A reproduction of Apple's bi-directional LSTM models for language identification in short strings

Mads Toftrup[*] [€], **Søren Asger Sørensen**[*] [€], **Manuel R. Ciosici**[$], and **Ira Assent**[€]
[€] Computer Science Department, Aarhus University
[$] Information Sciences Institute, manuelc@isi.edu

Abstract

Language Identification is the task of identifying a document's language. For applications like automatic spell checker selection, language identification must use very short strings such as text message fragments. In this work, we reproduce a language identification architecture that Apple briefly sketched in a blog post. We confirm the bi-LSTM model's performance and find that it outperforms current open-source language identifiers. We further find that its language identification mistakes are due to confusion between related languages.

1 Introduction

Automatic Language Identification is the task of identifying a document's language, an essential task for document classification and machine translation (Ling et al., 2013). General-purpose, open-source Language Identification tools like *langid.py* (Lui and Baldwin, 2012) and FastText (Grave, 2017) are the *de facto* standards for Language Identification in large documents.

During the last two decades, text messaging and social media have generated large amounts of short plain-text documents. Language identification on partial and complete short texts presents unique challenges (Jauhiainen et al., 2019). Successful Language Identification can support marketing, political, and socioeconomic analyses on large corpora of short texts such as tweets. Such analyses can, for example, study hate speech towards immigrants and women (Basile et al., 2019) or seek to understand support groups for smoking cessation (Prochaska et al., 2012).

On a smartphone, Language Identification on short texts can support several features. Language identification of incoming text messages can help virtual assistants read incoming text messages, which can be an essential tool for minorities such as visually impaired multilingual speakers.

Language identification can also help when typing short texts. Identifying language from the first few characters typed (a very short string) can allow a smartphone to select the correct spelling and grammar checker automatically. Such features motivated a team at Apple to study character-level Language Identification using bi-directional LSTMs (Apple, 2019).

This paper reproduces the architecture presented in an industry blog post (Apple, 2019) on Language Identification on extremely short strings (10 characters or less). The blog post briefly sketches the language identification system used by Apple's smartphones and computers. However, due to the use of internal, proprietary corpora, the architecture's performance cannot be compared with the current *de facto* standards for Language Identification: the open-source tools *langid.py* (Lui and Baldwin, 2012) and *FastText* (Joulin et al., 2017, 2016; Grave, 2017).

Our reproduction confirms the performance described in the original blog post (Apple, 2019). We go beyond mere reproduction and (1) compare the bi-LSTM model with the current *de facto* standards for Language Identification and (2) analyze performance on related languages. We find that the bi-LSTM is more accurate than out-of-the-box FastText and langid.py, even outperforming the re-trained FastText. Our results suggest that the bi-LSTM architecture could be an alternative to FastText and langid.py for Language Identification on short strings.[1]

[*] Equal contribution

[1] Our source code and models are available at https://github.com/AU-DIS/LSTM_langid. End-users can download our code as a library from the Python Package Index (PyPI) via https://pypi.org/project/LanguageIdentifier/.

Proceedings of the 16th Conference of the European Chapter of the Associationfor Computational Linguistics: Student Research Workshop, pages 36–42
April 19 - 23, 2021.

2 Related work

The simplest Language Identification methods discriminate using elementary distinguishing traits like unique character combinations, frequent or unique words, diacritics, or common n-grams (Dunning, 1994; Souter et al., 1994; Truică et al., 2015). Increasing model complexity, some Language Identification methods model sequences of words, characters, or bytes. Some methods focus on modeling the frequency of n-grams, e.g., frequency of character n-grams (Ahmed et al., 2004; Souter et al., 1994). Such methods outperform techniques based on unique words. Markov model-based approaches estimate the probability of a string based on n-grams of characters or bytes (Dunning, 1994), as is the case of langid.py (Lui and Baldwin, 2012, 2011). Due to its availability as an open-source library, langid.py is one of the most popular language identifiers.

Recent language identifiers increasingly use word representations. For example, in a blog post, Grave (2017) shows how to identify languages using FastText vectors (Bojanowski et al., 2016; Joulin et al., 2017, 2016), which model character n-grams. Language identification with FastText vectors is as performant as langid.py (Grave, 2017). Similar to langid.py, FastText language identification models are open-source and, therefore, popular.

LanideNN (Kocmi and Bojar, 2017) identifies languages in multilingual documents using a recurrent neural network with a single layer of gated recurrent units (GRU). Unlike Markov-based methods, recurrent neural network architectures do not model character sequences with a fixed window of context. The language identifier that Apple briefly sketched in a blog post (Apple, 2019) uses a recurrent neural network with a two-layer bi-directional LSTM to model character sequences. Apple's method differs from LanideNN in architecture complexity (two layers, LSTM cells instead of the simpler GRU cells) and in its focus. LanideNN works with long multilingual documents, whereas Apple classify extremely short monolingual strings.

In a survey, Jauhiainen et al. (2019) present more than the techniques above, discuss challenges, and identify remaining research questions. Among the remaining research questions are very short texts (the problem motivating Apple) and discrimination of related languages. In this paper, we go beyond reproducing Apple's work by analyzing the effect of related languages.

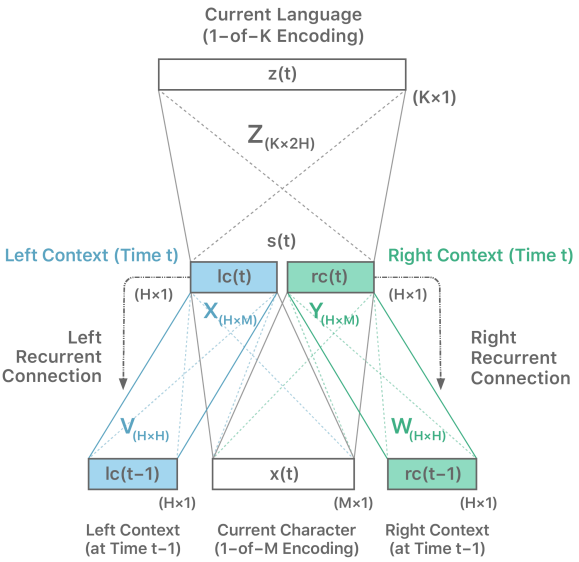

Figure 1: The bi-LSTM architecture. Figure reproduced from Apple (2019).

3 Model architecture

Figure 1 gives an overview of the two-layer bi-directional LSTM architecture powering Apple's products, as briefly sketched in a blog post (Apple, 2019).

The model takes as input strings of characters. In the following, we describe the left-to-right direction of the bi-directional LSTM. The right-to-left direction is identical but mirrored. In the first step, vector embeddings replace all characters in the input string. The network uses a single embedding for all languages since the language is unknown at this point. At each time step, the LSTM ingests a character's embedding and the hidden layer representation from the previous step. The per-character output from the left-to-right LSTM layer is concatenated with that of the right-to-left layer. The concatenated vectors pass to a second LSTM layer that is identical to the first but does not share parameters. After the second layer, the concatenated vectors go through a single linear layer, producing a distribution over all supported languages. The linear layer provides character-level language identification. In other words, for each input character, the network generates a probability distribution over the possible languages.

With the outputs from the linear layer, Apple (2019) state that *A max pooling style majority voting decides the dominant language of the string*. However, max pooling and majority voting are dif-

ferent techniques. A combination of the two is impossible as one cannot perform majority voting over outputs that have been max pooled, and vice versa. Instead, we sum over the linear layer's output values at each time step and *softmax* the summed output to obtain a prediction. We expect this approach to be what the original authors intended. The similarity between our reproduction's performance and what Apple report in the original blog post confirms our approach.

4 Data sets

Apple (2019) only mention the kind of data used in their experiments. Therefore, we use two large and openly available data sets of the same kind as Apple: a subset of OpenSubtitles (Lison and Tiedemann, 2016) to study performance on dialog; and Universal Dependencies (UD, Zeman et al., 2019) for prose. Following Apple, we trim strings to 50 characters per sample, with all samples starting at the beginning of a word, and remove special characters.

Apple test on 20 languages that use the Latin alphabet, but only show results on 9 of the 20 and do not specify the remaining 11 languages. Besides the 9 languages in the original blog post, we select 11 languages, some of which are closely related. Thus, our experimental setup[2] is similar to Apple's. Including closely related languages increases our data sets' difficulty but supports more interesting and more representative experiments. Specifically, it supports performance analysis on related languages, an open research question (Jauhiainen et al., 2019).

5 Experiments and results

We use five-fold cross-validation in all experiments. Following Apple (2019), we evaluate on strings of 10 characters. We test all models on the same strings.

We use the AdamW optimizer with default parameters in PyTorch; we set the character embedding dimension to 150 and the bi-LSTM's hidden dimension to 150; we train for 25 epochs using batches of 64 examples and use weighted cross-entropy for the loss function.

[2]The languages we use are: Catalan (ca), Czech (cs), Danish (da), French (fr), German (de), English (en), Spanish (es), Estonian (et), Finnish (fi), Croatian (hr), Hungarian (hu), Italian (it), Lithuanian (lt), Dutch (nl), Norwegian (no), Portuguese (pt), Polish (pt), Romanian (ro), Swedish (sv), and Turkish (tr).

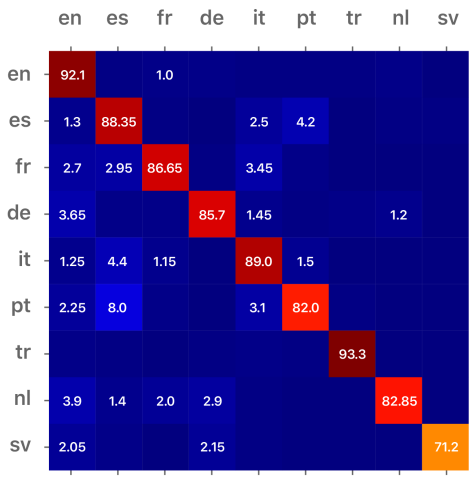

Figure 2: Apple (2019)'s original results.

Out-of-the-box, FastText and langid.py can identify more than our set of 20 languages. For fair evaluation, we limit the set of languages that the models output. For langid.py, we use a built-in method that limits the number of languages under consideration. For FastText, we take the probability distribution over all language predictions, extracting only the relevant 20. We use the large pre-trained FastText model[3]. When re-training FastText, we use 15 epochs, with a minimum n-gram length of one character and a maximum of six characters. We leave all other parameters at their default.

5.1 Comparison with original work

Figure 3 contains the results of our reproduction of the experiment in Figure (b) from Apple (2019), a confusion matrix of the bi-LSTM model trained and evaluated on the UD data set. Since Apple do not include averaged results, we use the confusion matrices for comparison. Figure 2 includes a copy of Figure (b) from Apple (2019) for easier comparison. We find that performance per language is similar between the two implementations. While in one case, accuracy is almost identical (Turkish, tr), for most languages, our implementation is either a few points of accuracy below (e.g., French, fr, -2.85 points, and Italian, it, -2.62) or above the original model (e.g., Dutch, nl, $+1.87$). For some languages, our implementation considerably underperforms the original (e.g., English, en, -7.4 points, and Spanish, es, -16.7). Our implementation considerably outperforms the original on German (de $+6.91$) and Swedish (sv $+7.54$).

[3]Available at https://fasttext.cc/docs/en/language-identification.html

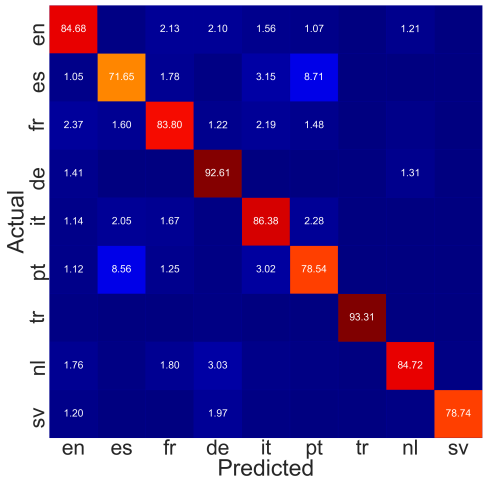

Figure 3: Confusion matrix for bi-LSTM on UD.

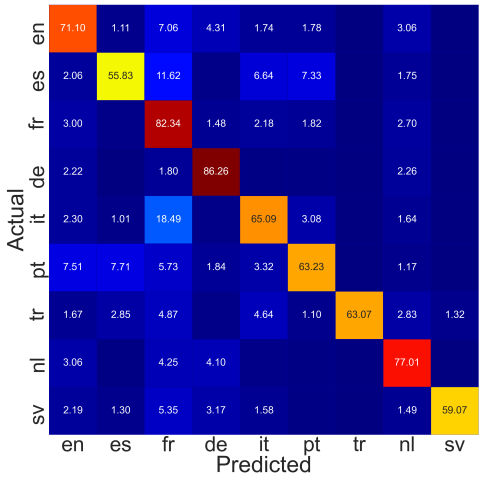

Figure 4: Confusion matrix for re-trained FastText on UD.

We attribute the difference in performance to randomness during training and differences in training data. The original blog post does not state the size nor language composition of the data set.

In Figure 3, we follow Apple and threshold values in the confusion matrix at 1.0. Thus, we can effortlessly compare error patterns. Interestingly, the patterns are almost identical. Both matrices show issues distinguishing between Italian (it) and Portuguese (pt), German (de) and Dutch (nl), French (fr) and English (en), and Italian (it) or Portuguese (pt) vs. Spanish (es) or French (fr). Unsurprisingly, most confusions appear for languages from the same families, Romance (es, fr, it, pt) and Germanic (de, nl).

5.2 Comparative analysis

In Tables 1 and 2, we include the comparative analysis results with the current *de facto* standards for Language Identification: FastText and langid.py. We use two weighing strategies for F1 to provide different insights. Macro-F1 averages the per-language results and considers languages equally important. Weighted-F1 takes into account the popularity of the different languages in the data sets. Weighted-F1 measures the performance on the data set, while macro-F1 illustrates language coverage as it is not affected by label frequency. In multi-class classification, micro-F1 equals accuracy. We, therefore, include only accuracy, denoted *acc@1*.

On both data sets, the bi-LSTM exceeds the weighted- and macro-F1 of langid.py, pre-trained FastText, and re-trained FastText. The performance difference between the bi-LSTM and the next best model (the re-trained FastText) also appears in the confusion matrix. Figure 4 shows that even the re-trained FastText exhibits confusion across all pairs. It also shows a strong bias towards some languages like English (en), French (fr), or Dutch (nl) regardless of the input language. All columns in Figure 4 that correspond to these languages exhibit confusion errors.

The OpenSubtitles data is more challenging than UD for out-of-the-box langid.py and FastText, but easier for bi-LSTM and re-trained FastText. Also, there is a considerable improvement from the pre-trained FastText to the re-trained FastText on both data sets. These observations suggest that (1) domain adaptation has a considerable impact on Fast-

	LSTM	**pFT**	**rFT**	**langid.py**
wF1	**87.41**	72.45	78.67	64.89
maF1	**79.22**	61.20	67.90	51.66
acc @1	**86.93**	70.45	77.92	61.73
acc @3	**96.07**	85.84	90.59	82.83
acc @5	**97.78**	90.92	94.45	88.99

Table 1: Results on UD. pFT = pre-trained FastText; rFT = re-trained FastText

	LSTM	**pFT**	**rFT**	**langid.py**
wF1	**91.38**	67.45	84.14	54.31
maF1	**91.38**	67.45	84.14	54.31
acc @1	**91.37**	67.73	84.13	53.47
acc @3	**98.14**	84.15	95.08	76.30
acc @5	**98.93**	89.31	97.38	84.22

Table 2: Results on OpenSubtitles. pFT = pre-trained FastText; rFT = re-trained FastText

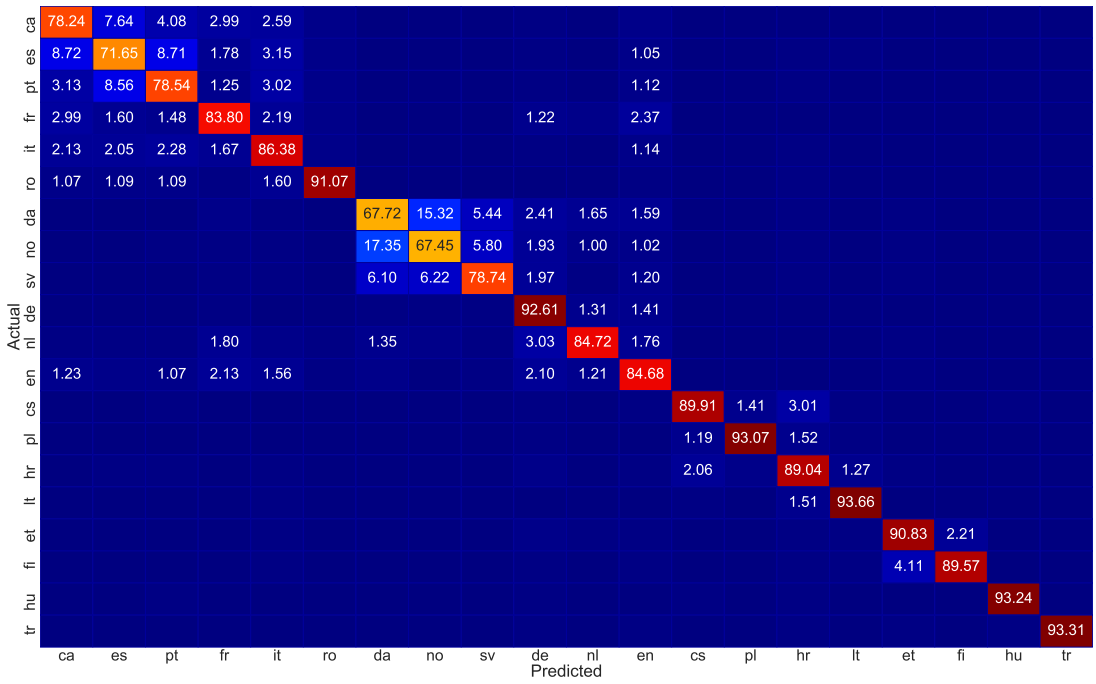

Figure 5: Confusion matrix for bi-LSTM on UD.

Text, and (2) that dialog is more difficult for the out-of-the-box models. OpenSubtitles contains subtitles of movies predominantly produced in English. Consequently, character names are also English-centered, e.g., Jane. Character names can appear in dialog, which might confuse the pre-trained models to assign such dialog lines to English, despite their translation.

5.3 Error analysis

Tables 1 and 2 show a jump from accuracy at the top of the list of prioritized predicted languages (*acc@1*) to accuracy at the top three (*acc@3*). For most models, a smaller jump follows to accuracy at the top five (*acc@5*). The sizeable jump indicates that, even when the models are wrong, the correct answer is usually among the top three. For example, from *acc@1* to *acc@3*, the bi-LSTM jumps 9.14 points on UD and 6.77 on OpenSubtitles, but only 1.71 and 0.79 from *acc@3* to *acc@5*. The gap from *acc@1* to *acc@3* is much larger for langid.py and FastText, illustrating a higher confusion. Recent work in language identification suggests that the accuracy gap might be a symptom of confusion of related languages (Haas and Derczynski, 2020).

To understand the bi-LSTM's jump in accuracy, we turn to the complete confusion matrix. In Figure 5, we show the confusion matrix of the bi-LSTM on all 20 languages in our experiments. There is intense confusion between highly similar languages. We observe three large clusters of confused languages: Romance (ca, es, fr, it, pt, ro), West Germanic (de, en, nl), and languages of Northern Europe (da, no, sv). More closely related languages are more confusing, for example, Catalan (ca) vs. Spanish (es) and Danish vs. Norwegian (no). The clusters of confusion between related languages indicate that, despite the bi-LSTM's improved performance, highly similar languages still pose a challenge.

5.4 Storage requirements

Apple (2019) also consider storage requirements. Our bi-LSTM uses 4 MB of storage, confirming the claims in the original blog post. The re-trained FastText model requires 1.5 GB of storage, but that could reduce to approximately 150 MB, following Joulin et al. (2016). langid.py's model is only 2.5 MB. Given its language identification performance and model size, the bi-LSTM is a great value proposition, especially on storage-constrained mobile devices, confirming Apple's use case scenario.

6 Conclusions

We have reproduced the bi-LSTM language identification architecture described in a blog post by Apple (2019). Our reproduction experiments confirm the performance claims in the original blog post. We evaluated the bi-LSTM against the *de*

facto open-source language identifiers in experiments on two openly available data sets. Our evaluation considered dialog and prose, and targeted twenty languages, including some highly similar languages such as Danish (da) and Norwegian (no) or Catalan (ca) and Spanish (es). Our experiments illustrate the difficulty of identifying the language in very short strings. The reproduced bi-LSTM outperformed FastText and langid.py on all measures, even when training FastText on the same data. However, we went beyond a straightforward reproduction and considered related languages. Our analysis shows that the bi-LSTM can easily confuse languages from the same family (e.g., Romance, West Germanic, or Scandinavian) and highly similar languages such as Catalan (ca) and Spanish (es). We publish our implementation's source code and make a trained model available as a library. In the future, we would like to consider avenues for improving the bi-LSTM architecture. For example, we would like to replace the majority voting mechanism in the bi-LSTM with a more robust alternative.

References

Bashir Elhaj Ahmed, Sung-Hyuk Cha, and Charles C. Tappert. 2004. Language identification from text using n-gram based cumulative frequency addition. In *Proceedings of Student/Faculty Research Day, CSIS, Pace University*.

Apple. 2019. Language identification from very short strings. Online: https://machinelearning.apple.com/research/language-identification-from-very-short-strings. Accessed: 2021-02-10.

Valerio Basile, Cristina Bosco, Elisabetta Fersini, Debora Nozza, Viviana Patti, Francisco Manuel Rangel Pardo, Paolo Rosso, and Manuela Sanguinetti. 2019. SemEval-2019 task 5: Multilingual detection of hate speech against immigrants and women in Twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 54–63, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*.

Ted Dunning. 1994. *Statistical identification of language*. Computing Research Laboratory, New Mexico State University Las Cruces, NM, USA.

Edouard Grave. 2017. Language identification. Online: https://fasttext.cc/blog/2017/10/02/blog-post.html. Accessed: 2020-09-24.

René Haas and Leon Derczynski. 2020. Discriminating Between Similar Nordic Languages. *arXiv preprint arXiv:2012.06431*.

Tommi Jauhiainen, Marco Lui, Marcos Zampieri, Timothy Baldwin, and Krister Lindén. 2019. Automatic language identification in texts: A survey. *Journal of Artificial Intelligence Research*, 65:675–782.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Tom Kocmi and Ondřej Bojar. 2017. LanideNN: Multilingual language identification on character window. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 927–936, Valencia, Spain. Association for Computational Linguistics.

Wang Ling, Guang Xiang, Chris Dyer, Alan Black, and Isabel Trancoso. 2013. Microblogs as parallel corpora. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 176–186, Sofia, Bulgaria. Association for Computational Linguistics.

Pierre Lison and Jörg Tiedemann. 2016. OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 923–929, Portorož, Slovenia. European Language Resources Association (ELRA).

Marco Lui and Timothy Baldwin. 2011. Cross-domain Feature Selection for Language Identification. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 553–561, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Marco Lui and Timothy Baldwin. 2012. langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 System Demonstrations*, pages 25–30, Jeju Island, Korea. Association for Computational Linguistics.

Judith J Prochaska, Cornelia Pechmann, Romina Kim, and James M Leonhardt. 2012. Twitter=quitter? an analysis of twitter quit smoking social networks. *Tobacco Control*, 21(4):447–449.

Clive Souter, Gavin Churcher, Judith Hayes, John Hughes, and Stephen Johnson. 1994. Natural language identification using corpus-based models. *HERMES-Journal of Language and Communication in Business*, 13:183–203.

Ciprian-Octavian Truică, Julien Velcin, and Alexandru Boicea. 2015. Automatic Language Identification for Romance Languages using Stop Words and Diacritics. In *17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 243–246.

Daniel Zeman, Joakim Nivre, Mitchell Abrams, and et al. 2019. Universal dependencies 2.5. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.