Romanian Syllabication using Machine Learning

Liviu P. Dinu[1,2], Vlad Niculae[1,3], and Octavia-Maria Șulea[1,2]

[1] Center for Computational Linguistics, University of Bucharest
[2] Faculty of Mathematics and Computer Science, University of Bucharest
[3] University of Wolverhampton
ldinu@fmi.unibuc.ro, vlad@vene.ro, mary.octavia@gmail.com

Abstract. The task of finding syllable boundaries can be straightforward or challenging, depending on the language. Text-to-speech applications have been shown to perform considerably better when syllabication, whether orthographic or phonetic, is employed as a means of breaking down the text into units bellow word level. Romanian syllabication is non-trivial mainly but not exclusively due to its hiatus-diphthong ambiguity. This phenomenon affects both phonetic and orthographic syllabication. In this paper, we focus on orthographic syllabication for Romanian and show that the task can be carried out with a high degree of accuracy by using sequence tagging. We compare this approach to support vector machines and rule-based methods. The features we used are simply character n-grams with end-of-word marking.

1 Introduction

In this paper we describe systems for solving end-of-the-line hyphenation in Romanian. The most challenging aspect is that of distinguishing between hiatus and diphthongs, an ambiguity exemplified in Table 1, as well as between the letter *i* which can surface either as a non-vocalic element, or as a proper vowel, affecting thus the syllable boundary. Although the words in the dataset we used for this task were marked for main stress, we ignored this information in order to build a system that works on plain text.

Table 1: Romanian hiatus/ diphthong ambiguity

sequence	hiatus		diphthong	
ai	*ha-i-nă*	(heinous)	*hai-nă*	(coat)
iu	*pi-u-li-ță*	(screw nut)	*piu-re*	(purée)
oa	*bo-ar*	(bull herder)	*oa-meni*	(humans)

2 Syllabication and end-of-the-line hyphenation

Syllabication or syllabification is the task of detecting syllable boundaries within words. The boundaries depend on the phonetic structure of those words and on phonotactic constraints of the respective language. End-of-the-line hyphenation, on the other hand, is the task of determining where to break up a word when it doesn't fit at the end of a line in a written text and this separation may or may not overlap with a syllable boundary, depending on how close its rules are to those of proper syllabication in a particular language. This is why some have dubbed the product of applying these end-of-the-line hyphenation rules to a whole word, and not just in one place to accommodate line length, as *orthographic syllabication* [2].

In Romanian, the rules for end-of-the-line hyphenation, as discussed in the orthographic dictionary [4], are not based on phonetics and, if applied to the whole word, may not render a proper phonetic syllabication, so a distinction can and should be made between phonetic and orthographic syllabication. When analyzing the orthographical rules listed in the dictionary, we see that they diverge from the rules of phonetic syllabication by often times going against the maximal onset principle and, thus, rendering different codas and onsets, but not affecting the nucleus. This means that the diphthong-hiatus ambiguity is maintained in and remains a source of problem for both types of syllabication. The Romanian orthographic syllabication rules also diverge from the phonetic ones in that they are mostly independent from stress placement since Romanian, unlike French, doesn't normally mark stress orthographically. In what follows we will take into account these aspects.

The main application for syllabication, whether phonetic or orthographic, has been shown to be in text-to-speech systems, where a need for letter-to-phoneme conversion is great. Specifically, the performance of the conversion has been shown [2] to increase for English when done on segments smaller than words, namely syllables, with the phonetic syllable increasing performance a bit more. For orthographic syllabication, structured SVMs [2] and CRFs [10] have been used. We investigated unstructured SVMs and CRFs when attempting to solve only the task of orthographical syllabification, due to it being much better resourced for Romanian. Attempts at solving the task of syllabication in Romanian have used rule-based algorithms [9] as well as contextual information (previous and following segments) with contextual grammars [5].

3 Approach

As mentioned previously, one difficulty of the syllabication task for Romanian, of either type, lies in the Romanian diphthong-hiatus contrast, which, although predictable for native speakers from diachronic information [3], is difficult to discriminate based only on synchronic information. This is where statistical machine learning techniques will come into play.

Table 2: Status difference in word-final i

part of speech	nucleic		in the coda	
verb	*ci-ti*	(to read)	*ci-teşti*	(you read)
noun	*ti-gri*	(tigers)	*e-levi*	(pupils)
adjective	*ne-gri*	(black)	*albi*	(white)

3.1 Classifiers

Linear Support Vector Machine (SVM) A linear support vector machine is a simple, unstructured learner that predicts one label for each input sample. SVMs are discriminative max-margin models optimizing the hinge loss.

$$L_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$$

The optimization problem can be approached using the *scikit-learn* stochastic gradient descent (SGD) SVM solver. The ℓ_2 regularization is parametrized in terms of α.

The objective function used by the SGD solver is:

$$E(w, b) = \sum_{i=1}^{M} L_{\text{hinge}}(y_i, wx_i + b) + \frac{\alpha}{2} \sum_{i=1}^{d} w_i^2$$

For SGD, the learning rate schedule used for the updates is given by:

$$\eta^{(t)} = \frac{1}{\alpha(t_0 + t)}$$

In the above, t is the time step, going from 1 to $M * N$ where M is the number of samples and N is the number of passes over the training set (epochs). In our case we used the default value of $N = 5$. t_0 is determined based on a heuristic proposed by Léon Bottou in the SvmSgd package[4], such that the expected initial updates are comparable with the expected size of the weights. This schedule is the default one in the *scikit-learn* stochastic gradient descent classifier and it performed well.

Sequence tagging Some problems can be structured as related sequential prediction problems, where the classification of one point is related to the classification of the ones right before or after it in the sequence. Such models are common in NLP because of the sequential nature of text, and are used for tasks like POS tagging. The Conditional Random Field (CRF) model has a learning algorithm [6] that minimizes the regularized negative conditional log likelihood

[4] `http://leon.bottou.org/projects/sgd`

of the data. We used L-BFGS optimization. The regularization parameter c is not halved like it is in the SVM formulation:

$$E(\theta) = -\sum_{i=1}^{M} \log p(y_i|x_i;\theta) + c\sum_{i=1}^{K} \theta^2$$

We implemented three systems for orthographic syllabication of Romanian words. The first is a simple rule-based approach which aims to implement as many of the rules [5] for orthographic syllabication given in [4] as possible. The second uses a linear SVM to decide what to do with every possible candidate split in the word. The third approach uses a CRF model. The features used for the two machine learning approaches are character n-grams.

The advantage of a sequence model over the SVM is that it has the modeling power to jointly analyze consecutive splits, whereas the SVM is limited to local knowledge. It is not aware of interaction between consecutive candidate splits within a same word (e.g. the CRF can learn that it should be rare to split on both sides of a single consonant).

3.2 Software

The software we use is the *scikit-learn* machine learning library for the Python scientific computing environment version 0.12.1 [8]. The library provides efficient text n-gram feature extraction using the sparse matrix implementation in SciPy[6]. We use the SVM implementation by stochastic gradient descent.

We also used *CRFsuite* version 0.12 [7] for its implementation of CRF inference and training.

3.3 Dataset and preprocessing

The dataset used is based on the morphological dictionary *RoMorphoDict* [1], which is available from the author and has two versions. The resource relevant to our task provides a long list of word forms along with their hyphenated form with accent marking. An online version of this second data resource is available for querying at `http://ilr.ro/silabisitor/`.

The dataset is annotated for accent by using Unicode accented characters. Unlike, for instance, French, in Romanian the accent is usually not written and therefore not usually available as input, so we strip it completely. We do not strip down to the ASCII character set though, because Romanian has the characters *ă, â, î, ș, ț* – these are not accented characters, but separate letters.

3.4 Features

The feature extraction consists of extracting character n-grams that form strings. This is parametrized by the n-gram size n. For example, we will consider $n = 3$,

[5] Also available, in Romanian, at `http://ilr.ro/silabisitor/reguli.php`

[6] `http://docs.scipy.org/doc/scipy/reference/sparse.html`

the word *dinosaur* and the split between *o* and *s*. The position induces two strings, *dino* and *saur* but we are only interested in the window of radius n around the split, so we are left with *ino* and *sau*. Since the bag-of-n-grams features we use for the SVM loses the order, we consider adding a special marker, obtaining *ino$* and *$sau*. The n-grams of length up to 3 are: *i, n, o, $, in, no, o$, ino, no$* and the analogous for the right hand side. This is then vectorized into counts, with the observation that features that occur with the frequency of 1 across all instances are obviously irrelevant and are ignored.

The vectorization produces one matrix for the left sides of the splits and one for the right side of the splits. These are stacked horizontally to give our data matrix. For the CRF, the feature extraction is the same, but the sparse vectorized representation is replaced with an input like:

```
1 c[-3]=i c[-2]=n c[-1]=o c[-3-2]=in c[-2-1]=no c[-3-2-1]=ino
  c[1]=s c[2]=a c[3]=u c[12]=sa c[23]=au c[123]=sau
```

The format above is the one accepted as input by *CRFsuite*. Because the feature names include the offset, the dollar marker would provide no useful information. The names could just as well be arbitrary: *CRFsuite* cannot understand that `c[-2-1]` means the bigram just before the split, but the values that a certain feature tends to take carry the discriminative information.

Both the SVM and the CRF were trained using ℓ_2 regularization, so we would have another system parameter that controls the amount of regularization applied, α. In the case of the SVM, we also investigate the difference in the results if the boundary marker *$* is used or not. These parameters are optimized using grid search to maximize the classification accuracy of individual splits using 3-fold cross-validation over the training set.

3.5 Generating training samples

The average word in our dictionary has 9.96 characters and 4.24 syllables. This means that each word generates around 9 training instances (possible splits), out of which we expect around 3 to be labeled as true, and the rest as false. Prior to generating training instances, we split the words into a training and test set, each consisting of $262,764$ words.

For each word of length n we generate $n-1$ instances, corresponding to each position between two letters of the word. Instances are labeled as positive if a hyphen can be inserted there, or negative if not.

This tagging method is called NB labelling [2], because we label each split as boundary (B or 1) or no boundary (N or 0). For example, the word *di-a-mant* (diamond) would be encoded as:

d	i	a	m	a	n	t
0	1	1	0	0	0	

A slightly more informative way of assigning labels, introduced also in [2], is to use numbered NB (#NB) tags: each split is labeled with the distance from the

Table 3: Results for hyphenation models. The scores are evaluated on the test set. In the numbered NB case, we are only evaluating with respect to class 0, the one marking the position of the hyphens. Correctly identifying all tags $0-7$ is not directly relevant for this application, but their presence helps.

model	n	marker	regularization	Hyphen acc.	Hyphen F_1	Word acc.
Rule	-	-	-	94.31%	92.12%	60.67%
Svm nb	4	yes	$\alpha = 10^{-5}$	98.72%	98.24%	90.96%
Svm #nb	4	yes	$\alpha = 10^{-6}$	98.82%	98.37%	91.46%
Crf nb	4	-	$c = 0.1$	99.15%	98.83%	94.67%
Crf #nb	4	-	$c = 1.0$	99.23%	98.94%	95.25%

last hyphen:

```
d  i  a  m  a  n  t
1  0  0  1  2  3
```

This way the class corresponding to boundaries is class 0, and classes $1, 2, ...$ are all negative. We are still dealing with a classification problem, as the classes are discrete. On our dataset the maximum class is 7. This helps if we have a structured sequence model, because from class k the Markov chain can only transition to class 0 or $k + 1$. An unstructured SVM cannot take advantage of this.

3.6 Results

Apart from the accuracy on the test data, we looked at precision and recall scores. All measures are relative to training instances, which correspond to splits within words. A more tough evaluation metric is at the word level: a word is considered correctly hyphenated if all of the splits within it were correctly classified. This score should show the advantage of using a sequence model.

The rule-based algorithm obtained 60.67% splitting accuracy at word level and 94.31% accuracy, within the word, at split level. The small accuracy at word level we found to not only be influenced by the ambiguity between hiatus and diphthongs or glide+vowel sequences, which cannot be distinguished based on the orthographic rules, but also by the difference in vocalic status of the word-final *i*, which can form a new nucleus or stay in the coda. Table 2 shows examples of word-final *i* either determining a new syllable or staying in the coda. Looking at the data relating to word-final *i* we see that it is not only dependant on phonetic constraints (i.e. it is vocalic after stop+liquid) but also on morphological ones (i.e. it is vocalic in infinitive-based forms). However, since the system does not have access to part of speech tags, we could not refine the rules of the algorithm any further. Because it is not a trained system, we could evaluate the rule-based system on the entire corpus (training and test data).

We found it to be consistent, scoring 60.66% word accuracy and 94.32% split accuracy.

We went on to evaluate the SVM system. The hyperparameters found to optimize it are $n = 4, \alpha = 10^{-5}$, `marker=true`. The grid we investigated ranged over $n \in \{1, 2, 3, 4\}$ and $\alpha \in \{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$. For the CRF system, we did not use marking, because the feature representation includes feature names that encode the offsets. The space over which we looked for n is the same, while for c we searched $\{0.01, 0.1, 1.0, 10.0\}$.

The best results and the associated parameter values are presented in Table 3. The trained models and scripts needed to apply on new data are made available at `https://github.com/nlp-unibuc/ro-hyphen`.

4 Conclusions

In this paper, we have looked at ways of detecting orthographic syllable boundaries, with the intent to rule over the diphthong-hiatus ambiguity which arises when attempting to implement a rule-based algorithm for either types of syllabication in Romanian. We've seen that this ambiguity along with status ambiguity of word final *i* causes great drop in performance for a rule-based algorithm (60.66% word level split accuracy vs. 94.32% split accuracy at syllable level), but is handled very well by an SVM or CRF system, with the CRF having the best accuracy at word level (95.25%).

The systems we used can be viewed with respect to a trade-off spectrum. The rules in the rule-based system can take any form and they can model very complex interactions between features. This model has the largest predictive power, but the rules are written by hand, therefore limiting its practicality and its performance. At the opposite end of the spectrum is the SVM classifier, which applies a simple linear decision rule at each point within a word, looking only at its direct context. This simple approach outperforms the rule-based system by being trained on large amounts of data. The sequence tagger is more successful because it exploits the data-driven advantage of the SVM, while having more modeling power. This comes at a cost in model complexity, which influences training and test times.

Acknowledgements

The contribution of the authors to this paper is equal. The work of Liviu P. Dinu was supported by a grant of the Romanian National Authority for Scientific Research, CNCS – UEFISCDI, project number PN-II-ID-PCE-2011-3-0959.

References

1. Barbu, A.M.: Romanian lexical databases: Inflected and syllabic forms dictionaries. In: Sixth International Language Resources and Evaluation (LREC'08) (2008)
2. Bartlett, S., Kondrak, G., Cherry, C.: Automatic syllabification with structured svms for letter to phoneme conversion. In: 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL 08: HLT). pp. 568–576. Association for Computational Linguistics, Columbus, Ohio (2008), http://www.aclweb.org/anthology/P/P08/P08-1065
3. Chitoran, I., Hualde, J.I.: From hiatus to diphthong: the evolution of vowel sequences in romance. Phonology 24, 37–75 (2007)
4. Collective: Dicționarul ortografic, ortoepic și morfologic al limbii române. Bucharest: Romanian Academy (2010), 2nd edition, revised. (In Romanian)
5. Dinu, L.P., Dinu, A.: A parallel approach to syllabification. In: Gelbukh, A.F. (ed.) CICLing. Lecture Notes in Computer Science, vol. 3406. Springer (2005)
6. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning. pp. 282–289. ICML '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001), http://dl.acm.org/citation.cfm?id=645530.655813
7. Okazaki, N.: CRFsuite: a fast implementation of Conditional Random Fields (CRFs) (2007), http://www.chokkan.org/software/crfsuite/
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (Oct 2011)
9. Toma, S.A., Oancea, E., Munteanu, D.: Automatic rule-based syllabication for Romanian. In: Proceedings of the 5th Conference on Speech Technology and Human-Computer Dialogue (2009)
10. Trogkanis, N., Elkan, C.: Conditional Random Fields for word hyphenation. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. pp. 366–374. Association for Computational Linguistics, Uppsala, Sweden (July 2010), http://www.aclweb.org/anthology/P10-1038