CoLi Final Project: Diacritics Restoration

Herbert Ullrich (**2576412**)

March 25, 2019

Abstract

The project examines the usability of Hidden Markov Models and Viterbi algorithm for the *diacritic restoration* problem. We discuss the possible HMM configurations for this task and arrive to the one with diacritical marks as *states* and n-grams of characters as possible *observations*.

We then give an implementation of such a system using NLTK and examine its accuracy on 10 Indo-European languages using standardized corpora for the diacritics restoration task.

To encourage further system examination, we provide a Jupyter notebook of demos along with a library of loadable pre-trained restorers to take the costly operations away from the user.

1 Introduction

Diacritic restoration is the task of turning the text without diacritics back into the natural language sentences *with* diacritics it was supposed to represent.

Of all motivations, let us mention the English QWERTY keyboard, frequently used by programmers, that trades off the diacritics for the speed of writing, thus creating ambiguities and losing accent information some diacritic marks may carry.

For a native speaker, *diacritics restoration* is typically a trivial task, as a single word typically only has a small set of meaningful diacritizations. From those, one can easily pick the correct (– intended) one, looking at its close context.

We are going to tackle this problem programmatically, trying to exploit the aforementioned feats. Hidden Markov Models come to mind when the narrow choice and a role of context is of importance.

1.1 Project structure

```
/
├── corpora ..... directory with corpus files, proper contents excluded from submission
│   └── {language code} ..... example structure for storing corpus files
│       ├── target_train.txt . training set for the tagger – with diacritics, not tokenized
│       └── target_test.txt ..... testing set – with diacritics, not tokenized
├── doc
│   └── documentation.tex ..... LaTeX sources for this documentation
├── pretrained ..... pretrained taggers loadable using HmmNgramRestorer.load()
│   └── {language code} ..... all models for given language
│       └── {n}-gram.pickle ..... model observing n-grams of characters
├── .gitignore
├── accents.py ..... module defining how to handle and compose the diacritics
├── demos.ipynb ..... Jupyter notebook with usage demos
├── detokenize_corpora.py ..... simple ad-hoc bulk detokenizer
├── diacritic_restorer.py .... main module of the project, defines tagger class
├── evaluate.py ..... script for pretrained taggers bulk evaluation
├── LICENSE.md
├── measurement_table.py ..... simple ad-hoc script turning raw data into table 1
├── pretrain.py .. module for bulk training taggers on given corpora, saving bin. results
└── README.md
```

The contents of the data directories `corpora` and `pretrained` were significantly reduced in order to enable the e-mail submission.

The models can be trained and tested on any diacritical corpora instead. However, if you want to recover the full data of the original project (thus enabling some of the demos), you can replace the stub directories with the following additional downloads:

corpora: `http://herbert.saarland/corpora.zip`

pretrained: `http://herbert.saarland/pretrained.zip`

1.2 Used corpora

We have decided to use the *Corpus for training and evaluating diacritics restoration systems* [2] proposed in 2018 as the standard training and testing set for the diacritics restoration task.

The training data sets come from localizations of Wikipedia (therefore having some language norms), testing sets may contain text from other websites too. Corpora were built for training the bidirectional character-level recurrent neural-network in [3], which is the current state-of-the-art. During the evaluation of this project, corpora are temporarily available on `http://herbert.saarland/corpora.zip`

2 Method

As decided in our final project assignment, we are going to use represent our problem using Hidden Markov Models, trained using the Baum-Welch algorithm for ML estimate. We will be finding the word diacritization using the Viterbi algorithm.

We have dropped the idea of implementing a competitive Naïve Bayes classifier in favor of training and testing our model on numerous languages.

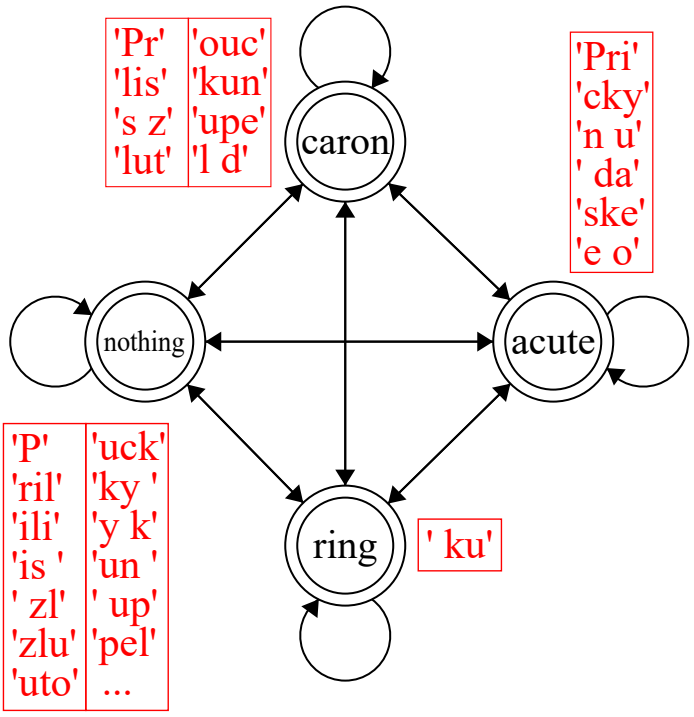

Figure 1: 3-gram based Hidden Markov Model emitting Czech sentence *Příliš žluťoučký kůň úpěl ďábelské ódy.* (*Possible observations* are placed in a red box counter clockwise from their respective HMM *states*)

2.1 Rejected approaches

The original idea was to have the HMM hidden states represent all **possible diacriticisations** of the **entire words** in the observable sequence. This method turned out to be inefficient due to its space complexity and to the fact that the most of the words of the language are actually the ones we did not observe in training data (Zipf's Law).

Having a **character-level** HMM, switching between the states denoting the diacritical marks turns to be vastly inaccurate[1], as the most common diacritization of every character alone is usually one with no diacritics. Using the ML estimation, we then arrive to a costly way to output a text without diacritics.

2.2 Accepted approach

The final approach we chose to follow was to be observing the **n-grams** of characters as occured in training data, tagging every such a n-gram with a diacritical mark over its last character, as in **figure 1**.

This approach significantly reduces the complexity (as there is only a constant number of HMM states – signs) while maintaining better accuracy which is rising with the n.

3 Implementation

For simplicity and performance optimization, we have decided to build the explained system as a wrapper of the NLTK's `HiddenMarkovModelTagger` and `HiddenMarkovModelTrainer` classes [1]. Our legacy in-house made HMM can be found at (might still get used in future if there is an optimization we can use it for):
`https://github.com/heruberuto/diacritic-restoration/blob/master/models/hmm.py`

Class `HmmNgramRestorer` serves as the facade for the NLTK's and offers additional (de)serialization using Pickle. Upon training, class `CorpusNgramBuffer` mimics the `list` NLTK is trying to iterate through, outputting a parsed tagged n-grams for each line of a buffered file, thus enabling training on collections that would not fit in memory.

Class `DiacriticsAccuracyCounter` is instantiated during the HMM testing, collecting scores for the accuracy experiments.

The core modules `accents.py` and `diacritics_restorer.py` were documented in detail using standardized Sphinx docstrings.

The scripts for detokenizing corpora, conducting experiments and generating outputs in `{evaluate,detokenize_corpora,evaluate,pretrain,measurement_table}.py` are documented in-line, sufficiently to their complexity.

4 Measuring accuracy

Please see the **table 1** for the result of our measuring experiments.

The measurement procedure can be found within class `DiacriticsAccuracyCounter` and can be instantiated running `evaluate.py`.

[1] in fact, see the unigram model's results in table 1

language	training sentences	diacritical words	n	accuracy			
				tag	word	dia-word	alpha-word
Croatian	802,610	14.3%	1	0.977	0.857	0	0.85
			2	0.98	0.874	0.206	0.868
			3	0.985	0.903	0.491	0.898
			4	**0.989**	**0.929**	**0.618**	**0.925**
Czech	952,909	47.8%	1	0.892	0.522	0	0.498
			2	0.904	0.549	0.141	0.527
			3	0.924	0.628	0.321	0.61
			4	**0.94**	**0.689**	**0.433**	**0.674**
Slovak	613,727	41.1%	1	0.922	0.589	0	0.564
			2	0.925	0.614	0.123	0.591
			3	0.938	0.669	0.274	0.65
			4	**0.951**	**0.727**	**0.42**	**0.71**
Irish	50,825	29.7%	1	0.942	0.703	0	0.695
			2	0.948	0.732	0.21	0.725
			3	0.957	0.778	0.403	0.772
			4	**0.961**	**0.8**	**0.447**	**0.795**
Hungarian	1,294,605	47.4%	1	0.906	0.526	0	0.507
			2	0.906	0.52	0.0792	0.501
			3	0.924	0.587	0.267	0.57
			4	**0.939**	**0.649**	**0.38**	**0.635**
Polish	1,069,841	31.6%	1	0.946	0.684	0	0.669
			2	0.949	0.701	0.164	0.686
			3	0.961	0.762	0.353	0.75
			4	**0.969**	**0.809**	**0.493**	**0.8**
Romanian	837,647	28.2%	1	0.95	0.723	0.0234	0.709
			2	0.953	0.742	0.164	0.729
			3	0.958	0.768	0.31	0.757
			4	**0.964**	**0.797**	**0.431**	**0.787**
French	1,818,618	16.6%	1	0.97	0.834	0	0.826
			3	0.971	0.84	0.0933	0.833
			4	**0.975**	**0.861**	**0.274**	**0.854**
Spanish	1,735,516	11.8%	1	0.981	0.882	0	0.88
			3	0.982	0.893	0.218	0.89
			4	**0.985**	**0.909**	**0.378**	**0.907**
Latvian	315,807	46.8%	1	0.915	0.532	0	0.502
			3	0.924	0.589	0.24	0.563
			4	**0.939**	**0.654**	**0.38**	**0.632**

Table 1: Measurements of per-tag (single diacritical mark) and per-word accuracy of an n-gram based HMM diacritic restorer. We call dia-word a word that contains at least one diacritical character. The dia-word percentage is listed for testing set. We call alpha-word a word with at least one alphabetic character.

We have measured 4 scores - the accuracy of tags (single diacritic character), the accuracy of words (at their entirety), accuracy of words with at least one diacritical mark, and the standard accuracy of alpha words (words with at least one alphabetic character).

We have observed that the accuracy rises significantly with the size of n-grams and that some languages are significantly easier (harder) than the others (e.g. Croatian).

That can be caused by a simplicity of diacritical rules and their disambiguity. Sadly, we could not conclude the experiments for our most accurate feasible models with $n = 6$ due to running out of memory during the tests.

5 Demos

Please see the attached demonstration sheet `demos.ipynb`, that covers all the basic functionalities of our system.

It also presents several pre-trained models we consider enjoyable to play around with.

6 Conclusions

As hard as we tried, the accuracy got nowhere near the current state-of-the-art systems. For example, our 4-gram Croatian model has an alpha-word accuracy of 92.5%, properly trained *Lexicon* model (that replaces every word with its most common observed diacritization) achieves 99.31% and the Náplava's neural networks ([3]) get 99.67%.

6.1 Recommendations

Throughout implementing and testing our models, we have made several remarks on the possible system improvements and further research motivations:

- **"Tailed"** n-grams – the HMM state could not be the diacritical mark over n-gram's last letter, but, think, the penultimate one — that would give the tagger some context from both sides and would further punish faults of "deciding too early" (tagging "...čhodit" even when "ch" behaves as a single letter with no possible diacritizations in Czech — and the trainings cover that)

- **Combine with Lexicon** – the HMM tagger could work well deciding ambiguities (or unobserved words) for the Lexicon system, thus creating system more accurate than both the original ones (if such an ambiguity decisioning is not implemented in *Lexicon* already as it is)

References

[1] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[2] Jakub Náplava, Milan Straka, Jan Hajič, and Pavel Straňák. Corpus for training and evaluating diacritics restoration systems, 2018. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[3] Jakub Náplava, Milan Straka, Jan Hajič, and Pavel Straňák. Diacritics Restoration Using Neural Networks. In Nicoletta Calzolari (Conference chair), Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 7-12, 2018 2018. European Language Resources Association (ELRA).