $\mathbf{x}_i$  is an input form,  $\mathbf{y}_i$  is the gold standard reference and  $\hat{\mathbf{y}}_i$  is the prediction of a specific system. Word accuracy is a strict measure that penalizes even tiny deviations from the gold standard, but is nowadays the most common in G2P and related fields.

### 4.1 Testing G2P Performance

Table 5 gives results when we train the systems on training sets of size 2,000, 5,000, and 10,000, respectively, and test them on a disjoint set of string pairs of size 28,609. Phonetisaurus is clearly best in all settings, with a margin of about 7–20 % over AliSeTra, depending on training set size. AliSeTra++ performs slightly worse than AliSeTra in two out of three training set size cases, indicating that the additional features tend to harm in this case, which could be due to the fact that the system now must estimate additional (possibly irrelevant) parameters and due to stronger overfitting given the additional degrees of freedom (see also the discussion below). In any case, difference between AliSeTra++ and AliSeTra is marginal. Both AliSeTra/++ and Phonetisaurus perform strikingly better than each of LemmaGen and Mate on the G2P conversion task. For example, at a training set of size 10,000, Phonetisaurus has about 66 % of the test inputs correct, while LemmaGen has only 6.82 % correct (i.e., accuracy is about 10 times higher for Phonetisaurus) and Mate 1 %.

**Table 5.** Word accuracy in % as a function of training set size. G2P data.

	2,000	5,000	10,000
AliSeTra++	38.33	51.98	61.26
AliSeTra	36.64	52.43	62.13
Phonetisaurus	<b>44.60</b>	<b>57.62</b>	<b>66.67</b>
LemmaGen	2.29	4.42	6.82
-last-4-chars	15.30	22.33	36.82
Mate	0.39	0.76	1.00
-on-training	89.17	97.49	95.26

Looking at the reasons for this discrepancy, we find that, as we have already outlined when introducing the systems, LemmaGen can essentially only

transform the endings of strings, which results in errors for virtually all long words. In fact, when we test accuracy comparing only the last four characters of  $\hat{\mathbf{y}}_i$  with the last four of  $\mathbf{y}_i$ , we find that the accuracy of LemmaGen increases substantially, but is still at a much lower level than either Phonetisaurus or AliSeTra. For instance, at a training set size of 10,000, word accuracy of Phonetisaurus (AliSeTra) is still 81 % (68 %) higher than that of LemmaGen, even in this favorable setting for LemmaGen. The performance of Mate is even (considerably) worse than that of LemmaGen. This is not due to the fact that input and output alphabets are different in G2P conversion, as the performance on the training data shows. Here, Mate achieves accuracy of up to 97 %, indicating that the system may be entirely overfitting the training data (e.g., storing training data instances as lexical entries and learning minimal transformation regularities).<sup>8</sup>

## 4.2 Testing Lemmatization Performance

Next, we investigate performance on the lemmatization task. Figure 1 shows learning curves—accuracy as a function of training set size—for all systems when the systems are exclusively trained and tested on verbs. We distinguish two modes of testing:

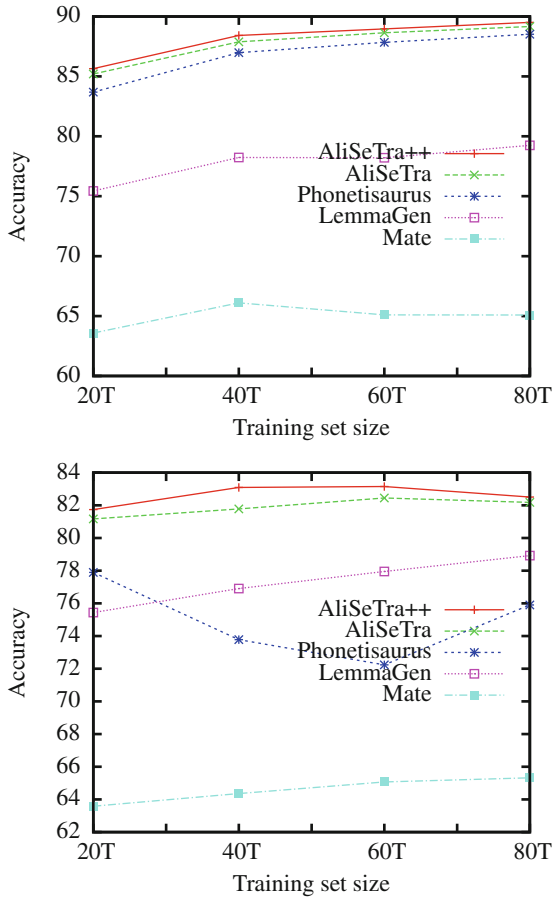
- *In-domain-testing*: In in-domain testing, training and test data contain forms that belong to the same lemma. For example, the training data might contain the form *amavisse*, while the test data might contain the form *amas*, both of which have *amo* as a lemma (however, no *form* in the test data also occurs in the training data).
- *Out-domain-testing*: In out-domain testing, the test data contains only forms whose lemmas do not underlie any form in the training data.

In-domain and out-domain testing intend to address different application scenarios. If a statistical lemmatizer is primarily used for lemmatizing out-of-vocabulary (OOV) forms,<sup>9</sup> out-domain-testing would be the more relevant criterion for suitability of the lemmatizer. In a less restricted application scenario for the lemmatizer, results of in-domain-testing would likely be the more relevant statistic, particularly if training sets are large enough, since in this case, most forms to be lemmatized in a text will either have been seen in the training data or, at least, may be expected to be morphologically related to forms in the training data.

The figure shows that AliSeTra++ performs best now, with a slight but consistent margin over AliSeTra. Particularly in out-domain testing, AliSeTra/++ perform substantially better than Phonetisaurus where difference in performance

<sup>8</sup> In fact, it seems that Mate simply stores input strings that occur fewer than 5 times, rather than learning substitution patterns from these (personal communication with Bernd Bohnet). Thus, the evaluation scenario adopted in this work puts Mate at a general disadvantage, since we generally train systems on arbitrary lists of word pairs selected from a lexicon rather than on the distributions found in ‘real’ text.

<sup>9</sup> E.g., when the lemmatizer is developed to assist a lexicon-based lemmatizer.



**Fig. 1.** Word accuracy as a function of training set size. Left: In-domain testing. Right: Out-domain testing. Verbs.

is between 5 and 12 %. Again, all three of the general string transduction systems have considerably higher word accuracies than either LemmaGen or Mate, but differences are much less pronounced than on the G2P conversion task. In fact, in out-domain-testing, LemmaGen performs even better than Phonetisaurus, for particular training set sizes. It is also worthy to note that LemmaGen and Mate perform relatively stable over in-domain vs. out-domain testing, while AliSeTra/++ and Phonetisaurus seem to suffer more from overfitting (put positively: can better adapt to the distribution of the training data).

As to why AliSeTra++ performs better than AliSeTra on the verb lemmatization task, an intuitive explanation would be that average segment length in the string transduction task is correlated with effectiveness of the intra-subsequence-character features, since longer segments are sparser and therefore harder to estimate as whole chunks. In fact, average length of a segment in verb lemma-



**Table 6.** Test set sizes.

	Verbs	Nouns	Adjectives
In-domain	20,000	20,000	20,000
Out-domain	206,347	39,454	83,295

tization is 1.53, with a maximum segment length of 10—cf. the aligned pair (*m-i-r-arenturque, m-i-r-o*)—while average length of a segment in the G2P data set is 1.14, with a maximum length of 2—cf. the aligned pair (*gu-a-r-a-n-t-ee-d, g-a-r-@-n-t-i-d*).

In Tables 7, 8, and 9, we report word accuracies for the different systems on the three word classes verbs, nouns, and adjectives. Each system is trained ten times, on randomly extracted and not necessarily disjoint training sets of size 40,000. These training sets contain only (word form, lemma) pairs that belong to the respective word classes. The tables report average and simple majority vote results when each system is tested on in-domain and out-domain data, as before. Test set sizes are indicated in Table 6. Generally, the same conclusions as for the values shown in Fig. 1 apply—namely, that AliSeTra/++ and Phonetisaurus perform considerably better than LemmaGen and Mate, and that AliSeTra/++ typically perform best among all systems (particularly in the out-domain setting).

**Table 7.** Word accuracy in % for different systems, *verbs*. Each system is trained on 10 random subsets of the training data of size 40,000 each. Average and simple majority vote results indicated. In bold: Statistically indistinguishable best performances.

	Avg-InDomain	Maj-InDomain	Avg-OutDomain	Maj-OutDomain
AliSeTra	87.89	89.07	81.78	82.94
AliSeTra++	<b>88.42</b>	<b>89.72</b>	<b>83.09</b>	<b>84.51</b>
Phonetisaurus	86.98	89.64	73.78	78.40
LemmaGen	78.23	81.45	76.91	80.19
Mate	66.10	67.98	64.36	66.63

**Error Analysis.** For verbs, typical errors are mismatches of *-or/-o* endings. Such distinctions are very hard to learn for the statistical lemmatizers because it requires to know whether a verb is a *deponent verb*, that is, lacks active forms. This can actually not be regularly predicted from the characters that constitute a form, but would require lexical knowledge. Also, mismatches between conjugation classes is a common source of error. For example, the verb *transpicio* (‘look through’) is third conjugation class, for which the *-i-* ending of the stem is characteristic. A wrong lemmatization might assign a form in the morphological paradigm of *transpicio* the lemma *transpico*, which contains the characteristic ending for the (most common) first conjugation class. For nouns, the decoding

**Table 8.** Word accuracy in % for different systems, *nouns*. Each system is trained on 10 random subsets of the training data of size 40,000 each. Average and simple majority vote results indicated. In bold: Statistically indistinguishable best performances.

	Avg-InDomain	Maj-InDomain	Avg-OutDomain	Maj-OutDomain
AliSeTra	<b>78.25</b>	79.98	74.11	75.84
AliSeTra++	77.76	<b>80.96</b>	<b>74.31</b>	<b>75.92</b>
Phonetisaurus	76.74	79.28	72.98	75.64
LemmaGen	75.37	78.09	72.74	75.85
Mate	72.90	73.53	70.26	71.98

**Table 9.** Word accuracy in % for different systems, *adjectives*. Each system is trained on 10 random subsets of the training data of size 40,000 each. Average and simple majority vote results indicated. In bold: Statistically indistinguishable best performances.

	Avg-InDomain	Maj-InDomain	Avg-OutDomain	Maj-OutDomain
AliSeTra	<b>92.13</b>	<b>93.18</b>	<b>87.21</b>	<b>87.94</b>
AliSeTra++	91.50	92.14	<b>87.28</b>	<b>87.90</b>
Phonetisaurus	91.80	<b>93.57</b>	84.57	86.09
LemmaGen	85.37	86.20	84.49	85.77
Mate	71.16	71.70	70.32	71.67

problem is even more difficult for the statistical lemmatizers because gender (e.g., *-us* (masculine) vs. *-um* (neuter) lemma ending) is, to a significant degree, arbitrary and therefore unpredictable. Moreover, many declension classes have identical endings for forms in certain slots of the morphological paradigms (e.g., *-is* ending in dative plural for nouns that belong to both first as well as second declension), which is another source of difficult-to-predict error. In many of these instances, a lexicon could act as a filtering device (see below).

As to why the joint  $m$ -gram modeling Phonetisaurus performs worse than the discriminative model in the lemmatization task but not the G2P setting, we find that there are significantly more distinct ‘graphones’ (pairs of corresponding input-output subsequences such as  $(a,a)$  or  $(arenturque,o)$ ) in a comparable quantity of aligned  $(\mathbf{x}, \mathbf{y})$  pairs (e.g., 1,217 vs. 725 distinct graphones in a lemma vs. G2P data set, respectively, in a list of 10,000 aligned pairs), so that the basic entity of the joint  $m$ -gram model, the graphone, is harder to estimate.

### 4.3 Text Evaluation

We also present an evaluation of ‘real-world’ lemmatization, i.e., when the presented lemmatizers are used for the lemmatization of word forms in the context of sentences. To this end, we extract all words in all sentences from Perseus [22]. We discard all upper-case forms (since upper case forms generally do not occur

**Table 10.** Word accuracy for different systems. Each is trained 10 times on 120,000 (word form, lemma) pairs as described in the text. Type accuracy is the fraction of unique word forms lemmatized correctly, while token accuracy also takes frequency of word forms into account (irregular forms, which are harder to learn, are typically much more frequent). In bold: Statistically indistinguishable best performances.

		AliSeTra++	Phonetisaurus	LemmaGen	Mate
Type	Average	<b>61.09</b>	57.67	53.07	46.71
	Majority	63.83	60.96	57.40	49.12
Token	Average	<b>54.38</b>	50.04	47.01	41.43
	Majority	57.07	52.09	52.11	44.12

in the above outlined training data), and all word forms that the Perseus gold standard does not classify as either verbs, adjectives, or nouns.

Instead of training lemmatizers separately on each word class and then trying to resolve the resulting ambiguity upon lemmatizing a new test form  $\mathbf{x}$ , we directly train lemmatizers on word forms from all word classes.<sup>10</sup> Hence, we train each lemmatizer on a total of 120,000 word forms, consisting of 40,000 verb pairs, 40,000 noun pairs, and 40,000 adjective pairs each. Results are shown in Table 10. As can be seen, AliSeTra++ performs again best and the ordering of systems is (about) the same as in the previous experiments, i.e., AliSeTra++ > Phonetisaurus > LemmaGen > Mate. We omit indicating the results of AliSeTra because the CRF typically takes hours to days to train on the given training set size.

To say a word on why results appear relatively weak compared to the previously outlined, we note the following:

- Perseus sometimes indicates lemma variants as gold standard for input forms (e.g., in Latin i/j and u/v alternation are typically considered free variants). We count such phenomena as errors although, from a linguistic point of view, these would not constitute real errors. Still, from an evaluation perspective, this does not matter since conditions for all systems are the same.
- Most importantly, note that our training distributions do not represent the actual distributions of word forms in text. For example, we include equal numbers of verbs, nouns, and adjectives in training, but such a distribution is not likely to hold for real text.
- We note, however, that when we increase training set size from 120,000 word pairs to more than 3 million word pairs, type accuracy for LemmaGen increases from about 53 % to more than 76 % and token accuracy increases from about 47 % to more than 74 %. Training the other lemmatizers on much more data would be the logical next step, but is omitted in this evaluation because training times are considerable.

<sup>10</sup> We also performed the alternative decoding strategy where lemmatizers are separately trained, but found it to perform worse.

**Table 11.** Combining a lexicon and statistical lemmatizer.

	Type	Token
Lexicon	72.21	69.11
Lexicon + AliSeTra++	76.22	71.38

**Table 12.** TreeTagger lemma token accuracy on a subpart of the PL and accuracy values when the lemmatizer is complemented by our trained lemmatizers.

	Token Accuracy
TreeTagger	86.23 %
TreeTagger + AliSeTra++	88.56 %
TreeTagger + LemmaGen	89.37 %

Finally, we address **two more questions** in the context of real-world lemmatization. The first is how a *lexicon* can be combined with the trained lemmatizers that we have outlined. To this end, we (a) lemmatize each word in Perseus with simple lexicon look-up: if the word form is in our lexicon (see Sect. 3), retrieve the corresponding lemma. If several lemmas are associated with the form, pick one of them randomly. This strategy leads to a word type accuracy of 72.21 % (average over ten runs). Most problems in this case are due to lemma ambiguity: while the lexicon has an OOV rate of only 1.03 %, each form is on average associated with 1.49 lemmas. For example, for the form *canis* the lexicon outputs the lemma suggestions *canes*, *cania*, *cano*, *canum*, *canus*, *canis*. Alternatively, (b) for each form to lemmatize, we let AliSeTra++ output its  $k$ -best lemma suggestions (we choose  $k = 10$ ) and choose the first-best that occurs in the lexicon. If none of the  $k$  best is in the lexicon, we simply choose the first-best suggestion of AliSeTra++. Table 11 shows that this leads to a type accuracy of 76.22 %, which is not only considerably better than AliSeTra++’s performance without a lexicon (61.09 %) but also better than the lexicon itself.

Secondly, we ask how our trained lemmatizer can complement an existing tagger or lemmatizer. To this end, we download the Latin TreeTagger from Gabriele Brandolini<sup>11</sup> and have it lemmatize (and tag) a subpart of the Patrologia Latinae (PL) [16].<sup>12</sup> This tagger and lemmatizer has a token lemma accuracy of 86.23 % on the indicated text. About 50 % of all errors are unknown word forms. If, for each unknown word form, we substitute the prediction of AliSeTra++ (trained on 120,000 pairs), or LemmaGen (trained on more than 3 million pairs), lemma token accuracy increases to 88.56 % and 89.37 %, respectively, which constitute improvements of about 2.7 % and 3.6 %, respectively. See Table 12 for a summary.

<sup>11</sup> Available at <http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/>.

<sup>12</sup> We could not use Perseus because the TreeTagger was trained on Perseus.

## 5 Conclusion

We have considered the statistical lemmatization problem in which lemmatizers are trained on (word form, lemma) pairs, which enables them to learn the morphological processes involved in lemmatization. We have investigated whether general purpose string-to-string transduction models are suitable for this task, and have seen that they typically perform (much) better than more restricted lemmatization techniques/heuristics based on suffix transformations. We have also investigated how the learned lemmatizers can complement lexicon-based systems, e.g., by tackling the OOV and/or the disambiguation problem.

Our next step is to train the described lemmatizers on the full size of our database, from which we expect huge accuracy gains, since they have concurrently been trained only on a tiny fraction of it (much less than 10%). For the CRF based models, training them on the full size of our database is actually a scaling challenge since they must then learn hundreds of millions of features, but this can be accommodated by training many subsystems on disjoint portions of the data and a subsequent aggregation step. For real world lemmatization, this also requires to train the systems on distributions that reflect those found in ‘real’ text rather than on random word pair lists retrieved from a lexicon, as done here. Finally, by combining the so-enhanced lemmatizers with our lexicon, very high accuracy lemmatizers can be expected.

## References

1. Bartlett, S., Kondrak, G., Cherry, C.: Automatic syllabification with structured SVMs for letter-to-phoneme conversion. In: McKeown, K., Moore, J.D., Teufel, S., Allan, J., Furui, S. (eds.) *ACL*, pp. 568–576. Association for Computational Linguistics, Morristown (2008)
2. Bisani, M., Ney, H.: Joint-sequence models for grapheme-to-phoneme conversion. *Speech Commun.* **50**(5), 434–451 (2008)
3. Bohnet, B.: Top accuracy and fast dependency parsing is not a contradiction. In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, Coling 2010 Organizing Committee, Beijing, China, pp. 89–97, August 2010. <http://www.aclweb.org/anthology/C10-1011>
4. Brill, E., Moore, R.C.: An improved error model for noisy channel spelling correction. In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, ACL 2000*, pp. 286–293. Association for Computational Linguistics, Stroudsburg (2000)
5. Daelemans, W., Groenewald, H.J., Huyssteen, G.B.V.: Prototype-based active learning for lemmatization. In: Angelova, G., Bontcheva, K., Mitkov, R., Nicolov, N., Nikolov, N. (eds.) *RANLP*, pp. 65–70. *RANLP 2009 Organising Committee/ACL*, Morristown (2009)
6. Dreyer, M., Smith, J., Eisner, J.: Latent-variable modeling of string transductions with finite-state methods. In: *EMNLP*, pp. 1080–1089. *ACL* (2008)
7. Eger, S.: Sequence segmentation by enumeration: an exploration. *Prague Bull. Math. Linguist.* **100**, 113–132 (2013)

8. Eger, S., vor der Brück, T., Mehler, A.: Lexicon-assisted tagging and lemmatization in Latin: a comparison of six taggers and two lemmatization methods. In: Latech 2015. Association for Computational Linguistics (2015, accepted)
9. Gesmundo, A., Samardzic, T.: Lemmatisation as a tagging task. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (vol. 2: Short Papers), pp. 368–372. Association for Computational Linguistics (2012). <http://aclweb.org/anthology/P12-2072>
10. Jiampojamarn, S., Cherry, C., Kondrak, G.: Joint processing and discriminative training for letter-to-phoneme conversion. In: Proceedings of ACL-08: HLT, pp. 905–913. Association for Computational Linguistics, Columbus, June 2008. <http://www.aclweb.org/anthology/P/P08/P08-1103>
11. Jiampojamarn, S., Cherry, C., Kondrak, G.: Integrating joint n-gram features into a discriminative training framework. In: NAACL-HLT, pp. 697–700. Association for Computational Linguistics (2010)
12. Juršič, M., Mozetič, I., Lavrač, N.: Learning ripple down rules for efficient lemmatization. In: Mladenić, D., Grobelnik, M. (eds.) Proceedings of the 10th International Multiconference Information Society, pp. 206–209. IJS, Ljubljana (2007)
13. Juršič, M., Mozetič, I., Lavrač, N.: LemmaGen: multilingual lemmatisation with induced ripple-down rules. *J. Univ. Comput. Sci.* **16**, 1190–1214 (2010)
14. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of 18th International Conference on Machine Learning, pp. 282–289. Morgan Kaufmann, San Francisco (2001)
15. Mehler, A., vor der Brück, T., Gleim, R., Geelhaar, T.: Towards a network model of the coreness of texts: an experiment in classifying Latin texts using the tlab Latin tagger. In: Biemann, C., Mehler, A. (eds.) Text Mining: From Ontology Learning to Automated text Processing Applications. Theory and Applications of Natural Language Processing, pp. 87–112. Springer, Berlin (2015)
16. Migne, J.P. (ed.): *Patrologiae Cursus Completus: Series Latina*, vol. 1–221. Chadwyck-Healey, Cambridge (1844–1855)
17. Nguyen, N., Guo, Y.: Comparisons of sequence labeling algorithms and extensions. In: Ghahramani, Z. (ed.) ICML. ACM International Conference Proceeding Series, vol. 227, pp. 681–688. ACM, New York (2007)
18. Novak, J.R., Minematsu, N., Hirose, K.: WFST-based grapheme-to-phoneme conversion: open source tools for alignment, model-building and decoding. In: Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing, pp. 45–49. Association for Computational Linguistics, Donostia-San Sebastián, July 2012. <http://www.aclweb.org/anthology/W12-6208>
19. Porter, M.: An algorithm for suffix stripping. *Program Electron. Libr. Inf. Syst.* **14**(3), 130–137 (1980)
20. Richmond, K., Clark, R.A.J., Fitt, S.: Robust LTS rules with the Combilex speech technology lexicon. In: INTERSPEECH, pp. 1295–1298. ISCA (2009)
21. Sherif, T., Kondrak, G.: Substring-based transliteration. In: Carroll, J.A., van den Bosch, A., Zaenen, A. (eds.) ACL. Association for Computational Linguistics, Morristown (2007)
22. Smith, D.A., Rydberg-Cox, J.A., Crane, G.R.: The Perseus project: a digital library for the humanities. *Literary and Linguistic Computing* **15**(1), 15–25 (2000). <http://llc.oxfordjournals.org/content/15/1/15>
23. Toutanova, K., Cherry, C.: A global model for joint lemmatization and part-of-speech prediction. In: Su, K.Y., Su, J., Wiebe, J. (eds.) ACL/IJCNLP, pp. 486–494. Association for Computational Linguistics, Morristown (2009)

Systems and Frameworks for Computational  
Morphology

Fourth International Workshop, SFCM 2015, Stuttgart,  
Germany, September 17-18, 2015. Proceedings

Mahlow, C.; Piotrowski, M. (Eds.)

2015, XIII, 185 p. 23 illus. in color., Softcover

ISBN: 978-3-319-23978-1