

Seeking robustness in a multilingual world: from pipelines to embeddings

Yerai Doval

Doctoral Thesis



UNIVERSIDADE DA CORUÑA

2019

Seeking robustness in a multilingual world: from pipelines to embeddings

Yerai Doval

Doctoral Thesis / 2019

Advisors: Manuel Vilares & Jesús Vilares

Ph.D. degree in Computer Science



This work is licensed under a Creative Commons Attribution 3.0 License.

Jesús Vilares Ferro, Associate Professor of the Department of Computer Science and Information Technologies of the University of A Coruña,

Jesús Vilares Ferro, Profesor Titular de Universidad del Departamento de Ciencias de la Computación y Tecnologías de la Información de la Universidade da Coruña,


Manuel Vilares Ferro, Full Professor of the Department of Computer Science of the University of Vigo,

Manuel Vilares Ferro, Catedrático de Universidad del Departamento de Informática de la Universidade de Vigo,

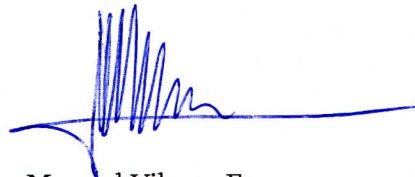
**hereby certify
certifican**

that the dissertation entitled *Seeking robustness in a multilingual world: from pipelines to embeddings*, submitted to Universidade da Coruña by Yeraí Doval Mosquera, has been carried out under our supervision and meets all the requirements for obtaining the *Ph. D. degree with International Mention*.

que el trabajo de tesis titulado Buscando robustez en un mundo multilingüe: de pipelines a embeddings, enviado a la Universidade da Coruña por Yeraí Doval Mosquera, ha sido llevado a cabo bajo nuestra supervisión y cumple con todos los requisitos para la obtención del título de Doctor en Computación con Mención Internacional.



Jesús Vilares Ferro



Manuel Vilares Ferro

A mi hermano. Lo hemos conseguido.

Agradecimientos

Quisiera agradecer de todo corazón a mis directores y tutor, Jesús, Manuel y Carlos, por haber depositado semejante confianza en mí. Su inestimable apoyo tanto a nivel personal como académico me ha permitido llegar al punto de estar escribiendo estas líneas.

Gracias a todos mis compañeros de laboratorio, y sobre todo a David, por ser además un modelo de investigador a seguir. También al grupo del café, por amenizar las mañanas con risas y conversaciones tan variadas.

Gracias a mi director de estancia, Steven, por haberme aceptado en la prestigiosa comunidad de la Universidad de Cardiff, donde he podido conocer a tan buenos compañeros como Luis y José. Ha sido gracias a vosotros que mi carrera como investigador se impulsó con nuevas ideas y ganas de hacer cosas. Cardiff tendrá siempre un lugar privilegiado en mi memoria.

Gracias a mis amigos de siempre por ser mi familia fuera de casa y obligarme a desconectar de vez en cuando (con diferentes grados de éxito). Mención también a todos aquellos que, por un motivo u otro, vinieron y se fueron en los últimos años dejando una huella imborrable en mí.

Pero lo más importante de todo, a mi familia. Gracias a mis padres por tantas cosas, pero sobre todo por su amor incondicional. Gracias a mi hermano pequeño por ser en muchos aspectos mi hermano mayor. Gracias a mis padrinos y mi abuela por haber hecho mi vida en casa lo maravillosa que es. A mis tíos, primos y demás familia cercana por haberme recibido siempre con los brazos abiertos.

Gracias

Abstract

In this dissertation, we study two approaches to overcome the challenges posed by processing user-generated non-standard multilingual text content as it is found on the Web nowadays.

Firstly, we present a traditional discrete pipeline approach where we preprocess the input text so that it can be more easily handled later by other systems. This implies dealing first with the multilinguality concern by identifying the language of the input and, next, managing the language-specific non-standard writing phenomena involved by means of text normalization and word (re-)segmentation techniques.

Secondly, we analyze the inherent limitations of this type of discrete models, taking us to an approach centred on the use of continuous word embedding models. In this case, the explicit preprocessing of the input is replaced by the encoding of the linguistic characteristics and other nuances of non-standard texts in the embedding space. We aim to obtain continuous models that not only overcome the limitations of discrete models but also align with the current state of the art in Natural Language Processing (NLP), dominated by systems based on neural networks.

The results obtained after extensive experimentation showcase the capabilities of word embeddings to effectively support the multilingual and non-standard phenomena of user-generated texts. Furthermore, all this is accomplished within a conceptually simple and modular framework which does not sacrifice system integration. Such embedding models can be readily used as a fundamental building block for state-of-the-art neural networks which are, in turn, used in virtually any NLP task.

Resumen

En esta tesis estudiamos dos enfoques para abordar los desafíos planteados de cara al procesamiento de contenidos textuales no estándar y multilingües generados por los usuarios del tipo que se pueden encontrar en la Web a día de hoy.

En primer lugar, presentamos un enfoque tradicional basado en *pipelines* discretos en el que el texto de entrada es preprocesado para facilitar su ulterior tratamiento por otros sistemas. Esto implica abordar el problema del multilingüismo, primero, identificando el idioma de la entrada para, seguidamente, tratar los fenómenos de escritura no estándar específicos de dicho idioma presentes en la entrada. Para ello se aplicarán técnicas de normalización del texto y (re-)segmentación de palabras.

En segundo lugar, analizamos las limitaciones inherentes a este tipo de modelos discretos, lo cual nos conduce a un enfoque centrado en el empleo de modelos continuos basados en *word embeddings* (i.e., representaciones vectoriales). En este caso, el preprocesamiento explícito de la entrada es sustituido por la codificación de las características lingüísticas y demás matices propios de los textos no estándar en el propio espacio de *embedding* (un espacio vectorial). Nuestro objetivo es obtener modelos continuos que no sólo superen las limitaciones de los modelos discretos, sino que también se alineen con el estado del arte actual del Procesamiento de Lenguaje Natural (PLN), dominado por sistemas basados en redes neuronales.

Los resultados obtenidos después de una extensa experimentación muestran la capacidad de las *word embeddings* para dar un soporte efectivo por sí mismas a los fenómenos multilingües y no estándar propios de los textos generados por usuarios. Además, todo esto se logra dentro de un marco conceptual simple y modular que no necesita sacrificar la integración de sistemas. Dichos modelos de *word embeddings* pueden emplearse fácilmente como un elemento fundamental en redes neuronales de última generación que, a su vez, son utilizadas en prácticamente cualquier tarea de PLN.

Resumo

Nesta tese estudamos dous enfoques para abordar os desafíos que presenta o procesamento de contidos textuais non estándar e multilingües xerado polos usuarios do tipo que se atopar na Web a día de hoxe.

En primeiro lugar, presentamos un enfoque tradicional baseado en *pipelines* discretos nos que preprocesamos o texto de entrada para facilitar a seu posterior tratamento por outros sistemas. Isto implica abordar o problema do multilingüismo, primeiro, identificando a lingua de entrada para, seguidamente, tratar o resto dos fenómenos de escritura non estándar específicos da lingua involucrados mediante técnicas de normalización do texto e (re-)segmentación de palabras.

En segundo lugar, analizamos as limitacións inherentes a este tipo de modelos discretos, o cal nos leva a un enfoque centrado no emprego de modelos continuos baseados en *word embeddings* (i.e., representacións vectoriais). Neste caso, o preprocesamento explícito da entrada substitúese pola codificación das características lingüísticas e demais matices propios dos textos non estándar no espazo de embedding mesmo (un espazo vectorial). O noso obxectivo é obter modelos continuos que non so superen as limitacións dos modelos discretos, senón que tamén se aliñen co estado da arte actual do Procesamento da Linguaxe Natural (PLN), dominado por sistemas baseados en redes neurais.

Os resultados obtidos tras unha ampla experimentación amosan a capacidade das *word embeddings* para dar un soporte efectivo por si mesmas aos fenómenos multilingües e non estándar propios de textos xerados por usuarios. Ademais, todo isto acádase dentro dun marco conceptual simple e modular que non precisa sacrificar a integración de sistemas. Estes modelos de *word embeddings* poden empregarse facilmente como un elemento fundamental en redes neurais de última xeración que, á súa vez, utilízanse en practicamente calquera tarefa de PLN.

Contents

I Introduction	1
1 Introduction	3
1.1 Motivation	3
1.1.1 User-generated texts and texting	4
1.1.2 Long-term implications	6
1.2 NLP for user-generated texts: domain adaptation	7
1.3 From discrete to continuous systems in NLP	8
1.3.1 A discrete preprocessing approach	9
1.3.2 A continuous word embeddings approach	10
1.4 Outline and contributions	11
2 Preliminary definitions	15
2.1 Terminology	15
2.2 Language model	16
2.3 Word embeddings	17
II Discrete pipeline approach	21
3 LID in Twitter	23
3.1 Domain particularities	23
3.2 LID resources	25
3.3 Evaluation	26
3.3.1 Language-level analysis	28
3.4 Towards multilingual integration	28
3.5 Related work	30
3.6 Conclusions	31

4	Microtext normalization	33
4.1	Architecture	34
4.2	Evaluation	36
4.3	Implementation	37
4.3.1	Configuration before W-NUT 2015	39
4.3.2	Adaptation for W-NUT 2015	40
4.4	Texting encoding	40
4.5	Related work	43
4.6	Conclusions	44
5	Word segmentation	47
5.1	Problem domain	48
5.2	System description	49
5.2.1	Language model	49
5.2.2	Beam search algorithm	52
5.3	Experiments	53
5.3.1	System implementation	54
5.3.2	Corpora	54
5.3.3	Results	55
5.4	Word embeddings	61
5.5	Related work	61
5.6	Conclusions	64
III	Interlude	67
6	Towards continuous models	69
6.1	Limitations	70
6.2	Alternatives	72
6.2.1	K -best pipelines and Bayesian networks	72
6.2.2	Non-linear pipelines	74
6.2.3	Graph-based solutions	75
6.3	Word embeddings	76
6.3.1	Embeddings as modular <i>and</i> integrated encoded knowledge	77
6.3.2	Tackling our current challenges	78
6.3.3	The downside of this approach	78

IV	Replacing preprocessing with embeddings	81
7	Cross-lingual word embeddings	83
7.1	Meeting in the middle	84
7.1.1	Bilingual models	86
7.1.2	Multilingual models	87
7.2	Experimental setting	88
7.2.1	Corpora and monolingual embeddings	88
7.2.2	Training dictionaries	89
7.2.3	Compared systems	89
7.3	Intrinsic evaluation	90
7.3.1	Cross-lingual performance	90
7.3.2	Monolingual performance	92
7.4	Extrinsic evaluation	93
7.4.1	Cross-lingual hypernym discovery	94
7.4.2	Cross-lingual natural language inference	95
7.5	Analysis	97
7.5.1	Studying word translations	97
7.5.2	Multilingual performance	98
7.6	Embedding models to replace language identification	100
7.7	Related work	102
7.8	Conclusions	104
8	Embeddings for noisy text	107
8.1	Noise-resistant embeddings	108
8.1.1	Modified skipgram model	110
8.2	Evaluation	112
8.2.1	Word embedding training	112
8.2.2	Intrinsic tasks: word similarity and outlier detection	112
8.2.3	Extrinsic tasks: the SentEval benchmark and Twitter SA	113
8.2.4	Dataset de-normalization	114
8.2.5	Results	115
8.3	Word segmentation	118
8.4	Related work	120
8.5	Conclusions	121

V Conclusion	123
9 Conclusions and future work	125
9.1 Future work	128
A Phonetic algorithms	159
A.1 Phonetic algorithms	160
A.1.1 Soundex	161
A.1.2 IBM Alpha Search Inquiring System	161
A.1.3 New York State Identification and Intelligence System	164
A.1.4 Match Rating Approach	164
A.1.5 Metaphone	164
A.1.6 Double Metaphone	165
A.1.7 Daitch-Mokotoff Soundex	165
A.1.8 Caverphone	165
A.1.9 Beider-Morse	165
A.1.10 Fuzzy Soundex	166
A.1.11 Lein	166
A.1.12 Onca	166
A.1.13 Phonex	166
A.1.14 Phonix	166
A.1.15 Roger Root	167
A.1.16 Census Modified Statistics Canada	167
A.1.17 Eudex	167
A.2 Implementation	168
A.3 Evaluation	168
A.3.1 Evaluation corpora	168
A.3.2 Experimental methodology	169
A.3.3 Results and discussion	171
A.4 Related work	176
A.5 Conclusions	177
B Cross-lingual analysis	179
B.1 Variables	180
B.1.1 Monolingual corpora	180
B.1.2 Bilingual supervision	180
B.1.3 Languages	181
B.1.4 Other variables	182

B.2	Evaluation	182
B.2.1	Bilingual dictionary induction	184
B.2.2	Cross-lingual semantic word similarity	184
B.2.3	Cross-lingual natural language inference	184
B.3	Analysis	185
B.4	Related work	190
B.5	Conclusions	191
B.6	Supplementary material: detailed results	191
C	Resumen largo en español	195
C.1	Motivación	196
C.2	PLN para textos generados por el usuario: adaptación al dominio	196
C.3	Enfoque discreto	197
C.3.1	Identificación del idioma	198
C.3.2	Normalización de microtexto	198
C.3.3	Segmentación de palabras	199
C.4	Limitaciones y transición	200
C.4.1	Propagación de errores	200
C.4.2	Fragmentación del contexto	202
C.4.3	Hacia una aproximación continua	203
C.5	<i>Embeddings</i> multilingüe	205
C.6	<i>Embeddings</i> robustas para microtextos	206
C.7	Conclusiones y trabajo futuro	206

List of Figures

- 1.1 Preprocessing pipeline for user-generated text. 9
- 2.1 Skipgram model (Mikolov et al., 2013). 18
- 2.2 Simplified example of a two-dimensional embedding space. 19
- 2.3 Simplified example of a two-dimensional bilingual embedding space. . . 20
- 4.1 Original pipeline and pipeline adapted for W-NUT 2015 integrated into the architecture of the system. 41
- 5.1 Simplified illustration of the algorithm execution, with $n = 2$ and $m = 1$. 50
- 5.2 Illustration of the architecture of our neural networks for word segmentation. 51
- 5.3 Validation error curves for some neural models on the English and Twitter training corpus. 57
- 5.4 Validation error curves for models which are *too* wide. 58
- 6.1 Sequential execution of a 3-best pipeline, where we use a beam search algorithm with $n = 2$ 73
- 6.2 The three possible non-linear pipelines formed by three steps and at least one feedback loop. 74
- 6.3 An example graph formed by four nodes (tasks) where we can prune one of its edges (dashed line). 75
- 7.1 Step by step integration of two monolingual embedding spaces. 86
- 8.1 Visualization of the adapted skipgram model where bridge-words have a lower impact than the original word in the context. 110
- 8.2 Performance of each considered model when going from standard texts to noisier ones on the extrinsic tasks. 117

B.1	<i>P</i> @1 performance of the unsupervised version of VecMap on dictionary induction across corpus types and language pairs.	187
B.2	Comparison between the dictionary induction performance (<i>P</i> @1) of VecMap and MUSE.	188

List of Tables

- 3.1 System tuning results. 26
- 3.2 Official results and positions in the rankings (pos). 27
- 3.3 Language-level results. 29

- 4.1 Results on the training and test datasets. 36

- 5.1 Example instances of the problem we are trying to solve as input/output pairs. 49
- 5.2 Precision results on the English (EN) and Twitter (Tw) development datasets by neural model architecture. 56
- 5.3 Precision results on development datasets by language and n-gram model order. 59
- 5.4 Average counts of words, characters, and bytes per instance in the development datasets. 59
- 5.5 Precision results on the test datasets by language and approach. 60

- 7.1 $P@k$ performance of different cross-lingual embedding models in the bilingual dictionary induction task. 90
- 7.2 Cross-lingual word similarity results in terms of Pearson (r) and Spearman (ρ) correlation. 92
- 7.3 Monolingual word similarity results in terms of Pearson (r) and Spearman (ρ) correlation. 93
- 7.4 Cross-lingual hypernym discovery results. 95
- 7.5 Accuracy on the XNLI task using different cross-lingual embeddings as features. 96
- 7.6 Word translation examples from English and Spanish, comparing VecMap with the bilingual and multilingual variants of Meemi. 97

7.7	Dictionary induction results obtained with multilingual Meemi over (Vec-Map _{ortho}).	99
7.8	Results obtained by the Corpus Concatenation (CC) and Artificial Code-Switching (ACS) models in the usual test corpora for Dictionary Induction (DI), Word Similarity (WS), and Cross-lingual Natural Language Inference (XNLI).	101
8.1	Spearman correlation results of word similarity on SCWS, wordsim353 (WS353), SimLex999 (SL999), and SemEval17 (Sem17) datasets.	115
8.2	Accuracy results of outlier detection on 8-8-8 and wiki-sem-500 (wiki) datasets.	116
8.3	Results of the extrinsic evaluation on the SentEval benchmark. The noise levels are <i>low</i> ($p_d = 0.3$), <i>mid</i> ($p_d = 0.6$), and <i>high</i> ($p_d = 1$).	118
8.4	Accuracy results of the extrinsic evaluation on SemEval (SE) Twitter SA datasets.	119
8.5	Spearman correlation averages on the new de-normalized STS* datasets, with $p_j = 0.5$ and $p_s = 0.1$	120
A.1	Example encodings for each of the phonetic algorithms analyzed (1): “nuff”-“enough” and “cntrtkxn”-“contradiction”	162
A.2	Example encodings for each of the phonetic algorithms analyzed (2): “da”-“the” and “onez”-“ones”.	163
A.3	Evaluation corpora statistics.	169
A.4	Results for the utdallas dictionary, ranked by F1.	171
A.5	Results for the unimelb dictionary, ranked by F1.	172
A.6	Results for the utdallas dictionary, this time ranked by recall.	173
A.7	Results for the unimelb dictionary, this time ranked by recall.	174
B.1	Statistics of the corpora used to train monolingual word embeddings: size (total number of tokens) and words (number of unique tokens).	181
B.2	Bilingual dictionary induction results using English as source language. Performance measured by $P@k$	183
B.3	Spearman correlation performance of various cross-lingual word embedding models in the cross-lingual word similarity task.	185
B.4	Accuracy in the cross-lingual natural language inference task (XNLI) using different cross-lingual word embedding models.	186

B.5	Absolute improvement (in percentage points) by applying the postprocessing (Meemi) over the two base models VecMap and MUSE on the cross-lingual word similarity task using Web corpora.	189
B.6	Bilingual dictionary induction results in the test sets of Conneau et al. (2018a).	192
B.7	Cross-lingual word similarity results in the SemEval-17 dataset (Camacho Collados et al., 2017).	193

Acronyms

NLP	Natural Language Processing
PoS	Part of Speech
ML	Machine Learning
IV	In-Vocabulary
OOV	Out-Of-Vocabulary
TBPTT	Truncated Back-Propagation Through Time
LID	Language IDentification
P	Precision
R	Recall
P@k	Precision at k
MRR	Mean Reciprocal Rank
MAP	Mean Average Precision
CCA	Canonical Correlation Analysis
Meemi	Meeting in the middle
NLI	Natural Language Inference
XNLI	Cross-lingual Natural Language Inference
SA	Sentiment Analysis
IPA	International Phonetic Alphabet

Part I

Introduction

Chapter 1

Introduction

In the present dissertation, we study modular, flexible, and integrated approaches for handling user-generated non-standard multilingual text as it is found nowadays on the Web in general and social media platforms in particular. We start with a traditional discrete pipeline approach, where we implement an explicit preprocessing of the input text by means of language identification, word segmentation,¹ and text normalization. Then, we analyze the inherent limitations of this and other similar discrete models to finally arrive at an approach based on continuous word representations, commonly referred to as word embeddings. In this case, the explicit preprocessing of the input is replaced by the encoding of the linguistic characteristics and other nuances of user-generated texts. The resulting continuous models not only overcome the limitations of discrete models, but also integrate to a greater extent in state-of-the-art Natural Language Processing (NLP) systems based on neural networks.

In this introductory chapter, we first establish the motivation behind our work, which revolves around the challenges posed by user-generated text on the Web. Then, we make a brief introduction of the three preprocessing tasks and corresponding approaches used to tackle them. Finally, we outline the storyline for the rest of this dissertation and highlight the main contributions.

1.1 Motivation

Since the last decade, social media users have produced the large amounts of text and other types of content which have led us to the Big Data era (Boyd and Crawford, 2012).

¹Although we will use the term “segmentation” throughout this work, in reality we perform a *re*-segmentation of the already (possibly badly) segmented input.

In reality, the Internet has been a user-centered platform from its beginnings, with applications such as E-mail in the 70s, BBS in the 80s, the Web and IRC chat in the 90s, social media networks in the 00s, and smartphone messaging apps in the 10s (e.g., WhatsApp, Telegram, etc.). What these technologies have in common is that they made it easier for more and more users across the globe to communicate and share information. These platforms continue to our day, either in their original or evolved forms; e.g., BBS gave way to forums and Facebook groups; and IRC is still in use but lost most of its users to MSN, WhatsApp, and many other chat applications. But most importantly, their use is showing no signs of stagnation as users share increasing amounts of pictures, videos, audio, and many forms of written text. In this work, we focus on the latter and, more specifically, on those texts coming from the Web and social media platforms, which can be exploited for many different purposes: e.g., using sentiment analysis to automatically process the opinions of users (Vilares et al., 2017b), social media marketing (Tuten and Solomon, 2017), or even predicting population health indicators (Signorini et al., 2011). Hereinafter, we refer to this kind of texts as **user-generated texts**.

Nowadays, Internet users produce and share all types of written content in a variety of services and platforms: Web articles, e-mails, chat messages, social media posts, etc. Particularly in the latter cases, these communications have in common two specific traits that differentiate them from most handwritten text and bring them closer to spoken language: **spontaneity** and **informality**. This results in a writing style heavily influenced by speaking habits such as, for example, elongated sounds to provide emphasis (e.g., “nooo”), frequent use of tag questions (e.g., ending phrases with “right?”), or the general use of sloppy or inappropriate language (e.g., “u was there” for “you were there”); i.e., Internet users tend to *write as they speak*.

Moreover, even when English is the predominant language of the Internet, it demonstrates a clear and increasing **multilinguality** by accommodating content in virtually any human language. To illustrate this, we should mention that the most popular social media platforms such as Facebook or Twitter are used by a large number of users across the globe who publish text content in their particular language, or languages, of choice.

1.1.1 User-generated texts and texting

As mentioned previously, the written language employed by Web and social media users is marked by three general traits shared with spoken language: spontaneity, informality, and multilingualism. This can be observed in chat applications (e.g., WhatsApp, Telegram), forums (e.g., Reddit, 4chan), and social networks (e.g., Facebook, Twitter), where this electronic text messaging is usually referred to as **texting**. In this section, we talk about

the static time-invariant traits of texting language which cause it to differ notably from standard written language (Baldwin et al., 2013) and derive in the so-called **texting phenomena** (Thurlow and Brown, 2003):

Spontaneity. The conversations in which users engage tend to occur without much previous preparation, in the same spontaneous way as spoken conversations occur when people casually meet in the street. In particular, the communication process usually happens in a restrained time span with each speaker/writer making short interventions, sometimes even interrupting each other. This promotes the use of *shorthands* and other custom abbreviations for long or common words and expressions whenever possible, in order to align the spatial characteristics of the text with the corresponding temporal constraints. In the same line, services like the SMS and some social media platforms, such as Twitter, further encourage brevity by imposing a limit to the number of characters in each message or post. As a result, it is often preferred to use “OTOH” instead of “on the other hand”, “fav” in place of “favorite”, or “sum1” for “someone”. Also, given the immediacy of this type of conversations, users tend to neglect the correct spelling of words, which is probably the reason why automatic spell checkers are ubiquitous in smartphones nowadays. Having said that, this form of communications may still have a formal or informal appearance, although the latter is more frequent. The resulting short-length texts are usually called **microtexts**.

Informality. Using a colloquial register in a spontaneous communication has significant implications in the writing style of Web and social media users; primarily, ignoring or deviating from the standard writing rules (official or de-facto), and acquiring more traits of informal spoken language. For example, the misuse of homophones (“your” and “you’re”), character repetition (“yeeeeeah”, “noooo”), or the use of onomatopoeia (“boom”, “quack”, “hahaha”, “argh”) and emoji (😄) are frequent in this type of writing. But these phenomena are not limited to the *write as you talk* principle, and users also play with the different linguistic dimensions of words to come up with genuinely creative writing styles. For instance, it is common to generate new spellings for existing words which still retain their approximate original pronunciations; e.g., “dat” instead of “that” or “gawd” in place of “god”. On the other hand, similarly-shaped characters may be used interchangeably with the original ones, such as in *leet speak*, where most of the original characters are replaced with numbers and other symbols; e.g., “n00b” instead of “noob” or “1337” in place of “leet” (“elite”). In any case, these phenomena produce new words which are similar to the originals in some sense, be it in their pronunciation or the shape of its constituent characters, also known as **lexical variants**.

Multilinguality. Since its very inception, the *lingua franca* of the Web has always been English, which is also the native language of hundreds of millions of people. However, the combined number of native speakers for the rest of the languages greatly exceeds that figure,² implying that most Web users must be, at least, bilingual. Not only that: these users may also have multiple native languages, as is the case for Galicians, Basques, and Catalans in Spain speaking their regional languages together with Spanish. In any case, this proximity of multiple languages in the Web, national, or personal scopes not only implies the ability or necessity to communicate in multiple languages but, most of the times, that languages interact with each other at different levels. Most frequently, this manifests in users intentionally or accidentally switching between languages, and for several reasons: covering for the lack of a native alternative to a foreign term or expression (e.g., there is no Spanish term for “selfie”), using the language in which the speaker feels more comfortable in a specific context (e.g., due to historical reasons, older Galicians use Spanish rather than Galician when they want to sound formal), reinforcing the cultural identity of the speaker (e.g., when Mexican-Americans use “loco” instead of “crazy”), or to compensate for the imperfect knowledge of a particular language by non-native speakers (e.g., “let’s go to my *casa*”, where the speaker does not know the English word “house”). With the exception of the former case, which corresponds to *word loan*, this mechanism of language swapping is often called **code-switching** (Yu et al., 2013).

1.1.2 Long-term implications

Beyond the stationary view of languages adopted until now, it is also interesting to consider the temporal dimension where they evolve to fit the changing communication needs and habits of their speakers. In the previous section, we have already described several mechanisms that modify and augment the non-standard lexicon of a language through alternative spellings, shorthands, and borrowed terms from other languages. This is, in fact, a continuous process which highlights the remarkable dynamism of texting language (Eisenstein, 2013), where social, cultural, and technological changes and advances are immediately reflected.

Interestingly, it can be argued that if the corresponding new words and expressions reached a long-term widespread use, they would become part of the language standard, fulfilling the requirement that languages are, first and foremost, a communication tool. Because of this, we can consider texting as the experimental ground supporting language evolution. As an example of this, consider the word “text”. Initially employed as a noun, it recently became widely used as a verb as a shorthand for “text messaging”; e.g., in “I will

²internetworldstats.com/stats7.htm

text you later”. Consequently, prestigious dictionaries such as Merriam-Webster or Oxford now include this new meaning of the word. As an example relating to the multilinguality of the Web, many technical terms from computer science such as “driver” are generally accepted in Spanish where, nonetheless, there exist proper native translations for them (“controlador”, in this case).

1.2 NLP for user-generated texts: domain adaptation

After discussing the linguistic aspects of our problem domain, which constitute the base motivation behind our work, we turn to look into their practical implications on current NLP systems, and how we can better take them into account when simultaneously tackling other common tasks such as Part-of-Speech (PoS) tagging, dependency parsing, or sentiment analysis. Furthermore, we pay special attention to obtaining solutions that are modular, flexible, and integrated.

One of the main concerns of the present work lies in the fact that NLP systems are usually tailored to standard texts: PoS taggers, dependency parsers, or text classification systems, for example, are frequently trained and evaluated on well-written texts, hence they do not account for the linguistic characteristics of Web and social media texts that we described previously. This leads to a performance penalty when dealing with this kind of texts. At this point, there are two general approaches to tackle this issue (Eisenstein, 2013):

System adaptation. Re-implement or re-design a particular model so that it is able to support social media texts. The strength of this approach is its high integration, avoiding the need for any form of normalization or standardization of the input, and therefore possible sources of errors that would propagate across our system. The disadvantage is that it would be necessary to adapt all systems to social media texts, which may not be trivial, as we are adding the new concern of supporting a wider range of inputs to an already complex system. Examples of this type of system are the PoS tagger (Owoputi et al., 2013) and dependency parser (Kong et al., 2014) from the TweetNLP framework.³

Input adaptation. Maintain the existing implementation of a model that only accepts standard texts as input and include a preprocessing step in which we adapt the input to conform to the standards. The advantage in this case is having separate models for separate concerns, lowering the complexity of each one of them and increasing the modularity of our solution. This form of sequential arrangement of steps is usually called a **pipeline** and

³<http://www.cs.cmu.edu/~ark/TweetNLP/>

is very common in NLP. On the other hand, its disadvantage is a low degree of integration, which causes error propagation from one step to the following, which may in turn arrive again at erroneous answers in a cumulative effect. Granted, there are methods to alleviate this problem which use non-linear arrangements of the steps in the pipeline or graphs (see Section 6.2 for an in-depth discussion).

In this work, we have transitioned from a strict take on the second approach to an intermediate solution between the two of them: instead of adapting the input, we adapt the internal representation in the system models. The goal is maintaining the modularity from the second while obtaining the integration benefits of the first one, namely lowering the chance of error propagation and improving the integration.

1.3 From discrete to continuous systems in NLP

During the course of this dissertation, our own solutions for handling user-generated text on the Web have evolved from traditional NLP approaches based on preprocessing (i.e., input adaptation), to those more in line with state-of-the-art advancements which solve the limitations presented by the former and eliminate the need for said preprocessing. Notably, this has been accomplished by abandoning mostly symbolic approaches, where the focus is on the surface forms of discrete linguistic elements such as words, in favor of fully continuous models that embed these elements in a high dimensional vector space and perform all necessary data transformations in this domain. More specifically, this implies going from a traditional pipeline approach, where the input is modified by a series of sequential steps, obtaining a partial solution after each one of them; to relying on adequately trained vector representations of words (word embeddings), which encode useful features about those words for our particular context; i.e., information about spelling variants, abbreviations, or multilinguality.

Furthermore, we have strived to obtain solutions which are both modular, flexible, and integrated. On one hand, the modularity enables the separation of concerns, making a complex task split into smaller subtasks easier to approach, while at the same time benefiting the reusability of our solutions. As already mentioned in the previous section, tackling the specific challenges of user-generated text independently from other NLP tasks, e.g., in a separated preprocessing stage, increases modularity at the cost of system integration for the simplest approaches. On the other hand, the flexibility of our models would be desirable to support the dynamic nature of natural languages; hence, their adaptability to changes. In this case, we resort to Machine Learning (ML) techniques that require the least amount possible of human intervention or supervision, which corresponds to the most expensive

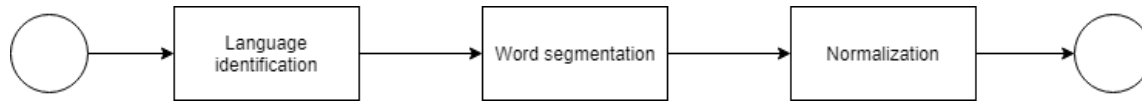


Figure 1.1: Preprocessing pipeline for user-generated text.

part of any development process. Consequently, we look for highly automated systems that are able to obtain performance improvements just by adding more training data in an unsupervised or semi-supervised fashion.

1.3.1 A discrete preprocessing approach

Firstly, we divided the writing phenomena described in Section 1.1.1 into two groups: monolingual and multilingual. Then, we considered the following preprocessing tasks to handle separately each of those groups (also depicted in Figure 1.1):

Language identification. This task handles the multilingual phenomena. At its basis, it consists in determining the language a text is written in. Although it is usually assumed that the whole text is written in one language, in our work we go from this monolingual case to finally supporting a cross-lingual⁴ scenario where words from multiple different languages can be used interchangeably. On the other hand, we will also consider shorter-than-usual sentences, as our focus will be on social media texts from platforms that impose a character length limit, such as Twitter. The language information obtained in this step would allow us to choose the correct monolingual modules afterwards, in an attempt to obtain the required support for multilingual user-generated text.

Microtext normalization. This is the central step that takes care of the monolingual phenomena in our preprocessing pipeline.⁵ Ideally, we would have as many normalization modules as languages considered, and choose between them according to the output of the language identification step. The goal of this task is obtaining a more standard version of the input text by normalizing each of its words. This would be achieved, for example, by expanding abbreviations (e.g., “OTOH” is normalized as “on the other hand”), correcting

⁴Multilingual and cross-lingual are used as synonyms throughout this work.

⁵Our preprocessing pipeline is assumed to be part of a bigger pipeline which would include a *main* task for which the input is adapted.

spelling variations (e.g., “dat” is replaced by “that”), or removing emphasis (e.g., “yeaahh” is normalized to “yeah”).

Word segmentation. This can be considered a subtask of text normalization that, in fact, should go before it in the pipeline. Since we are considering a word-level normalization task, it should be useful to start from a correctly segmented input text where words are clearly delimited. However, this might not be the case in user-generated texts: non-standard word segmentations can be used to provide emphasis (e.g., “im possible” instead of “impossible”) or they can just be the result of spelling mistakes (e.g., “noone” for “no one”). While normalizing word segmentation could be achieved in the main normalization task, we have decided here to perform this process in its own step of the pipeline before the microtext normalization step, in order to follow the modularity argument.

The modules that implement each of these preprocessing tasks were arranged in a pipeline, given the broad use of this structure in NLP. While this allows us to obtain the desired modularity, it does so in exchange of a low integration between the modules in the sequential steps, since information only flows in the forward direction. However, and as we further discuss in Section 6.2, the dependencies between tasks in the pipeline might not be entirely 1:1 and unidirectional; e.g., the result from the final step could be used to obtain a better result in the first step, which in turn would help the second one, and so on. To solve this issue, Valls-Vargas et al. (2015) proposed using non-linear pipelines, and Roth and Yih (2004) went further by replacing this structure with graphs of interconnected tasks. In both cases, the integration is improved with respect to the linear pipeline but, as we will also see below, they do not resolve the particular issues presented by our preprocessing scenario, where the order between preprocessing and main tasks in the pipeline has to be preserved if we want the former to be of any use.

1.3.2 A continuous word embeddings approach

Going beyond the pipeline and graph structures, modern NLP systems are increasingly relying on approaches based on neural networks (Plank et al., 2016; Eshel et al., 2017; Devlin et al., 2019), a particular type of machine learning models that have been obtaining state-of-the-art performance in the last decade thanks to the abundance of training data, computational resources, and new training algorithms. Similarly to a non-linear pipeline, these models are formed by a sequence of neuron layers connected in a feedforward fashion, sometimes also including recurrent links. Each of these layers perform a particular transformation on the input data, and the result is then fed to the next layer. Unlike

pipelines, neuron layers do not have to correspond with a specific task or being directly interpretable in a particular sense, generally. Given their great power and flexibility, neural networks tend to be trained as end-to-end systems by only providing the raw inputs and the corresponding desired outputs (hence, with no or very little preprocessing applied, eliminating the need for the tasks proposed earlier). If the number and type of layers and neurons per layer is adequate, these models have been shown to tackle complex tasks with minimal human intervention, encoding domain knowledge in the real-valued parameters of the network (Mikolov et al., 2013; Devlin et al., 2019; Ratner et al., 2018).

At this point, it would seem that using these powerful black boxes entails an inevitable loss of modularity, mostly when compared with a more interpretable structure such as the handcrafted pipeline. This is because we are now considering the system adaptation approach explained in the previous section, where the integration of different concerns (possible subtasks) in the model is now improved at the cost of its modularity. In reality, however, neural networks are able to avoid the compromise between modularity and integration that we have seen until now by facilitating *transfer learning* through mechanisms such as *pretraining* and *fine-tuning* (Erhan et al., 2010; Howard and Ruder, 2018). Basically, this allows us to train a model on a specific task, extract the information encoded in the network at the end of the process (i.e., the embeddings), and use it to train a different model for a different task, which can make use of that information to improve its performance. Indeed, this process can be repeated over more models and more tasks, accumulating more knowledge in the resulting networks.

In our context, we replace the preprocessing tasks considered earlier with the *pretraining* of embeddings that encode information about texting phenomena (described in Section 1.1.1). These embeddings would then be used to train other models that target specific NLP tasks; e.g., PoS tagging, sentiment analysis, or dependency parsing, which are now enhanced with knowledge about Web text phenomena. Notably, we replace the sharing of information across tasks through discrete symbols (e.g., words, tags, sentences, etc.) which constitute the inputs and outputs of each step in a pipeline, with continuous real-valued mathematical constructs (i.e., embeddings). We will analyze the advantages of this approach in Sections 3.4, 4.4, and 6.3.

1.4 Outline and contributions

This dissertation is organized in five parts plus one interlude chapter and three appendices. The main contents can be found between the second and fourth parts, as the proposed methods transition from discrete approaches to continuous ones, always orbiting around the considered tasks: language identification, word segmentation, and text normalization.

Complementary studies that support some of the main findings are included as appendices. We should note that, while these also contribute meaningful conclusions, they are not required to follow the main storyline of this dissertation. The interlude chapter included in-between those parts provides the theoretical justifications for the discrete-continuous transition. Finally, the present introductory part and the last one establish the discourse frame of our work.

In this outline, we enumerate the chapters in which this work is divided, highlighting their main contributions and the corresponding publications in scientific journals and conferences. It is worth noting that this work is a contextualized collection of previous published and unpublished research by the same author.

Part I

- **Chapter 1** describes the texting phenomena that characterize Internet speak, which make up for the motivation behind this work, the preprocessing tasks considered, and also introduces the approaches being studied in the following chapters.
- **Chapter 2** gathers relevant terminology for the domain of our work, and then introduces two important resources widely used not only here but in many other NLP systems: language models and word embeddings.

Part II

- **Chapter 3** presents the language identification task in the context of the TweetLID workshop (Zubiaga et al., 2014) and analyzes the performance of common language identification tools on tweets in the Iberian context. This chapter is based on:
 - Doval, Y., Vilares, D., and Vilares, J. (2014). Identificación automática del idioma en Twitter: adaptación de identificadores del estado del arte al contexto ibérico. In *Proc. of the Tweet Language Identification Workshop co-located with the 30th Conf. of the Spanish Society for Natural Language Processing, TweetLID@SEPLN 2014*, pages 39–43.
- **Chapter 4** proposes a simple approach for microtext normalization in the context of the W-NUT 2015 shared task 2 (Baldwin et al., 2015), based on the traditional two-step framework of candidate generation and selection, with a focus on modularity and adaptability. This chapter is based on:

- Doval, Y., Vilares, J., and Gómez-Rodríguez, C. (2015). LYSGROUP: Adapting a Spanish microtext normalization system to English. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 99–105.
- **Chapter 5** presents a word segmentation approach based on a search algorithm and a language model, and studies its performance when the latter component is implemented as a recurrent neural network and an n-gram model. This chapter is based on:
 - Doval, Y. and Gómez-Rodríguez, C. (2019). Comparing neural-and n-gram-based language models for word segmentation. *Journal of the Association for Information Science and Technology*, 70(2):187–197.

which was preceded by:

- Doval, Y., Gómez-Rodríguez, C., and Vilares, J. (2016). Spanish word segmentation through neural language models. *Procesamiento del Lenguaje Natural*, 57:75–82.

It is worth noting that bad word segmentation is treated here as a particular texting phenomenon. Hence, the relative ordering of this chapter and the previous one on microtext normalization does not follow the step sequence in the pipeline, but the process of going from general to particular which was also followed during research.

Part III

- **Chapter 6** analyzes, from a theoretical point of view, the inherent limitations of discrete pipelines and other similar approaches and how directly using word embeddings solves or bypasses the resulting issues. This theoretical analysis, which serves as the basis for the work presented in the following chapters, is novel work first published in this dissertation.

Part IV

- **Chapter 7** introduces a technique to improve the integration of cross-lingual embedding spaces obtained by aligning monolingual spaces. This chapter extends:
 - Doval, Y., Camacho-Collados, J., Espinosa-Anke, L., and Schockaert, S. (2018a). Improving cross-lingual word embeddings by meeting in the middle. In *Proc.*

of the 2018 Conf. on Empirical Methods in Natural Language Processing, *EMNLP 2018*, pages 294–304.

- **Chapter 8** describes an adaptation technique that improves the performance of existing monolingual word embedding models on noisy texts. It also presents a short study on the effect of bad word segmentation on the performance of word embeddings. The contents of this chapter have not been published at the moment of writing this dissertation. Once again, the relative ordering of this and the previous chapter does not follow the step sequence when obtaining robust multilingual word embeddings. Rather, it reflects the ordering of the chapters in Part II to show more clearly which continuous models replace their previous discrete counterparts, which in turn coincides with the ordering followed during research.

Part V

- Chapter 9 wraps up the storyline of this dissertation by presenting the most relevant conclusions and future lines of work.

Appendices

- **Appendix A** analyzes the performance of a wide range of phonetic algorithms for the task of normalization candidate generation tackled in Chapter 4. It is based on:
 - Doval, Y., Vilares, M., and Vilares, J. (2018b). On the performance of phonetic algorithms in microtext normalization. *Expert Systems with Applications*, 113:213–222.
- **Appendix B** presents a broad analysis of the factors usually involved in the bilingual alignment of monolingual embedding spaces as described in Chapter 7. This chapter is based on:
 - Doval, Y., Camacho-Collados, J., Espinosa-Anke, L., and Schockaert, S. (2019). On the robustness of unsupervised and semi-supervised cross-lingual word embedding learning. *arXiv preprint arXiv:1908.07742*.
- **Appendix C** provides a lengthy summary of this work in Spanish.

Chapter 2

Preliminary definitions

In this chapter, we first explain relevant terminology of extended use throughout this dissertation, and then introduce two crucial components used in several of the systems presented, as well as in many others across the NLP field: language models and word embeddings. Any other key element of this work is presented in-context throughout the main parts of this work; i.e., between Parts II and IV.

We also take the chance to remark that, in the first part of this work, we focus on the preprocessing part of a bigger NLP pipeline which includes *main* tasks; e.g., PoS tagging, dependency parsing, sentiment analysis, etc.

2.1 Terminology

This section introduces relevant terms that will be widely used throughout this work. Although some of these terms have already appeared in the previous chapter, we enumerate them here for easy reference:

- **User-generated text.** Any type of text written by users of a communication platform such as the Web, social media networks, chat applications, etc.
- **Microtext.** A form of short-length user-generated text which abounds in certain communication platforms such as SMS and microblogging social networks such as Twitter, where the number of characters per message is limited. Also frequent in chats and other communication platforms where users value the immediacy of the communications.
- **Standard word.** Any word registered by some recognized dictionary or other entity that regulates language usage.

- **Lexical variant** or *non-standard word*. Usually, an alternative spelling of a standard word, where phonological and morphological similarities between the variant and the original are exploited.
- **Standard** and **non-standard text**. By extension of the previous definitions, texts which are written following *de facto* or *de jure* regulations of language usage, or not, respectively. This covers using only standard words, in the first case, or including lexical variants, in the second.
- **Noisy** or **non-standard text**. The noise comes from the non-standard linguistic constructions employed (e.g., words, phrases).
- **Texting**. Short for “electronic text messaging”, which now covers any form of non-standard writing.
- **Texting phenomena** (Thurlow and Brown, 2003). Specific writing practices which deviate from the standard rules used while texting, including lexical variants and other non-standard constructs.
- **In-Vocabulary (IV)** and **Out-Of-Vocabulary word (OOV)**. In the first case, a standard word; in the second, a word not included in any recognized dictionary, which includes lexical variants and other types of words such as neologisms or new proper nouns not yet registered.
- **Code-switching** (Yu et al., 2013). The use of multiple languages in the same context frame, which most of the time results in words or phrases from different languages used interchangeably.

2.2 Language model

A **language model** is a probability distribution over sequences of linguistic tokens, which are usually characters or words, obtained from text corpora. Specifically, it models the conditional probability $p(x_t|x_{t-1}, \dots, x_{t-\rho})$, this is, the probability of the occurrence of some input token x_t given the previous ρ tokens in the input. With this information, we can also obtain the estimated likelihood or score of an input token sequence as the mean of the logarithmic probabilities of its constituent tokens:

$$\text{score}(x_t, x_{t-1}, \dots, x_0) = \frac{1}{t} \sum_{i=0}^t \log p(x_i|x_{i-1}, \dots, x_{i-\rho}) \quad (2.1)$$

Language models are usually implemented using two underlying structures: n-gram models (Brown et al., 1992; Stolcke, 2002; Heafield et al., 2013) or artificial neural networks (Mikolov and Zweig, 2012; Bengio et al., 2003; Kim et al., 2016).

An **n-gram model** stores historic data about n-grams, sequences of n tokens seen in a training corpus. To construct these models, we first have to set the order of the model n and, usually, a smoothing function which avoids assigning excessively high probabilities to frequent lower-order n-grams and compensates those with fewer, or even zero, counts. In this regard, Kneser-Ney smoothing (Heafield et al., 2013; Kneser and Ney, 1995) is generally considered to be the best option (Chen and Goodman, 1996). For this type of model, $\rho = n - 1$ in Equation 2.1.

Artificial neural networks are a type of machine learning model formed by several layers of processing units which perform basic operations, called neurons, and parametrized connections between neurons across layers. With respect to neural language models, the most usual design includes one or several hidden (middle) layers followed by a *softmax* layer of size equal to the number of token types considered, which are represented as *one-hot* vectors. At training time, the objective is to predict the next token in the input sequence which, given the 1-hot encoding, corresponds to assigning a probability of 1 to the occurrence of that token in the softmax, while the rest are 0. On the other hand, these networks are usually *recurrent* neural networks since their focus is on dealing with sequential data, such as text. Recurrent networks differ from traditional feedforward architectures in that they allow feedback loops in their structure, thus being able to use the output information corresponding to the input t when processing input $t + 1$, which fits the language modelling use case. In this case, the parameter ρ from Equation 2.1 is defined at training time as a hyperparameter of the neural network used for the Truncated Back-Propagation Through Time (TBPTT) (Principe et al., 1993).

Most recently, neural networks are being trained for language modelling under new paradigms, the most notable one being the *masked language model* (Devlin et al., 2019). Having said that, we will not go into a more detailed explanation of them as they are not used in this work.

2.3 Word embeddings

Word embeddings are vector representations of words usually defined over a \mathbb{R}^d embedding space, where d is the number of vector components or *embedding dimensionality*. They are usually obtained by traversing a large text corpus and accounting for the contexts, or groups of words, where specific words appear in. Then, following the *distributional hypothesis* of Firth (1957), similar embeddings (vectors) will be assigned to words occurring in similar

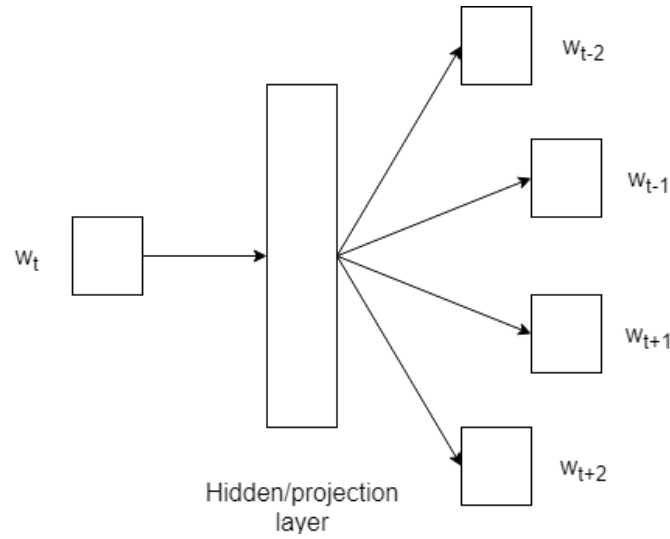


Figure 2.1: Skipgram model (Mikolov et al., 2013). The word embeddings are extracted from the projection (hidden) layer as the activation values obtained after a word is given as input.

contexts. This idea can be implemented in several ways. One option is through a word-word cooccurrence matrix where each line, or column, corresponds to a sparse representation of the corresponding word in the training corpus; i.e., each row contains the occurrence counts of a given word with the rest of the words in the vocabulary represented as columns. We can then treat the sparsity of the resulting matrix, which is induced by the *Zipfian distribution* of words (Zipf, 1949), through any matrix factorization technique. GloVe (Pennington et al., 2014), one of the most well-known word embedding models, takes this approach.

However, the majority of the work on word embeddings has sought to exploit the distributional hypothesis through a different approach: training neural networks in a specific manner so that we can then extract the internal representations they give to their inputs (e.g., words) at some hidden layer as our word embeddings. In general, starting from a random initialization of the network (i.e., yielding arbitrary embeddings for each word), the training procedure iteratively adjusts its parameters as it reads each word w_t from the training corpus and the various contexts it appears in $C = \{w_{t+j} / -c \leq j \leq c, j \neq 0\}$, with c being the number of words to the right and left of w_t . This adjustment can be performed through several training models or procedures. We now introduce the most common two:

1. Conditioning the network to learn a transformation that maps the embedding of a word w_t to the embeddings of the words in its context, as depicted in Figure 2.1. This

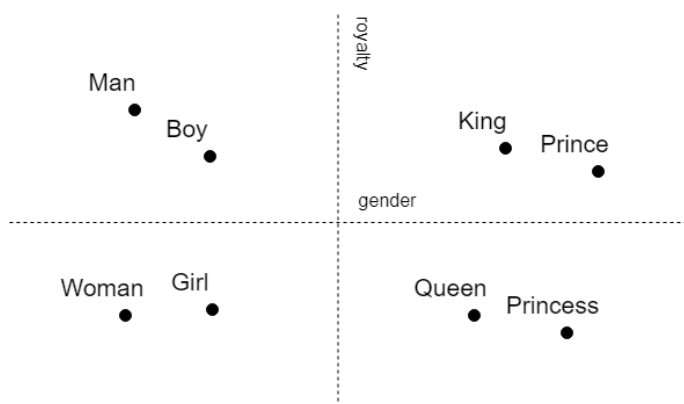


Figure 2.2: Simplified example of a two-dimensional embedding space.

is called the **skipgram model** (Mikolov et al., 2013), and we will use it extensively in Part IV.¹ More formally, the parameters of the network that would determine the activation values which constitute our word embeddings are obtained by optimizing the following objective function:

$$E_{sg} = \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \quad (2.2)$$

where T is the total number of input words, and we approximate $p(w_{t+j} | w_t)$ using negative sampling (Mikolov et al., 2013), a simplification of noise contrastive estimation (Gutmann and Hyvärinen, 2012).

2. A similar procedure but now mapping a context embedding, obtained by aggregating the embeddings of the words from C , to the embedding of w_t . This is called the **continuous bag of words model** (Mikolov et al., 2013).

The most widely used tools to build these models are word2vec (Mikolov et al., 2013) and fastText (Bojanowski et al., 2016). Also interestingly, this type of approach based on neural networks highlights the significance of the internal representations that they construct of their inputs, being important machine learning models not only because of their good performance when obtaining the final output at the last layer, but also because of the possibility to exploit their internal state.

The embeddings obtained by any of these methods have been shown to encode mean-

¹Or, more precisely, a modification of the original model.

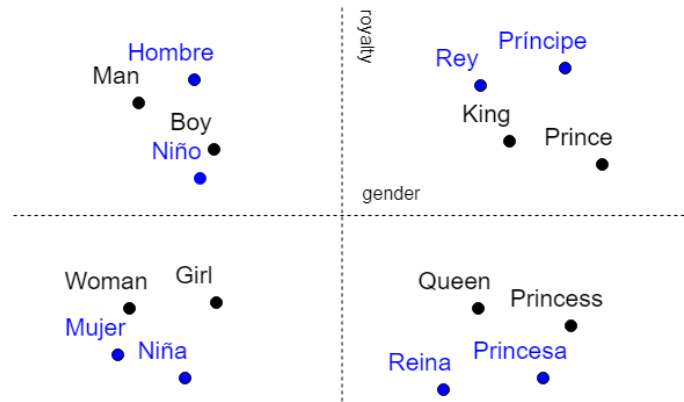


Figure 2.3: Simplified example of a two-dimensional bilingual embedding space.

ingful linguistic features of words, both semantic, syntactic, and even morphological. For instance, the placement of the word “king” in the embedding space relative to “man” is analogous to the one given for “queen” with respect to “woman”, indicating that the “royalty” semantic trait has been encoded in the embeddings, as we depict in Figure 2.2. Interestingly, these structures can be used as feature vectors that replace, or complement, the ones traditionally obtained by manual feature engineering. They have been successfully used in a wide range of NLP tasks such as dependency parsing (Bansal et al., 2014), information retrieval (Vulić and Moens, 2015), PoS tagging (Kutuzov et al., 2016), or sentiment analysis (Xiong et al., 2018), to name a few examples. Moreover, when these properties are extended to multiple languages so that, for example, “king” and “rey” (its Spanish translation) obtain similar vector representations, we call them **multilingual** or **cross-lingual word embeddings**, in contrast with the monolingual version seen until now. This extended scenario is depicted in Figure 2.3.

Finally, it is also worth mentioning that there have recently appeared new models such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) which go beyond the vector representations obtained by word2vec et al. In fact, the second one is trained on a language model task as mentioned at the end of the previous section, but using a different type of network which is also much larger than the ones considered in this work. Therefore, and once again, we will not go into a more detailed explanation of them as they are not used in this work.

Part II

Discrete pipeline approach

Chapter 3

Language identification in Twitter

We describe in this chapter our participation in TweetLID (Zubiaga et al., 2014, 2016) as part of our efforts to develop a language identification system for user-generated text which would allow us to tackle the multilinguality concern (see Section 1.1.1). The specific application domain proposed in this workshop involves tweets as text input and considers English and the following languages spoken in the Iberian Peninsula: Spanish, Portuguese, Galician, Basque, and Catalan. On this occasion, we resort to a simple approach where we perform a shallow adaptation of existing language identifiers for this particular use case.

We may define *language identification* (LID) as a particular case of the general classification problem where, given an input text (i.e., one or more words), the goal is to find out the language it was written in (Cavnar and Trenkle, 1994). This problem has been studied since the beginnings of NLP and it has already seen satisfactory results under certain controlled conditions; namely, when the input is constituted by long monolingual texts. Despite these achievements, certain application domains such as Twitter continue to pose a clear challenge for traditional approaches (Baldwin and Lui, 2010; Bergsma et al., 2012) since the operating conditions are very different in this case.

3.1 Domain particularities

As commented above, our operating context presents diverse particularities. First of all there is the nature of the entries, tweets of no more than 280 characters, which limits the amount of information available to perform language identification successfully, since length is a critical factor for this task (Baldwin and Lui, 2010). In addition to this, they are user-generated text often affected by the texting phenomena described in Section 1.1.1, thus introducing a fair amount of noise in an already information-scarce context.

Second, and also relating to the characteristics of microtexts and other user-generated content, is the multilinguality of this domain. This collides with the traditional assumption made in LID that entries are monolingual, implying that they are written in a single language (Baldwin and Lui, 2010) either fully or in the most part. However, on Twitter it is common to find multilingual tweets code-switching between several languages; e.g., “Goodbye, *hermano!*” (English + Spanish), and also short entries which can sometimes be *ambiguous*, with the possibility of belonging to any of several languages; e.g., “Boa noite” (Galician / Portuguese). We also consider the case of tweets written in an unknown language,¹ which should be labeled as such; e.g., “hyvä päivä” (Finnish, but categorized as “undetermined”). In any case, we would then speak of a *multi-label classification* problem (Tsoumakas and Katakis, 2007), where an entry can be simultaneously associated with several classes (languages, in our case). There is some work done in this area (Lui et al., 2014), but again with documents of a certain length.

The third factor relates to the availability of resources for our use case, apart from the development dataset given by the organizers of TweetLID (Zubiaga et al., 2014). On the one hand, some of the available traditional LID tools do not allow re-training, and usually do not include all the languages considered in the TweetLID task. On the other hand, for those that do allow adjusting them to our domain, we would need datasets to accomplish this, and although there are several publicly available ones (Lui and Baldwin, 2011, 2012; Majliš, 2012), few of them consider all the necessary languages, which is desirable in order to ensure a homogeneous behaviour. Also, it would be ideal to have datasets constituted by tweets, but these resources are even scarcer and do not cover the required languages (Bergsma et al., 2012; Lui and Baldwin, 2014).

Finally, the set of languages involved in this particular task, although small, is also challenging in itself. Let us consider Galician, which coexists with Spanish in the same geo-political region and also has a common root with Portuguese. This implies that it is usual for a Galician speaker to introduce Galician words when using Spanish, and vice versa, both voluntarily or by accident, something which is strengthened by the informal context of Twitter. Then, other factors such as the degree of language normativization, its dialectal variants, geographical situation, etc. lead us to find Galician speakers who use a more Castilianized Galician, and others who lean closer to Portuguese. On the other hand, the linguistic proximity between languages makes them share vocabulary, morphemes, etc., thus making classification even more difficult. Effectively, the problem presented at TweetLID is that of a multi-label classification in which there exists a clear overlapping between classes.

However, given our time and resource restrictions at the moment of the TweetLID workshop, together with the low number of multilingual tweets in the TweetLID development

¹In the sense that it is not a language modelled by the system.

dataset, we resorted to tackling language identification as a single-label classification problem by adapting existing language identifiers to our domain. We proposed two alternative solutions: the first one corresponds to the simplest scenario of using a single classifier and choosing the highest scored language for any given input, while the second consists of a voting scheme where, given a series of classifiers, the most frequently returned language is given as the final result (Lui and Baldwin, 2014).

3.2 LID resources

We have used existing LID resources whenever possible. Starting with the language identification tools, we considered the following ones which can be re-trained:

- **TextCat**. One of the simplest and most well-known LID tools, we use the implementation by van Noord (1997) of the classification algorithm designed of Cavnar and Trenkle (1994). In brief, it consists in ranking character n-grams using language models and an out-of-place metric.²
- **langdetect**. Implemented by Shuyo (2010), it is based on a Bayes classifier over character n-grams, adding simple input normalization mechanisms to reduce data sparsity.
- **langid.py**. Designed by Lui and Baldwin (2012), it also employs a Bayes classifier, this time in combination with byte-level n-gram features. According to the authors, this is a robust solution which, interestingly for us, outperforms its competition on microtexts.

We have preferred not to use pre-trained models for any of the above classifiers and, instead, train them ourselves in order to obtain a better control over the experiments, ensuring that all of the considered tools operate under the same conditions. For this task, we would need appropriate data resources for our use case which, as explained above, are not abundant in the public domain. Unfortunately, even the development dataset published by the organizers of TweetLID is rather small. Finally, we considered two data sources for our training corpora:

- The **European Constitutional Treaty**, or ECT (Unión Europea, 2004), although taking only from Part I to Part IV until Article IV-448, since these are the only ones available in Galician, Catalan, and Basque.³ This is a parallel corpus comprising 430KB of text, and 63,000 words per language.

²Some parts of the original code were rewritten in order to support UTF-8.

³<http://repositori.uji.es/xmlui/>

run	P	R	F1
Corpus selection			
TCE	50.6	54.0	47.7
Yali	55.9	61.0	54.8
Normalization selection			
Yali	55.9	61.0	54.8
Yali _{no1}	58.0	63.5	57.6
Yali _{no2}	58.9	64.4	58.2
Yali _{no3}	59.1	63.9	58.2
Yali _{no4}	59.0	64.5	58.3
Identifier selection			
TextCat	59.0	64.5	58.3
langid.py	41.3	25.2	22.9
langdetect	62.0	67.8	62.2
voter	42.3	26.6	24.9

Table 3.1: System tuning results (in bold the configurations selected in each phase).

- The **Yali Dataset Long** (Majliš, 2012), a comparable corpus (not parallel like the previous one) totalling 4.5MB of text, and more than 690,000 words per language.

It is worth mentioning here that the organizers of TweetLID distinguish between *constrained* and *unconstrained* tracks, which divide the presented systems into those that were trained only with the dataset they provided and those trained with any other dataset.

3.3 Evaluation

When deciding on the final configuration of our system, we carried out a series of setup experiments whose precision (P), recall (R), and F1 score ($F1$) results are shown in Table 3.1. These metrics are defined as the average precision, recall and F1 score for all languages considered, respectively:

$$P = \frac{1}{|L|} \sum_{l \in L} \frac{TP_l}{TP_l + FP_l} \quad (3.1)$$

$$R = \frac{1}{|L|} \sum_{l \in L} \frac{TP_l}{TP_l + FN_l} \quad (3.2)$$

run		<i>P</i>	<i>R</i>	<i>F1</i>	<i>pos</i>
constr	langdetect	73.2	73.4	63.9	9/12
	voter	61.0	58.2	49.8	12/12
unconstr	langdetect	68.2	68.8	58.1	5/9
	voter	58.8	59.0	57.1	7/9

Table 3.2: Official results and positions in the rankings (pos).

$$F1 = \frac{1}{|L|} \sum_{l \in L} \frac{2 \cdot TP_l}{2 \cdot TP_l + FP_l + FN_l} \quad (3.3)$$

where L is the set of languages considered and TP_l , FP_l , and FN_l are the numbers of true positives, false positives, and false negatives for a given language l , respectively (Zubiaga et al., 2014). In other words, precision is the number of times the correct language was identified over the total number of identifications of such language, and recall changes the denominator to the total number of entries (tweets in this case) in such language. On the other hand, we also used the development corpus provided by the organization (Zubiaga et al., 2014), but filtering special elements such as links, hashtags, emojis, etc., that would introduce noise (Tromp and Pechenizkiy, 2011). For this, we made use of a tool called Twokenize (Owoputi et al., 2013).

Firstly, we performed experiments to select the best dataset on which to train our LID models. Both the ECT and Yali datasets were tested with TextCat, obtaining the results that appear at the top of the Table 3.1, showing the superiority of Yali. Next, we applied a *light* normalization on the training corpora in order to reduce its noise levels: we passed the text to lowercase and eliminated digits (Yali_{no1}), and then we also removed diacritics (Yali_{no2}). The results obtained using the new training data show clear improvements. On the other hand, the texting phenomena were of particular interest to us given their language-specific nature. This is why we only eliminated character repetitions, trying to reduce the sequences of the same character to one (Yali_{no3}) or two characters (Yali_{no4}), the latter yielding better results as we show in the mid part of Table 3.1 (Normalization Selection). Finally, we selected the specific systems to be used in the competition: on the one hand the individual model that obtained the best results and, on the other, our approach based on voting. In both cases, the classifiers considered were TextCat, langid.py, and langdetect (see Section 3.2). The solution based on voting (voter) used all three of them. The obtained results are shown in the lower part of Table 3.1), with langdetect rising as the winner. This was surprising given the fact that Lui and Baldwin (2012) showed how langid.py is superior to langdetect on short texts.

The official results sent to the competition corresponded to langdetect and to our

solution based on voting, in both cases trained on the preprocessed training corpora. In the case of the constrained track, we trained the classifiers with the tweets of the development corpus provided by the organizers, while in the unconstrained track we used the Yali dataset. The results obtained are shown in Table 3.2, where langdetect improves the results of the voting scheme once again, although the difference is now much lower. The best results were obtained in the constrained setting, showing that using in-domain data (i.e., tweets) for training is more important than having a greater amount of out-of-domain data (i.e., standard text). In any case, the results obtained are not as good as those obtained in LID tasks on *regular* text (Lui and Baldwin, 2012).

3.3.1 Language-level analysis

We have also performed a language-level study and analyzed separately the case of monolingual and multilingual tweets, whose results are shown in Table 3.3. In the latter case, we did not take into account the ambiguous (AMB) or the undefined (UND) tweets. The monolingual results confirm the drop in performance for the solution based on voting (voter), which also has a more irregular behavior. In the case of langdetect, it shows a good behavior in the case of Spanish (ES) and ambiguous tweets (AMB). The precision is lower for Catalan (CA), Basque (EU), and English (EN), an unexpected result in the latter cases, since both languages are very different from the rest, which are Romance languages and, hence, should be easier to distinguish. The worst performance was obtained for Galician (GL), in most part due to the confusion with Spanish and Portuguese (PT), a problem we discussed previously; and for the undefined (UND) entries, whose recall was zero. Regarding the comparative analysis of monolingual versus multilingual tweets, the results obtained in terms of multilingual accuracy are even slightly better than in the monolingual case. At the same time, the recall obtained was practically half that of monolingual tweets, since almost all the multilingual ones contained two languages. Interestingly, these results indicate that the single language returned by our models was, in fact, one of the two languages in the tweet.

3.4 From discrimination to multilingual integration

The presented approach covers the absolute minimum requirement in our pipeline: it identifies the language a text is written in so that we can choose the adequate modules for the next steps. However, it does not directly tackle code-switching and, compared to the rest of the systems presented at TweetLID, it obtains relatively low performance scores. In this regard, it is worth remembering the challenging nature of the task at hand (Zubiaga et al.,

considered tweets		available (18397)			monolingual (16841)			multilingual (350)			
run	lang	P	R	F1	P	R	F1	P	R	F1	
constrained	langdetect	PT	82.8	90.4	86.4	86.4	90.7	88.5	77.3	68.0	72.3
		ES	97.7	81.5	88.9	98.9	82.3	89.9	98.7	50.5	66.8
		CA	56.4	89.2	69.1	65.4	92.5	76.6	67.4	34.1	45.3
		GL	23.1	84.1	36.2	24.8	85.3	38.4	13.6	33.3	19.4
		EN	69.6	82.2	75.4	81.9	92.3	86.8	96.5	31.1	47.0
		EU	56.1	84.8	67.5	69.4	95.8	80.5	97.9	46.1	62.7
		UND	100	0.8	1.7	-	-	-	-	-	-
		AMB	100	75.4	86.0	-	-	-	-	-	-
		GLOBAL	73.2	73.6	63.9	71.1	89.8	76.8	75.2	43.8	52.2
	voter	PT	50.7	70.7	59.1	52.8	71.1	60.6	23.9	44.0	31.0
		ES	93.7	66.8	78.0	95.0	67.6	79.0	94.8	35.6	51.8
		CA	48.7	73.2	58.5	52.6	75.7	62.1	67.6	29.4	41.0
		GL	9.1	46.7	15.3	10.6	47.3	17.3	5.6	22.2	8.9
		EN	52.7	65.5	58.4	58.8	72.5	65.0	88.5	30.5	45.4
		EU	32.8	71.3	44.9	36.5	79.1	50.0	83.3	44.1	57.7
		UND	100	0.3	0.6	-	-	-	-	-	-
		AMB	100	71.9	83.7	-	-	-	-	-	-
		GLOBAL	61.0	58.3	49.8	51.1	68.9	55.6	60.6	34.3	39.3
unconstrained	langdetect	PT	67.9	84.2	75.2	70.8	84.7	77.1	48.0	48.0	48.0
		ES	97.4	74.6	84.5	98.7	75.7	85.7	98.2	36.2	53.0
		CA	56.7	86.5	68.5	64.1	89.7	74.8	77.8	32.9	46.3
		GL	17.5	75.1	28.4	18.5	74.9	29.7	18.4	77.8	29.8
		EN	58.8	83.2	68.9	69.3	91.0	78.6	91.8	44.1	59.5
		EU	47.2	83.0	60.2	57.0	92.7	70.6	96.2	49.0	64.9
		UND	100	0.7	1.5	-	-	-	-	-	-
		AMB	100	63.5	77.6	-	-	-	-	-	-
		GLOBAL	68.2	68.9	58.1	63.1	84.8	69.4	71.7	48.0	50.3
	voter	PT	68.0	77.3	72.4	70.0	77.8	73.7	47.4	36.0	40.9
		ES	93.2	86.8	89.9	94.7	87.2	90.8	96.9	70.9	81.9
		CA	74.7	71.3	72.9	80.6	74.6	77.5	92.9	15.3	26.3
		GL	32.1	37.9	34.7	32.7	38.1	35.2	30.0	33.3	31.6
		EN	37.6	81.3	51.5	50.5	93.1	65.5	82.2	20.9	33.3
		EU	64.5	62.0	63.2	72.6	70.4	71.5	91.7	32.4	47.8
		UND	0.0	0.0	0.0	-	-	-	-	-	-
		AMB	100	56.5	72.2	-	-	-	-	-	-
		GLOBAL	58.8	59.1	57.1	66.9	73.5	69.0	73.5	34.8	43.6

Table 3.3: Language-level results.

2014), as recognized by the organizers themselves and supported by the low baseline scores obtained through Twitter metadata and *vanilla* TextCat. Moreover, only two of the systems presented considered the problem of code-switching but none of them were successful on this purpose. As explained before, the difficulties continue with discerning between similar languages frequently used together in a context where one of them is underrepresented compared to the rest, such as the case of Galician and Spanish. That said, and instead of looking for specific solutions to these problems, we posit if language identification is the best solution for our use case.

Essentially, a language identification step allows us to extract and consolidate the concern of *multilinguality* in a specific module of the pipeline, benefiting the modularity of the final

system by letting other modules to focus on one specific language at a time. But let us now assume that there exist some common intermediate language to which we can translate our multilingual inputs. In this scenario, we would replace the goal of discriminating between languages to integrating them in a common substrate (i.e., the intermediate language), so that the rest of the modules of the pipeline would be only concerned with processing input in such common language.

A possible implementation of this idea are the *multilingual embedding models* (Artetxe et al., 2018a; Conneau et al., 2018a; Ammar et al., 2016; Mikolov et al., 2013a) on which we base our work on this topic in Chapter 7. In short, they use real-valued vectors as intermediate representations encoding the linguistic properties of words from multiple languages; e.g., due to similar semantics, the numerical representations for “house” and its Spanish translation, “casa”, should be close to one another, so that processing texts containing any of those leads to similar results. In this case, code-switching ceases to be a problem and, in fact, plays in our favor serving as an important hint to relate word translations, since “house” and “casa” would be seen in similar contexts. This is why some methods to obtain multilingual word embeddings introduce *artificial* code-switching at training time (Luong et al., 2015; Wick et al., 2016) in order to strengthen the similarity between word translations and, thus, integrate both languages more tightly in the embedding space.

3.5 Related work

As already mentioned, the traditional setting for language identification involves long monolingual texts where the corresponding categorization problem can be tackled with simple techniques. The most famous one is the frequentist approach where we count the occurrence of character n-grams to build language profiles, and then compare the profiles of new documents with the previous ones to identify the language (Cavnar and Trenkle, 1994). Other common approaches involve training classifiers such as Naïve Bayes using character or byte n-grams (Baldwin and Lui, 2010; Lui and Baldwin, 2012; Shuyo, 2010) and various other combinations of features; e.g., Hammarström (2007); Ceylan and Kim (2009) employ dictionary and affix information, Giguët (1996) considered the characteristic tokenization patterns of each language, and Lins and Gonçalves (2004) used syntacting patterns rather than lexical ones. Alternatively, some authors employ more complex classifiers, such as Support Vector Machines (Baldwin and Lui, 2010; Lodhi et al., 2002), while others use Markov Models (Dunning, 1994) or Monte Carlo Sampling (Poutsma, 2001).

Language identification might be considered a solved problem in the previous scenario by some authors (McNamee, 2005). However, the progress on microtexts, and specifically tweets, is slower due to their unique set of characteristics, which calls for customized

methods for this domain. Laboreiro et al. (2013); Carter et al. (2013); Hong et al. (2011) show that adding new types of features in a classifier such as the language in which linked webpages are written, named entities mentioned, mentions, hashtags, replies, retweets, and other stylistic choices relating with texting phenomena (e.g., using “kkk” to represent laughter is unique to Brazilian Portuguese speakers), clearly improves the performance over a basic character n-gram classifier.

Regarding the systems presented at the TweetLID itself, they were mostly adaptations of traditional approaches (Zubiaga et al., 2014). To begin with, all of the systems used character n-gram features, with the exceptions of Hurtado et al. (2014) which included word information, and then Gamallo et al. (2014) went further by adding suffix information. For the classification part, Naïve Bayes (Gamallo et al., 2014) and Support Vector Machines (Hurtado et al., 2014; Porta, 2014) yielded the best results. Notably, these latter two also considered the multilingual dimension of the task, although the best results were obtained by the monolingual models of Gamallo et al. (2014), which shows that multilinguality was not determinant on this occasion. In any case, most of the presented approaches output confidence scores for each considered language, and thus Hurtado et al. (2014) and Porta (2014) trained a threshold parameter to not only include the single best matching language, but also those that surpassed the threshold.

3.6 Conclusions

In this chapter, we have presented our approach for the language identification task in the context of the TweetLID shared task (Zubiaga et al., 2014). Our proposed solution goes through adapting and re-training existing LID tools using various corpora of our choice, so that they all share the same starting point in this regard. With respect to the results, we can see how existing tools that report good performance in other more traditional domains or even short texts are not as accurate in the current context. Specifically, `langid.py`, which was initially evaluated on microtexts obtaining better results than its competitors (including `langdetect`), is not able to obtain remarkable results in our use case. We can conclude that the problem of language identification is not a solved problem in NLP, given the difficulties of existing tools to tackle the particularities of our experimental settings.

Finally, we question whether using language identification to support multilingual input in our preprocessing pipeline is the optimal solution. Instead, we consider using an intermediate language to which translate our multilingual inputs; in our case, by using multilingual word embeddings.

Chapter 4

Microtext normalization

As explained in Section 1.2, when dealing with user-generated content such as tweets and other similar sources for NLP or text mining purposes, texting phenomena make automatic processing of this type of content difficult, as most of the available resources tend to be designed to deal with plain standard text (Gimpel et al., 2011; Ritter et al., 2011; Foster et al., 2011).

In this chapter, we take the input adaptation or *normalization* path to overcome the challenges posed by lexical variants in microtexts and other user-generated content (Eisenstein, 2013). This is, given an input text affected by texting phenomena, we perform a correction at the lexical level which tries to make it adhere in a higher degree with the corresponding writing standards. Applying this as a preprocessing step in an NLP pipeline, we discharge later models such as PoS taggers or dependency parser of the responsibility to handle texting-related concerns, favoring the modularity of our solutions.

Specifically, we describe the microtext normalization¹ system we have used to participate in the W-NUT 2015 shared task 2 (Baldwin et al., 2015). Similarly to TweetLID (see previous chapter), in this shared task there are also two tracks: *constrained* and *unconstrained*, which affects the choices of resources to be used. On this occasion, we follow a mostly symbolic approach and focus on its modularity and adaptability, which is why we also describe in great extent most of its design and implementation details at the end of the chapter.

¹This preprocessing task usually refers to *microtexts* in its name since they constitute its main focus, but is not limited to normalizing short texts.

4.1 Architecture

Our normalization system was developed taking as basic premises its modularity and adaptability, two general design concerns of our work (see Section 1.2) which allow us to tackle complex problems and support multiple dynamic domains more easily.

As a starting point, we took a previous prototype for Spanish tweet normalization (Vilares et al., 2013) which, although functional, did not comply with our requirements. The proposed approach, however, shares its symbolic nature with the previous one, implying that the behaviour of the system is mostly defined by hand and the application of machine learning techniques is limited. Because of this, special attention has been paid to design and implementation details that optimize maintainability, a desirable property of systems where human intervention is required for a foreseeable future, which we leave for Section 4.3. In the following paragraphs, we focus on the fundamental ideas behind our approach that will interest a more NLP-centered reader.

Normalization can be considered as a superset of spell-checking (Aw et al., 2006), where lexical variants not only originate from simple spelling mistakes such as “tkae”-“take”, but include acronym expansion (e.g., “smh”-“shaking my head”), creative spelling (e.g., “100t”-“loot” or “dawg”-“dog”), or special tokens not to be processed such as hashtags, smileys, or URLs. On top of this, normalization also assumes an automated correction process where it is not enough to obtain candidate words to replace those badly spelled, but also to select the correct ones given the particular context in each case. Following this reasoning, our central idea for a normalization system is extending the capabilities of an already existing spell-checker to cover the characteristic phenomena of user-generated text. Hence, we would need to use a filter for special tokens, a spell-checker, auxiliary candidate generators based on phonetic algorithms or normalization dictionaries, and finally a candidate selector. More specifically, these components may be split and located into three sequential stages in a normalization pipeline, based on the two-stage approach of Han and Baldwin (2011):²

Input preprocessing. This is the only addition to the traditional two-stage approach. Here, we filter special elements that do not need normalization and would interfere in the process, such as hashtags, smileys, or URLs. This is achieved through the use of regular expressions and the PoS tags obtained by the Twitter-focused NLP toolkit ark-tweet-nlp (Owoputi et al., 2013).

²Note that this leads to using a general pipeline for the preprocessing tasks considered in this work (see Section 1.3.1) and also a *subpipeline* for the normalization part.

Candidate generation. For each word in the input not filtered in the previous step, we generate a set of normalization candidates (which includes the original word) by using spell-checkers, phonetic algorithms, and normalization dictionaries. The former two components can be merged together, as is the notable case of *aspell* (Atkinson, 2011) and the Double Metaphone (Philips, 2000), which we use here. Since only *off-the-shelf* tools are permitted in the constrained track, we used *aspell* with its default dictionary but filtering its retrieved candidate corrections taking as reference the canonical lexicon; i.e., only those candidates that could be found on this lexicon were taken into account. For the unconstrained run, we integrated the canonical lexicon provided into *aspell*.

Interestingly, many authors have noted the weight of a good phonetic processing of non-standard texts (Kobus et al., 2008b; Beaufort et al., 2010; Xue et al., 2011; Han et al., 2013a; Schulz et al., 2016). For this reason we also conducted a study of the most famous phonetic algorithms publicly available for their use in normalization candidate generation, which can be consulted in Appendix A.³

On the other hand, the normalization dictionaries are usually formed by pairs of (non-standard variant - standard word), and can be obtained automatically (Sridhar, 2016; Hassan and Menezes, 2013; Bertaglia and Nunes, 2016; Gouws et al., 2011) or manually, as in our case, where we concatenate the two normalization dictionaries and training data provided. Our idea is to cover frequent texting phenomena not already considered by the other candidate generators; mainly, acronyms (e.g., “lol”-“laughing out loud”) and slang (e.g., “curve”-“turn someone down”).

Candidate selection. Lastly, we select one normalization candidate for each input word by using context information; i.e., we choose the candidates that are most likely to appear next to their surrounding words, which are candidates themselves. To do this, we use a word-based n-gram language model trained on unlabeled text, which stores the probability of the occurrence of some input word w_t given the previous ρ words in the input, as introduced in Section 2.2 (replacing *words* with *tokens*). With this information, we use a search algorithm, in this case a Viterbi decoder (Viterbi, 1967), which navigates the candidate lattice and chooses the optimal path; i.e., sequence of candidates for each input word. Specifically, we train a Kneser-Ney bigram ($\rho = 1$, since $\rho = n - 1$) language model (Kneser and Ney, 1995) with the BerkeleyLM tools (Pauls and Klein, 2011) and the normalized tweets from the shared task training dataset.

In our current implementation, this is the only place where we score the candidates obtained in the previous step, and we will see how this is not an effective solution. However,

³As already mentioned in Section 1.4, while the contents of this Appendix are important in the context of microtext normalization, they are not required to follow the main storyline of which this is a part.

	<i>P</i>	<i>R</i>	<i>F1</i>
training			
constr	89.6	87.5	88.5
unconstr	89.1	87.4	88.3
test			
constr	46.5	62.8	53.4
unconstr	45.9	63.0	53.1

Table 4.1: Results on the training and test datasets.

we have considered this approach which can rely mostly on unsupervised methods in order to make our system easily adaptable to new domains and also to the ever-evolving nature of texting language (Eisenstein, 2013), lowering the requirement for costly human annotators. The only supervised component currently in use would be the normalization dictionary, which can be nonetheless obtained through fully automated means.

Unfortunately, regular words in a non-standard text are not always clearly delimited, which can be a consequence of a spelling mistake (e.g., “an dno”-“and no”, “wasnot”-“was not”) or an intentional variation (e.g., “im possible”-“impossible” to convey emphasis). Since our system operates at the word level, this phenomenon disaligns input and output, causing many input words to correspond to an output word (*n-1 mapping*) or an input word correspond to many output words (*1-n mapping*). With our current approach, we only provide explicit support for some form of 1-n mapping when the non-standard word is an acronym (e.g., “brb”-“be right back”). In order to tackle this problem following our modular philosophy, we could use a word segmentation step before word normalization that corrects word delimiter placement. This part of our preprocessing pipeline will be described in the next chapter.

4.2 Evaluation

The upper part of Table 4.1 shows the results obtained for the *training* corpus. It should be noted that these correspond to an *overfitted* system, since we inadvertently used a language model built using the whole training dataset (for candidate selection) in our 10-fold cross-validation framework. Nevertheless, this also gave us an interesting clue to the main performance bottleneck of our system, as we will discuss below.

The performance metrics are again precision (*P*), recall (*R*), and F1 score (*F1*) but adapted to the present context. In this case, precision is the number of correctly normalized words over the total of normalizations performed, recall changes the denominator to the

total number of words to be normalized, and F1 is defined by aggregating the former two:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \quad (4.1)$$

The lower part of Table 4.1 shows the results obtained for the *test* corpus. At the sight of these figures, which differ considerably from the previous ones, we decided to analyze them in more detail. For this purpose, we obtained a recall metric on the scope of the generated candidates in order to see how many times the correct candidate was among the ones considered by the system. The resulting ratio amounted to 0.87, a more consistent figure with respect to those shown in the case of the training results. Therefore, we can conclude that the performance bottleneck of our system is the candidate selection process, which is heavily influenced by the language model in use.

In this respect, additional experiments were conducted by extending our unconstrained configuration with the unigrams and bigrams from the Web 1T 5-gram v1 English language model made available by Google (Brants and Franz, 2006). However, the resulting performance was unsatisfactory, and we ended up discarding this option. According to our analysis, the cause for this seems to be the great differences, at both the lexical and syntactical levels, between the texts used to build this model, which could be considered as *regular* texts, and those corresponding to tweets, which agrees with the observations of (Chrupała, 2014). As illustrative examples of this type of expressions we can take “I like them girls” and “Why you no do that?”, which are lexically correct but not syntactically valid, so language models built using regular texts will not recognize them. In the case of our previous preliminary experiments on Spanish, this difference was not so clear.

We have also observed a clear tendency to *overnormalize* the input, this is, to normalize standard input words which are being correctly used. Specifically, this amounts to the 45% of the total number of errors in the test dataset (constrained track), where a 65% of the total number of words should be left unchanged. Again, having the language model as the only scoring mechanism seems insufficient for this case, and we could have introduced a weighting factor to favor the original input words in the candidate selection process. This would mean increasing the low precision of our system at the expense of some amount of recall.

4.3 The normalization pipeline: implementation details

We decided to give our system an *object oriented* approach as opposed to the *imperative* approach of the original prototype. At its core, our normalization system is structured as a pipeline of *processors*, modules that together implement the stages described in Section 4.1.

During its development, we have followed good engineering practices and made extensive use of *design patterns*. Among them, we highlight the use of the *decorator* pattern which, in our context, represents a simple pipeline, allowing us to dynamically stack an arbitrary number of processors. Its combination with the *composition* pattern lets us group them into *stages*, which enable the definition of particular processor sequences while still sharing the same basic processor interface, thus preserving the flexibility of the *decorator*. Thereby, the resulting structure allows for the dynamic construction of different pipeline configurations of varying complexity and different levels of abstraction, not being restricted to the original settings.

The application of the *template* pattern allowed us to factorize great part of the common processes of the components, such as the sequential iteration through all the input tweets, which most of the processors perform. This resulted in a great homogenization of the code, thus simplifying maintenance and allowing us to focus our efforts on the specific implementation of the processing methods in each case.

Moreover, some processors make use of external tools capable of being changed even at runtime—something of special interest in multilingual environments. It should also be possible to integrate them into other external components, so that their logic can be reused by others. All this involves decoupling the processors from the specific implementations of the external components employed, which we have achieved through the use of the *inversion of control* pattern.

Furthermore, communication between the components of the pipeline is done through structured text files, allowing us to gain flexibility as we can integrate and exchange with ease new processing modules regardless of their particular implementation (Vilares et al., 2013). In this case we have used XML along with an implementation of the *abstract factory* pattern for its construction and parsing. This also facilitates possible future migrations to other data representation languages, such as JSON.

Finally, we have created a *dynamic configuration subsystem* based on XML files that allows us to define and instantiate the particular structure of the pipeline on which we want to process the tweets. The advantages of such a subsystem are clear, both for system maintainability and testing:

1. It improves the multilingual support of the system by enabling the definition of configurations that use processors and resources designed for a particular language.
2. It allows for experimentation in a simple, agile, and documented manner, since the configuration file itself also serves as documentation.
3. It avoids the necessity of modifying the system source code.

4.3.1 Configuration before W-NUT 2015

The processor configuration for Spanish tweet normalization derives from that one used by the initial prototype for its participation in the TweetNorm 2013 task (Alegria et al., 2013). We now enumerate the processors used to implement the stages described in Section 4.1.

- `FreelingProcessor`, which reads the input data in the TweetNorm 2013 format and uses Freeling (Padró and Stanilovsky, 2012) to perform the tokenization, lemmatization, and PoS tagging (although these tags are not currently in use) of the text of the tweet.
- `MentionProcessor`, `HashtagProcessor`, `URLProcessor`, and `SmileyProcessor`, which act as filters for OOVs we do not want to consider for normalization.
- `LaughESProcessor`, which normalizes laugh string representations, as in “ja”-“jajaja”.
- `PhoneticProcessor`, which uses a phonetic table to map characters to their phonetic equivalent strings, such as “x”-“por”.⁴
- `SMSDictionaryProcessor`, which looks for normalization candidates in an SMS dictionary, for example “tb”-“también” (“too”/“also”).
- `AspellProcessor`, which obtains normalization candidates using the spell checker `aspell` (Atkinson, 2011), as in “polémica”-“polemik” (“controversy”). It should be noted that this tool has been customised with a new phonetic table for Spanish, based on the Metaphone algorithm (Philips, 1990; Mosquera, 2011) and a new Spanish dictionary extracted from Wikimedia resources.⁵
- `AffixESProcessor`, which identifies and normalizes affix-derived Spanish forms of base words, also supporting phonetical writing, as in the case of “chikiyo”-“chiquillo” (“little boy”), obtained from “chico” with the diminutive suffix “-illo” (“little”/“small”).
- `NGramProcessor`, which calculates the scores of those most likely normalization candidates according to the Viterbi algorithm (Manning and Schütze, 1999, Ch. 9) taking as reference the Web 1T 5-gram v1 (Brants and Franz, 2006) Spanish language model.
- `CandidateProcessor`, which selects the top-scoring candidate for each word.
- `ResultProcessor`, which dumps the tweet data obtained by the system to a file using the required format.

⁴The character “x” resembles the multiplication (times) sign ×, which in Spanish is read as “por”.

⁵<http://wikimediafoundation.org>

4.3.2 Adaptation for W-NUT 2015

In general, the adaptation process revolved around implementing new processors and integrating new resources to account for the requirements of this new use case, such as the use of English instead of Spanish or the new I/O data format, while leaving the base structure of the system untouched. This was precisely the main goal during the initial refactoring process. The resulting configuration includes the following new processors:

- `WNUTTweetProcessor`, which parses the structured input (now in JSON format instead of plain text) and obtains the system representation of the tweets.
- `ArkTweetProcessor`, which uses the `ark-tweet-nlp` PoS tagger to obtain the morphosyntactic information of the input tweet tokens.
- `WNUTFilterProcessor`, which filters out all those terms that should not be normalized according to the task rules (mentions, hashtags, URLs, etc.) using regular expressions and the information obtained in the previous step.
- `LowerCaseProcessor`, which takes all the candidate forms of a token and lowercases them. `AspellCProcessor`, a constrained version of the original `AspellProcessor` described above.
- `WNUTNgramProcessor`, which is similar to the previous `NGramProcessor` but with some added modifications to fit the particularities of our new custom language model.
- `WNUTResultProcessor`, which dumps all tweet data generated by the system in the required output format (JSON).

We show in Figure 4.1 a graphical representation of the architecture of the system both before (left side) and after (right side) the adaptation.

4.4 From normalization to texting phenomena encoding

In the context of the W-NUT 2015, our normalization pipeline did not obtain as good results as expected. The best performers were mostly systems that exploited the training data to a higher extent through various supervised models (Jin, 2015; Supranovich and Patsepnia, 2015; Min and Mott, 2015). In our case, we relied on the language model at the last stage of the pipeline to select normalization candidates in an attempt to place less emphasis on the human supervision signal and thus make the system more easily adaptable to changes (Bertaglia and Nunes, 2016; Eisenstein, 2013; Han et al., 2013a). However, we

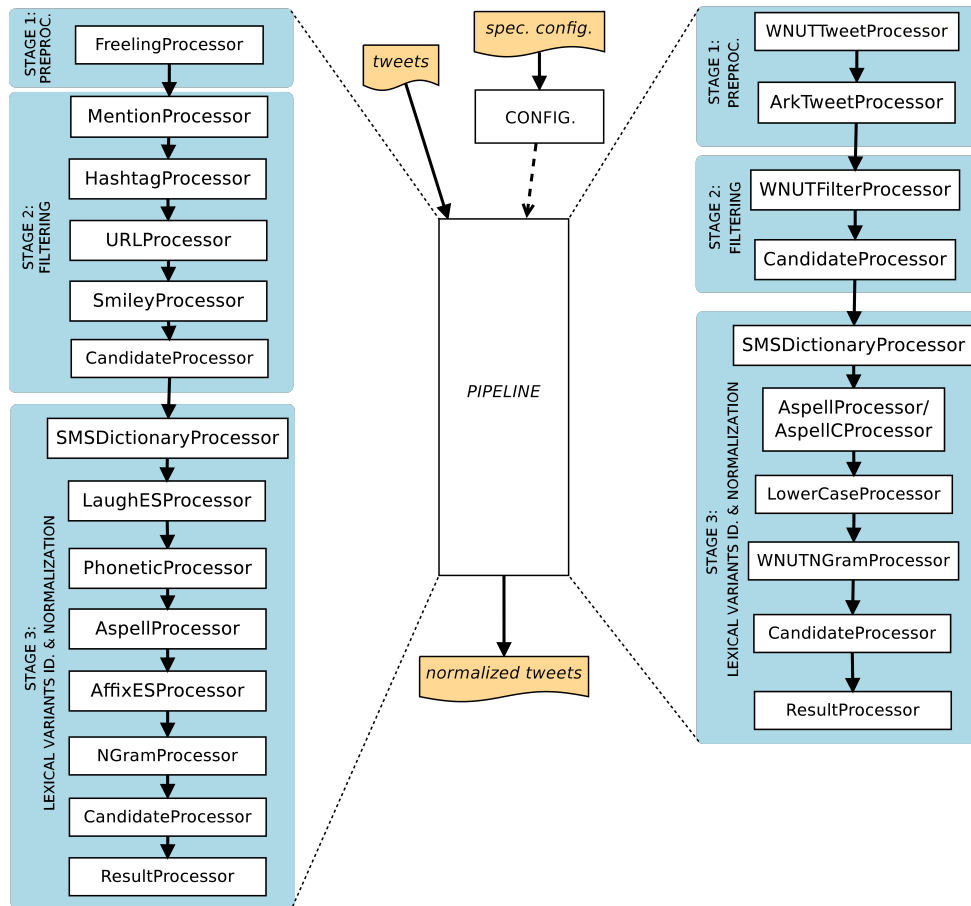


Figure 4.1: Original pipeline (left) and pipeline adapted for W-NUT 2015 (right) integrated into the architecture of the system.

have already shown that a language model trained on either large amounts of originally standard text or a small amount of normalized tweets is not able to choose the adequate candidates even when they were generated in previous steps. A possible solution would be to train a higher order n-gram model on large amounts of tweets containing only IV words, following (Han et al., 2013a; Yang and Eisenstein, 2013). Alternatively, we could also introduce scoring information in the candidate-generation steps by using an error model (Shannon, 1948). Thus, the probability of a particular word normalization is not only determined by its context but also given the frequency of that particular normalization in a training corpus. In principle, this makes our approach more dependent on an explicit supervision signal, which we would prefer to avoid.

Most interestingly, however, is observing that a series of state-of-the-art normalization systems are now relying on word embedding models to obtain and also select normalization candidates in the traditional two-stage architecture (Bertaglia and Nunes, 2016; van der Goot and van Noord, 2017; Ansari et al., 2017; Sridhar, 2016). When these models are trained on large amounts of standard and non-standard text, they tend to cluster together lexical variants, which can be then exploited in the normalization process; e.g., “yes”, “yeah”, and “yeeahh” will probably have similar vector representations, so that given the latter we can use the former as normalization candidates, selecting the second one afterwards based on other criteria such as edit distance⁶ or any other lexical similarity metric.

Taking this into account, it appears as if the rest of the modules of a normalization system fulfill two roles with respect to the embedding model:

1. Tackling low frequency non-standard variants in the training corpus which appear far away from each other in the embedding space (e.g., “dawwgg” may not be frequently used compared to “dawg” for the standard word “dog”, hence their vectors will not be similar).⁷
2. Transforming, or decoding, the continuous representations of words into their associated discrete linguistic symbols (i.e., words), needed as output for this task.

At this point, we can once again consider, analogously to the previous chapter for language identification, if normalization is the only option that we have when dealing with the noise in user-generated texts. As we will now see, this is not the case.

Let us assume that word embeddings can ultimately encode this noise in an effective way, as it appears to be the case given their usefulness not only in the normalization systems mentioned earlier, but also when using them instead of explicit preprocessing (van der Goot et al., 2017). Essentially, we would replace the normalization dictionary induction of those approaches with the direct use of the corresponding word embeddings. Then, the transformation of embeddings into words at the last step can be dismissed altogether if we consider that the following systems receiving our output will convert it back to a continuous representation, as in all the neural-based models that comprise the current state of the art in NLP and many other fields. On this basis, we can bypass the second concern mentioned in the previous paragraph. Now, regarding the low frequency variants which may not be easily captured by traditional distributional models, we can cover them by augmenting the training procedure of word embeddings to more explicitly encode the information relating

⁶Minimum number of character additions, removals, substitutions, or transpositions to transform one string into another (Levenshtein, 1966).

⁷Note, however, that they are *morphologically* similar, and this is exploited by some embedding models, as we will see in Chapter 8.

to spelling variants, as we will discuss in Chapter 8. The result is a word embedding model which can be used instead of, or complementing, a microtext normalization system.

It is worth noting that this solution retains a high degree of modularity and adaptability: word embeddings can be trained separately from the final system where they are used, and obtaining them only requires unlabeled data, which can be easily gathered at any point in time for virtually any domain. In turn, this highlights the possibility of having a much simpler system compared to the solution presented in this chapter.

Finally, both our current normalization system and most word embedding models operate at the word level, requiring in principle that words in the input text are correctly delimited. In reality, and as we will see in Section 8.3, word embeddings that exploit subword information seem to support word joining partially, relaxing the accuracy requirements for a word segmentation step such as the one described in the following chapter.

4.5 Related work

There are three traditional formulations for microtext normalization, also referred to as *metaphors* by Kobus et al. (2008a): spell-checking, machine translation, and speech recognition. In the first case, the microtext normalization problem is tackled through a *noisy channel model* (Shannon, 1948) where, given a noisy text N , its correspondent normalization S , an error model $P(N|S)$, and a (unigram) language model $P(S)$, we want to find $\operatorname{argmax} P(N|S)P(S)$. The error model may be built on information such as edit operations (Brill and Moore, 2000) or various graphemic and phonemic features (Choudhury et al., 2008; Toutanova and Moore, 2002). Similarly to spell-checking, this approach operates at the word level, does not take word context into consideration, and assumes that words in a standard vocabulary should not be corrected; in other words, it does not consider *real-word errors* such as in “Eye like NLP”, where the standard word “Eye” is incorrectly used due to having the same pronunciation as “I”, the correct word for this context.

In order to perform context-dependant corrections, approaches akin to statistical machine translation explore the use of phrase-level translations between a source (non-standard) and a target (standard) language (Aw et al., 2006), which nonetheless are adjusted for the similarities between the non-standard and the standard language. On the other hand, given the notable influence that spoken language has on microtexts, we may process the input words assuming that their spelling is closer to their pronunciation rather than to their standard spelling. Speech recognition techniques are useful in this scenario (Kobus et al., 2008a). In any case, the three formulations for microtext normalization are not mutually exclusive and can be combined to exploit the advantages of each one of them, as is usual for most systems (Schulz et al., 2016; Kobus et al., 2008a; Beaufort et al., 2010).

More recently, the celebration of the W-NUT 2015 workshop (Baldwin et al., 2015) has brought much attention to microtext normalization. The 2015 edition, where we participated with the work presented in this chapter, featured mostly supervised machine learning models that achieved high scores (Jin, 2015; Supranovich and Patsepnia, 2015; Min and Mott, 2015), although a simpler approach mainly based on normalization lexicons was presented by (Beckley, 2015) which also obtained competitive results. Some of the participant machine learning models were already based on neural networks (Leeman-Munk et al., 2015; Min and Mott, 2015; Wagner and Foster, 2015) at a time when they were still not popular in NLP. While obtaining good results, they could not prove to be significantly better than traditional statistical models, such as Random Forests (Jin, 2015) or Conditional Random Fields (Akhtar et al., 2015; Supranovich and Patsepnia, 2015).

It is also interesting to distinguish between data-driven supervised approaches, such as those presented at W-NUT 2015, and unsupervised ones, which remove the need for human annotations or pre-built dictionaries and are more easily adaptable to new domains and the evolution of the texting language (Eisenstein, 2013). Generally, these approaches exploit lexical similarity metrics to identify and score normalization candidates, word distribution to automate the construction of normalization lexicons, and language models to aid in candidate selection (Sridhar, 2016; Hassan and Menezes, 2013; Gouws et al., 2011; Han et al., 2013a). Given the proliferation of this type of approaches, where our own normalization system would ultimately correspond, together with the impact of neural networks, we decided to focus on the word distribution part when modelled as word embeddings (Bertaglia and Nunes, 2016; van der Goot and van Noord, 2017; Ansari et al., 2017; Sridhar, 2016). As we will explain in Section 4.4, this led us through a different research path, where word embeddings are not only the means, but also the goal itself.

Finally, there have been some efforts in microtext normalization for languages other than English; e.g., Spanish (Alegria et al., 2013, 2015), French (Kobus et al., 2008a; Beaufort et al., 2010), Dutch (Schulz et al., 2016; van der Goot and van Noord, 2017), Brazilian Portuguese (Bertaglia and Nunes, 2016; Duran et al., 2015), Arabic (Duwairi et al., 2014), Chinese (Wang and Ng, 2013), or Japanese (Ikeda et al., 2016). This is important to note as the appropriate normalization procedure is highly dependant on specific language features, such as phonetics (Satapathy et al., 2017). Furthermore, some approaches also focus on the modular design of the system (Schulz et al., 2016; van der Goot and van Noord, 2017).

4.6 Conclusions

We have presented in this chapter the tweet normalization system used to participate in the W-NUT 2015 shared task 2. Our approach is based on the traditional two-step framework

of candidate generation, for which we use a spell checker and a normalization dictionary, and candidate selection, implemented through a word-level language model and a search algorithm. We also strove for a modular and adaptable solution, obtained through the use of the appropriate design patterns.

The analysis of the results obtained at this point showed a wide margin for improvements. However, instead of working in the same direction as most existing normalization systems, we propose directly using word embeddings which encode texting phenomena. By doing so, we may overcome the encountered obstacles in a way that better aligns with the current state of the art in NLP, consisting mostly of neural-based end-to-end approaches.

Chapter 5

Word segmentation

The normalization system presented in the previous chapter operates at the word level. However, word boundaries are not always properly used in non-standard texts, and they are even fully omitted in special tokens such as hashtags which, although not a target for normalization, can still be interesting to analyze. Following the *divide and conquer* principle of splitting complex problems into smaller and simpler ones, we consider here word segmentation as a separate task in our preprocessing pipeline, preceding the normalization step.

Our proposed approach consists of a beam search algorithm assisted by a byte or character-level language model, which is implemented using a neural network or an n-gram model, respectively. The beam search algorithm goes through the input one token at a time (in this case, a byte or character) generating, for each step, a set of partial segmentation candidates by checking the likelihood of the current candidates and the probability of the next token corresponding to a word boundary in the language model. Then, at the end of each step, the n best candidates are chosen as input for the next one.

In our experiments, we have compared the described approach with the Microsoft Word Breaker (Wang et al., 2011) and the WordSegment Python module (Jenks, 2017). The languages considered for our tests were English, Spanish, German, Turkish, and Finnish, and we also included a test set comprised of English tweets. The latter three languages are known for their complex morphology, with Turkish and Finnish being agglutinative languages and thus conforming a greater challenge for a segmentation system (Manning et al., 2008, Chapter 2).

Overall, our approach was able to outperform both the Word Breaker and WordSegment for all of the languages considered, with the sole exception of a tie with WordSegment in one of the Spanish datasets. But most notably for us, our systems obtained notable improvements in such an interesting case as the Twitter dataset. Looking at the performance

obtained by the different types of language models used, we surprisingly see strong numbers for the simpler and faster n-gram model, which was in several cases on par with the more sophisticated neural model.

5.1 Problem domain

We define here *word* as a sequence of characters delimited by special boundary characters. This is an important concept in NLP as there are several tasks and systems which rely on word level information to achieve their goals. For instance, *tokenization*, a common preprocessing stage in many pipelines that splits an input sentence into its constituent tokens, requires that boundary characters are correctly used to be able to identify the corresponding tokens. In this context,¹ a *token* is usually a word or a group of words which constitutes the basic element to process in a particular task, such as in entity recognition (Tjong Kim Sang and De Meulder, 2003), PoS tagging (Ratnaparkhi, 1996), sentiment analysis (Vilares et al., 2017b), or in most word embedding models (Mikolov et al., 2013; Pennington et al., 2014; Bojanowski et al., 2016)

While an English speaker can easily discern the words in “thepricewasfair”, a machine can only see a sequence of characters—or more precisely, bytes—, corresponding to one long word. By explicitly executing a *word segmentation* step, the machine inserts word boundary characters between sequences of characters that would end up constituting words in some particular language. The resulting text, in our example “the price was fair”, can now be further processed at the word level, as these elements are now clearly isolated from each other. It is worth noting that this scheme also works for incorrectly segmented texts such as “th e pricew asf air” by first removing all the word boundaries. We show some example instances of the problem we are trying to solve in Table 5.1.

Our main objective is to develop a word segmentation system which can be used before the microtext normalization step in our preprocessing pipeline system described in the previous chapter. In this context, the biggest challenge comes from the type of texts we are dealing with, which are still affected by texting phenomena. These introduce a great amount of *data sparsity* in the problem at hand, as “hiiii” is not *exactly* the same as “hii” or “hi” but must be treated equally by the segmenter as one word. Because of this, we abandon the word level processing of these texts and opt instead for a character or byte level approach in order to tackle the resulting data sparsity problem.

¹Not in our case, where a token is either a byte or a character.

Input	Output
mostvaluablepuppets	most valuable puppets
webdesign	web design
RantsAndRaves	Rants And Raves
thankU	thank U
work allday	work all day
o m g u serious ??	omg u serious??
Safe way is very rocknroll tonight	Safeway is very rock n roll tonight

Table 5.1: Example instances of the problem we are trying to solve as input/output pairs.

5.2 System description

Before going into details, it is important to note that we will view the input text as a sequence of bytes when using the neural model and as a sequence of characters when using the n-gram model. We will refer to either a byte or a character as a *token*.

Our approach is conformed by two components: the beam search algorithm and the language model. The search algorithm acquires the input text and firstly removes all word boundary tokens. Then it analyzes the resulting text one token at a time, deciding whether a word boundary token would be appropriate in that position. If it is, two partial segmentation candidates may be generated, with and without the boundary. At some point, the number of candidates exceeds some predefined upper limit n , the beam width, and the n best candidates are chosen to continue the process. When the whole input is processed by this algorithm, m candidates from the currently n best are chosen as the final result. Figure 5.1 shows a simplification of the described procedure.

All decisions taken by the algorithm are based on the information retrieved from the language model, which estimates the likelihood of sequences of tokens. More precisely, given an input token and a history of ρ previous tokens, the language model approximates the probability distribution over all the possible token values for the next token in the sequence.

5.2.1 Language model

We use both recurrent neural networks and n-gram models to implement this component, which have been already introduced in Section 2.2. In this section, we delve into the details of the former type of model, as its inherently higher complexity calls for a more thorough analysis.

Given their focus on dealing with sequential data, such as text, recurrent neural networks

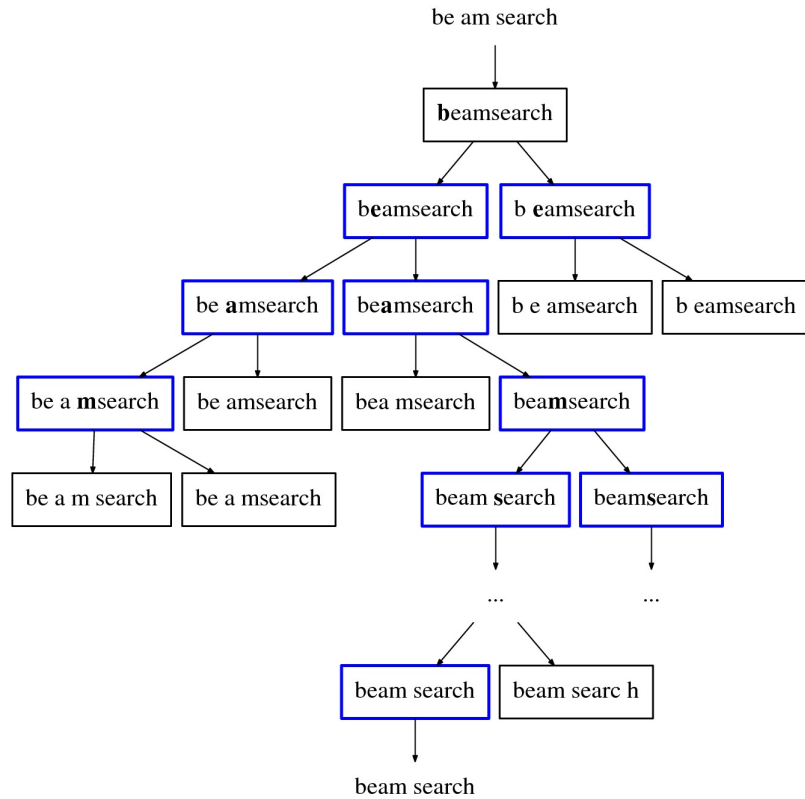


Figure 5.1: Simplified illustration of the algorithm execution, with $n = 2$ and $m = 1$.

are particularly fitting for language modelling. This is reflected in their wide use in several NLP tasks, such as machine translation (Johnson et al., 2017), dependency parsing (Vilares and Gómez-Rodríguez, 2017), question answering (Iyyer et al., 2014), or language modelling (Sundermeyer et al., 2012). Recurrent neural networks differ from traditional feed-forward networks in that they allow to use the output information corresponding to the input t when processing input $t + 1$. The existence of different types of recurrent networks comes from the different designs of their recurrent units. In our case, we have used LSTM units (Hochreiter and Schmidhuber, 1997) for the construction of our neural networks as they have proven very effective for language modelling (Sundermeyer et al., 2012). These LSTM units contain a memory cell, which stores information from past computations, and three *gates* which control the information stored in the memory as well as the output of the

whole unit.

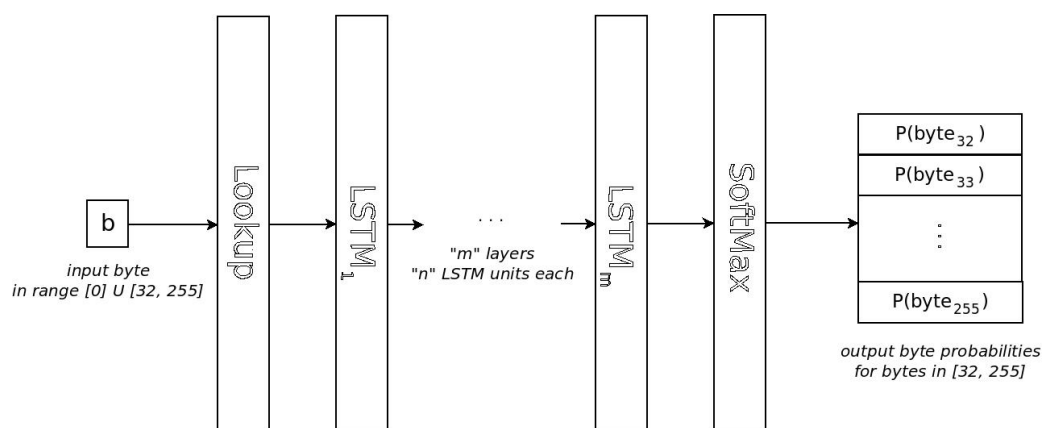


Figure 5.2: Illustration of the architecture of our neural networks. The *Lookup* layer transforms a number (byte) into a tensor suitable for the first LSTM layer. The *LSTM* layers apply a non-linear transformation to their inputs. The *SoftMax* layer computes the output probabilities using the output from the last LSTM layer.

Using a byte-level approach we can reuse the same network design for multiple languages, as the character set is not a parameter in the design process. This can also be an advantage for languages with large character sets, as fixing a smaller output size for the *softmax* operation in the last layer of the network avoids the bottleneck issues caused by this operation on large vocabularies. Furthermore, to reduce even more the complexity of the problem, we do not consider those byte values corresponding to non-printable characters except for the null byte 0, which can be used as padding in the input sequence. Assuming an Unicode encoding such as the popular UTF-8, these are the values in the range [1, 31]. The resulting neural networks receive as input one byte at a time from a given sequence and output the (logarithmic) probabilities for each of the possible next bytes in the sequence. The general architecture of these networks is depicted in Figure 5.2.

As explained before, we avoid the data sparsity problem by using language models that work at the byte and character level instead of the word level, and also by using neural networks. These latter models transform the sparse discrete input data, usually *one-hot* vectors, into continuous representations which encode meaningful information about the relations between the inputs and outputs of the network (Kim et al., 2016) (the same phenomenon used to obtain word embeddings, as introduced in Section 2.3). Under this assumption, three words such as “hiiii”, “hii”, and “hi” would end up having *similar*

continuous representations, as they are morphologically similar and would appear in similar contexts.

However, as powerful constructions as recurrent neural networks are, they tend to *overfit* the training data (Zaremba et al., 2014). To overcome this issue, several measures may be used, of which we have chosen Batch Normalization (Ioffe and Szegedy, 2015) and keeping the network as small as possible while retaining a good precision in the task at hand.

5.2.2 Beam search algorithm

Now we describe the beam search algorithm using a functional approach. For all the following functions, we define the threshold parameter t , beam width b , number of final results m , word boundary element wb , and scoring function $score$ as global constants in order to avoid long function signatures and improve readability. Additionally, the $+$ and \oplus symbols are used as the operators for string and list concatenation, respectively, s_i denotes the character at position i from the string s , $s_{i,j}$ denotes the substring of s going from index i to j , and $l_{i,j}$ the sublist of l going from index i to j .

The first function we define is $segment^*(part, txt)$, which recursively processes one token at a time from the input sequence. It takes two arguments: a list of partial results for the already-processed text and the remaining text to segment with no word boundaries and length l . The path this function takes depends on the emptiness of its first and second arguments, as shown below:

$$segment^*(part, txt) := \begin{cases} part & txt = \emptyset \\ segment^*((txt_1), txt_{2,l}) & part = \emptyset \\ segment^*(beam(part, txt_1), txt_{2,l}) & \text{otherwise} \end{cases} \quad (5.1)$$

If the second one is \emptyset , we have the base case and the recursion stops returning the current partial results list. If the first argument is \emptyset , the function bootstraps a partial results list and calls itself appropriately to begin the recursive process. This is the way that the function should be called the first time. The remaining case is the main recursive case, where $beam(part, c)$ takes as second argument the current token in the sequence and obtains a new list of n best partial results including that token in the segmentation:

$$beam(part, c) := top_n(xpd(part, c), b) \quad (5.2)$$

With respect to the $xpd(part, c)$ (expand) function used by the former and defined as:

$$\text{xpd}(\text{part}, c) := \begin{cases} \emptyset & \text{part} = \emptyset \\ (\text{part}_1 + c) \oplus (\text{bnd}(\text{part}_1, c)) \oplus \text{xpd}(\text{part}_2, |\text{part}|, c) & \text{otherwise} \end{cases} \quad (5.3)$$

it enables us to obtain new candidates by recursively traversing the partial results list and generating at least one more candidate for each of the existing ones by appending the next token c to them. It may also generate a second candidate if the call to the function $\text{bnd}(x, c)$ (boundary) does not return \emptyset . This boundary function is defined as:

$$\text{bnd}(x, c) := \begin{cases} x + wb + c & \text{score}(x + wb + c) > t \\ \emptyset & \text{otherwise} \end{cases} \quad (5.4)$$

and enables us to check, using the scoring function score , if a candidate x with a word boundary in the last position followed by c is likely or not. This is, whether its associated score (likelihood as returned by the scoring function) is greater than the threshold parameter t . If it is, it will return this new candidate.

With respect to top_n , the other function used by beam, it is the function that selects the n best partial results:

$$\text{top}_n(\text{part}, n) := \text{sort}((x, y) \mapsto \text{score}(x) > \text{score}(y), \text{part})_{1,n} \quad (5.5)$$

The final step would be to create a wrapper function that acts as an entry point to the system through the correct call to segment and then select the top m partial results to serve as final results according to the following definition:

$$\text{segment}(\text{txt}, t, b, wb, \text{score}, m) = \text{top}_n(\text{segment}^*(\emptyset, \text{r}(\text{txt})), m) \quad (5.6)$$

where r is a function that removes the word boundary characters from its input text:

$$\text{r}(\text{str}) := \text{filter}((x) \mapsto x \neq wb, \text{str}) \quad (5.7)$$

5.3 Experiments

In this section we describe the implementation details and followed procedure for validating our approach.

5.3.1 System implementation

We have implemented two versions of the system just described, one in Lua and the other in Python, due to the availability of the tools that we use to implement the language models.² Torch,³ a scientific framework with support for neural networks is available for Lua, and kenlm,⁴ a toolkit for n-gram language modelling, has bindings for easy usage in Python.

Torch only includes by default the tools to build feed forward neural networks, so in order to use it for recurrent neural networks we have imported the package rnn (Léonard et al., 2015). We also used the Adam optimization algorithm (Kingma and Ba, 2014) from the optim⁵ Lua package.

Regarding the kenlm toolkit, it is straightforward to use. The generation of the n-gram models was performed from the command line, while their integration with the search algorithm took place inside a Python script.

For the training and evaluation of the neural models, we tried to take full advantage of the parallelization features of Torch. Thus, all computations are performed in batches by a GPU, in our case a GTX Titan X (2015).

Another implementation detail not previously specified is the extra numeric parameter *win*. It defines the number of previous tokens from the current position in the input to use for the score computation. This is, instead of computing $\text{score}(x_t, x_{t-1}, \dots, x_0)$, we compute $\text{score}(x_t, x_{t-1}, \dots, x_{t-\text{win}})$ (see Equation 2.1 in Section 2.2). As this scoring operation is costly for the neural language model, this parameter allows us to seek a compromise between execution time and accurate scoring. It is worth noting that the value assigned to *win* does not have to be necessarily the same as the one used for ρ (see Section 5.2.1).

5.3.2 Corpora

The data used for training the models, both for our system and WordSegment, was obtained from several sources. For English, German, Turkish, and Finnish, we used the monolingual training datasets corresponding to the 2016 news from the WMT17 shared task.⁶ The English corpus was also augmented with tweets from the training dataset at <http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip>.

For each one of our datasets, we shuffled the lines⁷ and selected the first ten million lines (at most) for training and the last three hundred (six hundred in the case of English)

²All implemented code is available at <http://www.grupocole.org/software/VCS/segmnt/>.

³<http://torch.ch/>

⁴<https://kheafield.com/code/kenlm/>

⁵<https://github.com/torch/optim>

⁶<http://www.statmt.org/wmt17/>

⁷A line is defined as a sequence of characters delimited by newline characters.

for validation. We then removed special tokens such as microblog mentions, hashtags, and URLs, as they constitute counter-examples of the tokens we want to obtain in our results. For the Finnish and Turkish datasets, we also removed the SGML tags and some resulting blank lines.

In the case of Spanish, we employed a training corpus based on the Wikipedia dump from 2015/02/28 preprocessed using the `wikiextractor`⁸ and with all the Wikipedia markup expressions removed. From the result, we selected the first four million lines.

As test data, we used the monolingual testing datasets corresponding to the 2013 news from the WMT17 shared task for English and German, and the ones corresponding to 2016 for Turkish and Finnish as there is no test data from 2013 available for these languages. The preprocessing performed was the same as described above for the training corpus. The difference here is that we kept the English tweets test corpus, obtained from the same source as the training corpus, separated from the news test corpus.

It is also important to note that, unfortunately, we did not have enough resources available at the time to normalize some aspects of the tweets used for testing. These cause that a correct segmentation be labeled as incorrect, such as with the output “no way” for the reference “noway”. This has a greater impact on our approach and `WordSegment` than on the `Word Breaker`, as it considers a wider range of input characters, which may be incorrectly positioned in the reference. As an example, the “-” character may be particularly difficult to test properly as it tends to appear arbitrarily surrounded by whitespaces in informal contexts. To alleviate this and provide a fair comparison, we add a particular test case for these systems where we only consider the correct positioning of word boundaries around alphanumeric tokens. We also give the precision score for the corresponding *strict* test case where we consider the positioning of all word boundaries.

For Spanish we used two test datasets. One of them is based on the same Wikipedia article dump used for training where we randomly select 1000 short lines from the last 25% of the lines in the corpus. The second one is obtained in almost the same way as the former, but in this case lines are kept noticeably longer, as the random selection considers lines regardless of their length. Given the exact match scoring scheme that we will use, this should pose a greater challenge for all the systems in the benchmark.⁹

5.3.3 Results

In our tests, we focus on precision as the performance metric. Precision is defined here as the number of correctly segmented input instances over the total number of inputs given

⁸<https://github.com/attardi/wikiextractor>

⁹All dataset preprocessing scripts are available at <http://www.grupocole.org/software/VCS/segmnt/>.

	EN	Tw
512×2	82.0 ₂₅₁₀	75.6 ₁₂₉₉
512×3	82.3 ₃₁₂₇	78.3 ₁₄₇₂
757×3	82.0 ₃₇₆₆	79.0 ₁₈₀₀
1021×3	82.3 ₄₄₈₂	81.3 ₂₅₇₁
1500×3	86.6 ₈₂₃₅	82.6 ₄₀₀₀

Table 5.2: Precision results on the English (EN) and Twitter (Tw) development datasets by neural model architecture specified as $n \times m$, where m is the number of hidden layers and n the number of neurons per layer. Elapsed times in seconds are also shown in subscript.

to the system. A correct segmentation means that every word boundary in the output is correctly placed, otherwise it is deemed incorrect, i.e., our precision metric is an exact match score over whole lines. For example, given the input “Shel ikes music”, the only correct answer is “She likes music”, and any other possible answers such as “She like s music” would be incorrect.

We began our experiments by training some neural models and checking their performance in the development set throughout this process. We tried smaller networks first and progressively increased their parameter count until we reached bigger models with reasonable size and performance, both in precision and time metrics. Due to the high costs of training neural networks, which usually took *days* to complete, we were far from exhaustive in our exploration of the hyperparameter space. For this same reason, these initial experiments were only conducted with the English (EN) and Twitter (Tw) corpora. The best models for English and Twitter would then be used for the remaining languages: German (DE), Turkish (TR), Finnish (FI), and Spanish (ES).

In Figure 5.3 we show validation error curves for a few relevant models and in Table 5.2 the precision numbers obtained by those in the segmentation task. The results obtained for the tweets *strict* case are around 20 points below those shown in the table (see Section 5.3.2). As we expected, lower validation error numbers can be obtained with bigger networks, which generally translates into higher precision figures.

For standard text, relatively good precision can be obtained even by the smallest network. If we want to see the real benefits of more complex networks, we have to look at the precision numbers for a more difficult setup such as the tweets dataset. These networks, containing a higher number of parameters, are better suited for handling the greater sparseness in microtext data.

However, as shown in Figure 5.4, it seems that our networks cannot grow indefinitely in width (number of neurons per layer) with respect to depth (number of layers), since this

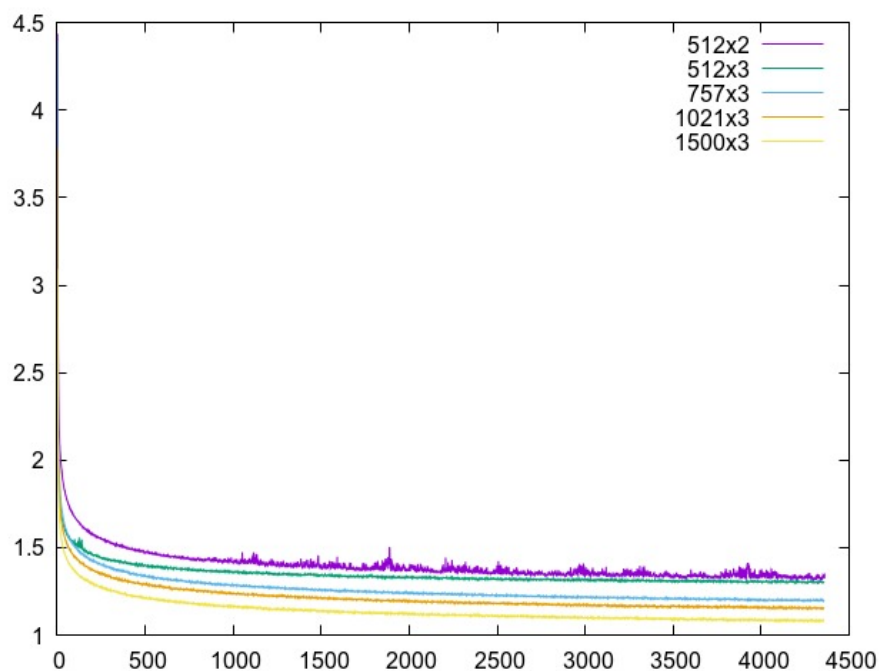


Figure 5.3: Validation error curves for some neural models on the English and Twitter training corpus. The architecture is specified as $n \times m$, where m is the number of hidden layers and n the number of neurons per layer.

may cause serious instability issues during the training process.

Consequently, in order to insert more neurons per layer, it would be also necessary to insert more layers into the network at some point. Moreover, given the long training process required by these bigger models and the precision numbers obtained, we decided to stop at networks of 3 layers and 1500 neurons per layer.

Next, we built our n -gram models, a process which took *minutes* even for the largest of these models. The only mandatory parameter we had to adjust here was the order of the model, n , for which we considered the values 4, 6, 8, 10, and 12. The precision numbers for each of these models on the development datasets can be seen in Table 5.3. In this case, the results obtained in the *strict* test are around 30 points below those shown in the table. As with the number of parameters of neural models, our n -gram models also benefit from higher values of n . After the large performance gain when moving from order 4 to order 6, the growth slows down until it stops or reverses for most languages when going from order 10 to order 12. On the other hand, these models are usually notably affected by the sparsity

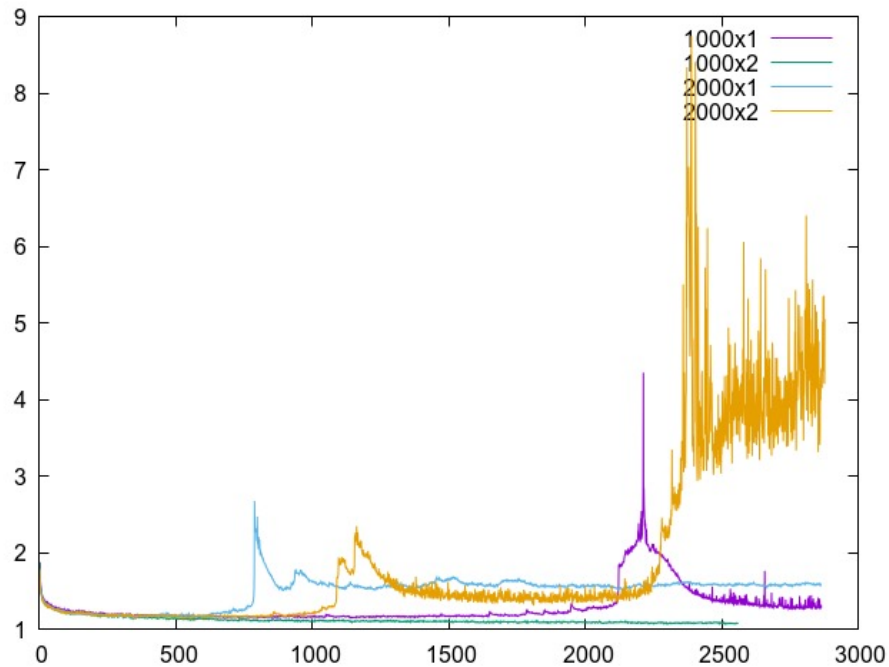


Figure 5.4: Validation error curves for models which are *too* wide. The architecture is specified as $n \times m$, where m is the number of hidden layers and n the number of neurons per layer.

of the training data, and using higher n values yields exponentially larger models. To avoid this, we can prune those n -grams with frequency counts lower than a specified threshold value at the expense of possibly lowering the performance of the model. In our tests, we used a pruning value of 5 for the 8-, 10-, and 12-grams without any noticeable performance drop.

For the reasons explained above, we decided to stop constructing bigger models at $n = 12$. This is similar to the reasoning applied in the case of neural models, adding the relatively small performance gain when going from $n = 10$ to $n = 12$.

In order to account for the difference in execution time for different languages, Table 5.4 shows the average counts of words, characters, and bytes per line (instance) in the development datasets.

Before obtaining the precision figures presented, we also tuned the search algorithm parameters for each type of language model using the development datasets. For the neural model, these were $t = 8, b = 10, win = 64$, while the n -gram model required

n	EN	DE	TR	FI	Tw	ES
4	37.6 ₅₇	34.0 ₄₇	51.3 ₄₉	33.6 ₃₀	48.6 ₁₈	37.0 ₂₇₉
6	86.6 ₆₈	77.3 ₅₅	94.0 ₅₄	82.0 ₃₃	77.3 ₂₅	72.6 ₃₄₄
8	89.0 ₇₀	84.6 ₅₉	92.6 ₅₆	88.3 ₃₅	77.6 ₂₅	78.0 ₃₈₆
10	90.6 ₈₇	89.3 ₇₂	93.0 ₆₁	91.6 ₄₁	79.3 ₃₂	79.3 ₄₂₅
12	89.3 ₁₀₅	89.3 ₉₂	93.0 ₆₅	92.6 ₄₉	79.0 ₄₁	79.3 ₄₅₁

Table 5.3: Precision results on development datasets by language and n-gram model order. Elapsed times in seconds are also shown in subscript.

	EN	DE	TR	FI	Tw	ES
words	20.0	15.2	13.6	10.0	13.1	42.1
chars	120.2	106.5	103.4	90.3	74.1	261.0
bytes	120.2	108.0	112.9	93.5	74.1	266.3
chars/word	6.0	7.0	7.6	9.0	5.7	6.2
bytes/char	1.0	1.0	1.1	1.0	1.0	1.0

Table 5.4: Average counts of words, characters, and bytes per instance in the development datasets.

$t = 10, b = 500, win = \infty$ to guarantee good performance across datasets. These numbers imply that the neural model is able to make better decisions at each step of the segmentation algorithm and thus requires carrying fewer partial candidates in order to finally decide on a good one. The opposite seems to be true for the n-gram model, which needs a wide range of possibilities available at each step and thus a higher b value, in proportion with the length of the text input.

Then, we compared our system, WordSegment, and the Microsoft Word Breaker (as of April 2017) against each other using the test datasets. In order to interact with the latter system, we used a slightly modified version of the demo code available at its website, transcribing or removing non-ASCII characters and adapting the formatting of the output to make it compatible with the rest of our evaluation scripts. The best precision numbers on the development datasets were obtained with $n = 3$ and the *Bing body corpus*. It is worth noting that, as this system works only with alphanumeric characters, the test gold standard was filtered accordingly. This implies that the failure surface for the Word Breaker is much smaller compared to that of our systems and WordSegment, as the former does not consider possibly troublesome characters such as “ ” or “-”.

Results are shown in Table 5.5, where we can see that both the n-gram and neural models were able to obtain higher precision numbers than both WordSegment and Word

Model	EN	DE	TR	FI	Tw	ES _S	ES _L
10-gram	91.4	82.2	77.6	77.5	75.7	92.4	80.3
12-gram	91.3	82.4	77.5	78.4	76.5	92.7	77.1
1021×3-neural	90.4	88.5	82.3	75.3	78.1	89.4	75.9
1500×3-neural	92.2	88.5	81.1	77.3	78.1	86.7	76.5
WordSegment	86.6	72.0	55.1	57.7	54.2	92.7	61.6
Word Breaker	88.7	38.5	15.4	8.9	62.6	88.0	66.5

Table 5.5: Precision results on the test datasets by language and approach.

Breaker on the test datasets. The sole exception is the tie between our 12-gram model and WordSegment in the smaller Spanish dataset (ES_S), which is resolved in our favor in the longer Spanish dataset (ES_L). For further detail, we have published more verbose outputs at <http://www.grupocole.org/software/VCS/segmnt/>.

WordSegment suffers from data sparsity problems as we can infer from the heavy performance drop in the Turkish, Finnish, and Twitter test corpora. For the Word Breaker, the performance gap was smaller in the case of standard English and Spanish. We believe that this is due to the language model of the Word Breaker having seen many more tokens from these languages than from German, Turkish, or Finnish, where it performed noticeably worse than its competitors.

More interestingly, we also come up well above in the English tweets dataset, a particularly challenging domain. This time, the reason might be the acute data sparsity present in this kind of texts, whose word vocabulary includes the wide range of texting-induced variations of standard words, which are handled well by the character-based approach.

Regarding the neural models, we can see that our currently biggest network does not perform much better than the second biggest across the test bench. In some cases, such as with Turkish and most notably Spanish, it is actually worse.

We can also observe that the performance of the n-gram models was close to the neural models, most notably for the Finnish language, and even surpassed them by a noticeable margin in the case of Spanish. Given the great attention and good results obtained by neural models in the literature, we expected the opposite to be true. To add more merits to the n-gram models, we should also mention their (quite) faster operation, both in training and evaluation time, compared to the neural models.

The main reasons why we did not try bigger or more sophisticated neural models are the good results we already achieve and the long training processes and slow operation times we obtain with our current biggest models, even on such a powerful GPU as the Titan X (2015).

We have also observed that it may be possible to apply this approach in a *cross-lingual environment*, even though our training and testing corpora are monolingual. The short English phrase conformed by common words “You are not welcome” in the Finnish test corpus is correctly segmented, as well as English named entities in the German and Turkish corpora such as “The Particle Adventure” or “The Voice”. It is then reasonable to assume that, given a suitable cross-lingual training corpus, it should be possible to address a truly cross-lingual scenario where sentences mix words from different languages.

5.4 Segmentation and continuous word embedding models

First of all, and in contrast to what we have seen in previous chapters, in the case of word segmentation we are able to obtain state-of-the-art results. This is crucial if we take into account that *words* are the central linguistic units on which not only our discrete pipeline operates, but also the word embedding models that we have been proposing until now to overcome the limitations of former approaches. When not considering other character-based models apart from the segmenter itself, this step is the minimum preprocessing that can be useful in both our discrete and continuous approaches. On the other hand, Wu et al. (2016) have shown that some form of text segmentation can be generally beneficial, instead of relying entirely on character-based approaches. Their own strategy, based on *word pieces*, also depends on words being correctly delimited.

Indeed, our word segmentation system may introduce segmentation errors in its output, and one could argue that these would distort the behaviour of the following pipeline steps or lead us to collect the wrong word embeddings, which has the potential to cancel out the benefits obtained in the latter case. As we will see in Section 8.3, embedding models that take subword information into account seem to support word joining to a certain extent. In this setting, using a word segmenter with a slight tendency to join words, for example through a threshold parameter as in (Doval et al., 2016) or even the raw input directly if we consider the low frequency of splits, since joins are more usual in special elements such as hashtags or URLs, can be considered good practical solutions.

5.5 Related work

Word segmentation is an important preprocessing step in several natural language processing systems, such as machine translation (Koehn and Knight, 2003), information retrieval (Alfonseca et al., 2008), or speech recognition (Adda-Decker et al., 2000). On the other hand, most Asian languages, while retaining the concept of *word*, do not use word boundary

characters in their writing systems to separate these elements. As a result, the application of word segmentation for these languages has drawn a lot of attention from the research community, with abundant work in recent years (Zheng et al., 2013; Pei et al., 2014; Chen et al., 2015; Xu and Sun, 2016).

Beyond the Asian context, we can also find European languages with highly complex morphology such as German, Turkish, or Finnish, which can also benefit from a conceptually different word segmentation procedure (Alfonseca et al., 2008; Koehn and Knight, 2003). In these cases, and mainly for agglutinative or compounding languages (Krott et al., 2007), new words are usually created just by joining together previously known words. A system with a vocabulary lacking these new words may still be able to process them if some sort of word segmentation system is in place. However, it is worth noting that this is a slightly different kind of word segmentation, as it is concerned with extracting the base words which conform a compound word. In contrast, our approach focuses on separating all words, compound or not, from each other.

Moving on to the Web domain, there are special types of tokens which can also be targeted by a segmentation system. The first ones to appear, and an essential concept for the Web itself, are URLs (Chi et al., 1999; Wang et al., 2011). These elements do not admit literal whitespaces in their formation, but most of the time do contain multiple words in them. Words may be separated by a special encoding of the whitespace character like percent-encoding or a different encoding that uses URL-safe characters. Most other times, words are just joined together with no boundary characters, and thus the requirement for a segmentation process arises.

Then, with the advent of the Web 2.0, the use of special tokens called *hashtags* in social media became very common (Srinivasan et al., 2012; Maynard and Greenwood, 2014). Like URLs, hashtags may also be conformed by multiple words. Unlike those, these elements do not use any word boundary character(s) between words, thus the use of a segmentation system seems more advantageous in this case.

The segmentation procedure that most of the previous work follows can be summarized in two steps. First, they scan the input to obtain a list of possible segmentation candidates. This step can be iterative, obtaining lists of candidates for substrings of the input until it is wholly consumed. Sets of predefined rules (Koehn and Knight, 2003) or other resources such as dictionaries and word or morpheme lexicons (Kacmarcik et al., 2000) may be used for the candidates generation. Then, for the second step they select the best or n best segmentation candidates as their final solution. In this case, they resort to some scoring function, such as the likelihood given by the syntactic analysis of the candidate segmentations (Wu and Jiang, 1998) or the most probable sequence of words given a language model (Wang et al., 2011).

Some other techniques, usually employed in the Chinese language, consider the word

segmentation task as a tagging task (Xue, 2003). Under this approach, the objective of the segmentation system is to assign a tag to each character in the input text, rendering the word segmentation task as a sequence labeling task. The tags mark the position of a particular character in a candidate segmented word, and usually come from the following set: beginning of word, middle of word, end of word, or unique-character word.

More recently, neural network based approaches have joined traditional statistical ones based on Maximum Entropy (Low et al., 2005) and Conditional Random Fields (Peng et al., 2004). These models may be used inside the traditional sequence tagging framework (Zheng et al., 2013; Pei et al., 2014; Chen et al., 2015) but, more interestingly, they also enable new approaches for word segmentation. Cai and Zhao (2016) obtain segmented word embeddings from the corresponding candidate character sequences and then feed them to a neural network for scoring. Zhang et al. (2016) consider a transition-based framework where they process the input at the character level and use neural networks to decide on the next action given the current state of the system: append the character to a previous segmented word or insert a word boundary. Both of these approaches use recurrent neural networks for the segmentation candidates generation and beam search algorithms to find the best segmentation obtained.

Outside the Chinese context, one of the most popular state-of-the-art systems for word segmentation in multiple languages is the Microsoft Word Breaker from the Project Oxford (Wang et al., 2011). Its original paper defines the word segmentation problem as a Bayesian Minimum Risk Framework. Using a uniform risk function and the *Maximum a posteriori* decision rule, they define the a priori distribution, or segmentation prior, as a Markov n-gram. For the a posteriori distribution, or transformation model, they consider a binomial distribution and a word length adjusted model. Finally, they solve the optimization problem posed by the decision rule using a word synchronous beam search algorithm.

The language model they use for the a priori distribution is presented in (Wang et al., 2010). This is a word-based smoothed backoff n-gram model constructed using the CALM algorithm (Wang and Li, 2009) with the Web crawling data of the Bing search engine.¹⁰ Some particular features of this model are that all the words are first lowercased and their non-ASCII alphanumeric characters transformed or removed to fit in this set, and also that it is being continuously updated with new data from the Web. However, the aggressive preprocessing performed by this system may result in limitations in two particular domains: microtexts and non-English languages. For the first case, data sparsity may pose a problem for a word-based n-gram language model. This type of model would have to see every possible variation of a standard word in order to process it appropriately. As an example, an appearance of the unknown word “hii” would mean using the <UNK> token instead of the

¹⁰<https://www.bing.com/>

information stored for the equivalent standard known word “hi”, which constitutes some loss of information. Then, if the also unknown word “theere” occurs, it would mean that the system has failed to use any relevant information to process the input. Hence, the input “hiitheere” could be incorrectly segmented into “hii thee ere”, a more likely path for the model given the known token “thee”

On the other hand, working only with lowercased ASCII alphanumeric characters leaves non-Latin alphabets out of the question—although Latin transcriptions could be used—and limits the overall capacity of the system due to the loss of information from the removed or replaced characters. For instance, consider “momsday” in the context of text normalization. The n-gram model would give higher likelihood to “mom s day” as it would have seen the token “mom” very frequently, both when appearing on its own and when swapping the “ ’ ” by a word boundary character in “mom’s”, obtaining “mom s day”. However, we prefer in this case “moms day” as the most likely answer, not only because it can be the correct answer but also because we can later correct the first word to include the apostrophe if needed and/or appropriate using a text normalization system.

The WordSegment Python module (Jenks, 2017) is an implementation of the ideas covered in (Norvig, 2009) that we have also used as baseline in this work. It is based on 1-gram and 2-gram language models working at the word level which are paired with a Viterbi algorithm for decoding. The system first obtains segmentation candidates which are scored using the n-gram models, and then the best sequence of segmented words is selected using the Viterbi algorithm. A clear advantage of this system for our work is that we can easily train its n-gram models from scratch in order to adapt it for our text domains/languages. This provides us with a better comparative framework than the Word Breaker.

5.6 Conclusions

In this chapter, we have presented an approach to tackle the word segmentation problem consisting of two components: a beam search algorithm, which generates and chooses over possible segmentation candidates incrementally while scanning the input one token at a time; and a language model working at the byte or character level, which enables the algorithm to rank those candidates. We have considered recurrent neural networks and n-gram models to implement the language model.

Our aim was to build a word segmentation system which can be used in the context of microtexts, a domain where data sparsity can be a problem to traditional approaches based on word n-grams, such as the popular Microsoft Word Breaker or the WordSegment Python module. We solve this issue by using byte- and character-level language models, and also by

taking advantage of the ability of neural networks to transform their discrete sparse inputs into continuous representations which encode similarities between inputs.

In our experiments, we explored possible configurations for our systems by adjusting the search algorithm parameters and the language model hyperparameters. The languages we considered for training and testing were English, German, Turkish, Finnish, Spanish, and also English tweets. Then we compared the performance of the different configurations of our system, WordSegment, and the Microsoft Word Breaker. The best neural models obtain the best precision figures overall on the test datasets.

Most surprisingly, the performance of the simpler n-gram models was close to their neural counterparts while being noticeably faster. Compared to WordSegment and Word Breaker, our approach obtained better results overall.

Part III

Interlude

Chapter 6

From discrete to continuous models

In previous chapters, we have described methods to tackle the preprocessing tasks considered in this work: language identification, word segmentation, and text normalization; through a discrete traditional pipeline approach. When analyzing their current limitations and proposing alternatives to overcome them, we have observed an interesting trend not unique to our use case: transitioning from discrete approaches to continuous models, such as word embeddings, along with dropping the assumption that explicit preprocessing tasks are even needed to achieve our final goals, opens up promising avenues for research, which in the end are more aligned with the current state-of-the-art techniques in NLP.

Taking each preprocessing task in isolation, we have observed that, in principle, the different concerns that each of them tackle can be encoded in a continuous embedding space. Solving these concerns in a discrete approach implies performing one or multiple transformations on the input which take care of those concerns explicitly. On the contrary, in the embedding space, these concerns together with other useful information are implicitly encoded into mathematical structures such as vectors, so that other systems may exploit this multidimensional view of discrete linguistic symbols to attain their objectives. Specifically, the *multilinguality* concern is tackled in the discrete setup by tagging chunks of texts with the language they are written in, whereas the continuous embedding approach assigns similar vectors to words in different languages with similar linguistic features (e.g., semantic, syntactic, etc.), so that models afterwards can treat them in a similar way. In the case of non-standard language, the discrete approach normalizes words so that they conform to standard linguistic rules, whereas the continuous embedding approach, once again, encodes the similarity between non-standard and standard variants of a word in their corresponding similar representations.

Outside the scope of each individual preprocessing task, our simple pipeline approach also carries some limitations of its own which relate to the low integration achieved between

tasks. This is caused by the strict sequential nature of this structure, which prevents using the information obtained by latter steps early on to improve the results. However, and as we will see below, discrete alternatives to the pipeline are not suitable for our preprocessing scenario, while continuous representations of words are once again rendered as a promising solution.

In this chapter, we first analyze the limitations of the pipeline approach; next, we argue that discrete alternatives such as non-linear pipelines and graphs may solve those limitations for some NLP tasks, but not in our case; finally, we arrive at a solution employing word embeddings which avoids said limitations altogether. But before going into details, it is important to note three points with respect to our argumentation:

1. It applies mainly to settings including several preprocessing tasks with an unclear sequential ordering. In other cases, pipelines might still be a good solution or even the only feasible one.
2. Word embeddings and pipelines are not mutually exclusive, and it should be possible to use them together in the same solution. In fact, taking into account the broad definition of *pipeline* as a mere sequence of steps, we will show in Chapter 7 how we can obtain and improve multilingual embeddings through a 3-step pipeline. In this case, however, we use the term *framework* to clearly distinguish between the discrete and continuous approaches, leaving *pipeline* to refer to its classical symbolic use-case.¹
3. It is not our goal to prove that the word embedding approach obtains better performance than the discrete pipeline system, but rather that it removes an inescapable performance upper bound of the discrete approach while taking better advantage of the latest state-of-the-art neural network models.

6.1 Limitations of the pipeline approach

Pipelines, a common structure for composing NLP models, sacrifice system integration in order to obtain high modularity. This latter property benefits reusability and separation of concerns, thus making complex problems more tractable by splitting them into smaller sub-problems that can be tackled in isolation to obtain better sub-results. On the other hand, integration is needed to guarantee an optimal performance of the final system as a whole. The low integration present in the pipeline approach used until now manifests itself in the following challenges:

¹Hence, we avoid considering a multilayer neural network as a pipeline.

Error propagation. This is an immediate consequence of any feedforward cascade or sequential structure which relies on first-best solutions in each of its constituent steps. Given a multistep pipeline, the error at the last step is not merely the sum of the errors at previous ones, but a multiple of that sum. In the intermediate steps, there are two sources for output errors which interact with each other: an incorrect input, which is likely to derive in an incorrect answer; and the inaccuracies of the model used in the step, since we make the realistic assumption that no model obtains 100% accuracy in any NLP task.

For example, given the Galician non-standard sentence “o meu móbil é bnito” (“my mobile phone is nice”) and its misclassification as Spanish text by a language identifier, a possible incorrect outcome of the subsequent normalization module or spell checker would be “o mejor móvil e Benito” (“or better phone and Benito”), a nonsensical sentence formed by words in the standard Spanish vocabulary. In this case, the incorrect identification of the language causes the misnormalization of the first four words, while the error in the last one is a direct mistake of the normalization module itself, as the correct answer would be “bonito”.

As we will see later, this problem has been already studied in the literature and several techniques and different architectures (non-linear, graph) exist to overcome error propagation. However, our particular use case is not exactly the one depicted for these alternatives, and hence they do not solve this problem without conflicting with the modularity constraint. As our preferred solution, we will rely instead on word embeddings to improve the integration of our systems while maintaining a high degree of modularity, as we will describe in Section 6.3.

Context fragmentation. A more specific problem to our setting, it consists in having heterogeneous input data for a specific task, where each homogeneous part has to be processed by a different, specialized model. This is effectively our case when considering code-switching; that is, when different parts of the input are written in different languages. In that case, the input has to be split in its constituent monolingual parts which the corresponding monolingual model will process in isolation. Clearly, when processing a part of the text written in a certain language, the corresponding model will not have access to the neighboring parts written in other languages, lowering the amount of contextual information available to it. As an example, consider the sentence “let’s go to my *ksa*”. Here, we have no context available to normalize the word “ksa” (for the Spanish “casa”, “house”), as the surrounding words are written in a different language and hence are processed by a different model. Two possible solutions to avoid this problem would be:

1. Using only *multilingual models* in our pipeline and remove the language identification step, something that goes frontally against modularity.

2. *Homogenizing the input* by translating all the parts into one common language, thus allowing us to focus on a single language in the remaining steps, but also introducing into our pipeline the complexities of machine translation, and an additional task. However, a derived solution would be to replace words with their corresponding vector representations (embeddings), which would serve as an intermediate language that homogenizes context.

6.2 Discrete alternatives to the linear pipeline

The error propagation problem described in the previous section is inherent to the simplest pipeline designs. Specifically, this is caused by its first-best, feedforward nature: the input to a given step is always the unique best solution obtained by the previous step, and information between steps only flows in the forward direction over one single path. Instead, we would prefer a structure where all the information obtained in any step can be used by others to improve their results, thus alleviating the problem of error propagation.

To better illustrate this, let us consider integrating a relation extraction and a named entity recognition model into such a structure. Also, note that these models have been trained for their respective tasks independently of each other. The goal of the integrating structure would be to use the information from one of them to influence the results obtained by the other, or at least be able to select a coherent solution taking into account the outputs from both models. For instance, if the first model identifies the *live_in* relation (or it is one of its candidate results), it should be more likely that the second one identifies the corresponding entities involved in the relation as *person* and *location*, respectively, instead of the inverse (*location* and *person*).² Likewise, if the second model identifies the *person* and *location* entities (or they are part of their candidate results), it should be more likely that the first one prefers the local prediction *live_in* rather than *birthplace_of*.

6.2.1 *K*-best pipelines and Bayesian networks

One common alternative is to consider a *k*-best pipeline (Wellner et al., 2004; Sutton and McCallum, 2005), where each step outputs a ranking of its *k* best predictions. The next step takes these and outputs its own set of *k* candidate answers for each one of them. The resulting candidate lattice is then traversed using a search algorithm which extracts the *k*-best results which, in turn, are fed to the next step, and the process is repeated. An example of this is shown in Figure 6.1.

²We use the terms “first” and “second” solely to distinguish between the two tasks, without implying any specific ordering.

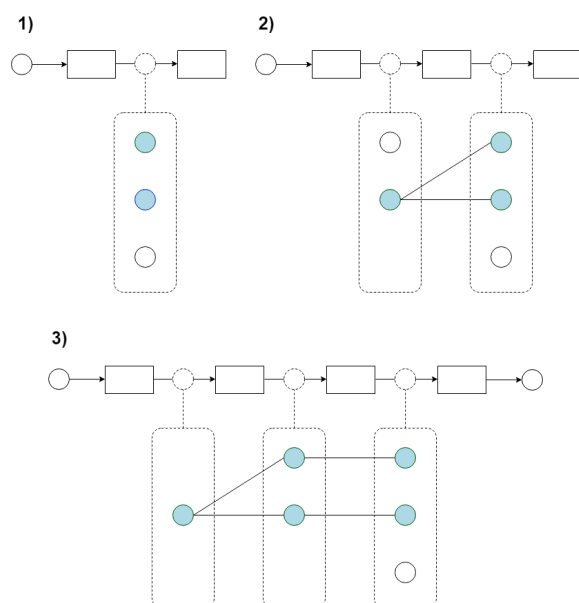


Figure 6.1: Sequential execution of a 3-best pipeline, where we use a beam search algorithm with $n = 2$ (as described in Section 5.2.2). In the first step, the top three answers (3-best) given by the first module are considered for the beam search, which trivially chooses the top two ($n = 2$, in blue). In the second step, the next module outputs its top three answers, but now the beam search looks for the top two best combinations of the previous two answers and the present three. Subsequential steps follow this same logic.

This scheme allows us to tackle the error propagation problem to some extent, as we are no longer limited to a 1-best cascading process where all the decisions are local to each step. In this case, we can use some extra information to choose from k possible results which combine outputs from several steps of the pipeline, looking to optimize some global criteria. However, if the parameter k is not large enough, we might miss on output combinations that would lead to better overall solutions down the line. On the contrary, if k is too large, the process can become intractable due to the high number of candidate solutions. Furthermore, once a step is executed and its k -best results are obtained, they cannot be changed by following steps; i.e., information from latter steps cannot be used to refine the output of preceding steps.

Finkel et al. (2006) discuss the limitation of k -best pipelines when selecting a good value for k and propose in turn to model them as *Bayesian networks*. Under this approach, each step or task in the network is represented as a variable with an associated probability distribution over its output domain (e.g., labels). The goal is to approximate a full swipe

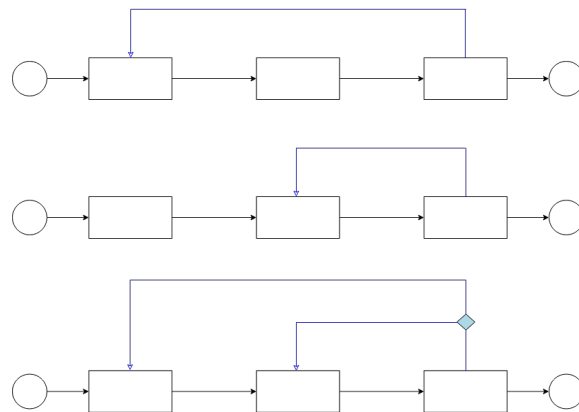


Figure 6.2: The three possible non-linear pipelines formed by three steps and at least one feedback loop.

over the space of solutions by drawing multiple samples of each distribution at each step conditioned on the samples from the previous one. The final solution is chosen by a classifier among the samples drawn at the last step. While this approach might solve the limitations imposed by the k parameter, further alleviating error propagation, it is still a feedforward construct, since samples are *conditioned* on previous ones, and information cannot flow backwards.

6.2.2 Non-linear pipelines

Non-linear pipelines, depicted in Figure 6.2, allow information to flow backwards and affect the outcomes of previous steps through the use of feedback loops and an iterative refinement process of the results. Hollingshead and Roark (2007) show a parsing pipeline where the performance of a coarse parser is improved by introducing constraints obtained by a shallow parser beforehand and a fine parser afterwards, which in turn improves the overall performance of the pipeline. Valls-Vargas et al. (2015) implements a pipeline for character role identification where the module at its last step feeds-back information to a previous coreference resolution model. In this case, they show performance gains in the coreference resolution step, but not in character role identification, the main task. This, together with the discouraging results reported by Samuelsson et al. (2008), highlight the difficulty of constructing this type of pipelines where all of its constituent models are benefited by their integration.

In general, the design of non-linear pipelines is dependant on the tasks included,

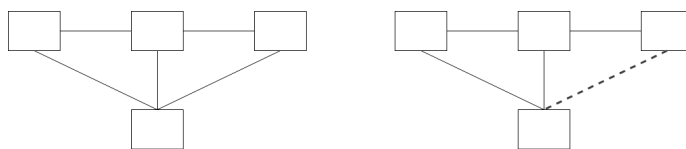


Figure 6.3: An example graph formed by four nodes (tasks) where we can prune one of its edges.

connecting specific steps through feedback loops where the information from the latter can be effectively used, or integrated, into the former, usually as complementary input. One consequence is that there still remains a strict linearity inside these loops, which might contribute to error propagation. Another consequence is that it is not trivial to add or replace modules in the pipeline (e.g., at which step should they be located) or establish feedback loops, as: (1) it is likely that the step receiving the new information has to be modified to integrate it effectively, leading to a loss of modularity in favor of a higher degree of integration, since its internal operation is no longer independent from the following steps of the pipeline where it is coincidentally included; and (2) it might be feasible to use information from one step to affect another one, but not the reverse (Finkel et al., 2006).

6.2.3 Graph-based solutions

Going beyond the pipeline design, some authors have instead explored using graphs to completely remove the sequential ordering constraints of the tasks, and consequently eliminate the error propagation problem of feedforward structures. The result is akin to a mixture between a non-linear pipeline and a k -best approach where all of the steps are connected with each other using bidirectional links and the best local solutions are chosen based on global constraints. Roth and Yih (2004) proposed precisely this approach, modelled as a linear optimization problem where the final results for each task depend on global constraints introduced at decision time. They call this process *global inference*, and it solves not only the limitations seen for non-linear pipelines when using information from other tasks to influence the outcome of a given one, but also the error propagation seen for both k -best and non-linear pipelines, as there is no fixed ordering.

Building upon this model, Marciniak and Strube (2005) study a general scenario where more than two tasks are considered. They include a method to prune the graph edges between unrelated tasks in order to simplify the optimization problem, which is done by discarding the links between models whose local predictions are weakly correlated. We illustrate this in Figure 6.3.

Unfortunately, while this approach seems to correctly balance modularity and integration, solving error propagation effectively, it is based on a false premise when considering a preprocessing setup: that there is *no* sequential ordering between tasks. As their name suggests, *preprocessing* tasks imply that they are located *before* the main tasks, and thus that a strict dependence emerges between the two types of tasks. In this scenario, we would still want some information from the main tasks to flow back to the preprocessing part in order to reduce error propagation. Consequently, the advantages seen for this approach do not fully transfer to our use case.

6.3 The continuous approach: word embeddings

All of the proposed alternatives to the 1-best linear pipeline share the same goal and general methodology: alleviating error propagation through higher integration of the constituent models. However, we have seen that they still carry some set of limitations in our specific problem domain derived from the non-trivial task of balancing modularity and integration. On top of this, we must also highlight that they do not explicitly consider the issue of context fragmentation described in Section 6.1, and that there is another specific practical consideration playing an important role against them. Current state-of-the-art NLP systems are mostly based on machine learning techniques which operate on real-valued *feature vectors* that describe the discrete input elements over a multidimensional continuous domain. As a consequence, assuming that we use these machine learning models to implement the steps in our pipeline, each one of them performs a discrete-to-continuous (encoding) and then a continuous-to-discrete (decoding) transformation: a discrete input element is expanded into its continuous feature vector to be processed inside the model and, then, it is *collapsed* back to a discrete element to be used as input for the next stage or as the final result of the pipeline, unless we are considering a regression task. Note here that the effective processing of the input is *not performed* on its discrete form, but its continuous representation, and that a discrete output can only be useful at the end of the pipeline since the user requires an *interpretable* result. If we now consider that preprocessing tasks are, by definition, located before the main tasks in which the user is interested, it should be questioned if it is even worth obtaining intermediate discrete outputs in each step of the pipeline.

In fact, by not forcing the collapsing of a multidimensional continuous representation into a discrete one-dimensional element after each step of a pipeline, we avoid the multiple *lossy* encoding-decoding transformations which contribute to the problem of error propagation of the previous discrete approaches. From this point of view, *k*-best models and Bayesian networks try to minimize this information loss by collapsing to *k* elements instead

of just one, and using probability distributions on the possible outcomes, respectively. In contrast, when we use the continuous representations obtained internally by each model directly, *no information* is lost.

6.3.1 Embeddings as modular *and* integrated encoded knowledge

In our case, we use words as our basic linguistic elements on which to construct continuous representations, or embeddings. As already explained in Section 2.3, *word embeddings* encode word features (e.g., morphological, syntactic, semantic, etc.) in real-valued vectors. Notably, they can be used to initialize the input layer of neural network models, the current state of the art in NLP, in which case they are considered as *pre-trained* parameters which can be *fine-tuned*, or left unchanged, during the training of those neural models. In any of these cases, pre-trained word embeddings have been shown to help them achieve performance improvements and even reduce the need for labeled training data (Devlin et al., 2019; Peters et al., 2017; Collobert et al., 2011).

This showcases the modular aspect of word embeddings and other parts of neural networks, in general, since they can be used as a means for transferring the knowledge obtained in multiple tasks into other domains, allowing for an incremental acquisition process if correctly fine-tuned (Howard and Ruder, 2018). In fact, Chapter 7 will show how to obtain multilingual embeddings from monolingual models, which is a clear example of this incremental process, and where these models do not only gain a new feature (i.e., multilinguality) but also see their monolingual performance improved.

This property can also be considered as the counterpart of *feedback* loops in non-linear pipelines, which increase the degree of integration across tasks. The crucial difference here is that the integration of multitask information is generally accomplished through the usual automated training procedure based on the *backpropagation algorithm*, which replaces the manual implementation of custom feedback loops. While this can be performed either incrementally (Hashimoto et al., 2017; Phang et al., 2018) or in a single step (Luong et al., 2016; Subramanian et al., 2018), some authors note that one should take into account the characteristics of the training data in use along with the specific tasks involved in order to obtain the best results (Benton et al., 2017; Martínez Alonso and Plank, 2017; Bingel and Søgaard, 2017). Fortunately, Ratner et al. (2018) propose a solution that avoids those pitfalls and restrictions from previous approaches, allowing it to scale effortlessly to a greater number of tasks by using training data of unknown quality. This renders it as a clear demonstration of the ability of neural networks, and word embeddings as part of those networks, to automatically and effectively integrate information from an arbitrary number of tasks.

Hence, taking all of this into account, word embeddings seem to strike an adequate balance between modularity and integration while maintaining the amount of human work to accomplish this to a minimum. These properties of the continuous approach make it stand out when compared to discrete models, and serve as the justification for our paradigm shift.

6.3.2 Tackling our current challenges

In Section 6.1, we have briefly introduced word embeddings as a means to solve the limitations presented by linear pipelines. Specifically, error propagation can be tackled by an improved system integration, as described above, and context fragmentation can be solved by considering translating, or transforming, the input words into multilingual word embeddings (Artetxe et al., 2018a; Conneau et al., 2018a; Ammar et al., 2016; Mikolov et al., 2013a) which constitute a common *intermediate language*. In this case, word translations are close to each other in the embedding space (e.g., “casa”-“house”, “gato”-“cat”, etc.), making it possible to use models trained on these multilingual continuous representations directly without requiring an explicit machine translation step to homogenize language context.

Regarding the steps included in our preprocessing pipeline (see Section 1.3.1), word embeddings may be used to encode the concerns raised by the multilinguality and texting phenomena of user-generated texts and replace their traditional explicit processing (see Sections 3.4 and 4.4). For instance, multilingual word embeddings may be used instead of the language identification step, given the property previously mentioned. Furthermore, the normalization step, which deals with noisy variants of words, may also be discarded if we consider that lexical variants of words in a particular language should have similar vector representations when trained on large amounts of standard and non-standard texts (Bertaglia and Nunes, 2016; van der Goot and van Noord, 2017; Ansari et al., 2017; Sridhar, 2016). All of this means, in principle, that we could obtain embeddings for, e.g., “frnd”, “friend”, “amigo”, and “amgo”, which are close to each other in the embedding space, indicating that they should be treated similarly by other NLP models while still being able to distinguish between them.

6.3.3 The downside of this approach

However, there is one immediately apparent drawback of using word embeddings together with neural network models: the loss of *interpretability*, an issue affecting the vast majority of machine learning models. In a discrete pipeline, the outcome from each step is usually formed by human-readable discrete symbols, enabling an easier analysis and debugging

of the system, even when each step is implemented by an opaque model. On the contrary, in a fully continuous approach, the resulting machine learning models operate entirely on their own numerical domains from start to finish, where they encode the information they receive on their own terms to obtain the desired results. Having said that, there has been a good amount of work towards deciphering the internal state of complex neural networks by visualizing neuron activations under controlled conditions (Yosinski et al., 2015; Karpathy et al., 2015; Li et al., 2016). Here we can also include the study of linear structures emerging in a word embedding space which make these models so interesting (Mikolov et al., 2013b). Furthermore, the inspection is likely to be facilitated by the incremental construction of complex neural networks through pre-training and fine-tuning, following the same reasoning as before about splitting complex problems to make them easier to solve.

In the following chapters, we will describe our word embedding alternatives to overcome the challenges posed by user-generated texts, which replace the limiting discrete approaches seen until now.

Part IV

Replacing preprocessing with embeddings

Chapter 7

Cross-lingual word embeddings

Word embeddings are one of the most widely-used resources in NLP, capable of modelling relevant morphological, syntactical, and semantic features of words without any form of human supervision. In the beginning, all efforts went towards obtaining good monolingual embeddings, which encoded information about words in a particular language, hence obtaining isolated models for each language. More recently, however, a popular research direction has been broadening the coverage of embedding models from one to multiple languages integrated into the same embedding space, extending the properties found in monolingual models to a multilingual domain. The resulting cross-lingual embeddings do not only play a central role in multilingual NLP tasks and knowledge transfer from resource-rich languages (typically English); most interestingly for us, their properties make them suitable for tackling the context fragmentation problem present in code-switched texts, as already discussed in Section 6.1.

In general, there exist two approaches to obtain multilingual models: training a multilingual model from scratch (Luong et al., 2015; Wick et al., 2016) or training first various monolingual models and then integrating them into a common multilingual space (Mikolov et al., 2013a; Artetxe et al., 2016). Given the significant amount of recent work focusing on the latter approach and its greater degree of modularity, in this chapter we describe a method to improve the integration obtained by these cross-lingual alignments.

In order to align monolingual word embedding spaces, some state-of-the-art models rely on orthogonal transformations that map monolingual embeddings from a source language into the embedding space of a target language while enforcing monolingual invariance; i.e., the internal structure of the monolingual spaces is preserved during the transformation. The alignment is guided by a collection of word translations serving as anchor points gathered in a bilingual dictionary, which may be obtained manually or through automated means (Conneau et al., 2018a; Artetxe et al., 2018b). Overall, this can be considered an

adequate solution if we assume that the embedding spaces for different languages are approximately isometric (Barone, 2016), although it has been argued that this is not always the case (Søgaard et al., 2018; Kementchedjheva et al., 2018). Yet, and perhaps surprisingly, orthogonal mappings outperform more flexible linear and non-linear methods in many multilingual and monolingual evaluation tasks (Artetxe et al., 2016), showing that the internal monolingual structure of an embedding space is, in fact, a highly relevant property. As a consequence, finding an unconstrained linear or non-linear mapping which optimizes both monolingual and multilingual performance without degrading the former does not seem to be a trivial task. In principle, this would prevent us from exploiting cross-lingual information to improve monolingual performance any further, which should be achievable according to a previous work by Faruqi and Dyer (2014).

In this chapter, we propose using an unconstrained linear transformation to refine cross-lingual alignments which, in principle, allows us to combine the advantages of orthogonal transformations with the potential benefit of allowing monolingual spaces to mutually affect their internal structures. In reality, our method can be applied over any form of cross-lingual alignment, although the most notable gains and complete range of benefits are obtained when operating over orthogonal transformations, as we will see below. Our experimental analysis reveals that, in general, this combination of an initial transformation followed by a fine-tuning step outperforms the base approaches in both multilingual and monolingual evaluation tasks. The performance increase in the case of monolingual tasks indicates that the proposed transformation leads to improvements in the monolingual spaces, which, as already mentioned, is not possible when using orthogonal mappings alone.

Furthermore, our method generalizes easily to an arbitrary number of languages, thus learning truly multilingual vector spaces which extend the benefits of the bilingual case considered in most previous work. Nonetheless, we have also carried out a broad analysis of the factors usually involved in bilingual alignments using the methods described here, which we leave for Appendix B.

7.1 Fine-tuning by meeting in the middle

The method we are proposing is included in a 3-step framework composed of (1) a word embedding model such as word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), or fastText (Bojanowski et al., 2016) that obtains the monolingual embedding spaces for each language, independently of one another; (2) a bilingual method to obtain the initial alignment of the monolingual spaces; and (3) our own method which improves the integration of the monolingual spaces obtained in the previous step. The initial bilingual alignment of monolingual spaces is usually modelled as an orthogonal linear transformation (Artetxe

et al., 2016; Conneau et al., 2018a; King et al., 2015), where a matrix \mathbf{W} is learned such that it minimizes the following objective:¹

$$Q = \sum_{i=1}^n \|\mathbf{x}_i \mathbf{W} - \mathbf{z}_i\|^2 \quad (7.1)$$

where we write \mathbf{x}_i for the vector representation of some word x_i in the source language and \mathbf{z}_i is the vector representation of the translation z_i of w_i in the target language. When we add the orthogonality constraint $\mathbf{W}\mathbf{W}^T = \mathbf{1}$, the resulting optimization problem is known as the *orthogonal Procrustes problem*, whose exact solution can be computed efficiently, similarly to any regular least-squares regression problem. Note that this approach relies on a bilingual dictionary containing the training pairs $(x_1, z_1), \dots, (x_n, z_n)$. However, once the matrix \mathbf{W} has been learned, for any word w in the source language, we can use $\mathbf{x}\mathbf{W}$ as a prediction of the vector representation of the translation of w . In particular, to predict which word in the target language is the most likely translation of the word w from the source language, we can then simply take the word z whose vector \mathbf{z} is closest to the prediction $\mathbf{x}\mathbf{W}$. In this work, we use two well-known methods to obtain the bilingual alignments: VecMap (Artetxe et al., 2018a) and MUSE (Conneau et al., 2018a).

The third step in our multilingual integration framework constitutes our main contribution. After the initial bilingual alignment described in the previous paragraph, we propose to apply a post-processing step which aims at improving the integration of the resulting bilingual space. To this end, we learn an unconstrained linear transformation that maps word vectors from one space onto the average of that word vector and the vector representation of its translation according, once again, to a given bilingual dictionary. We call this approach *Meemi (Meeting in the middle)*,² which is illustrated in Figure 7.1. In particular, the figure shows the three-step nature of our approach: (1) we first obtain the monolingual spaces; (2) we then learn an initial transformation (in this case, orthogonal, using VecMap or MUSE), which aligns the two monolingual spaces as much as possible without changing their internal structure; and (3) finally, we map both spaces into a middle ground between them through a non-orthogonal linear transformation. Since we already start this step from aligned spaces, the changes applied by this transformation are expected to be relatively small. Notably, this approach can naturally be applied to more than two monolingual spaces given some specific conditions, as we will see in Section 7.1.2. But first, let us consider the traditional bilingual case.

¹We will see shortly how we can obtain truly multilingual models starting with bilingual alignments.

²Code is available at <https://github.com/yeraidm/meemi>. This page will be updated with pre-trained models for new languages and tailored code to apply our models in cross-lingual tasks.

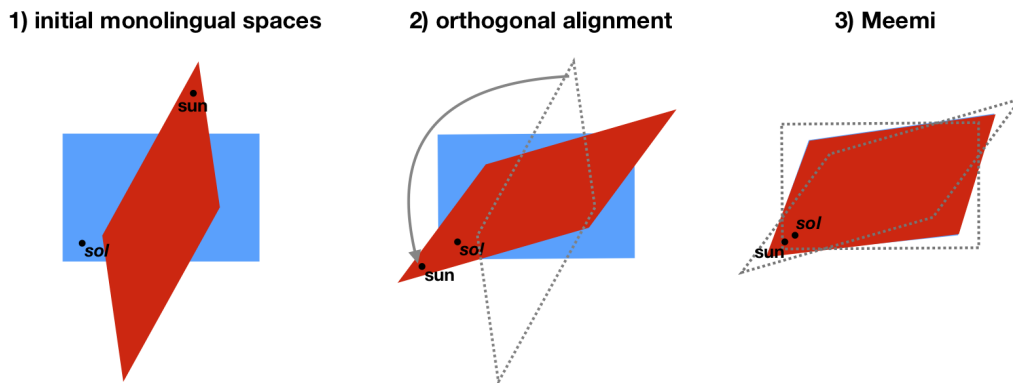


Figure 7.1: Step by step integration of two monolingual embedding spaces: (1) obtaining isolated monolingual spaces, (2) aligning these spaces through an orthogonal linear transformation, and (3) map both spaces using an unconstrained linear transformation learned on the averages of translation pairs.

7.1.1 Bilingual models

Let D be the given bilingual dictionary, encoded as a set of word pairs (w, w') . Using the pairs in D as training data, we learn a linear mapping \mathbf{X} such that $\mathbf{w}\mathbf{X} \approx \frac{\mathbf{w} + \mathbf{w}'}{2}$ for all $(w, w') \in D$, where we write \mathbf{w} for the vector representation of word w in the given (aligned) monolingual space. This mapping \mathbf{X} can then be used to predict the averages for words outside the given dictionary. To find the mapping \mathbf{X} , we solve the following least squares linear regression problem:

$$E_{bi} = \sum_{(w, w') \in D} \left\| \mathbf{w}\mathbf{X} - \frac{\mathbf{w} + \mathbf{w}'}{2} \right\|^2 \quad (7.2)$$

Similarly, we separately learn a mapping \mathbf{X}' such that $\mathbf{w}'\mathbf{X}' \approx \frac{\mathbf{w} + \mathbf{w}'}{2}$.

It is worth pointing out that we experimented with several variants of this linear regression formulation. For example, we also tried using a multilayer perceptron to learn non-linear mappings, and we experimented with several regularization terms to penalize mappings that deviate too much from the identity mapping. None of these variants, however, were found to improve on the much simpler formulation in Equation 7.2, which can be solved exactly and efficiently. Furthermore, one may wonder whether the initial alignment is actually needed, since e.g., Coates and Bollegala (2018) obtained high-quality meta-embeddings without such an alignment set. However, when applying our approach directly to the initial monolingual non-aligned embedding spaces, we did not obtain competitive

results in our current setting.

On the other hand, we do consider a weighted variant of Meemi where the linear model is trained on weighted averages based on word frequency. Specifically, let f_w be the occurrence count of word w in the corresponding monolingual corpus, then $\frac{w+w'}{2}$ is replaced by:

$$\frac{f_w \mathbf{w} + f_{w'} \mathbf{w}'}{f_w + f_{w'}} \quad (7.3)$$

The intuition behind this weighted model is that the embedding for the word w might be more relevant (i.e., encode the underlying concept more effectively) than the one for w' , given that it is more frequent in its training corpus compared to the other, or the reverse. Suppose, for instance, that $w = w'$, and w is the name of a Spanish city. Then, we may expect to see more occurrences of w in a Spanish corpus than in an English corpus, and hence the training algorithm could use more context information to refine the Spanish embedding.

7.1.2 Multilingual models

To apply Meemi in a multilingual setting, we exploit the fact that bilingual orthogonal methods such as VecMap (without re-weighting) and MUSE do not modify the target monolingual space but only apply the transformation to the source. Hence, by simply applying this method to multiple language pairs while fixing the target language (i.e., for languages l_1, l_2, \dots, l_n , we construct pairs of the form (l_i, l_n) with $i \in \{1, \dots, n-1\}$), we can obtain a multilingual space in which all of the corresponding monolingual models are aligned with the same target embedding space. And this is truly the central requirement of this multilingual approach: that the initial mapping allows us to map all the corresponding monolingual spaces into a common shared multilingual space. Note, however, that if we applied a re-weighting strategy, as suggested by Artetxe et al. (2018a) for VecMap, the target space would no longer remain fixed for all source languages and would instead change depending on the source considered in each case.

While most previous work has been limited to bilingual formulations, multilingual models involving more than two languages have already been studied by Ammar et al. (2016). They use an approach based on Canonical Correlation Analysis (CCA) which also designates one specific language as reference for all the mappings. Having said this, we decided to use the bilingual approaches of VecMap and MUSE given their current state-of-the-art consideration.

Formally, let D be a multilingual dictionary, encoded as a set of tuples (w_1, w_2, \dots, w_n) , where n is the number of languages. Using the tuples in D as training data, we learn a

linear mapping \mathbf{X}_i for each language, such that $\mathbf{w}_i \mathbf{X}_i \approx \frac{\mathbf{w}_1 + \dots + \mathbf{w}_n}{n}$ for all $(w_1, \dots, w_n) \in D$. This mapping \mathbf{X}_i can then be used to predict the averages for words in the i th language outside the given dictionary. To find the mappings \mathbf{X}_i , we solve the following least squares linear regression problem for each language:

$$E_{multi} = \sum_{(w_1, \dots, w_n) \in D} \left\| \mathbf{w}_i \mathbf{X}_i - \frac{\mathbf{w}_1 + \dots + \mathbf{w}_n}{n} \right\|^2 \quad (7.4)$$

Note that while a weighted variant of this model can straightforwardly be formulated, we will not consider this in the following experiments.

7.2 Experimental setting

In this section we explain the common training settings for all experiments. Firstly, the corpora and other training details used when learning the initial monolingual embeddings are presented in Section 7.2.1. Then, the bilingual and multilingual dictionaries which serve as supervision signals are explained in Section 7.2.2. Finally, all compared systems are listed in Section 7.2.3.

7.2.1 Corpora and monolingual embeddings

Instead of using comparable corpora such as Wikipedia, as in the majority of the previous work (Artetxe et al., 2017; Conneau et al., 2018a), we make use of independent corpora extracted from the Web. This represents a more realistic setting where alignments are harder to obtain, as already noted by Artetxe et al. (2018b). For English, we use the 3B-word UMBC WebBase Corpus (Han et al., 2013b), containing over 3 billion words. In the case of Spanish, we use the Spanish Billion Words Corpus (Cardellino, 2016), consisting of over a billion words. For Italian and German, we use the itWaC and sdeWaC corpora from the WaCky project (Baroni et al., 2009), containing 2 and 0.8 billion words, respectively.³ For Finnish and Russian, we use their corresponding Common Crawl monolingual corpora from the Machine Translation of News Shared Task 2016,⁴ composed of 2.8 and 1.1 billion words, respectively. Finally, for Farsi we leverage the newswire Hamshahri corpus (AleAhmad et al., 2009), composed of almost 200 million words.

³The same English, Spanish, and Italian corpora are used as input corpora for the hypernym discovery SemEval task (see Section 7.4.1).

⁴<http://www.statmt.org/wmt16/translation-task.html>

In a preprocessing step, all corpora were tokenized using the Stanford tokenizer (Manning et al., 2014) and lowercased. Next, we trained fastText word embeddings (Bojanowski et al., 2016) on the preprocessed corpora for each language. The dimensionality of the vectors was set to 300, using the default values for the remaining hyperparameters.

7.2.2 Training dictionaries

We use the training dictionaries provided by Conneau et al. (2018a) as supervision. These bilingual dictionaries were compiled using the internal translation tools from Facebook. To make the experiments comparable across languages, we randomly extracted 8,000 training pairs from these splits for all language pairs considered, as this is the size of the smallest available dictionary. For completeness we also present results for fully unsupervised systems (see the following section), which do not take advantage of any dictionaries.

7.2.3 Compared systems

In the bilingual case, we use the supervised and unsupervised variants of VecMap and MUSE to obtain the base alignments and then apply *plain* Meemi and weighted Meemi on the results. For VecMap, we consider both its orthogonal version (VecMap_{ortho}) (Artetxe et al., 2016) and the non-orthogonal multi-step procedure (VecMap_{multistep}) (Artetxe et al., 2018a). It is worth noting that the unsupervised version of VecMap (VecMap_{uns}) (Artetxe et al., 2018b) is based on the non-orthogonal procedure, while both variants of MUSE (MUSE and MUSE_{uns}, supervised and unsupervised, respectively) (Conneau et al., 2018a) are based on orthogonal transformations.

The multilingual case is formed exclusively by Meemi models, where we follow the procedure described in Section 7.1.2 for all seven languages considered: English (EN), Spanish (ES), Italian (IT), German (DE), Finnish (FI), Farsi (FA), and Russian (RU). Note that while in the bilingual case we can use any VecMap variant, in the multilingual setting we can only use the orthogonal methods, this is, VecMap_{ortho}, MUSE, and MUSE_{uns}, although we only consider the former at the moment.

Finally, we will write *Meemi* (M) to refer to the model obtained by applying Meemi after the base method M , where M may be any variant of VecMap or MUSE. Similarly, we will write Meemi_w (M) in those cases where the weighted version of Meemi is used, and Meemi-multi for the multilingual models.

7.3 Intrinsic evaluation

In this section we assess the intrinsic performance of our post-processing techniques in cross-lingual and monolingual settings.

7.3.1 Cross-lingual performance

We evaluate the performance of all compared cross-lingual embedding models on standard purely cross-lingual tasks, namely dictionary induction and cross-lingual word similarity.

Model	English-Spanish			English-Italian			English-German		
	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$
VecMap _{uns}	34.8	60.6	67.0	31.4	53.7	60.7	23.2	42.7	50.2
MUSE _{uns}	31.4	51.2	57.7	31.4	51.2	57.7	20.8	38.7	46.6
VecMap _{ortho}	32.6	58.1	65.8	32.9	56.5	63.4	22.8	42.8	50.4
Meemi (VecMap _{ortho})	33.9	60.7	67.4	33.8	58.8	65.6	23.7	45.0	52.9
Meemi _w (VecMap _{ortho})	33.4	60.9	67.4	33.1	58.5	66.3	22.9	44.3	52.5
Meemi-multi (VecMap _{ortho})	33.4	60.9	67.1	33.7	58.1	65.5	23.0	44.5	52.8
VecMap _{multistep}	33.8	60.7	68.4	33.7	58.9	66.5	24.1	45.3	53.6
Meemi (VecMap _{multistep})	33.8	61.4	68.4	33.7	59.0	66.8	23.4	45.7	53.6
Meemi _w (VecMap _{multistep})	33.2	60.9	68.1	32.5	58.2	66.2	22.8	44.8	53.1
MUSE	32.5	58.2	65.9	32.5	56.0	63.2	22.4	40.9	48.9
Meemi (MUSE)	33.9	60.7	68.4	33.8	58.4	65.6	23.7	45.3	52.3
Meemi _w (MUSE)	33.3	61.2	68.2	33.0	58.8	65.3	22.8	44.4	52.3

Model	English-Finnish			English-Farsi			English-Russian		
	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$
VecMap _{uns}	0.1	0.5	0.7	19.7	34.6	40.4	13.8	30.9	38.6
MUSE _{uns}	23.7	45.0	52.9	18.1	32.8	37.8	14.4	31.2	38.5
VecMap _{ortho}	22.1	44.5	52.9	18.5	33.6	40.5	15.6	35.5	44.2
Meemi (VecMap _{ortho})	24.8	48.9	57.7	20.0	37.1	43.8	19.0	40.5	49.9
Meemi _w (VecMap _{ortho})	22.6	48.3	56.5	19.8	35.2	41.6	17.4	39.9	49.4
Meemi-multi (VecMap _{ortho})	23.1	48.3	57.2	21.0	37.9	44.4	18.8	41.7	50.5
VecMap _{multistep}	22.5	48.4	57.5	20.8	36.1	43.4	18.2	40.2	49.5
Meemi (VecMap _{multistep})	24.0	50.8	58.9	20.0	36.9	42.4	19.3	41.5	50.6
Meemi _w (VecMap _{multistep})	21.6	48.3	57.2	21.5	38.5	43.7	17.4	40.9	49.7
MUSE	20.0	40.1	48.3	17.4	31.6	37.6	15.5	35.6	44.1
Meemi (MUSE)	23.0	46.1	54.0	19.3	36.0	41.7	18.7	40.5	49.7
Meemi _w (MUSE)	21.7	46.9	55.0	19.5	33.8	39.8	18.1	40.0	49.5

Table 7.1: $P@k$ performance of different cross-lingual embedding models in the bilingual dictionary induction task.

Bilingual dictionary induction

Also referred to as *word translation*, this task consists in automatically retrieving the word translations in a target language for words in a source language. Acting on the corresponding cross-lingual embedding space which integrates the two (or more) languages of a particular test case, we obtain the nearest neighbors to the source word in the target language as our translation candidates. The performance is measured with *precision at k* ($P@k$), the proportion of test instances where the correct translation candidate for a given source word was among the k highest ranked candidates. The nearest neighbors ranking is obtained by using cosine similarity as the scoring function. For this evaluation we use the corresponding test dictionaries released by Conneau et al. (2018a).

Table 7.1 shows the results attained by a wide array of models. We can observe that the best figures are generally obtained by Meemi over the bilingual VecMap models. Unsurprisingly, the impact of Meemi is more apparent when used in combination with the orthogonal base models. On the other hand, using the weighted version of Meemi (Meemi_w) does not seem to be particularly beneficial on this task, with the only exception of English-Farsi. In general, the performance of unsupervised models (VecMap_{uns} and MUSE_{uns}) is competitive in closely-related languages such as English-Spanish and English-German but considerably under-perform for distant languages, especially on English-Finnish and English-Russian.

Finally, the results obtained by the multilingual model that includes all seven languages considered, i.e., Meemi-multi (VecMap_{ortho}), improve over the base orthogonal model, but they do not improve over the results of our bilingual model. We further discuss the impact of adding languages to the multilingual model in Section 7.5.2.

Cross-lingual word similarity

This task consists in measuring the semantic similarity between pairs of words and comparing the obtained scores to a gold standard given by human annotators. In a word embedding space, the similarity between two words can be measured through a distance or similarity metric between the corresponding vectors in that space, such as cosine similarity. In this case, and in contrast to monolingual similarity (see below), words in a given pair (a, b) belong to different languages; e.g., a may belong to English and b to Farsi, which in turn happen to be integrated into the same cross-lingual space. For this task, we make use of the SemEval-17 multilingual similarity benchmark (Camacho Collados et al., 2017), considering the four cross-lingual datasets that include English as target language, but discarding multi-word expressions. Performance is computed in terms of Pearson and Spearman correlation with respect to the gold standard.

Model	EN-ES		EN-IT		EN-DE		EN-FA	
	r	ρ	r	ρ	r	ρ	r	ρ
VecMap _{uns}	71.1	70.5	69.2	68.8	70.9	70.4	35.7	33.4
MUSE _{uns}	71.7	71.6	69.4	69.4	70.3	70.0	29.6	23.8
VecMap _{ortho}	71.6	71.6	70.2	70.1	70.9	70.7	29.2	23.7
Meemi (VecMap _{ortho})	72.3	72.0	71.2	70.7	72.5	72.1	35.3	31.6
Meemi _w (VecMap _{ortho})	72.1	72.0	70.0	69.7	70.5	70.2	34.2	30.2
Meemi-multi (VecMap _{ortho})	73.9	73.4	71.6	71.0	72.5	72.2	39.6	37.2
VecMap _{multistep}	72.8	72.4	71.6	71.2	72.7	72.2	36.5	31.7
Meemi (VecMap _{multistep})	72.1	71.5	71.1	70.9	72.6	72.3	40.4	39.0
Meemi _w (VecMap _{multistep})	71.5	71.2	69.7	69.8	70.3	70.3	39.6	40.8
MUSE	71.9	71.9	70.4	70.4	70.5	70.2	29.7	23.9
Meemi (MUSE)	72.5	72.3	71.5	71.1	72.5	72.1	36.4	33.0
Meemi _w (MUSE)	72.3	72.2	70.4	70.0	70.5	70.4	33.6	28.9

Table 7.2: Cross-lingual word similarity results in terms of Pearson (r) and Spearman (ρ) correlation.

As we can see in Table 7.2, our fine-tuned models clearly outperform the original orthogonal mappings of VecMap and MUSE, and also their unsupervised variants. Furthermore, Meemi-multi (VecMap_{ortho}) proves a consistently good performance across the board, being superior to its bilingual counterpart while tying with VecMap_{multistep} for Italian and German. On the other hand, and similarly to the dictionary induction task, Meemi seems to be essential when considering distant languages, as is the present case with Farsi, where Meemi (VecMap_{multistep}) obtains almost four more points than its competitive base model (going from 36.5% to 40.4%). This result is not entirely surprising, as with more similar languages we could expect a lesser impact in the transformation performed by Meemi.

7.3.2 Monolingual performance

As already mentioned at the beginning of this chapter, orthogonal transformations do not modify the internal structure of monolingual embedding spaces. For this reason, they cannot exploit the integration of cross-lingual information to improve the monolingual structures, as noted by Faruqui and Dyer (2014), and thus their performance will not change in monolingual tasks. However, methods that break this orthogonality constraint, such as Meemi and VecMap (with re-weighting), should report different performance figures in these cases. In order to test this, we use the monolingual word similarity task, where we measure the semantic similarity between two words from the same language. Similarly

Model	English		Spanish		Italian		German		Farsi	
	r	ρ	r	ρ	r	ρ	r	ρ	r	ρ
VecMap _{uns}	72.8	72.3	70.2	70.4	67.8	68.1	70.6	70.2	23.5	21.1
MUSE _{uns}	74.2	74.2	70.5	71.9	67.4	69.2	69.8	69.8	21.1	17.3
VecMap _{ortho}	74.1	73.9	70.0	71.5	67.2	69.0	70.1	70.1	21.1	18.2
Meemi (VecMap _{ortho})	74.4	73.9	71.6	72.1	69.0	69.4	71.1	70.7	24.3	22.5
Meemi _w (VecMap _{ortho})	74.4	74.0	71.8	71.8	68.2	68.8	68.8	68.9	28.5	29.8
Meemi-multi (VecMap _{ortho})	75.1	74.3	73.0	72.9	70.1	70.4	70.7	70.7	27.3	26.0
VecMap _{multistep}	73.8	73.3	71.8	72.0	69.6	69.7	71.8	71.2	24.8	22.2
Meemi (VecMap _{multistep})	73.3	72.6	71.7	71.6	69.4	69.8	71.1	71.0	27.3	26.2
Meemi _w (VecMap _{multistep})	73.5	72.9	70.9	70.6	67.2	68.4	67.0	67.8	27.3	25.6
MUSE	74.2	74.2	70.5	71.9	67.4	69.2	69.8	69.8	21.1	17.3
Meemi (MUSE)	74.6	74.1	71.9	72.4	69.5	69.9	71.0	70.6	24.6	22.5
Meemi _w (MUSE)	74.5	74.4	71.7	71.8	68.5	68.9	68.3	68.2	27.0	25.5
fastText	72.3	72.4	69.0	70.2	66.3	67.5	71.0	70.3	24.3	20.6
Human upper bound	<i>89.3</i>	-	<i>89.0</i>	-	<i>90.0</i>	-	<i>91.6</i>	-	<i>90.6</i>	-

Table 7.3: Monolingual word similarity results in terms of Pearson (r) and Spearman (ρ) correlation.

to the cross-lingual case, the evaluation is performed in terms of Spearman and Pearson correlation with respect to human judgements, and we use the monolingual datasets from the SemEval-17 task (English, Spanish, German, and Farsi). The results provided by the original monolingual fastText embeddings are also reported as baseline.

Table 7.3 shows the results on monolingual word similarity, where our multilingual model clearly stands out obtaining the best overall results for English, Spanish, and Italian, and improving over the base VecMap_{ortho} model on the rest. With the sole exception of German, where the multi-step framework of Artetxe et al. (2018a) proves effective, the plain Meemi transformation outperforms the base models, both for VecMap and MUSE.

7.4 Extrinsic evaluation

We complement the intrinsic evaluation experiments, which are typically a valuable source for understanding the properties of the vector spaces, with downstream cross-lingual tasks. For the remainder of this section, we will focus on the orthogonal model of VecMap (i.e., VecMap_{ortho}), in combination with the proposed Meemi strategy, both in bilingual and multilingual settings. For the latter case, we consider all six languages: Spanish, Italian, German, Finnish, Farsi, and Russian; while keeping English as the target language.

The tasks considered are cross-lingual hypernym discovery and cross-lingual natural language inference.

7.4.1 Cross-lingual hypernym discovery

Hypernymy is an important lexical relation with a direct impact on downstream NLP tasks such as semantic search (Hoffart et al., 2014; Roller and Erk, 2016), question answering (Prager et al., 2008; Yahya et al., 2013), or textual entailment (Geffet and Dagan, 2005). In addition, hypernyms are the backbone of taxonomies and lexical ontologies (Yu et al., 2015), which are in turn useful for organizing, navigating, and retrieving online content (Bordea et al., 2016). Here, we propose to evaluate the quality of the considered cross-lingual vector spaces in the extrinsic task of bilingual hypernym discovery, i.e., given an input word in a given language (e.g., “cat”), retrieve or discover its most likely set of valid hypernyms in another language (e.g., “animal”, “mamífero”, “felino”, etc.).⁵ Intuitively, by leveraging a bilingual vector space condensing the semantics of two languages, one of them being English, the need for large amounts of training data in the target language may be reduced.

The base model for this task is a linear transformation trained on English hyponym-hypernym pairs (Espinosa-Anke et al., 2016), which is afterwards used to predict the most likely set of hypernyms given a new term. Training and evaluation data come from the SemEval 2018 Shared Task on Hypernym Discovery (Camacho-Collados et al., 2018). Note that current state-of-the-art systems aimed at modelling hypernymy (Shwartz et al., 2016; Bernier-Colborne and Barriere, 2018) combine large amounts of annotated data along with language-specific rules and cue phrases such as Hearst Patterns (Hearst, 1992), both of which are generally scarcely available, if at all, for languages other than English. We also consider augmenting our models with extra data from the target language in each case.

The results shown in Table 7.4 correspond to the models trained on English data only (11,779 hyponym-hypernym pairs), and then those augmented models informed with relatively few training pairs (500, 1K, and 2K) from the target languages. Evaluation is conducted with the same metrics as in the original SemEval task: Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), and Precision at 5 ($P@5$). These measures explain the behavior of a model from complementary prisms, namely how often at least one valid hypernym was highly ranked (MRR), and in cases where there is more than one correct hypernym, to what extent they were all correctly retrieved (MAP and $P@5$).

The first noticeable trend is the better performance of the unsupervised VecMap version versus its orthogonal counterpart. Nevertheless, we obtain consistent gains over both VecMap variants when applying Meemi, across all configurations for the two language pairs considered. In fact, the weighted version (Meemi_w) brings an increase in performance between 4 and 5 MRR points and between 1 and 2 MAP in the English-only training setting

⁵Spanish translations for “animal”, “mammal”, and “feline”, respectively.

Train data	Model	Spanish			Italian		
		MAP	MRR	P@5	MAP	MRR	P@5
EN	VecMap _{uns}	5.8	15.0	5.5	5.2	12.4	4.8
	VecMap	5.3	13.6	5.2	3.9	9.0	3.6
	Meemi (VecMap)	6.5	16.8	6.1	5.5	12.7	5.3
	Meemi _w (VecMap)	7.7	20.6	7.2	7.0	17.2	6.4
	Meemi-multi (VecMap)	5.9	15.4	5.5	6.0	13.9	5.7
EN + 500	VecMap _{uns}	5.9	14.5	5.7	5.9	14.1	5.6
	VecMap	5.6	13.9	5.6	4.9	11.5	4.7
	Meemi (VecMap)	7.0	17.6	6.7	6.2	14.2	5.9
	Meemi _w (VecMap)	7.7	20.8	7.2	7.3	17.4	6.8
	Meemi-multi (VecMap)	6.1	15.8	6.0	6.4	14.8	6.1
EN + 1K	VecMap _{uns}	6.3	15.1	6.0	5.9	13.9	5.6
	VecMap	5.7	14.4	5.6	5.2	12.6	4.9
	Meemi (VecMap)	7.1	17.8	6.7	6.6	15.1	6.4
	Meemi _w (VecMap)	7.9	21.2	7.4	7.3	17.3	6.9
	Meemi-multi (VecMap)	6.4	16.5	6.2	6.5	14.6	6.2
EN + 2K	VecMap _{uns}	6.2	14.2	6.1	6.5	15.1	6.1
	VecMap	5.7	14.0	5.4	6.0	14.0	5.7
	Meemi (VecMap)	7.0	17.2	6.7	7.1	16.1	6.8
	Meemi _w (VecMap)	7.7	20.2	7.2	7.4	17.3	7.2
	Meemi-multi (VecMap)	6.4	16.1	6.2	6.8	15.4	6.4

Table 7.4: Cross-lingual hypernym discovery results. In this case: VecMap = VecMap_{ortho} .

for Spanish and Italian, respectively. This is in contrast to the intrinsic evaluation, where the weighted model did not seem to provide noticeable improvements over the *plain* version of Meemi. Finally, concerning the fully multilingual model, the experimental results suggest that, while still better than the orthogonal baselines, it falls short when compared to the weighted bilingual version of Meemi. This result suggests that exploring weighting schemes for the multilingual setting may bring further gains, but we leave this for future work.

7.4.2 Cross-lingual natural language inference

The task of natural language inference (NLI) consists in detecting entailment, contradiction, or neutral relations in pairs of sentences. In our case, we test a *zero-shot* cross-lingual transfer setting (XNLI) where a system is trained with English corpora and is then evaluated on a different language.

Model	EN-ES	EN-DE
VecMap _{uns}	45.5	44.4
VecMap _{ortho}	43.9	43.6
Meemi (VecMap _{ortho})	44.9	43.8
Meemi _w (VecMap _{ortho})	40.4	43.5
Meemi-multi (VecMap _{ortho})	46.6	45.5
VecMap _{multistep}	44.4	37.7
Meemi (VecMap _{multistep})	44.2	43.2
Lower bound	38.0	33.4

Table 7.5: Accuracy on the XNLI task using different cross-lingual embeddings as features.

We base our approach on the assumption that better aligned cross-lingual embeddings should lead to better XNLI models, and that the impact of the input embeddings may become more apparent in simple methods; as opposed to, for instance, complex neural network architectures. Hence, and also to account for the coarser linguistic granularity of this task (being a sentence classification problem rather than word-level), we employ a simple bag-of-words approach where a sentence embedding is obtained through word vector averaging.

We then train a linear classifier⁶ to predict one of the three possible labels in this task, namely *entailment*, *contradiction*, or *neutral*. We use the full MultiNLI English corpus (Williams et al., 2018) for training and the Spanish and German test sets from XNLI (Conneau et al., 2018b) for testing. For comparison, we also include a lower bound obtained by considering English monolingual embeddings for input; in this case fastText trained on the UMBC corpus, which is the same model used to obtain multilingual embeddings.

Accuracy results are shown in Table 7.5. The main conclusion in light of these results is the remarkable performance of the unsupervised VecMap model and, most notably, multilingual Meemi for both Spanish and German, clearly outperforming the orthogonal and multistep bilingual mapping baselines. Our results are encouraging for two reasons. First, they suggest that, at least for this task, collapsing several languages into a unified vector space is better than performing pairwise alignments. And second, the inherent benefit of having one single model accounting for an arbitrary number of languages.

⁶The codebase for these experiments is that of SentEval (Conneau and Kiela, 2018).

crazy			telegraph		
VecMap	Meemi	Meemi-multi	VecMap	Meemi	Meemi-multi
loco	loco	chifladas	telégrafo	telegráfico	telegraph
tonto	loca	locos	telégrafos	telégrafo	telegraaf
enloquecere	enloquecí	loca	telegráfico	telegráfico	telegraphone
locos	enloquecías	estúpidas	telegráfica	telegraf	telegráfico
enloqueci	locos	alocadas	telegrafo	telegráfico	telégrafo
conventions			discover		
VecMap	Meemi	Meemi-multi	VecMap	Meemi	Meemi-multi
convenciones	internaciones	convenios	descubrirá	descubre	descubr
internacional7	1972naciones	reglas	descubr	descubrir	descubrirán
convención	protocolos	convención	descubrirán	descubriendo	descubrirnos
1961naciones	convenios	normas	descubren	descubra	descubrira
internacionales3	1961naciones	legislacionesnacionales	descubriron	descubrira	descubrire
remarks			lyon		
VecMap	Meemi	Meemi-multi	VecMap	Meemi	Meemi-multi
astrométricos	lobservaciones	observaciones	rocquigny	beaubois	marcigny
observacionales	mediciones	observacionales	rémilly	bourgmont	lyon
astrométricas	lasobservaciones	observacional	martignac	marcigny	pierreville
astronómicas	deobservaciones	predicciones	beaubois	rémilly	jacquemont
predicciones	susobservaciones	mediciones	chambourcy	jacquemont	beaubois

Table 7.6: Word translation examples from English and Spanish, comparing VecMap with the bilingual and multilingual variants of Meemi. For each source word, we show its five nearest cross-lingual synonyms. Bold translations are correct, according to the source test dictionary (see Section 7.3.1).

7.5 Analysis

We complement our quantitative evaluation, covering intrinsic and extrinsic tasks, with a qualitative analysis which aims at discovering the most salient properties of the transformation performed by Meemi and their linguistic implications. We perform this analysis using the examples in Section 7.5.1 and different combinations of languages in the case of the multilingual model in Section 7.5.2.

7.5.1 Studying word translations

Table 7.6 lists a number of examples where, for a source English word, we explore its highest ranked *cross-lingual synonyms* (or word translations) in a target language. We select Spanish as a use case.

Let us study the examples listed in Table 7.6, as they constitute illustrative cases of linguistic phenomena which go beyond correct or incorrect translations. First, the word “crazy” is correctly translated by both VecMap and Meemi; “loco” (masculine singular),

“locos” (masculine plural), or “loca” (femenine) being standard translations, with no further connotations, of the source word. However, the most interesting finding lies on the fact that for Meemi-multi, the preferred translation is a colloquial (or even vulgar) translation which was not considered as correct in the gold test dictionary. The Spanish word “chifladas” translates to English as “going mental” or “losing it”. Similarly, we would like to highlight the case of “telegraph”. This word has two major senses, namely that of a message transmitter, but also as a proper noun that refers to media outlets (several newspapers have the word “telegraph” in their name). VecMap and Meemi (correctly) translate this word into the common translation “telégrafo” (the transmission device), whereas Meemi-multi does prefer its named-entity sense.

Other cases, such as “conventions” and “discover” share the property of being common ambiguous nouns. A common outstanding pattern can be observed in that in both cases candidate translations are either misspellings of the correct translation (“descubr”, instead of “descubrir”, for “discover”) or misspellings involving tokens conflating two words whose compositional meaning is actually a correct candidate translation for the source word; e.g., “legislaciones nacionales” (“national rulings”) for “conventions”. Finally, the last example of this batch, the word “remarks”, is interesting because it allows us to interpret the semantics of candidate translations when ambiguity causes major disruptions. “Remark” translates in Spanish to “observación”, which in turn has an astronomical sense; “astronomical observatory” translates to “observatorio astronómico”.

7.5.2 Multilingual performance

In this section, we assess the benefits of our proposed multilingual integration (see Section 7.1.2). To this end, we measure fluctuations in performance as more languages are added to the initially bilingual model. Thus, starting from a bilingual embedding space obtained with VecMap_{Ortho}, we apply Meemi over a number of aligned spaces, which ultimately leads to a fully multilingual space containing the following languages: Spanish, Italian, German, Finnish, Farsi, Russian, and English. This latter language is used as the target embedding space for the orthogonal transformations due to it being the richest in terms of resource availability.

To avoid a lengthy and overly exhaustive approach where all possible combinations from two to seven languages are evaluated, we opted for conducting an experiment where languages are added one by one in a fixed order, starting from the languages which are closer to English in terms of language family and alphabet: Spanish, Italian, and German, and then Finnish, Farsi, and Russian. However, this approach does not allow us to use, for example, the English-Farsi test set until reaching the fifth step. To solve this, in these cases

Languages	English-Spanish			English-Italian			English-German		
	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$
x -en (VecMap _{ortho})	32.6	58.1	65.8	32.9	56.5	63.4	22.8	42.8	50.4
x -en	33.9	60.7	67.4	33.8	58.8	65.6	23.7	45.0	52.9
es- x -en	34.2	60.8	68.2	33.3	58.1	66.5	23.9	45.9	53.2
es-it- x -en	34.1	61.2	68.1	33.8	58.9	66.7	23.8	45.8	53.1
es-it-de- x -en	34.2	61.3	68.3	33.9	58.8	66.5	23.9	45.6	53.4
es-it-de-fi- x -en	33.6	60.9	67.5	33.8	58.0	65.8	23.1	44.7	52.7
es-it-de-fi-fa-ru-en	33.4	60.9	67.1	33.7	58.1	65.5	23.0	44.5	52.8
Languages	English-Finnish			English-Farsi			English-Russian		
	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$
x -en (VecMap _{ortho})	22.1	44.5	52.9	18.5	33.6	40.5	15.6	35.5	44.2
x -en	24.2	48.8	57.7	20.0	37.1	43.8	19.0	40.5	49.9
es- x -en	24.7	50.1	58.4	21.1	37.9	43.9	17.9	40.2	49.3
es-it- x -en	24.1	51.1	59.2	20.9	37.6	44.5	18.9	41.6	50.6
es-it-de- x -en	23.9	50.2	58.5	21.0	37.7	44.9	18.9	41.5	50.8
es-it-de-fi- x -en	23.5	48.6	57.5	21.2	37.5	44.0	19.1	42.1	51.4
es-it-de-fi-fa-ru-en	23.1	48.3	57.2	21.0	37.9	44.4	18.8	41.7	50.5

Table 7.7: Dictionary induction results obtained with multilingual Meemi over (VecMap_{ortho}). The sequence in which source languages are added to the multilingual models is: Spanish, Italian, German, Finnish, Farsi, and Russian (English is the target). The x indicates the use of an alternative model that includes the required language to evaluate on the specific dataset in each case. We also include the scores of the original VecMap_{ortho} as baseline.

we replace the next language in the sequence that would be considered by default with the language needed to run the corresponding test set. For instance, when adding Italian as the second source language, thus obtaining the model Spanish-Italian-English, and using it for the English-Spanish and English-Italian test sets, we also consider replacing it with either German, Finnish, Farsi, or Russian in order to allow the testing of a trilingual model on the rest of the test sets. In Table 7.7 we show the results obtained by the multilingual models in bilingual dictionary induction.

The best results are achieved when more than two languages are involved in the training, which correlates with the results obtained in the rest of the tasks and highlights the ability of Meemi to successfully exploit multilingual information to improve the quality of the embedding models involved. In general, the performance fluctuates more significantly when adding the first language to the bilingual models and then stabilizes at a similar level to the bilingual case when adding more distant languages.

7.6 Embedding models to replace language identification

Going back to the multilinguality concern in user-generated texts, one of the central points of this dissertation, we now have two options to overcome its derived difficulties: (1) discriminating texts based on the language(s) they are written in to process them with the adequate modules afterwards, which amounts to the task of language identification in a preprocessing pipeline as described in Chapter 1; or (2) directly using a cross-lingual embedding model which translates words in any of the considered languages into a common language of vectors, so that other NLP modules only have to deal with this *universal* language. As already discussed in Chapter 6, we lean towards the second approach due to its postulated advantages in our use case. However, we should mention here an added difficulty affecting the cross-lingual models obtained in this chapter, and which is caused by the existence of *cross-lingual homographs*. These are words that exist in multiple languages where they have the same or different meanings. For example, the term “probable” means ‘likely to occur’ in both languages; whereas the term “sensible” means ‘showing good sense’ in English but ‘sensitive’ in Spanish. Consequently, they also have multiple embedding representations in our cross-lingual embedding spaces, one per language where they are used. We consider two possible solutions to this:

1. Use word-sense disambiguation techniques (Navigli, 2009) to select the correct *language-sense* embedding in a given context. This might be viewed as a form of *soft* language identification, although it is worth noting that context fragmentation, one of the main limitations of the preprocessing approach that used *hard* language identification, is still solved within this approach. Then, having all the context available for disambiguation can make this task easier than traditional language identification.
2. Obtain unique word embeddings for homographs. Camacho-Collados et al. (2019) show that directly using the average of homograph embeddings yields surprisingly good results in multiple tasks. However, we have not seen the same trend in the results for the experimental setup in this chapter, which leads us to think that averaging might not be an adequate general solution.

Having said that, one could argue that focusing on other methods to obtain cross-lingual word embeddings which avoid the homograph problem altogether would have been preferable.

One such method is just training a regular embedding model on a concatenation of monolingual corpora in different languages, as is the case of multilingual BERT (Devlin et al., 2019).⁷ Going one step further, we can apply *artificial* code-switching to these

⁷<https://github.com/google-research/bert/blob/master/multilingual.md>

Spanish-English						
	DI			WS		XNLI
	$P@1$	$P@5$	$P@10$	r	ρ	
Meemi	33.9	60.7	67.4	72.3	72.0	44.9
CC	16.5	16.6	16.6	56.5	60.0	41.0
ACS	16.5	17.0	17.4	62.8	64.2	44.2
German-English						
	DI			WS		XNLI
	$P@1$	$P@5$	$P@10$	r	ρ	
Meemi	23.7	45.0	52.9	72.5	72.1	43.8
CC	16.0	16.7	16.9	59.9	62.4	33.7
ACS	16.0	17.8	19.5	59.6	59.4	39.5

Table 7.8: Results obtained by the Corpus Concatenation (CC) and Artificial Code-Switching (ACS) models in the usual test corpora for Dictionary Induction (DI), Word Similarity (WS), and Cross-lingual Natural Language Inference (XNLI). We include the figures from the bilingual Meemi ($\text{VecMap}_{\text{ortho}}$) model for comparison.

corpora where we replace words in a multilingual dictionary with their translations to pass the contextual information obtained for a given language to the rest (Luong et al., 2015; Wick et al., 2016). However, we have found that the performance of this approach across tasks is not promising when compared to the bilingual Meemi ($\text{VecMap}_{\text{ortho}}$), as we can see in Table 7.8. Furthermore, the training time of this type of models grows rapidly with the number of languages considered, especially when we induce context-switching; from minutes for a bilingual model to weeks for a 7-language model, rendering it impractical for us. Hence, in general, this approach has a more limited modularity compared to our current 3-step framework while obtaining lower performance figures.

As opposed to this, the alignment of monolingual spaces is a well-studied subject at this point (Artetxe et al., 2018a; Conneau et al., 2018a; Ammar et al., 2016; Mikolov et al., 2013a; Lu et al., 2015; Lazaridou et al., 2015) and is also a fine example of the benefits of modularity, which allows the complex problem of obtaining cross-lingual embeddings to be more easily studied and improved upon. Along this line, adding a new language to Meemi does not affect the initially aligned monolingual spaces and only requires four inexpensive steps: (1) obtaining the initial alignment for the corresponding new monolingual space, (2) re-calculating the means, (3) obtaining new linear mappings, and (4) applying them to the embedding spaces. This would also cover for introducing major changes to an already existing monolingual model, which could lead to a full re-training of that part of the multilingual space.

On the other hand, we do not consider those cross-lingual embedding approaches that require comparable or parallel corpora (Søgaard et al., 2015; Mogadala and Rettinger, 2016; Gouws et al., 2015; Luong et al., 2015). In an attempt to reduce human supervision as much as possible and improve adaptability. In this sense, we build on methods that can minimize the required external supervision signal and even bootstrap their own (Artetxe et al., 2017, 2018b; Conneau et al., 2018a), rendering them as unsupervised in practice.

Finally, one could argue that language identification is still necessary to gather monolingual corpora for the base embeddings used in our cross-lingual framework. While it is true that we need to discriminate the training data by language, the conditions under which we do this are far more advantageous than those in the usual language identification setting. For once, we may limit the data crawling to sources that we know produce text in a particular language or set of languages, including multilingual websites such as Wikipedia (hence its wide-spread use in multilingual tasks) or tweets written by specific users that are known to speak a certain language. The second advantage is that a high classification accuracy is no longer needed since misclassifying entire documents or being unable to properly handle code-switching are no longer big concerns. In fact, having some amount of language mixing could be even beneficial for the cross-lingual alignments (Camacho-Collados et al., 2019). By comparison, in a typical language identification scenario, we are forced to give the best answer possible to whatever input is provided.

7.7 Related work

The approach of aligning two isolated monolingual embedding spaces, obtained through the usual word embedding models such as word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), or fastText (Bojanowski et al., 2016), was popularized by Mikolov et al. (2013a), who showed that a high-quality alignment between two monolingual spaces can be obtained by simply learning a linear mapping from the word embedding space of the source language into the word embedding space of the target language.

Specifically, they proposed to learn a matrix \mathbf{W} which minimizes the objective shown in Equation 7.1, modelling this as a least-squares regression problem. The restriction to linear mappings might intuitively seem overly strict. However, it was found that higher-quality alignments can be found by being even more restrictive. In particular, Xing et al. (2015) suggested to normalize the word vectors in the monolingual spaces, and restrict the matrix \mathbf{W} to an orthogonal matrix (i.e., imposing the constraint that $\mathbf{W}\mathbf{W}^T = \mathbf{1}$). Another approach was taken by Faruqui and Dyer (2014), who proposed to learn linear transformations \mathbf{W}_s and \mathbf{W}_t , which respectively map vectors from the source and target language word embeddings onto a shared vector space. They used CCA to find the transformations \mathbf{W}_s

and \mathbf{W}_t which minimize the dimension-wise covariance between $\mathbf{X}\mathbf{W}_s$ and $\mathbf{Z}\mathbf{W}_t$, where \mathbf{X} is a matrix whose rows are $\mathbf{x}_1, \dots, \mathbf{x}_n$ and similarly \mathbf{Z} is a matrix whose rows are $\mathbf{z}_1, \dots, \mathbf{z}_n$.

Note that while the aim of Xing et al. (2015) is to avoid making changes to the cosine similarities between word vectors from the same language, Faruqui and Dyer (2014) specifically want to take into account information from the other language with the aim of improving the monolingual embeddings themselves. Artetxe et al. (2016) propose a model which combines ideas from Xing et al. (2015) and Faruqui and Dyer (2014). Specifically, they use the formulation in Equation 7.1 with the constraint that \mathbf{W} be orthogonal, as in Xing et al. (2015), but they also apply a preprocessing strategy called mean centering which is closely related to the model from Faruqui and Dyer (2014).

On top of this, in Artetxe et al. (2018a) they propose a multi-step framework in which they experiment with several pre-processing and post-processing strategies. These strategies include: (1) whitening, which involves applying a linear transformation to the word vectors such that their covariance matrix is the identity matrix; (2) re-weighting each coordinate according to its cross-correlation, which means that the relative importance of those coordinates with the strongest agreement between both languages is increased; (3) de-whitening, inverting the whitening step to restore the original covariances; and (4) dimensionality reduction, which is seen as an extreme form of re-weighting in which those coordinates with the least agreement across both languages are simply dropped. They also consider the possibility of using orthogonal mappings of both embedding spaces into a shared space, rather than mapping one embedding space onto the other, where the objective is based on maximizing cross-covariance.

Other approaches that have been proposed for aligning monolingual word embedding spaces include models which replace the objective shown in Equation 7.1 with a max-margin objective (Lazaridou et al., 2015) and models which rely on neural networks to learn non-linear transformations (Lu et al., 2015).

A central requirement of the aforementioned methods is that they need a sufficiently large bilingual dictionary. Several approaches have been proposed to address this limitation, showing that high-quality results can be obtained in a purely unsupervised way. For instance, Artetxe et al. (2017) propose a method that can work with a small synthetic seed dictionary; for example, a dictionary only containing pairs of identical numerals (1,1), (2,2), (3,3), etc. To this end, they alternately use the current dictionary to learn a corresponding orthogonal transformation and then use the learned cross-lingual embedding to improve the synthetic dictionary. This improved dictionary is constructed by assuming that the translation of a given word w is the nearest neighbor of $\mathbf{x}\mathbf{W}$ among all words from the target language.

The same authors subsequently improved their approach (Artetxe et al., 2018b), ob-

taining state-of-the-art results without even assuming the availability of a synthetic seed dictionary. The key idea underlying their approach, called VecMap, is to initialize the seed dictionary in a fully unsupervised way based on the idea that the histogram of similarity scores between a given word w and the other words from the source language should be similar to the histogram of similarity scores between its translation z and the other words from the target language.

Another approach which aims to learn bilingual word embeddings in a fully unsupervised way, called MUSE, is proposed in (Conneau et al., 2018a). The main difference with VecMap lies in how the initial seed dictionary is learned. For this purpose, MUSE relies on adversarial training (Goodfellow et al., 2014), similar as in earlier models (Barone, 2016; Zhang et al., 2017) but using a simpler formulation, based on the model of Equation 7.1 with the orthogonality constraint on \mathbf{W} . The main intuition is to choose \mathbf{W} such that it is difficult for a classifier to distinguish between word vectors \mathbf{z} sampled from the target word embedding and vectors $\mathbf{x}\mathbf{W}$, with \mathbf{x} sampled from the source word embedding.

7.8 Conclusions

In this chapter, we have presented Meemi, a post-processing method that improves the integration of cross-lingual embedding spaces previously obtained by aligning isolated monolingual spaces. Our initial goal was to improve the bilingual alignments obtained by state-of-the-art cross-lingual methods such as VecMap (Artetxe et al., 2018a) and MUSE (Conneau et al., 2018a). We do this by applying an unconstrained linear transformation on their results which is learned by mapping word translations in the constituent aligned monolingual spaces into their average representations. Notably, we went beyond the usual bilingual setting and showed how Meemi can be naturally extended to map embeddings for an arbitrary number of languages into a single shared vector space. In this case, we use orthogonal methods in the first alignment step of our framework which only transform the embedding space of the source language while leaving the target space intact, which becomes the multilingual embedding space.

Regarding the evaluation, we include not only the usual Indo-European languages such as English, Spanish, Italian, and German, but also other distant languages such as Finnish, Farsi, and Russian. The results obtained show that Meemi is able to improve the results achieved by the base methods, with significant gains when applied over orthogonal variants and also when considering distant languages. At the task level, we observed that: models based on VecMap_{multistep} stand out in dictionary induction, Meemi_w (VecMap) obtains impressive results in cross-lingual hypernym discovery, and Meemi-multi(VecMap_{ortho}) is remarkably good both at word similarity and cross-lingual natural language inference. In

light of these results, we are particularly encouraged by the multilingual models, which demonstrate that bringing together more than two languages in a shared vector space is highly beneficial in several cases.

With respect to tackling the multilinguality concern in user-generated texts, we should be able to replace the explicit preprocessing step of language identification with the use of multilingual embeddings, which provide a common intermediate language to represent words in any of the languages considered. Furthermore, this approach also resolves the problem of context fragmentation, which is notably less trivial to solve with the language identification alternative.

Chapter 8

Word embeddings for noisy texts

Research on monolingual word embeddings has mainly focused on improving their performance on standard corpora, disregarding the difficulties posed by user-generated text, which is usually affected by texting phenomena (see Section 1.1.1).

In the previous chapter, we proposed a method to encode the multilinguality concern of this domain into cross-lingual word embeddings. Now, we turn to the remaining productive texting phenomena which give rise to the wide array of lexical variants, and how they can also be encoded in an embedding space. By doing this, we seek to benefit current end-to-end approaches (Bordes et al., 2016; Klein et al., 2017; Schmitt et al., 2018) which exploit the raw data from the source without applying explicit preprocessing steps, in an attempt to harness every bit of information for the specific task at hand while avoiding the error propagation problem described earlier.

In this regard, a normalization step usually alters the original information encoded in the input text, although in a way that would benefit the next stages of the pipeline. For instance, if we normalize “nooooo” to “no”, the emphatic connotation of the first word is lost, which could be useful for a system that looks for this kind of marks, such as sentiment analysis or opinion mining. In this case, it is important to highlight the existence of *intentionality* when using one form over the other, in contrast with accidentally introducing spelling mistakes in the writing.¹

Granted, a task or system tailored to normalized inputs which cannot exploit the nuances of non-standard texts will probably benefit from using “no” instead of “nooooo”. For example, in PoS tagging, emphasis, and other texting phenomena are not likely to help in obtaining the correct tags; quite the contrary, in fact (Gimpel et al., 2011; Hovy et al., 2014). This aligns with the results obtained by van der Goot et al. (2017), who used a state-of-the-art

¹Spelling mistakes may still convey some sort of information such as the educational level of the writer.

normalization system called MoNoise (van der Goot and van Noord, 2017) before the PoS tagging step to improve its performance. But more interestingly for us, they also found that using a *pretrained* word2vec model over the *unnormalized* input together with the tagger was competitive with the previous normalization alternative. The main reason is that, as already introduced in Section 4.4, word embedding models cluster lexical variants together, thus reducing the sparsity that would be otherwise tackled by a normalization system. Hence, it is not entirely clear whether a normalization approach outperforms the direct use of a simple word embedding model even on this kind of tasks. On the other hand, if we take into account that the PoS tagger was implemented as a bi-directional recurrent neural network (Bilty; Plank et al. 2016), we have found another example in favor of using word embeddings as initialization parameters for more complex neural networks.

In this chapter, we introduce an adaptation of the skipgram model proposed by Bojanowski et al. (2016) to train word embeddings that better integrate word variants (otherwise considered noisy words) at training time. This can be regarded as an analogous incremental improvement over fastText to what this one was over word2vec. Then, we perform an evaluation on a wide array of intrinsic and extrinsic tasks, comparing their performance to that of well-known embedding models such as word2vec and fastText on both standard and noisy English texts. The results show a clear improvement over the baselines in semantic similarity and Sentiment Analysis (SA) tasks, with a general tendency to retain the performance of the best baseline on standard texts and outperform them on noisy texts. Our ultimate goal in this chapter is to improve the performance of traditional embedding models in the context of noisy texts. This would eliminate, or at least alleviate, the need for the usual microtext normalization step, and act as a good starting point for modern end-to-end NLP approaches.

8.1 Towards noise-resistant word embeddings

Word embedding models such as word2vec, GloVe, or fastText are able to cluster word variants together when given a big enough training corpus that includes standard and non-standard language (Sumbler et al., 2018). That is, given enough examples where “friend” (standard word), “freind” (spell-checking error), “frnd” (phonetic-compressed spelling), and even “dog” or “dawg” (street-talk) appear in similar contexts, these words will be translated to similar vector representations. Taking advantage of this fact, many state-of-the-art microtext normalization systems use word embeddings in their pipelines (Bertaglia and Nunes, 2016; van der Goot and van Noord, 2017; Ansari et al., 2017; Sridhar, 2016), both when generating and selecting normalization candidates for the input words.

The problem with this approach is that the contexts where those example words appear

are also likely to be affected by the same phenomena as the words themselves. For example, “friend” might appear in phrases such as “that’s my best friend” or “friend for life”, while “frnd” in others such as “dats my bst frnd” or “frnd 4 lifee”. This can make it difficult for the embedding algorithm to find the semantic similarity between “friend” and “frnd” when only relying on the assumption that the training corpus is big and diverse enough to effectively convey this variability. However, not all of the embedding algorithms are equally affected by this, as those which take subword information into account may have an advantage: in our example, the similar morphology shared by the word variants may be exploited by algorithms such as fastText, which uses character n-grams, to give them more similar vector representations.

In this chapter we present a modification of the skipgram model proposed by Bojanowski et al. (2016), in turn a modification of the original by Mikolov et al. (2013), which tries to improve the clustering of standard words and their noisy variants. This is attained through the use of *bridge-words*, normalized derivatives of the original words from the training corpus where one of their constituent characters is removed.² By using these new words at training time in addition to the original ones, our objective is to increase the similarity between word variants, using those bridge-words as intermediate terms that match the words we want to cluster together. For example, “friend” and “freind” have in common the bridge-words “frind” and “frend”. Even if the original words do not appear in the same context in the training corpus, using the bridge-words in place of the originals allows for indirect paths to be discovered: “friend”-“frind”-“freind” and “friend”-“frend”-“freind”. In the case of “friend” and “frnd”, and assuming that we use an embedding algorithm that exploits subword information, as we propose here, the higher morphological similarities of the latter with respect to the bridge-words “frend” and “frind” benefits their grouping together in the same cluster. Notably, it should be also possible to apply analogous modifications to the ones described here to other training models, such as the continuous bag of words (Mikolov et al., 2013).

It is worth pointing out that we did not consider the latest state-of-the-art models such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) as it would not be feasible to apply analogous modifications to these large and complex models at this point. On the other hand, although we currently consider a monolingual English setup, our method should be suitable for any other language with a similar concept of *character*, in contrast to those based on logograms such as Chinese.

²In the sense that these are intermediate (or normalized) representations that tie together otherwise isolated terms, they may resemble the *index terms* used in information retrieval. Since there is no index in our case, we will not refer to them as such.

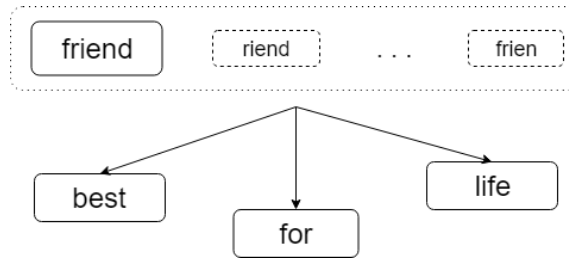


Figure 8.1: Visualization of the adapted skipgram model where bridge-words have a lower impact than the original word in the context.

8.1.1 Modified skipgram model

The skipgram model found in tools like word2vec and fastText establishes that for each word in a text it should be possible to predict those in their corresponding contexts (Mikolov et al., 2013). As a consequence, the words that appear in similar contexts end up represented by similar vectors, so that the transformation learned by the model can effectively map one group of words onto the other.

Based on the skipgram model from fastText, our proposal aims at increasing the similarity between standard words and their noisy counterparts by introducing extra words at training time: the *bridge-words* of the original terms. To obtain them, for each word in the training corpus, we first lowercase, strip diacritics, and remove character successive repetitions,³ and then obtain one bridge-word for each remaining character in the term by removing one different character each time. Note that this procedure is exclusively applied to obtain all the bridge-words, and the unprocessed corpus will be used during training. Formally, let \mathcal{V} be the word vocabulary extracted from the training corpus so that $\mathcal{V} = \{w_1, w_2, \dots, w_n\}$ with n the size of the vocabulary. The set of bridge-words is then defined as $\mathcal{B} = \{b_{1,1}, b_{1,2}, \dots, b_{1,|w_1|}, \dots, b_{n,1}, b_{n,2}, \dots, b_{n,|w_n|}\}$, where $|w_i|$ is the length of word w_i , and $b_{i,j}$ is the bridge-word obtained by first normalizing as described earlier and then removing the character at position j from the word $w_i \in \mathcal{V}$.⁴

These new words are used in addition to the original words when predicting their context in the skipgram model training, as depicted in Figure 8.1. For example, in the phrase “that’s my best Friëndd ever”, the objective is not only to predict “that’s”, “my”,

³This applies both to standard and non-standard repetitions (e.g., “success” vs. “daaammn”), obtaining a common denominator for when users make the mistake of removing standard repetitions (e.g., from “success” to “succes”) or add repetitions to provide emphasis (e.g., from “damn” to “daaammn”). The resulting words are very similar and can still be read mostly in the same way. An analogous reasoning is used in the case of lowercasing and stripping diacritics.

⁴It is possible that $\mathcal{V} \cap \mathcal{B} \neq \emptyset$.

“best”, and “ever” using the word “Friendd”, but also using the derived bridge-words “riend”, “fiend”, “frend”, “frind”, “fried”, and “frien”. This idea of removing one character at a time is similar to the one used in the tool SymSpell⁵ to speed up spell-checking, where it replaces the exhaustive approach of considering all possible edit operations.⁶

In our case, bridge-words are not interesting *per se* but as *intermediaries* between other words. We do not require that they coincide with real words with which they would establish a direct connection; in fact, we assume that these connections will be indirect most of the time. For instance, we do not consider the substitution operations that would construct “tome” and “tame” from “time”, which would explicitly connect the three, but only “tme”, which can be obtained from the three of them by removing one character, linking them together indirectly.

It is important to observe that these bridge-words also constitute artificial noise introduced in our training process that could play a harmful role. As an example, the word “fiend” appears as a bridge-word for “friend”, while also being a standard word from the English dictionary without much semantic relation to the concept of friendship. Because of this, bridge-words should not have the same impact as the original words when tuning the parameters of the model. We propose two mechanisms for lowering the weight of bridge-words in the training process: (1) introducing them randomly, with a fixed probability p_b , instead of for all the original words; and (2) reducing the impact in the objective function by adding a weighting factor. Formally, let w_x be an input word of length $|w_x|$, b_j the bridge-word for w_x when the character at position j is removed, w_y a target word in the context of w_x , H a random variable with $P(H = 1) = p_b$ and $P(H = 0) = 1 - p_b$, $h \sim H$, λ the weight factor, and $E_{ft}(w_x, w_y)$ the objective function of the skipgram model from fastText, then our new objective function, E_{robust} , is defined as:

$$E_{robust} = E_{ft}(\mathbf{w}_x, \mathbf{w}_y) + h \cdot \lambda \cdot \sum_{j=1}^{|w_x|} E_{ft}(\mathbf{b}_j, \mathbf{w}_y) \quad (8.1)$$

where \mathbf{w}_x , \mathbf{w}_y , and \mathbf{b}_j are the vector representations of the corresponding input, target and bridge words.

In any case, the proposed technique does not rule out the requirement of a training corpus where standard and noisy variants of words are used. Rather, it enhances the capacity of already existing models (in this case, the skipgram model from fastText) to *bridge* or further interconnect these word variants.⁷

⁵<https://github.com/wolfgarbe/SymSpell>

⁶Addition, removal, substitution, and transposition of characters.

⁷The corresponding source code is available at <https://github.com/yeraidm/bridge2vec>.

8.2 Evaluation

We use multiple intrinsic and extrinsic evaluation tasks to study the performance of our approach together with word2vec and fastText. The models are trained using the same unprocessed corpus of Web text and tweets. Starting with the usual word similarity task, we also include outlier detection (Camacho-Collados and Navigli, 2016), most of the extrinsic tasks from the SentEval benchmark (Conneau and Kiela, 2018), and then we add Twitter SA from various editions of the SemEval workshop. Ideally, we should see that our embeddings are able to retain the performance of “vanilla” fastText embeddings (Bojanowski et al., 2016) for standard and less-corrupted text, while outperforming them on noisier texts, and that word2vec (Mikolov et al., 2013) is at a disadvantage in this case.

8.2.1 Word embedding training

In this work, we use a combination of Web corpora, specifically the UMBC corpus (Han et al., 2013b), and tweets collected through the Twitter Streaming API from dates between October 2015 and July 2018. It is worth noting that we did not perform any preprocessing or normalization step over the resulting corpus, and the final dataset is formed by 64.653M lines and 3.3B tokens, of which 24.558M are unique.

We employed a modified version of the skipgram model from fastText which incorporates the changes described in Section 8.1.1 together with a vanilla version and a word2vec baseline, using the default hyperparameters for all models. In the case of the proposed model, we train four instances in order to take a first look at the influence of the hyperparameters introduced: probability of introducing a bridge-word (p_b) and weight for bridge-words in the objective (λ). The combinations are $(p_b = 1, \lambda = 1)$, $(p_b = 0.5, \lambda = 1)$, $(p_b = 1, \lambda = 0.1)$, and $(p_b = 0.5, \lambda = 0.1)$. In this work, we do not perform hyperparameter optimization, and those values were selected according to the initial hypothesis that a decreased impact of bridge-words in the training process should be beneficial to the model.

8.2.2 Intrinsic tasks: word similarity and outlier detection

The first intrinsic evaluation task is the well-known semantic word similarity task, which has already been introduced in Section 7.3.1. Here we use the monolingual variant, which consists in scoring the similarity between pairs of words in the same language, measured through the cosine similarity of their corresponding word embeddings. The evaluation is performed using the Spearman correlation between the list of similarity scores obtained and the gold standard. In this case, we use the wordsim353 (Finkelstein

et al., 2002), SCWS (Huang et al., 2012), SimLex999 (Hill et al., 2015), and SemEval17 (monolingual) (Camacho Collados et al., 2017) evaluation datasets.

The second task is outlier detection, which consists in identifying the word that does not belong in a group of words according to their pair-wise semantic similarities. As an example, “snake” would be an outlier in the set “german shepherd”, “golden retriever”, and “french bulldog” since, in spite of also being an animal, it is not a dog. In this case, we use the 8-8-8 (Camacho-Collados and Navigli, 2016) and wiki-sem-500 (Blair et al., 2016) datasets, and measure the proportion of times in which the outlier was successfully detected (i.e., the accuracy) as performance metric.

8.2.3 Extrinsic tasks: the SentEval benchmark and Twitter SA

Since it is not evident that performance on intrinsic tasks translates proportionally to extrinsic tasks (Faruqui et al., 2016; Chiu et al., 2016), where word embeddings are used as part of bigger systems, we resort to the SentEval benchmark (Conneau and Kiela, 2018) in order to evaluate our embeddings in a more realistic setup. The tasks included in this benchmark evaluate sentence embeddings, which can be obtained from word embeddings using an aggregating function, which can go from the simple bag of words to the more complex neural-based models InferSent (Conneau et al., 2017) or GenSen (Subramanian et al., 2018). Additionally, some tasks require a classifier to be trained on the sentence embeddings in order to obtain an output of the desired type. In both cases, we maintain a simple approach where we focus on the raw performance of the word embeddings rather than the models used on top of them. This means using the bag of words model to obtain sentence representations, which simply averages the corresponding word embeddings from each sentence, and then linear regression for the classification tasks.

SentEval includes 17 extrinsic tasks, of which we use 16, and 10 probing tasks. The first group includes semantic textual similarity (STS 2012-2016, STS Benchmark, and SICK-Relatedness), natural language inference (SICK-Entailment and SNLI), sentiment analysis (SST, both binary and fine-grained), opinion-polarity (MPQA), movie and product review (MR and CR), subjectivity status (SUBJ), question-type classification (TREC), and paraphrase detection (MRPC). The second group is formed by tasks that evaluate other linguistic properties which could be found encoded in sentence embeddings, such as sentence length, depth of the syntactic tree, or the number of the subject of the main clause. For a more detailed description of these tasks together with references to the original sources, see (Conneau and Kiela, 2018).⁸ In general, for the similarity tasks, the performance is measured using Spearman correlation, while in the rest of the cases, which correspond to

⁸We already introduced the natural language inference task in Section 7.4.2.

classification tasks, the accuracy of the classification is obtained. Unfortunately, we leave image-caption retrieval task (COCO) out of our test bench as it is not possible to access the source texts, which would be needed for the processing that we perform as described in the next section.

Finally, we also evaluate on the SA datasets released in the SemEval workshops by Nakov et al. (2013) (task 2, subtask B), Rosenthal et al. (2014) (task 9, subtask B),⁹ and Nakov et al. (2016) (task 4, subtasks B, D, C, and E). These already include noisy texts in the form of tweets, thus they are not processed in the same way as the following datasets are processed, as explained below. However, since we still use the SentEval code, we did filter the neutral/objective tweets in ternary SA datasets. We also performed downsampling on the 2016 training and development datasets, both binary and fine-grained, in order to compensate for the substantial unbalance across instance classes. This is important as the test datasets are also skewed in the same manner, and it lead the classifiers to adjust to this bias to obtain unrealistic results. In the case of the binary task, we equated the positive instances with the number of negative ones, while in the case of the fine-grained task we used a fixed maximum number of 500 instances per class, given the huge gap between the least frequent class (accounting for 71 instances) and the most frequent one (including 2876 instances).¹⁰

8.2.4 Dataset de-normalization

Since we could not find noisy text datasets for such a wide variety of evaluation tasks as the ones from the SentEval benchmark, we decided to *de-normalize* (i.e., introduce artificial noise into) these standard datasets, while also keeping the originals of the benchmark, in order to cover the case of noisy texts in the extension needed by this work. The procedure consists in randomly replacing every word in the texts by a noisy variant with some fixed probability. The noisy variants are obtained from two publicly available normalization dictionaries formed by (non-standard, standard) word pairs, utdallas and unimelb, that were released in the first (2015) edition of the W-NUT workshop (Baldwin et al., 2015).

For the word similarity and outlier detection datasets, this probability p_d was fixed to 1; i.e., we modify all the words in the test set which appear in our normalization dictionaries (which cover 78.61% of them). In the case of the SentEval datasets, we created three versions for each one of them: a heavily corrupted version ($p_d = 1$), a more balanced version ($p_d = 0.6$), and a less noisy one ($p_d = 0.3$). As an example, from the original sentence “A man is playing a flute” we obtain “aa woma isz playiin thw flute”, “aa mann is

⁹In this case, we use the training data from the previous edition.

¹⁰Other datasets used in this work are also unbalanced, although to a significantly lesser extent and with no such measurable impact on the results.

standard				
	SCWS	WS353	SL999	Sem17
word2vec	64.7	69.1	32.2	68.2
fastText	65.4	72.7	33.5	70.3
ours	65.1	73.1	33.8	70.4
noisy				
	SCWS	WS353	SL999	Sem17
word2vec	13.0	13.7	-10.9	11.2
fastText	35.2	38.1	7.3	37.2
ours	42.1	44.2	16.4	43.1

Table 8.1: Spearman correlation results of word similarity on SCWS, wordsim353 (WS353), SimLex999 (SL999), and SemEval17 (Sem17) datasets.

playng da flute”, and “aa wman is playing the flute”, in each respective case. The Twitter SA datasets, on the other hand, were not de-normalized.

Furthermore, we perform 10 de-normalization runs over the intrinsic tasks datasets and three over the extrinsic ones, obtaining multiple noisy versions of each dataset. By averaging the results over the different de-normalizations, we try to neutralize extreme measurements that can be caused by different noisy variants of words.

8.2.5 Results

Our currently best model is obtained with the hyperparameter combination ($p_b = 0.5$, $\lambda = 0.1$), which in some way validates our hypothesis that bridge-words should be introduced in a restrained fashion. In general terms, this model has a similar performance to fastText in the standard case, while outperforming both word2vec and fastText in noisy setups, with wider margins towards noisier texts.

Intrinsic evaluation. Table 8.1 shows the results on the intrinsic word similarity task. On standard words, fastText and our model obtain similar performance, both surpassing that of word2vec. On non-standard words, however, our model is able to consistently outperform fastText in every dataset, while word2vec falls further behind possibly due to its lack of support for OOV words in this scenario, as 48.77% of the unique noisy test words are not included in the vocabulary of the word2vec model.

In the case of the results for outlier detection, shown in Table 8.2, we obtained mixed results. On the 8-8-8 dataset, our model outperforms the baselines both in the standard and noisy scenarios, although with visibly lower margins than in the case of semantic

	standard		noisy	
	8-8-8	wiki	8-8-8	wiki
word2vec	59.4	53.8	22.8	39.3
fastText	65.6	49.0	31.7	41.1
ours	67.2	47.8	33.3	41.1

Table 8.2: Accuracy results of outlier detection on 8-8-8 and wiki-sem-500 (wiki) datasets.

similarity. However, on the wiki-sem-500 dataset, word2vec outperforms its competitors on standard words and does not lose much performance on the noisy setup. The latter may be explained by the low amount of successfully denormalized words, with just 7.5% of the total (compared to 52.2% on the 8-8-8 dataset), which also hints to the tie between fastText and our model.

Extrinsic evaluation. Given the considerable amount of tasks and datasets included in the SentEval benchmark, we decided to group similar tasks and datasets and show the aggregated results instead of following an exhaustive approach. In this case, and given the variability in dataset sizes, we use a weighted average as the aggregation function.

First of all, we show in Figure 8.2 the dynamic behaviour of each model when going from standard texts to noisier ones. In this case, we divided the tasks into two groups based on the performance metric: Spearman correlation or accuracy. The first one encompasses the semantic similarity and relatedness tasks (STS*¹¹ and SICK-Relatedness) and the second one the rest of the tasks. Except in the case of word2vec on the first group (yellow lines and crosses), all the models start from a very similar position in the standard scenario. Then, the performance begins its downward trend, where our model starts to stand out above the baselines. As we go towards noisier texts, our model manages to stay above the rest of the lines, increasing the distance margin up until the last stretch.

Next, Table 8.3 shows in greater detail the performance of each model in a less aggregated view. In this case, datasets have been grouped by task as described in Section 8.2.3. As we can see, our model is on par with the baselines on standard texts, with a few interesting exceptions: (1) it is able to obtain some advantage on sentiment analysis, which fastText also obtains over word2vec; (2) on question-type classification, word2vec obtains the best performance, and still clearly outperforms fastText on the lowest noise level, although not our model; and (3) on the probing tasks, word2vec takes the lead again, this time by a smaller margin.

¹¹The star notation is used here as a wildcard character; STS* accounts in this particular case for all task names that start with the string “STS”.

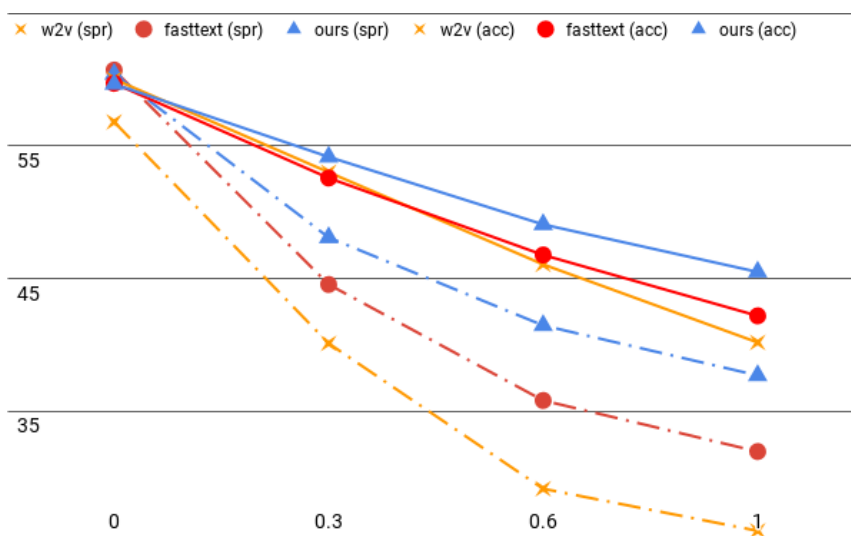


Figure 8.2: Performance of each considered model when going from standard texts to noisier ones on the extrinsic tasks. In lines and dots is the aggregated performance on semantic similarity and relatedness tasks (Spearman correlation). In continuous lines is the aggregated performance on the rest of the tasks (accuracy).

Regarding noisy texts, our model is clearly superior on semantic similarity and relatedness, as we had already seen before, and it also outperforms the baselines on the rest of the tasks, with wider margins on noisier texts, but with the sole exception of paraphrase detection. In this surprising case, word2vec outperforms both fastText and our model obtaining better accuracy on texts with the highest level of noise compared to the previous step. It appears that, with the proper training (and hence, vocabulary), word2vec remains a strong baseline on extrinsic tasks, even in the case of noisy texts, where the level of noise has to be increased notably in order for fastText to obtain a clear advantage. This can also be observed following the continuous lines in Figure 8.2. On the other hand, the weakness seen on word semantic similarity (Table 8.1) relating to OOV words does not seem to translate to extrinsic tasks, where having more context and hence a higher chance of finding IV words mitigates the problem, as we can see in the semantic similarity and relatedness results (Table 8.3).

Finally, in Table 8.4 we show the results obtained on the SemEval Twitter SA datasets. In this case, word2vec continues to display a strong performance, fastText loses the advantage it had on the SentEval benchmark for the same SA task, and our approach is able to revert this performance loss to outperform, once again, both of the baselines. At this point, we can observe how fastText is inferior to word2vec on a real-world social media setting, when

	standard	noisy			standard	noisy		
		<i>low</i>	<i>mid</i>	<i>high</i>		<i>low</i>	<i>mid</i>	<i>high</i>
	Semantic sim. & rel.				Binary classification			
word2vec	56.8	40.1	29.2	26.0	81.5	78.8	76.1	71.4
fastText	60.7	44.6	35.8	32.0	81.8	79.0	75.8	72.1
ours	60.4	48.1	41.5	37.7	81.6	79.4	77.7	74.1
	Sentiment analysis				Entailment			
word2vec	57.9	55.4	50.5	42.8	66.3	54.9	48.8	35.8
fastText	58.8	55.8	52.6	47.0	66.2	54.0	50.0	40.2
ours	59.3	56.9	54.6	51.1	66.3	55.1	51.4	48.1
	Question-type classification				Paraphrase detection			
word2vec	79.4	65.6	53.3	35.0	72.6	67.0	61.2	65.5
fastText	74.8	62.5	52.1	41.8	72.3	62.7	57.1	56.8
ours	73.4	67.5	59.4	49.5	72.9	66.9	60.2	56.3
	Probing tasks							
word2vec	58.2	51.5	45.3	39.3				
fastText	57.8	51.2	45.9	41.0				
ours	57.7	52.8	48.4	43.6				

Table 8.3: Results of the extrinsic evaluation on the SentEval benchmark. The noise levels are *low* ($p_d = 0.3$), *mid* ($p_d = 0.6$), and *high* ($p_d = 1$).

we may have expected the opposite at first. But, for this same reason, it is remarkable to see our approach taking the lead despite being a modification of fastText, which also demonstrates the benefit of including the bridge-words at training time. Having said that, it would be relevant to investigate if higher performance figures can be obtained by modifying the skipgram model from word2vec.

8.3 The importance of word segmentation

In principle, when using word embeddings we assume that the input text is correctly segmented into words. However, suppose that this is not the case, and that it goes beyond frequent de-normalization instances where words are joined or merged together; e.g., “noway”-“no way”, “yesplease”-“yes please”; or even instances where the individual words cannot be immediately recovered such as with “tryna”-“trying to”, or “whatchu”-“what are/do you”. Instead, in the present case we will consider sentences like “theproblem was veryclear” or “the prob lem was very clea r”. A possible solution would be to perform a word segmentation preprocessing step (see Chapter 5) before obtaining the corresponding word

	SE13 B	SE14 B	SE16 BD	SE16 CE
word2vec	84.3	88.3	77.4	35.1
fastText	83.3	88.1	76.5	33.7
ours	84.8	88.6	78.4	35.5

Table 8.4: Accuracy results of the extrinsic evaluation on SemEval (SE) Twitter SA datasets.

embeddings, which would imply introducing again the notion of sequential tasks together with the risk of error propagation, as explained in Section 6.1.

However, let us now consider that the two operations involved in bad word segmentation (i.e., word joining and splitting) might not have the same impact on the process of obtaining relevant word embeddings. If we take into account that models such as fastText, and by extension the modification presented in this chapter, use subword information to construct word embeddings, we might argue that *joining words together* may be moderately supported by these models, as they would still consider the words inside the merging as character n-grams modelled during training. On the contrary, *splitting words* would be more problematic, as it removes parts of a word which could be crucial to obtain the adequate vector representation.

To check this hypothesis, we have devised new experiments using new de-normalized versions of the STS* datasets from the SentEval benchmark, which we have divided into two sets: *join* and *split*. In the former, we randomly removed word delimiters from input sentences with a fixed probability p_j , while in the latter we added delimiters between word characters with a lower probability p_s , $p_s < p_j$, in order to account for the higher amount of non-delimiter characters.

The results obtained, which are shown in Table 8.5, seem to support our hypothesis. Therefore, using a word segmenter with a slight tendency to join words (e.g., through a threshold parameter as shown by Doval et al. (2016)) or even the raw input directly (taking into account the low frequency of splits, while joins are frequent in special elements such as hashtags or URLs), can be considered good practical solutions so long as we use embedding models that exploit subword information. Nonetheless, the latter option is specially relevant for us, since it shows that we may finally dispense with any form of input preprocessing for languages that delimit words; English in our current case. But even in the case of Chinese, where words are not explicitly delimited and word segmentation is a well-studied and complex subject (see Section 5.5), it has been recently shown that this preprocessing step might not be necessary. Meng et al. (2019) propose directly operating over Chinese characters rather than *strict* words. We highlight this strictness property as characters are frequently used as words themselves, but not always. That solution obtains

	join_{ρ}	split_{ρ}
word2vec	11.0	18.3
fastText	39.2	18.7
ours	39.2	17.2

Table 8.5: Spearman correlation averages on the new de-normalized STS* datasets, with $p_j = 0.5$ and $p_s = 0.1$.

better results than other systems that require a previous word segmentation step, even when all of them are implemented as state-of-the-art neural networks. In our case, this shows that we could relax the definition of a *word*, and obtain the embeddings at the character- or sequence-of-characters level.

8.4 Related work

Word embeddings have been at the forefront of NLP research since the past decade, although the first application of vector representations of words dates back to the work of Rumelhart et al. (1986). More recently, word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) were the first models to attain wide use, which take words as basic and indivisible units, implying that the word vocabulary is fixed at training time and any unknown word would be given the same vector representation, regardless of its context or any other intrinsic property. To address the limitations of word2vec and GloVe with OOV words, where morphologically-rich languages such as Finnish or Turkish are specially affected, new models appeared which take subword information into account. The type of subword information used varies in each particular approach: some of them require a preprocessing step to extract morphemes (Luong et al., 2013), while others employ a less strict approach by directly using the characters (Ling et al., 2015; Kim et al., 2016) or character n-grams (Bojanowski et al., 2016; Wieting et al., 2016) that form the words.

When targeting noisy texts from social media, such as tweets from Twitter, previous work relies solely on the high coverage that can be obtained from training in an equally noisy domain (Sumbler et al., 2018). An exception to this rule is the work from Malykh et al. (2018), where they try to obtain robust embeddings to misspelled words (one or two edit operations away from the correct form) by using a new neural-based model. In this case, the flexibility is obtained by an encoding of the prefix, suffix, and set of characters that form each word. By using this set of characters in the encoding, where the specific order between them is disregarded, this approach achieves some form of robustness to low-level noise, while the prefix and suffix part encodes most of the semantic information.

The main difference of our approach is that we are not proposing a whole new model but a generic technique to adapt existing ones. This could be applied to many others, including that from Malykh et al. (2018) itself. Furthermore, we evaluate our embeddings in the context of non-standard texts, a noisier medium than the slightly misspelled standard texts regarded in (Malykh et al., 2018).¹²

Lastly, if we consider standard and non-standard texts as pertaining to different languages, our approach would be similar to (Luong et al., 2015), where the authors also adapt the skipgram model to obtain bilingual embeddings. In this work, they start with comparable bilingual corpora and automatically calculate alignments between words across languages. At training time, they use the words from alignment pairs interchangeably in the texts from each language, requiring each word to predict not only the context in its own language but also the context in the other language. In our case, we only consider one training corpus and create a set of bridge-words that act as alignments between standard words and their noisy counterparts. On the other hand, the weight given to these new words in the objective function is $\lambda < 1$ as they represent noisy examples, whereas in (Luong et al., 2015) the words from the other language are given more weight ($\lambda > 1$).

8.5 Conclusions

In this chapter, we have proposed a modification of the skipgram model from fastText intended to improve the performance of word embedding models on noisy texts as they are found on social media, while retaining the performance on standard texts. These embeddings can be used inside end-to-end approaches which eliminate the need for preprocessing steps that modify the original input and, as explained in previous chapters, could introduce errors in the pipeline. On the other hand, van der Goot et al. (2017) has also shown that normalizing noisy texts does not yield a clear improvement over the plain usage of word embeddings in certain cases.

Existing word embedding models such as word2vec and fastText rely on the fact that an adequate training corpus, mixing standard and non-standard text, will be provided in order to implicitly cope with this kind of texts. In our case, we also exploit this situation and introduce a new set of words in the training process, called bridge-words, whose objective is to connect standard words with their noisy counterparts. To the best of our knowledge, this is the first attempt at explicitly dealing with this type of noisy texts at the word embedding

¹²Unfortunately, we could not include this approach in our test bench as, probably due to differences in the development environment setup, we were not able to train new models nor extract embeddings through pretrained models using the latest version of the code at https://gitlab.com/madrugado/robust-w2v/tree/py3_launch.

level, going beyond the support for OOV words of models such as fastText.

We have evaluated the performance of the proposed approach together with word2vec and fastText baselines on a wide array of intrinsic and extrinsic tasks. In addition to the usual word similarity task, we include outlier detection, 26 tasks from the SentEval benchmark, and Twitter SA from various editions of the SemEval workshop. The results show that, while the performance of our best model on standard texts is mostly preserved when compared to the baselines, it generally outperforms them on noisier texts with wider margins as the level of noise increases. On the extrinsic tasks, in fact, word2vec performs surprisingly well, and it is only clearly bested by fastText once the highest levels of noise are reached. This implies that, given a large enough corpus, word2vec remains a strong baseline for embedding models.

Finally, embedding models that take subword information into account also support, to some extent, bad word segmentation in the form of word joining. Given the low frequency of this phenomenon outside of special elements such as URLs and hashtags, together with the comparatively lower frequency of word splits, we can conclude that the explicit word segmentation step in our pipeline may be even skipped altogether, eliminating the need for any explicit preprocessing step for languages that delimit words.

Part V

Conclusion

Chapter 9

Conclusions and future work

In this dissertation, we have studied two approaches to overcome the challenges posed by user-generated texts in NLP: a traditional, discrete approach which addresses each individual concern through a modular framework, in the form of a sequential preprocessing pipeline, and then a continuous approach that encodes those concerns in real-valued vectors representing words acting as the new building block for other systems. Specifically, we have analyzed the inherent limitations of the first approach, which may impose an upper bound to their achievable performance, in order to justify a transition to the second one. This latter approach is not only free from those limitations, but is also aligned with machine learning state-of-the-art NLP models which already use feature-based representations for linguistic elements such as words, which account for the vast majority of them.

User-generated text as we can find it on the Web, chat applications, or social media platforms, is well-known for its writing style, which usually contains lexical variants and other non-standard linguistic constructs; as well as for its multilinguality and code-switching, since several languages might be used together in a relatively small context frame. Furthermore, this is the domain which supports language evolution nowadays, serving as a testing ground for new terms and concepts. These circumstances make any form of automatic processing notably difficult. Despite this, user-generated texts still constitute a highly valuable data stream to be exploited in multiple NLP applications, such as opinion mining (Vilares et al., 2017a), reputation surveillance (Law et al., 2017), political analysis (Vilares et al., 2015), health surveillance (Karisani and Agichtein, 2018), crime prediction (Gerber, 2014) or disaster management (Rudra et al., 2016), to name just a few. At this point, we have two options to overcome the challenges posed by texting phenomena (Eisenstein, 2013): adapt systems to support this type of texts or adapt the texts themselves to be more easily processed by any system.

System adaptation has the benefit of a high degree of integration, as a particular process

is globally optimized based on the raw, non-standard input, and the desired output for the main task at hand. However, the modularity of this approach is equally low, as the process is implemented using a single model which has to deal with several different tasks simultaneously and is hardly reusable in other settings. On the contrary, the input adaptation path has the benefit of being modular, as we decouple input preprocessing from the main process, and enables the reuse of the preprocessing part. Analogously, it also obtains a low degree of integration, since it is not easy to optimize the final system that includes the preprocessing and main processing steps as a whole, which may ultimately impact its performance. Besides all this, we should also seek some flexibility in order to easily support the dynamism of languages, meaning that our solution should adapt to changes in the domain with minimal human intervention.

Specifically, in Part II we presented our proposals to explicitly tackle the preprocessing tasks mentioned earlier organized into a pipeline:

- In the case of **language identification** (Chapter 3), we adapted existing tools to the domain of the TweetLID workshop (Zubiaga et al., 2014), which accounts for tweets written in the languages spoken in the Iberian peninsula alongside English.
- Likewise, our **microtext normalization** approach (Chapter 4) was developed on the occasion of the W-NUT 2015 shared task 2 (Baldwin et al., 2015), where we presented a simple proposal based on the two-step framework of Han and Baldwin (2011): a candidate generation step implemented by a spell checker and a normalization dictionary, followed by a candidate selection process supported by a word-level language model.
- In both of the previous tasks, we analyzed the shortcomings of our approaches and acknowledged the complexity of the problems at hand. Then, instead of improving our solutions by moving towards other successful approaches, we suggested that the underlying difficulties may be confronted from a different perspective by using word embeddings.
- For **word segmentation** (Chapter 5), we studied the performance of different types of language models, used in combination with a search algorithm that tries to find the optimal segmentation for an input text where word delimiters are removed. In this case, we obtained remarkable results both with neural-based and n-gram models, which did surprise us given the simplicity of the latter compared with the theoretical flexibility and power of the former.
- Furthermore, we have also found that obtaining the correct strict segmentation of an

input text into words might not be necessary to attain a practical solution based on continuous models.

Then, in Chapter 6 we discussed the limitations of a preprocessing pipeline approach and concluded that, rather than focusing on adapting the input, we could instead adapt continuous representations of lexical elements, such as word embeddings, to encode the characteristics of user-generated texts. These embeddings are then used in other machine learning models as part of their internal representation of the input, hence establishing a direct link between the raw input (although through a more effective representation) and the main processing components. Because of this, we can consider this approach as a hybrid between input and system adaptation, benefiting from the modularity of the former and the integration of the latter. On the other hand, we should also note the change of focus from the *discrimination* of the right answers to their *integration* alongside other relevant answers. That is, we are moving from deciding on the specific language a word belongs to or the correct standard word from which a lexical variant is derived, to giving similar vector representations not only to words in different languages relating to similar concepts, but also to any derived lexical variant used similarly to its standard counterpart.

Finally, in Part IV we showed how to encode into word embeddings the various concerns raised by user-generated texts, which in previous chapters had been explicitly tackled through preprocessing tasks:

- The **multilinguality** concern of this type of text can be encoded in cross-lingual word embeddings (Chapter 7). There are multiple approaches to obtain them, and we have presented a technique to improve the integration of multilingual embedding spaces obtained through the alignment of two or more monolingual spaces.
- The remaining concerns which relate to **texting phenomena** can be already encoded in word embeddings to some extent given a training corpus that mixes standard and non-standard texts (Chapter 8). Given this observation, we proposed an adaptation for existing embedding models which facilitates the encoding of the similarities between lexical variants.

It must be noted that our aim was not on obtaining a performance improvement over traditional discrete approaches at this point, but to show that continuous models are able to overcome some crucial limitations of those that would impose an upper bound on their performance and usefulness.

9.1 Future work

After transitioning from discrete to continuous models in the course of this dissertation, the future lines of research will mostly focus on improving the latter at two levels:

Multilingual embeddings

- Obtain singleton representations for cross-lingual homographs. This issue has been extensively discussed in Section 7.6.
- As an alternative to the previous point, look into the feasibility of selecting *language-sense* embeddings during evaluation.
- Continue to explore the possibilities of post-processing multilingual models, investigating their impact in different tasks. Given the fact that going from restrictive orthogonal transformations to the more unconstrained Meemi transformation seems clearly beneficial in the integration of monolingual models, it remains to be seen whether some form of constrained non-linear transformation can be successfully applied on the current models obtained with Meemi.
- Include more languages in our multilingual models and continue to analyze the possible performance improvements.

Robust embeddings

- Train embedding models for languages other than English which have normalization lexicons or training data available, such as Spanish using the resources published by TweetNorm organizers (Alegria et al., 2013, 2015).
- Adapt other embedding models to make them more robust to noisy text, such as the state-of-the-art models ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019), or even apply the adaptation to models that distill the knowledge of those two into smaller packages (Tang et al., 2019).
- It would be also interesting to consider other types of bridge-words such as phonetic codes obtained from a phonetic algorithm like the Metaphone (Philips, 1990). To this end, the study of such algorithms in the context of normalization candidate generation shown in Appendix A will surely be useful.

- Our approach should also be orthogonal to other techniques that enhance the performance of word embeddings, such as the ones described by Mikolov et al. (2018), which could also be applied to the models obtained in this work.

Although there is no reason to think that both approaches would not work well together, since the former was tested on fastText embeddings and the latter is just a modification of that same model, we will conduct experiments to analyze the performance of combining both of them and compare it with discrete approaches.¹ Lastly, we also intend to investigate how to support dynamically evolving vocabularies, which should greatly increase the adaptability of our models. At this respect, we might take a hybrid approach where the monolingual part is frequently updated, taking inspiration from the Gavai living lexicon (Sahlgren et al., 2016), but the multilingual alignments are updated less often.

¹Note, again, that obtaining higher performance than discrete approaches was not an objective of this work.

Bibliography

- Adda-Decker, M., Adda, G., and Lamel, L. (2000). Investigating text normalization and pronunciation variants for German broadcast transcription. In *Proc. of the 6th Int. Conf. on Spoken Language Processing, ICSLP 2000 / INTERSPEECH 2000*, pages 266–269.
- Ahmad, W. U., Zhang, Z., Ma, X., Hovy, E., Chang, K.-W., and Peng, N. (2018). On Difficulties of Cross-Lingual Transfer with Order Differences: A Case Study on Dependency Parsing. In *Proc. of the 16th Annual Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018*.
- Akhtar, M. S., Sikdar, U. K., and Ekbal, A. (2015). IITP: Hybrid approach for text normalization in Twitter. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 106–110.
- Al-Rfou, R., Perozzi, B., and Skiena, S. (2013). Polyglot: Distributed Word Representations for Multilingual NLP. In *Proc. of the 17th Conf. on Computational Natural Language Learning, CoNLL 2013*, pages 183–192.
- AleAhmad, A., Amiri, H., Darrudi, E., Rahgozar, M., and Oroumchian, F. (2009). Hamshahri: A standard Persian text collection. *Knowledge-Based Systems*, 22(5):382–387.
- Alegria, I., Aranberri, N., Comas, P. R., Fresno, V., Gamallo, P., Padró, L., San Vicente, I., Turmo, J., and Zubiaga, A. (2015). TweetNorm: A benchmark for lexical normalization of Spanish tweets. *Language resources and evaluation*, 49(4):883–905.
- Alegria, I., Aranberri, N., Fresno, V., Gamallo, P., Padró, L., San Vicente, I., Turmo, J., and Zubiaga, A. (2013). Introducción a la tarea compartida Tweet-Norm 2013: Normalización léxica de tuits en español. In *Proc. of the Tweet Normalization Workshop, Tweet-Norm 2013, co-located with 29th Conf. of the Spanish Society for Natural Language Processing, SEPLN 2013.*, pages 1–9.

- Alfonseca, E., Bilac, S., and Pharies, S. (2008). Decomposing Query Keywords from Compounding Languages. In *Proc. of the 46th Annual Meeting of the ACL: Short Papers, HLT-Short '08*, pages 253–256.
- Ammar, W., Mulcaire, G., Tsvetkov, Y., Lample, G., Dyer, C., and Smith, N. A. (2016). Massively multilingual word embeddings. *arXiv preprint arXiv:1602.01925*.
- Ansari, S. A., Zafar, U., and Karim, A. (2017). Improving text normalization by optimizing nearest neighbor matching. *arXiv preprint arXiv:1712.09518*.
- Artetxe, M., Labaka, G., and Agirre, E. (2016). Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proc. of the 2016 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2016*, pages 2289–2294.
- Artetxe, M., Labaka, G., and Agirre, E. (2017). Learning bilingual word embeddings with (almost) no bilingual data. In *Proc. of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017*, pages 451–462.
- Artetxe, M., Labaka, G., and Agirre, E. (2018a). Generalizing and improving bilingual word embedding mappings with a multi-step framework of linear transformations. In *Proc. of the 32th AAAI Conf. on Artificial Intelligence, AAAI 2018*, pages 5012–5019.
- Artetxe, M., Labaka, G., and Agirre, E. (2018b). A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*, pages 789–798.
- Atkinson, K. (2011). GNU Aspell (rel. 0.60). Available at: <http://aspell.net>.
- Aw, A., Zhang, M., Xiao, J., and Su, J. (2006). A phrase-based statistical model for SMS text normalization. In *Proc. of the Main Conf. Poster Sessions of the Int. Committee on Computational Linguistics and the Association for Computational Linguistics, COLING-ACL 2006*, pages 33–40.
- Baldwin, T., Cook, P., Lui, M., MacKinlay, A., and Wang, L. (2013). How noisy social media text, how diffrent social media sources? In *Proc. of the 6th Int. Joint Conf. on Natural Language Processing, IJCNLP 2013*, pages 356–364.
- Baldwin, T., de Marneffe, M.-C., Han, B., Kim, Y.-B., Ritter, A., and Xu, W. (2015). Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 126–135.

- Baldwin, T. and Lui, M. (2010). Language Identification: The Long and the Short of the Matter. In *Proc. of the 11th Annual Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2010*, pages 229–237.
- Bansal, M., Gimpel, K., and Livescu, K. (2014). Tailoring Continuous Word Representations for Dependency Parsing. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014*, volume 2, pages 809–815.
- Barone, A. V. M. (2016). Towards cross-lingual distributed representations without parallel text trained with adversarial autoencoders. In *Proc. of the 1st Workshop on Representation Learning for NLP, RepL4NLP 2016*, pages 121–126.
- Baroni, M., Bernardini, S., Ferraresi, A., and Zanchetta, E. (2009). The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. *Language resources and evaluation*, 43(3):209–226.
- Beaufort, R., Roekhaut, S., Cougnon, L.-A., and Fairon, C. (2010). A Hybrid Rule/Model-based Finite-state Framework for Normalizing SMS Messages. In *Proc. of the 48th Annual Meeting of the Association for Computational Linguistics, ACL 2010*, pages 770–779.
- Beckley, R. (2015). Bekli: A Simple Approach to Twitter Text Normalization. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 82–86.
- Beider, A. (2008). Beider-Morse phonetic matching: An alternative to Soundex with fewer false hits. *Avotaynu: the International Review of Jewish Genealogy (Summer 2008)*.
- Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3:1137–1155.
- Benton, A., Mitchell, M., and Hovy, D. (2017). Multitask Learning for Mental Health Conditions with Limited Social Media Data. In *Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics, EACL 2017*, volume 1, pages 152–162.
- Bergsma, S., McNamee, P., Bagdouri, M., Fink, C., and Wilson, T. (2012). Language Identification for Creating Language-specific Twitter Collections. In *Proc. of the 2nd Workshop on Language in Social Media, LSM 2012*, pages 65–74.
- Bernier-Colborne, G. and Barriere, C. (2018). CRIM at SemEval-2018 Task 9: A Hybrid Approach to Hypernym Discovery. In *Proc. of the 12th Int. Workshop on Semantic Evaluation, SemEval 2018*, pages 722–728.

- Bertaglia, T. F. C. and Nunes, M. d. G. V. (2016). Exploring word embeddings for unsupervised textual user-generated content normalization. In *Proc. of the 2nd Workshop on Noisy User-generated Text, W-NUT 2016*.
- Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., and Fienberg, S. (2003). Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23.
- Bingel, J. and Søgaard, A. (2017). Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics, EACL 2017*, volume 2, pages 164–169.
- Bisani, M. and Ney, H. (2008). Joint-sequence models for grapheme-to-phoneme conversion. *Speech communication*, 50(5):434–451.
- Blair, P., Merhav, Y., and Barry, J. (2016). Automated generation of multilingual clusters for the evaluation of distributed representations. *arXiv preprint arXiv:1611.01547*.
- Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *Transactions of the Association of Computational Linguistics*, 5(1):135–146.
- Bordea, G., Lefever, E., and Buitelaar, P. (2016). Semeval-2016 task 13: Taxonomy Extraction Evaluation (TExEval-2). In *Proc. of the 10th Int. Workshop on Semantic Evaluation, SemEval 2016*, pages 1081–1091.
- Bordes, A., Boureau, Y.-L., and Weston, J. (2016). Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.
- Boyd, D. and Crawford, K. (2012). Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon. *Information, Communication & Society*, 15(5):662–679.
- Branting, L. K. (2003). A comparative evaluation of name-matching algorithms. In *Proc. of the 9th Int. Conf. on Artificial Intelligence and Law, ICAIL 2003*, pages 224–232.
- Brants, T. and Franz, A. (2006). Web 1T 5-gram Version 1 (ref. LDC2006T13). DVD. Distributed by Linguistic Data Consortium.
- Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *Proc. of the 38th Annual Meeting on Association for Computational Linguistics - ACL '00*, pages 286–293.

- Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based N-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479.
- Cai, D. and Zhao, H. (2016). Neural word segmentation learning for Chinese. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, volume 1, pages 409–420.
- Camacho Collados, J., Pilehvar, M. T., Collier, N., and Navigli, R. (2017). SemEval-2017 Task 2: Multilingual and cross-lingual semantic word similarity. In *Proc. of the 11th Int. Workshop on Semantic Evaluation, SemEval 2017*, pages 15–26.
- Camacho-Collados, J., Delli Bovi, C., Espinosa-Anke, L., Oramas, S., Pasini, T., Santus, E., Shwartz, V., Navigli, R., and Saggion, H. (2018). SemEval-2018 Task 9: Hypernym Discovery. In *Proc. of the 12th Int. Workshop on Semantic Evaluation, SemEval 2018*, pages 712–724.
- Camacho-Collados, J., Doval, Y., Martínez-Cámara, E., Espinosa-Anke, L., Barbieri, F., and Schockaert, S. (2019). Learning Cross-lingual Embeddings from Twitter via Distant Supervision. *arXiv preprint arXiv:1905.07358*.
- Camacho-Collados, J. and Navigli, R. (2016). Find the word that does not belong: A framework for an intrinsic evaluation of word vector representations. In *Proc. of the 1st Workshop on Evaluating Vector-Space Representations for NLP, RepEval 2016*, pages 43–50.
- Cardellino, C. (2016). Spanish Billion Words Corpus and Embeddings. Available at <http://crscardellino.me/SBWCE/>.
- Carter, S., Weerkamp, W., and Tsagkias, M. (2013). Microblog language identification: overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation*, 47(1):195–215.
- Cavnar, W. B. and Trenkle, J. M. (1994). N-Gram-Based Text Categorization. In *Proc. of the 3rd Annual Symposium on Document Analysis and Information Retrieval, SDAIR 1994*, pages 161–175.
- Ceylan, H. and Kim, Y. (2009). Language identification of search engine queries. In *Proc. of the Joint Conf. of the 47th Annual Meeting of the ACL and the 4th Int. Joint Conf. on Natural Language Processing of the AFNLP, ACL-AFNLP 2009*, volume 2, pages 1066–1074.
- Chen, S. F. and Goodman, J. (1996). An Empirical Study of Smoothing Techniques for Language Modeling. In *Proc. of the 34th Annual Meeting of the Association for Computational Linguistics, ACL 1996*, pages 310–318.

- Chen, X., Qiu, X., Zhu, C., and Huang, X. (2015). Gated Recursive Neural Network for Chinese Word Segmentation. In *Proc. of the 53rd Annual Meeting of the ACL and the 7th Int. Joint Conf. on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL-IJCNLP 2015*, volume 1, pages 1744–1753.
- Chen, X., Qiu, X., Zhu, C., Liu, P., and Huang, X. (2015). Long Short-Term Memory Neural Networks for Chinese Word Segmentation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing, EMNLP 2015*, pages 1197–1206.
- Chi, C.-H., Ding, C., and Lim, A. (1999). Word Segmentation and Recognition for Web Document Framework. In *Proc. of the Eighth Int. Conf. on Information and Knowledge Management, CIKM '99*, pages 458–465.
- Chiu, B., Korhonen, A., and Pyysalo, S. (2016). Intrinsic evaluation of word vectors fails to predict extrinsic performance. In *Proc. of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 1–6.
- Choudhury, M., Saraf, R., Jain, V., Sarkar, S., and Basu, A. (2008). Investigation and Modeling of the Structure of Texting Language. *International Journal on Document Analysis and Recognition*, 10(3-4):63–70.
- Christen, P. (2006). A comparison of personal name matching: Techniques and practical issues. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE Int. Conf. on*, pages 290–294.
- Chrupała, G. (2014). Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 680–686.
- Coates, J. and Bollegala, D. (2018). Frustratingly easy meta-embedding—computing meta-embeddings by averaging source word embeddings. In *Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 194–198.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.
- Conneau, A. and Kiela, D. (2018). Senteval: An evaluation toolkit for universal sentence representations. In *Proc. of the 11th Int. Conf. on Language Resources and Evaluation, LREC-2018*, pages 1699–1704.

- Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proc. of the 2017 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2017*, pages 670–680.
- Conneau, A., Lample, G., Ranzato, M., Denoyer, L., and Jégou, H. (2018a). Word translation without parallel data. In *Proc. of the 6th Int. Conf. on Learning Representations, ICLR 2018*.
- Conneau, A., Rinott, R., Lample, G., Williams, A., Bowman, S. R., Schwenk, H., and Stoyanov, V. (2018b). XNLI: Evaluating Cross-lingual Sentence Representations. In *Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2018*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of the 2019 Annual Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, pages 4171–4186.
- Doval, Y., Camacho-Collados, J., Espinosa-Anke, L., and Schockaert, S. (2018a). Improving cross-lingual word embeddings by meeting in the middle. In *Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 294–304.
- Doval, Y., Camacho-Collados, J., Espinosa-Anke, L., and Schockaert, S. (2019). On the robustness of unsupervised and semi-supervised cross-lingual word embedding learning. *arXiv preprint arXiv:1908.07742*.
- Doval, Y. and Gómez-Rodríguez, C. (2019). Comparing neural-and n-gram-based language models for word segmentation. *Journal of the Association for Information Science and Technology*, 70(2):187–197.
- Doval, Y., Gómez-Rodríguez, C., and Vilares, J. (2016). Spanish word segmentation through neural language models. *Procesamiento del Lenguaje Natural*, 57:75–82.
- Doval, Y., Vilares, D., and Vilares, J. (2014). Identificación automática del idioma en Twitter: adaptación de identificadores del estado del arte al contexto ibérico. In *Proc. of the Tweet Language Identification Workshop co-located with the 30th Conf. of the Spanish Society for Natural Language Processing, TweetLID@SEPLN 2014*, pages 39–43.
- Doval, Y., Vilares, J., and Gómez-Rodríguez, C. (2015). LYSGROUP: Adapting a Spanish microtext normalization system to English. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 99–105.

- Doval, Y., Vilares, M., and Vilares, J. (2018b). On the performance of phonetic algorithms in microtext normalization. *Expert Systems with Applications*, 113:213–222.
- Dunning, T. (1994). Statistical identification of language. Technical report, Computing Research Laboratory, New Mexico State University Las Cruces.
- Duran, M. S., Nunes, M. d. G. V., and Avanço, L. (2015). A normalizer for UGC in Brazilian Portuguese. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 38–47.
- Duwairi, R. M., Marji, R., Sha’ban, N., and Rushaidat, S. (2014). Sentiment Analysis in Arabic Tweets. In *Proc. of the 5th Int. Conf. on Information and Communication Systems, ICICS 2014*.
- Eisenstein, J. (2013). What to do about bad language on the Internet. In *Proc. of the 2013 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2013*, pages 359–369.
- Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.
- Eshel, Y., Cohen, N., Radinsky, K., Markovitch, S., Yamada, I., and Levy, O. (2017). Named Entity Disambiguation for Noisy Text. In *Proc. of the 21st Conf. on Computational Natural Language Learning, CoNLL 2017*, pages 58–68.
- Espinosa-Anke, L., Camacho-Collados, J., Delli Bovi, C., and Saggion, H. (2016). Supervised Distributional Hypernym Discovery via Domain Adaptation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing, EMNLP 2016*, pages 424–435.
- Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proc. of the 14th Conf. of the European Chapter of the Association for Computational Linguistics, EACL 2014*, pages 462–471.
- Faruqui, M., Tsvetkov, Y., Rastogi, P., and Dyer, C. (2016). Problems with evaluation of word embeddings using word similarity tasks. In *Proc. of the 1st Workshop on Evaluating Vector-Space Representations for NLP, RepEval 2016*, pages 30–35.
- Finkel, J. R., Manning, C. D., and Ng, A. Y. (2006). Solving the problem of cascading errors: Approximate bayesian inference for linguistic annotation pipelines. In *Proc. of the 2006 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2006*, pages 618–626.

- Finkelstein, L., Evgeniy, G., Yossi, M., Ehud, R., Zach, S., Gadi, W., and Eytan, R. (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, 20(1):116–131.
- Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Special Volume of the Philological Society*.
- Foster, J., Cetinoglu, O., Wagner, J., Le Roux, J., Nivre, J., Hogan, D., and Van Genabith, J. (2011). From news to comment: Resources and benchmarks for parsing the language of Web 2.0. In *Proc. of 5th Int. Joint Conf. on Natural Language Processing, IJCNLP 2011*, pages 893–901.
- Fuentes, A. A. G., Parra, I. P., Quevedo-Torrero, J. U., and Perez, R. D. (2016). Comparative Analysis of Phonetic Algorithms Applied to Spanish. In *Proc. of the 2016 Int. Conf. on Computational Science and Computational Intelligence, CSCI 2016*, pages 1180–1185.
- Gadd, T. (1988). ‘Fishing fore werds’: phonetic retrieval of written text in information systems. *Program*, 22(3):222–237.
- Gadd, T. (1990). Phonix: The algorithm. *Program*, 24(4):363–366.
- Gálvez, C. (2006). Identificación de nombres personales por medio de sistemas de codificación fonética. *Encontros Bibli: revista eletrônica de biblioteconomia e ciência da informação*, 22:105–116. Available at <http://www.redalyc.org/articulo.oa?id=14702209>.
- Gamallo, P., Garcia, M., Sotelo, S., and Campos, J. R. P. (2014). Comparing Ranking-based and Naive Bayes Approaches to Language Detection on Tweets. In *Proc. of the Tweet Language Identification Workshop co-located with the 30th Conf. of the Spanish Society for Natural Language Processing, TweetLID@SEPLN 2014*, pages 12–16.
- Geffet, M. and Dagan, I. (2005). The distributional inclusion hypotheses and lexical entailment. In *Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics, ACL 2005*, pages 107–114.
- Gerber, M. S. (2014). Predicting crime using Twitter and kernel density estimation. *Decision Support Systems*, 61:115–125.
- Giguët, E. (1996). The stakes of multilinguality: Multilingual text tokenization in natural language diagnosis. In *Proc. of the PRICAI Workshop on Future Issues for Multilingual Text Processing*.

- Gill, L., Goldacre, M., Simmons, H., Bettley, G., and Griffith, M. (1993). Computerised linking of medical records: methodological guidelines. *Journal of Epidemiology & Community Health*, 47(4):316–319.
- Gill, L. E. and Baldwin, J. A. (1987). *Textbook of Medical Record Linkage*, chapter Methods and Technology of Record Linkage: Some Practical Considerations, pages 39–54.
- Gimpel, K., Schneider, N., O’Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., and Smith, N. A. (2011). Part-of-speech Tagging for Twitter: Annotation, Features and Experiments. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, ACL-HLT 2011*, volume 2, pages 42–47.
- Glavaš, G., Litschko, R., Ruder, S., and Vulić, I. (2019). How to (Properly) Evaluate Cross-Lingual Word Embeddings: On Strong Baselines, Comparative Analyses, and Some Misconceptions. In *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, ACL 2019*, pages 710–721.
- Godin, F., Vandersmissen, B., De Neve, W., and Van de Walle, R. (2015). Named Entity Recognition for Twitter Microposts using Distributed Word Representations. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 146–153.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proc. of the 2014 Conf. on Neural Information Processing Systems, NIPS 2014*, pages 2672–2680.
- Gouws, S., Bengio, Y., and Corrado, G. (2015). Bilbowa: Fast bilingual distributed representations without word alignments. In *Proc. of the 32nd Int. Conf. on Machine Learning, ICML 2015*.
- Gouws, S., Hovy, D., and Metzler, D. (2011). Unsupervised mining of lexical variants from noisy text. In *Proc. of the 1st Workshop on Unsupervised Learning in NLP*, pages 82–90.
- Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361.
- Hammarström, H. (2007). A fine-grained model for language identification. In *Proc. of the ACM SIGIR 2007 Workshop on Improving Non English Web Searching, iNEWS 2007*, pages 14–20.

- Han, B. and Baldwin, T. (2011). Lexical normalisation of short text messages: makin sens a #twitter. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, ACL-HLT 2011*, volume 1, pages 368–378.
- Han, B., Cook, P., and Baldwin, T. (2013a). Lexical normalization for social media text. *ACM Transactions on Intelligent Systems and Technology*, 4(1):5:1–5:27.
- Han, L., Kashyap, A., Finin, T., Mayfield, J., and Weese, J. (2013b). UMBC EBIQUITY-CORE: Semantic textual similarity systems. In *Proc. of the 2nd Joint Conf. on Lexical and Computational Semantics, *SEM 2013*, volume 1, pages 44–52.
- Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. (2017). A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *Proc. of the 2017 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2017*, pages 1923–1933.
- Hassan, H. and Menezes, A. (2013). Social text normalization using contextual graph random walks. In *Proc. of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013*, volume 1, pages 1577–1586.
- Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable Modified Kneser-Ney Language Model Estimation. In *Proc. of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013*, pages 690–696.
- Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proc. of the 14th Int. Conf. on Computational Linguistics, COLING 1992*, pages 539–545.
- Hill, F., Reichart, R., and Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.
- Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.
- Hoffart, J., Milchevski, D., and Weikum, G. (2014). STICS: searching with strings, things, and cats. In *Proc. of the 37th Int. ACM SIGIR Conf. on Research & Development in Information Retrieval*, pages 1247–1248.
- Hollingshead, K. and Roark, B. (2007). Pipeline iteration. In *Proc. of the 45th Annual Meeting of the Association of Computational Linguistics, ACL 2007*, pages 952–959.
- Holmes, D. and McCabe, M. C. (2002). Improving precision and recall for Soundex retrieval. In *Proc. of the Int. Conf. on Information Technology: Coding and Computing, ITCC 2002*, pages 22–26.

- Hong, L., Convertino, G., and Chi, E. H. (2011). Language matters in Twitter: A large scale study. In *Proc. of the 5th Int. AAAI Conf. on Weblogs and Social Media, ICWSM 2011*.
- Hood, D. (2002). Caverphone: Phonetic matching algorithm. Technical Paper CTP060902, University of Otago. Available at <http://caversham.otago.ac.nz/files/working/ctp060902.pdf>.
- Hood, D. (2004). Caverphone revisited. Technical Paper CTP150804, University of Otago. Available at <http://caversham.otago.ac.nz/files/working/ctp150804.pdf>.
- Hovy, D., Plank, B., and Søgaard, A. (2014). When POS data sets don't add up: Combatting sample bias. In *Proc. of the 9th Int. Conf. on Language Resources and Evaluation, LREC 2014*, pages 4472–4475.
- Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*, pages 328–339.
- Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *Proc. of the 50th Annual Meeting of the Association for Computational Linguistics, ACL 2012*, volume 1, pages 873–882.
- Hurtado, L. F., Pla, F., Giménez, M., and Arnal, E. S. (2014). ELiRF-UPV en TweetLID: Identificación del Idioma en Twitter. In *Proc. of the Tweet Language Identification Workshop co-located with the 30th Conf. of the Spanish Society for Natural Language Processing, TweetLID@SEPLN 2014*, pages 35–38.
- Ikeda, T., Shindo, H., and Matsumoto, Y. (2016). Japanese text normalization with encoder-decoder model. In *Proc. of the 2nd Workshop on Noisy User-generated Text, W-NUT 2016*, pages 129–137.
- Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456.
- Iyyer, M., Boyd-Graber, J. L., Claudino, L. M. B., Socher, R., and Daumé III, H. (2014). A Neural Network for Factoid Question Answering over Paragraphs. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing, EMNLP 2014*, pages 633–644.
- Jenks, G. (2017). WordSegment. Available at: <http://www.grantjenks.com/docs/wordsegment/>.

- Jin, N. (2015). NCSU-SAS-Ning: Candidate generation and feature engineering for supervised lexical normalization. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 87–92.
- Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., et al. (2017). Google’s multilingual neural machine translation system: enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.
- Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (2nd ed.)*. 2 edition.
- Kacmarcik, G., Brockett, C., and Suzuki, H. (2000). Robust Segmentation of Japanese Text into a Lattice for Parsing. In *Proc. of the 18th Conf. on Computational Linguistics, COLING 2000*, volume 1, pages 390–396.
- Karisani, P. and Agichtein, E. (2018). Did You Really Just Have a Heart Attack?: Towards Robust Detection of Personal Health Mentions in Social Media. In *Proc. of the 2018 World Wide Web Conf., WWW 2018*, pages 137–146.
- Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.
- Kementchedjheva, Y., Ruder, S., Cotterell, R., and Søgaard, A. (2018). Generalizing Procrustes Analysis for Better Bilingual Dictionary Induction. In *Proc. of the Conf. on Computational Natural Language Learning, CoNLL 2018*, pages 211–220.
- Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Proc. of the 30th AAAI Conf. on Artificial Intelligence, AAAI 2016*, pages 2741–2749.
- Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. In *Proc. of the 2014 Int. Conf. on Learning Representations, ICLR 2014*.
- Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. of the 2017 Annual Meeting of the Association for Computational Linguistics, System Demonstrations, ACL 2017*, pages 67–72.
- Klementiev, A., Titov, I., and Bhattacharai, B. (2012). Inducing crosslingual distributed representations of words. In *Proc. of the 24th Int. Conf. on Computational Linguistics, COLING 2012*, pages 1459–1474.

- Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing, ICASSP 95*, volume 1, pages 181–184.
- Kobus, C., Yvon, F., and Damnati, G. (2008a). Normalizing SMS: are two metaphors better than one? In *Proc. of the 22nd Int. Conf. on Computational Linguistics*, volume 1, pages 441–448.
- Kobus, C., Yvon, F., and Damnati, G. (2008b). Transcrire les SMS comme on reconnaît la parole. In *Actes de la 15e Conférence sur le Traitement Automatique des Langues, TALN 2008*, pages 128–138.
- Koehn, P. and Knight, K. (2003). Empirical Methods for Compound Splitting. In *Proc. of the 10th Conf. of the European Chapter of the ACL, EACL 2003*, volume 1, pages 187–193.
- Kong, L., Schneider, N., Swayamdipta, S., Bhatia, A., Dyer, C., and Smith, N. A. (2014). A dependency parser for tweets. In *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2014*, pages 1001–1012.
- Krott, A., Schreuder, R., Harald Baayen, R., and Dressler, W. U. (2007). Analogical effects on linking elements in German compound words. *Language and cognitive processes*, 22(1):25–57.
- Kutuzov, A., Velldal, E., and Øvrelid, L. (2016). Redefining part-of-speech classes with distributional semantic models. In *Proc. of the 20th SIGNLL Conf. on Computational Natural Language Learning, CoNLL 2016*, pages 115–125.
- Laboreiro, G., Bošnjak, M., Sarmiento, L., Rodrigues, E. M., and Oliveira, E. (2013). Determining Language Variant in Microblog Messages. In *Proc. of the 28th Annual ACM Symposium on Applied Computing, SAC 2013*, pages 902–907.
- Lait, A. J. and Randell, B. (1996). An assessment of name matching algorithms. Technical report, University of Newcastle upon Tyne, Department of Computing Science. Available at <http://homepages.cs.ncl.ac.uk/brian.randell/Genealogy/NameMatching.pdf>.
- Law, D., Gruss, R., and Abrahams, A. S. (2017). Automated defect discovery for dishwasher appliances from online consumer reviews. *Expert Systems with Applications*, 67:84–94.
- Lazaridou, A., Dinu, G., and Baroni, M. (2015). Hubness and pollution: Delving into cross-space mapping for zero-shot learning. In *Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Int. Joint Conf. on Natural Language Processing, ACL-IJCNLP 2015*, pages 270–280.

- Leeman-Munk, S., Lester, J., and Cox, J. (2015). NCSU_SAS_SAM: Deep encoding and reconstruction for normalization of noisy text. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 154–161.
- Léonard, N., Waghmare, S., Wang, Y., and Kim, J.-H. (2015). rnn: Recurrent library for Torch. *arXiv preprint arXiv:1511.07889*.
- Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710.
- Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2016). Visualizing and understanding neural models in NLP. In *Proc. of the 2016 Annual Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2016*, pages 681–691.
- Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fernandez, R., Amir, S., Marujo, L., and Luis, T. (2015). Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *Proc. of the 2015 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2015*, pages 1520–1530.
- Lins, R. D. and Gonçalves, P. (2004). Automatic language identification of written texts. In *Proc. of the 2004 ACM Symposium on Applied Computing, SAC 2004*, pages 1128–1133.
- Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444.
- Low, J. K., Ng, H. T., and Guo, W. (2005). A maximum entropy approach to Chinese word segmentation. In *Proc. of the 4th SIGHAN Workshop on Chinese Language Processing*, pages 448–455.
- Lu, A., Wang, W., Bansal, M., Gimpel, K., and Livescu, K. (2015). Deep multilingual correlation for improved word embeddings. In *Proc. of the 2015 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2015*, pages 250–256.
- Lui, M. and Baldwin, T. (2011). Cross-domain Feature Selection for Language Identification. In *Proc. of 5th Int. Joint Conf. on Natural Language Processing, IJCNLP 2011*, pages 553–561.
- Lui, M. and Baldwin, T. (2012). langid.py: An off-the-shelf language identification tool. In *Proc. of the 2012 Annual Meeting of the Association for Computational Linguistics, ACL*

- 2012, *System Demonstrations*, pages 25–30. Tool available at: <https://github.com/saffsd/langid.py>.
- Lui, M. and Baldwin, T. (2014). Accurate Language Identification of Twitter Messages. In *Proc. of the 5th Workshop on Language Analysis for Social Media, LASM 2014*, pages 17–25.
- Lui, M., Lau, J. H., and Baldwin, T. (2014). Automatic detection and language identification of multilingual documents. *Transactions of the Association for Computational Linguistics*, 2:27–40.
- Luong, M., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2016). Multi-task Sequence to Sequence Learning. In *Proc. of the 4th Int. Conf. on Learning Representations, ICLR 2016*.
- Luong, T., Pham, H., and Manning, C. D. (2015). Bilingual word representations with monolingual quality in mind. In *Proc. of the 1st Workshop on Vector-Space Modeling for NLP, RepEval 2016*, pages 151–159.
- Luong, T., Socher, R., and Manning, C. (2013). Better word representations with recursive neural networks for morphology. In *Proc. of the Seventeenth Conf. on Computational Natural Language Learning*, pages 104–113.
- Lynch, B. T. and Arends, W. L. (1977). Selection of a surname coding procedure for the SRS record linkage system. *Washington, DC: US Department of Agriculture, Sample Survey Research Branch, Research Division*.
- Majliš, M. (2012). Yet Another Language Identifier. In *Proc. of the Student Research Workshop at the 13th Conf. of the European Chapter of the Association for Computational Linguistics, EACL 2012*, pages 46–54. Tool available at <http://ufal.mff.cuni.cz/tools/yali/> (visitada en julio 2014).
- Malykh, V., Logacheva, V., and Khakhulin, T. (2018). Robust Word Vectors: Context-Informed Embeddings for Noisy Texts. In *Proc. of the 4th Workshop on Noisy User-generated Text, W-NUT 2018*, pages 54–63.
- Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proc. of 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, System Demonstrations*, pages 55–60.
- Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*.

- Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*.
- Marciniak, T. and Strube, M. (2005). Beyond the pipeline: Discrete optimization in nlp. In *Proc. of the 9th Conf. on Computational Natural Language Learning, CoNLL 2005*, pages 136–143.
- Martínez Alonso, H. and Plank, B. (2017). When is multitask learning effective? Semantic sequence prediction under varying data conditions. In *Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics*, volume 1, pages 44–53.
- Maynard, D. and Greenwood, M. A. (2014). Who cares about Sarcastic Tweets? Investigating the Impact of Sarcasm on Sentiment Analysis. In *Proc. of the 9th Int. Conf. on Language Resources and Evaluation, LREC 2014*, pages 4238–4243.
- McNamee, P. (2005). Language Identification: A Solved Problem Suitable for Undergraduate Instruction. *Journal of Computing Sciences in Colleges*, 20(3):94–101.
- Meng, Y., Li, X., Sun, X., Han, Q., Yuan, A., and Li, J. (2019). Is Word Segmentation Necessary for Deep Learning of Chinese Representations? In *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, ACL 2019*, pages 3242–3252.
- Mikolov, T., Corrado, G., Chen, K., and Dean, J. (2013). Efficient estimation of word representations in vector space. *Proc. of the International Conference on Learning Representations, ICLR 2013*, pages 1–12.
- Mikolov, T., Grave, E., Bojanowski, P., Puhersch, C., and Joulin, A. (2018). Advances in pre-training distributed word representations. In *Proc. of the 11th Int. Conf. on Language Resources and Evaluation, LREC-2018*.
- Mikolov, T., Le, Q. V., and Sutskever, I. (2013a). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.
- Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proc. of the 2013 Annual Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HTL 2013*, pages 746–751.
- Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *Proc. of the 2012 IEEE Spoken Language Technology Workshop, SLT 2012*, pages 234–239.

- Min, W. and Mott, B. (2015). NCSU_SAS_WOOKHEE: a deep contextual long-short term memory model for text normalization. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 111–119.
- Mogadala, A. and Rettinger, A. (2016). Bilingual word embeddings from parallel and non-parallel corpora for cross-language text classification. In *Proc. of the 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2016*, pages 692–702.
- Moore, G. B. (1977). *Accessing individual records from personal data files using non-unique identifiers*, volume 13.
- Mosquera, A. (2011). The Spanish Metaphone Algorithm (Algoritmo del Metáfono para el español). Code available at: <https://github.com/amsqr/Spanish-Metaphone>.
- Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F., and Stoyanov, V. (2016). SemEval-2016 Task 4: Sentiment analysis in Twitter. In *Proc. of the 10th Int. Workshop on Semantic Evaluation, SemEval 2016*, pages 1–18.
- Nakov, P., Rosenthal, S., Kozareva, Z., Stoyanov, V., Ritter, A., and Wilson, T. (2013). SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *Proc. of the 7th Int. Workshop on Semantic Evaluation, SemEval 2013*, pages 312–320.
- Navigli, R. d. R. L. S. (2009). Word sense: A Survey. *ACM Computing Survey*.
- Norvig, P. (2009). Natural language corpus data. In Segaran, T. and Hammerbacher, J., editors, *Beautiful Data*, pages 219–242.
- Odell, M. and Russell, R. C. (1918). The Soundex coding system. US Patents 1261167.
- Odell, M. K. (1956). The profit in records management. *Systems*, 20(20).
- Okazaki, N. and Tsujii, J. (2010). Simple and Efficient Algorithm for Approximate Dictionary Matching. In *Proc. of the 23rd Int. Conf. on Computational Linguistics, COLING 2010*, pages 851–859.
- Owoputi, O., O’Connor, B., Dyer, C., Gimpel, K., Schneider, N., and Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *Proc. of the 2013 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2013*, pages 380–390.

- Padró, L. and Stanilovsky, E. (2012). Freeling 3.0: Towards Wider Multilinguality. In *Proc. of the 8th Int. Conf. on Language Resources and Evaluation, LREC 2012*, pages 2473–2479. Toolkit available at: <http://nlp.lsi.upc.edu/freeling/>.
- Parmar, V. P. and Kumbharana, C. (2014). Study existing various phonetic algorithms and designing and development of a working model for the new developed algorithm and comparison by implementing it with existing algorithm(s). *International Journal of Computer Applications*, 98(19).
- Pauls, A. and Klein, D. (2011). Faster and Smaller N-gram Language Models. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2011*, volume 1, pages 258–267. BerkeleyLM source code available at <https://code.google.com/p/berkeleylm/>.
- Pei, W., Ge, T., and Chang, B. (2014). Max-Margin Tensor Neural Network for Chinese Word Segmentation. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014*, volume 1, pages 293–303.
- Peng, F., Feng, F., and McCallum, A. (2004). Chinese Segmentation and New Word Detection Using Conditional Random Fields. In *Proc. of the 20th Int. Conf. on Computational Linguistics, COLING 2004*, pages 562–568.
- Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing, EMNLP 2014*, volume 14, pages 1532–1543.
- Peters, M. E., Ammar, W., Bhagavatula, C., and Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of the 2018 Annual Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018*, pages 2227–2237.
- Phang, J., Févry, T., and Bowman, S. R. (2018). Sentence encoders on STILTs: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.
- Philips, L. (1990). Hanging on the Metaphone. *Computer Language*, 7(12):39–43.
- Philips, L. (2000). The double Metaphone search algorithm. *C/C++ users journal*, 18(6):38–43.

- Pinto, D., Vilariño, D., Alemán, Y., Gómez, H., Loya, N., and Jiménez-Salazar, H. (2012). The soundex phonetic algorithm revisited for sms text representation. In *Text, Speech and Dialogue*, pages 47–55.
- Plank, B., Søgaard, A., and Goldberg, Y. (2016). Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*.
- Porta, J. (2014). Twitter Language Identification using Rational Kernels and its potential application to Sociolinguistics. In *Proc. of the Tweet Language Identification Workshop co-located with the 30th Conf. of the Spanish Society for Natural Language Processing, TweetLID@SEPLN 2014*, pages 17–20.
- Poutsma, A. (2001). Applying Montecarlo techniques to language identification. In *Computational Linguistics in the Netherlands 2001*, pages 179–189.
- Prager, J., Chu-Carroll, J., Brown, E. W., and Czuba, K. (2008). Question answering by predictive annotation. In *Advances in Open Domain Question Answering*, pages 307–347.
- Principe, J., Principe, J. C., and Kuo, J.-M. (1993). Backpropagation Through Time with Fixed Memory Size Requirements. In *Proc. of the 3rd Workshop on Neural Networks for Signal Processing, IEEE-SP 1993*, pages 207–215.
- Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*, volume 1, pages 133–142.
- Ratner, A., Hancock, B., Dunnmon, J., Goldman, R. E., and Ré, C. (2018). Snorkel MeTal: Weak Supervision for Multi-Task Learning. In *Proc. of the 2nd Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2018*, pages 3:1–3:4.
- Ritter, A., Clark, S., Etzioni, O., et al. (2011). Named entity recognition in tweets: an experimental study. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing, EMNLP 2011*, pages 1524–1534.
- Roller, S. and Erk, K. (2016). Relations such as hypernymy: Identifying and exploiting Hearst patterns in distributional vectors for lexical entailment. In *Proc. of the 2016 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2016*, pages 2163–2172.
- Rosenthal, S., Ritter, A., Nakov, P., and Stoyanov, V. (2014). SemEval-2014 Task 9: Sentiment analysis in Twitter. In *Proc. of the 8th Int. Workshop on Semantic Evaluation, SemEval 2014*, pages 73–80.

- Roth, D. and Yih, W.-t. (2004). A linear programming formulation for global inference in natural language tasks. In *Proc. of the 8th Conf. on Computational Natural Language Learning, CoNLL@NAACL-HLT 2004*, pages 1–8.
- Ruder, S., Vulić, I., and Søgaard, A. (2017). A survey of cross-lingual word embedding models. *arXiv preprint arXiv:1706.04902*.
- Rudra, K., Banerjee, S., Ganguly, N., Goyal, P., Imran, M., and Mitra, P. (2016). Summarizing situational tweets in crisis scenario. In *Proc. of the 27th ACM Conf. on Hypertext and Social Media, HT 2016*, pages 137–147.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.
- Sahlgren, M., Gyllensten, A. C., Espinoza, F., Hamfors, O., Karlgren, J., Olsson, F., Persson, P., Viswanathan, A., and Holst, A. (2016). The Gavagai Living Lexicon. *Proc. of the Tenth International Conference on Language Resources and Evaluation, LREC 2016*.
- Samuelsson, Y., Täckström, O., Velupillai, S., Eklund, J., Fišel, M., and Saers, M. (2008). Mixing and blending syntactic and semantic dependencies. In *Proc. of the 12th Conf. on Computational Natural Language Learning, CoNLL 2008*, pages 248–252.
- Satapathy, R., Guerreiro, C., Chaturvedi, I., and Cambria, E. (2017). Phonetic-based microtext normalization for twitter sentiment analysis. In *Proc. of the 2017 IEEE Int. Conf. on Data Mining Workshops, ICDMW 2017*, pages 407–413.
- Schmitt, M., Steinheber, S., Schreiber, K., and Roth, B. (2018). Joint aspect and polarity classification for aspect-based sentiment analysis with end-to-end neural networks. In *Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 1109–1114.
- Schulz, S., Pauw, G. D., Clercq, O. D., Desmet, B., Hoste, V., Daelemans, W., and Macken, L. (2016). Multimodular text normalization of Dutch user-generated content. *ACM Transactions on Intelligent Systems and Technology*, 7(4):61.
- Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423.
- Shuyo, N. (2010). Language Detection Library for Java. Tool available at <http://code.google.com/p/language-detection/>.

- Shwartz, V., Goldberg, Y., and Dagan, I. (2016). Improving hypernymy detection with an integrated path-based and distributional method. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, pages 2389–2398.
- Signorini, A., Segre, A. M., and Polgreen, P. M. (2011). The use of Twitter to track levels of disease activity and public concern in the us during the influenza a H1N1 pandemic. *PloS one*, 6(5):1–10.
- Smith, S. L., Turban, D. H., Hamblin, S., and Hammerla, N. Y. (2017). Offline bilingual word vectors, orthogonal transformations and the inverted softmax. In *Proc. of the 5th Int. Conf. on Learning Representations, ICLR 2017*.
- Snae, C. (2007). A comparison and analysis of name matching algorithms. *International Journal of Applied Science. Engineering and Technology*, 4(1):252–257.
- Søgaard, A., Agić, Ž., Martínez Alonso, H., Plank, B., Bohnet, B., and Johannsen, A. (2015). Inverted indexing for cross-lingual NLP. In *Proc. of the 2015 Annual Meeting of the Association for Computational Linguistics, ACL 2015*, pages 1713–1722.
- Søgaard, A., Ruder, S., and Vulić, I. (2018). On the limitations of unsupervised bilingual dictionary induction. In *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*, volume 1, pages 778–788.
- Sridhar, V. K. R. (2016). Unsupervised text normalization using distributed representations of words and phrases. In *Proc. of the 1st Workshop on Vector-Space Modeling for Natural Language Processing, RepEval 2016*, pages 8–16.
- Srinivasan, S., Bhattacharya, S., and Chakraborty, R. (2012). Segmenting Web-domains and Hashtags Using Length Specific Models. In *Proc. of the 21st ACM Int. Conf. on Information and Knowledge Management, CIKM '12*, pages 1113–1122.
- Stolcke, A. (2002). SRILM an extensible language modeling toolkit. In *Proc. of the 7th Int. Conf. on Spoken Language Processing, ICSLP 2002*, pages 901–904.
- Subramanian, S., Trischler, A., Bengio, Y., and Pal, C. J. (2018). Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning. In *Proc. of the 6th Int. Conf. on Learning Representations, ICLR 2018*.
- Sumbler, P., Viereckel, N., Afsarmanesh, N., and Karlgren, J. (2018). Handling Noise in Distributional Semantic Models for Large Scale Text Analytics and Media Monitoring. *Proc. of the 4th Workshop on Noisy User-generated Text, W-NUT 2018, Abstract*.

- Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM Neural Networks for Language Modeling. In *Proc. of the 13th Annual Conf. of the Int. Speech Communication Association, INTERSPEECH 2012*, pages 194–197.
- Supranovich, D. and Patsepnia, V. (2015). IHS_RD: Lexical normalization for English tweets. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 78–81.
- Sutton, C. and McCallum, A. (2005). Joint parsing and semantic role labeling. In *Proc. of the 9th Conf. on Computational Natural Language Learning, CoNLL 2005*, pages 225–228.
- Taft, R. L. (1970). Name search techniques. Special Report 1, Bureau of Systems Development, New York State Identification and Intelligence System.
- Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., and Qin, B. (2014). Learning sentiment-specific word embedding for Twitter sentiment classification. In *Proc. of the 2014 Annual Meeting of the Association for Computational Linguistics, ACL 2014*, pages 1555–1565.
- Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. (2019). Distilling task-specific knowledge from BERT into simple neural networks. *arXiv preprint arXiv:1903.12136*.
- Thurlow, C. and Brown, A. (2003). Generation Txt? The sociolinguistics of young people’s text-messaging. *Discourse Analysis Online*.
- Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of the 7th Conf. on Natural Language Learning, CoNLL@HLT-NAACL 2003*, volume 4, pages 142–147.
- Toutanova, K. and Moore, R. C. (2002). Pronunciation modeling for improved spelling correction. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics, ACL 2002*, page 144.
- Tromp, E. and Pechenizkiy, M. (2011). Graph-based n-gram language identification on short texts. In *Proc. of the 20th Belgian Dutch Conf. on Machine Learning, Benelearn 2011*, pages 27–34.
- Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13.
- Tuten, T. L. and Solomon, M. R. (2017). *Social media marketing*.
- Unión Europea (2004). Tratado por el que se establece una Constitución para Europa. *Diario Oficial de la Unión Europea*. Available at <http://eur-lex.europa.eu/legal-content/ES/ALL/?uri=OJ:C:2004:310:TOC>.

- Upadhyay, S., Faruqui, M., Dyer, C., and Roth, D. (2016). Cross-lingual models of word embeddings: An empirical comparison. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, volume 1, pages 1661–1670.
- Valls-Vargas, J., Zhu, J., and Ontañón, S. (2015). Narrative hermeneutic circle: Improving character role identification from natural language text via feedback loops. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence, IJCAI 2015*, pages 2517–2523.
- van der Goot, R., Plank, B., and Nissim, M. (2017). To normalize, or not to normalize: The impact of normalization on Part-of-Speech tagging. In *Proc. of the 3rd Workshop on Noisy User-generated Text, W-NUT 2017*, pages 31–39.
- van der Goot, R. and van Noord, G. (2017). MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144.
- van Noord, G. (1997). Textcat. Source code. Source code available at: <http://odur.let.rug.nl/~vannoord/TextCat/>.
- Vilares, D., , Gómez-Rodríguez, C., and Alonso, M. A. (2017a). Universal, unsupervised (rule-based), uncovered sentiment analysis. *Knowledge-Based Systems*, 118:45–55.
- Vilares, D., Alonso, M. A., and Gómez-Rodríguez, C. (2017b). Supervised sentiment analysis in multilingual environments. *Information Processing & Management*, 53(3):595–607.
- Vilares, D. and Gómez-Rodríguez, C. (2017). A non-projective greedy dependency parser with bidirectional LSTMs. In *Proc. of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, CoNLL 2017*, pages 152–162.
- Vilares, D., Thelwall, M., and Alonso, M. A. (2015). The megaphone of the people? Spanish SentiStrength for real-time analysis of political tweets. *Journal of Information Science*, 41(6):799–813.
- Vilares, J., Alonso, M. A., and Vilares, D. (2013). Prototipado Rápido de un Sistema de Normalización de Tuits: Una Aproximación Léxica. In *Proc. of the Tweet Normalization Workshop at the 29th Congreso de la Sociedad Española de Procesamiento del Lenguaje Natural, SEPLN 2013*, pages 76–80.
- Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.
- Vulić, I. and Korhonen, A. (2016). On the role of seed lexicons in learning bilingual word embeddings. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, volume 1, pages 247–257.

- Vulić, I. and Moens, M.-F. (2015). Monolingual and Cross-Lingual Information Retrieval Models Based on (Bilingual) Word Embeddings. In *Proc. of the 38th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, SIGIR 2015*, pages 363–372.
- Wagner, J. and Foster, J. (2015). DCU-ADAPT: learning edit operations for microblog normalisation with the generalised perceptron. In *Proc. of the 1st Workshop on Noisy User-generated Text, W-NUT 2015*, pages 93–98.
- Wang, K. and Li, X. (2009). Efficacy of a constantly adaptive language modeling technique for web-scale applications. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing, ICASSP 2009*, pages 4733–4736.
- Wang, K., Thrasher, C., and Hsu, B.-J. P. (2011). Web Scale NLP: A Case Study on URL Word Breaking. In *Proc. of the 20th Int. Conf. on World Wide Web, WWW 2011*, pages 357–366.
- Wang, K., Thrasher, C., Viegas, E., Li, X., and Hsu, B.-j. P. (2010). An overview of Microsoft Web N-gram corpus and applications. In *Proc. of the NAACL HLT 2010 Demonstration Session, HLT-DEMO 2010*, pages 45–48.
- Wang, P. and Ng, H. T. (2013). A Beam-Search Decoder for Normalization of Social Media Text with Application to Machine Translation. In *Proc. of the 2013 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2013*, pages 471–481.
- Wellner, B., McCallum, A., Peng, F., and Hay, M. (2004). An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proc. of the 20th Conf. on Uncertainty in Artificial Intelligence, AUI 2004*, pages 593–601.
- Wick, M., Kanani, P., and Pocock, A. (2016). Minimally-Constrained Multilingual Embeddings via Artificial Code-Switching. *Proc. of the 30th AAAI Conference on Artificial Intelligence, AAAI 2016*.
- Wieting, J., Bansal, M., Gimpel, K., and Livescu, K. (2016). Charagram: Embedding Words and Sentences via Character n-grams. In *Proc. of the 2016 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2016*, pages 1504–1515.
- Williams, A., Nangia, N., and Bowman, S. (2018). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018*, volume 1, pages 1112–1122.

- Wu, A. and Jiang, Z. (1998). Word segmentation in sentence analysis. In *Proc. of the 1998 Int. Conf. on Chinese Information Processing*, pages 169–180.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.
- Xing, C., Wang, D., Liu, C., and Lin, Y. (2015). Normalized word embedding and orthogonal transform for bilingual word translation. In *Proc. of the 2015 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2015*, pages 1006–1011.
- Xiong, S., Lv, H., Zhao, W., and Ji, D. (2018). Towards Twitter sentiment classification by multi-level sentiment-enriched word embeddings. *Neurocomputing*, 275(C):2459–2466.
- Xu, J. and Sun, X. (2016). Dependency-based Gated Recursive Neural Network for Chinese Word Segmentation. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, volume 2, pages 567–572.
- Xue, N. (2003). Chinese word segmentation as character tagging. *Computational Linguistics and Chinese Language Processing*, 8(1):29–48.
- Xue, Z., Yin, D., and Davison, B. D. (2011). Normalizing microtext. In *Proc. of the 2011 AAAI Workshop on Analyzing Microtext, AAAI 2011*, volume WS-11-05 of *AAAI Workshops*, pages 74–79.
- Yahya, M., Berberich, K., Elbassuoni, S., and Weikum, G. (2013). Robust question answering over the web of linked data. In *Proc. of the 22nd ACM Int. Conf. on Information & Knowledge Management, CKIM 2013*, pages 1107–1116.
- Yang, X., Macdonald, C., and Ounis, I. (2018). Using word embeddings in Twitter election classification. *Information Retrieval Journal*, 21(2):183–207.
- Yang, Y. and Eisenstein, J. (2013). A Log-Linear Model for Unsupervised Text Normalization. In *Proc. of the 2013 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2013*, pages 61–72.
- Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. In *Proc. of the 2015 ICML Workshop on Deep Learning, ICML 2015*.

- Yu, L.-C., He, W.-C., Chien, W.-N., and Tseng, Y.-H. (2013). Identification of code-switched sentences and words using language modeling approaches. *Mathematical Problems in Engineering*, 2013. Article ID 898714.
- Yu, Z., Wang, H., Lin, X., and Wang, M. (2015). Learning term embeddings for hypernymy identification. In *Proc. of the 2015 Int. Joint Conf. on Artificial Intelligence, IJCAI 2015*, pages 1390–1397.
- Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.
- Zhang, M., Liu, Y., Luan, H., and Sun, M. (2017). Adversarial training for unsupervised bilingual lexicon induction. In *Proc. of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017*, pages 1959–1970.
- Zhang, M., Zhang, Y., and Fu, G. (2016). Transition-based neural word segmentation. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, pages 421–431.
- Zheng, X., Chen, H., and Xu, T. (2013). Deep Learning for Chinese Word Segmentation and POS Tagging. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing, EMNLP 2013*, pages 647–657.
- Zipf, G. K. (1949). *Human behavior and the principle of least effort*.
- Zubiaga, A., San Vicente, I., Gamallo, P., Pichel, J. R., Alegria, I., Aranberri, N., Ezeiza, A., and Fresno, V. (2016). TweetLID: A benchmark for tweet language identification. *Language Resources and Evaluation*, 50(4):729–766.
- Zubiaga, A., Vicente, I. S., Gamallo, P., Pichel, J. R., Alegría, I., Aranberri, N., Ezeiza, A., and Fresno, V. (2014). Overview of TweetLID: Tweet Language Identification at SEPLN 2014. In *Proc. of the Tweet Language Identification Workshop co-located with the 30th Conf. of the Spanish Society for Natural Language Processing, TweetLID@SEPLN 2014*, volume 1228, pages 1–11.

Appendix A

Phonetic algorithms and microtext normalization

As described in Chapter 4, a common framework to tackle the microtext normalization problem consists of two main sequential steps: candidate generation and candidate selection. During the candidate generation step, it is common to resort to mechanisms that exploit similarities between those words in the input and those in the dictionary of the language in order to find possible candidates. Interestingly, it is common for users to abuse the phonetic features of their language in order to shorten or customize the spelling of words. For instance, to an English speaker, the pairs “sumone”-“someone” or “u”-“you” sound the same or quite similarly, which is also the case of the emphasis example “dooooo it” when we do not take into account the prolonged /u/. Therefore, it would be highly desirable to count on a mechanism which could provide us with this kind of phonetic-based matchings, and this is precisely the purpose of the so-called *phonetic algorithms* (Odell and Russell, 1918). Given an input written word, these algorithms obtain phonetic codes that approximately indicate the way it is pronounced in a particular language (i.e., they are *language-dependent*). Thus, these algorithms can be used to match together words that differ in their written form but not that much in their pronunciation, as in the case of our previous examples.

However, contrary to expectations, a strict phonetic matching approach based on grapheme-to-phoneme transcription (Bisani and Ney, 2008) using, for example, International Phonetic Alphabet (IPA) transcriptions (Jurafsky and Martin, 2009, Ch. 7 “Phonetics”), would not be useful in this context, as its output codes would be too specific for our needs. For instance, the processing of “beat”-“bit” or “but”-“bought” would result in slightly different codes accounting for their slightly different pronunciations,¹ thus preventing their

¹This kind of word pairs are formally known as *minimal pairs*.

matching. So, in the context of microtext normalization, the use of a fuzzy phonetic-based matching seems a better choice.

Surprisingly, despite the importance of such phonetic processing of microtexts (Kobus et al., 2008b; Beaufort et al., 2010; Xue et al., 2011; Baldwin et al., 2013; Schulz et al., 2016) and the existence of many phonetic algorithms publicly available for the English language, we have also noticed the lack of any extensive comparison study of the performance of these algorithms for our problem domain. In this context we have decided to conduct our own study to compare the performance of several phonetic algorithms on this task. Our main objective is to facilitate future developers and researchers the choice of the phonetic algorithm to be used in a particular microtext normalization setup. By improving the process of normalization candidate generation, the potential effectiveness of the ulterior selection process and, consequently, that of the whole normalization process should be notably increased. As a result, the input noise introduced into subsequent information processing systems should be greatly reduced.

It should be remembered that, although most of the research work on text normalization has been focusing on English (Baldwin et al., 2015; Han and Baldwin, 2011; Xue et al., 2011), there is also interest in applying it to other languages (see Section 4.5). With this in mind, the decision of using English in our experiments was mostly due to the public availability of both evaluation corpora and ready-to-use implementations for a wide range of phonetic algorithms.

A.1 Phonetic algorithms

In microtext normalization, the candidate generation step needs to account for highly productive texting phenomena (see Section 4.1). This implies that it cannot be handled by traditional edit-distance based approaches alone, as in the case of “nuff”-“enough”, “da”-“the”, or “str8”-“straight”, for example. Hence, the phonetic processing of microtexts turns out to be of key importance in order to obtain meaningful sets of normalization candidates (Beaufort et al., 2010). This task can be accomplished through the use of so-called phonetic algorithms.

A phonetic algorithm transforms an input written word into a phonetic code which roughly indicates the way that term is pronounced in a particular language. It is important to highlight the *approximate* nature of these codes, as its purpose is to match words with *similar* pronunciations, and also the language-dependent aspect of these algorithms.

In this appendix, we study the most popular state-of-the-art phonetic algorithms designed for the English language in the context of the microtext normalization task. It is worth noting that most of them were originally designed for the task of personal name matching, although

it is fair to assume that the phonetic phenomena initially considered would also be useful in matching other types of similarly sounding words. Consequently, it will be interesting to analyze the performance of these algorithms in the task of microtext normalization when generating normalization candidates.

Next, the different phonetic algorithms considered for this study are introduced. The most relevant features of each algorithm, their variations, and the relations existing between them are detailed. Examples of their output encodings are also shown in Tables A.1 and A.2 for comparison. Notice that we have tried to be concise, keeping in mind that the objective of this work is not to study the algorithms themselves but to analyze their behaviour in the context of microtext normalization. Should the reader wish to go into detail about any particular algorithm, appropriate references have been included.

A.1.1 Soundex

Considered the first phonetic algorithm in history, the well-known and widely used Soundex algorithm (Odell and Russell, 1918; Odell, 1956) mainly encodes the consonants of an input word using numerical digits, but also encodes both consonants and vowels in the first position using that same character. The different digits used are related to the place of articulation of the consonant (Jurafsky and Martin, 2009, Ch. 7: “Phonetics”), so the labial consonants “b”, “f”, “p”, and “v” are encoded as the number 1, for example. Before any other rule is applied, the algorithm checks for character sequences represented by the same number and chooses either to retain the first of those characters or the full sequence depending on other characters in the context. Its codes have a fixed length of four characters, padded with trailing 0’s when needed.

Soundex conforms the basis for many other modern phonetic algorithms. These newer algorithms mostly try to address its poor precision, as in the refined version of the original algorithm (Ref. Soundex),² which is also tested here. This revised version does not impose a length limit on the encodings and takes vowels more into consideration for the encoding.

A.1.2 IBM Alpha Search Inquiring System

Popularly known as Alpha SIS (Moore, 1977), this algorithm uses two different conversion tables, one for the first characters of the input word and the other for the rest. The encodings are conformed by a fixed number of 14 numerical digits, appending trailing 0’s as padding for shorter words. If two characters with the same phonetic code are adjacent, only the first one will be used, but a third character would be retained. Alpha SIS also focuses on the

²<http://commons.apache.org/proper/commons-codec/>

Algorithm	nuff	enough	cntrtkxn	contradiction
Soundex	N100	E520	C536	C536
Ref. Soundex	N802	E08040	C38696358	C308690603608
Alpha SIS	0280000000000000	1270000000000000	07214172000000, 06214172000000, 07214167200000, 06214167200000	07214171200000, 06214171200000, 07214161200000, 06214161200000
NYSIIS	NAF	EMAG	CNTRIC	CANTRA
Rev. NYSIIS	NAF	EMAG	CNTRTCXN	CANTRADACTAN
MRA	NF	ENGH	CNTKXN	CNTCTN
Metaphone	NF	ENKH	KNTRTKXN	KNTRTKXN
D. Metaphone	NF,NF	ANK,ANK	KNTR,KNTR	KNTR,KNTR
D-M Soundex	670000	065000	463935	463934
Caverphone 1	NF1111	ANF111	KNTKN	KNTRIK
Caverphone 2	NF11111111	ANF1111111	KNTTKN1111	KNTRTKSN11
Beider-Morse	nuf	iindg,iinog,iinug,iindg, indgx,inag,inog,inogx, iinug,iinugx	kntrrtgzn, kntrtkzn, tzntrrtgzn,tzntrtkzn	kontradition,kontradikt, kontradition,kontradikt, kontradition,kontradikt, kontradition,kontradikt, tsontradition,tsontradition, tsontradition,tsontradition, tsontradition,tsontradition, tsontradition,tsontradition
F. Soundex	N1	E5	C536375	K536395
Lein	N400	E250	C213	C213
Onca	N100	E520	C536	C536
Phonex	N1	A52	C5325	C5363235
Phonix	N5,7	V5,7	C253632,285	K2536323,5
Phonix Comm	N700	v700	C536	K536
RogerRoot	02800	12700	07214	07214
StatCan	NF	ENGH	CNTR	CNTR
Eudex	648518346341351492	15564440312494426116	437444691622462482	1593606864933817373

Table A.1: Example encodings for each of the phonetic algorithms analyzed (1): “nuff”-“enough” and “cntrtkxn”-“contradiction”

Algorithm	da	the	onez	ones
Soundex	D000	T000	0520	0520
Ref. Soundex	D60	T60	00805	00803
Alpha SIS	0100000000000000	0100000000000000	1200000000000000	1200000000000000
NYSIIS	D	TH	ON	ON
Rev. NYIIS	D	T	ON	ON
MRA	D	TH	ONZ	ONS
Metaphone	T	0	ONS	ONS
D. Metaphone	T, T	0, T	ANS, ANS	ANS, ANS
D-M Soundex	300000	300000	064000	064000
Caverphone 1	T11111	T11111	ANS111	ANS111
Caverphone 2	TA111111111	T111111111	ANS11111111	ANS11111111
Beider-Morse	da, di, do	t, ti	Ynis, Ynits, oni, onis, onis, onits, unis	Ynis, oni, onis, onis, unis
F. Soundex	D	T	059	059
Lein	D000	T000	0250	0250
Onca	D000	T000	0500	0500
Phonex	D	T	A52	A5
Phonix	D, 3	T, 3	V5, 8	V, 58
Phonix Comm	D000	T000	v800	v800
RogerRoot	01000	01000	12000	12000
StatCan	D	TH	ONZ	ONS
Eudex	864691128455135232	1008806316530992128	10664523917614514324	10664523917614514196

Table A.2: Example encodings for each of the phonetic algorithms analyzed (2): “da”-“the” and “onez”-“ones”.

encoding of consonantal sounds, although vowels are encoded if appearing at the beginning of a word. It also may return multiple alternative encodings as output.

A.1.3 New York State Identification and Intelligence System

Commonly known as NYSIIS (Taft, 1970), its encoding procedure makes use of a small letter alphabet instead of numerical digits. Its more complex ruleset with respect to the Soundex algorithm allows for the processing of notable character n-grams such as “sch” or “rd”, performing different actions depending on their context characters being vowels or not, and on characters being at the end of the word. The first character of the word is maintained as-is while the rest of the vowels are replaced with the letter “a”. Finally, the code is truncated to its first six characters.

A revised version of this algorithm (Rev. NYSIIS), which adds new rules in order to obtain higher precision codes, is also available.³

A.1.4 Match Rating Approach

Usually referred to as MRA for short, in this case we are not only talking about a phonetic algorithm but also about a particular comparison scheme for the phonetic codes (Moore, 1977). The encoding rules are quite simple: delete all vowels—except the first one if the word begins with it—, remove character repetitions, and reduce the length of the code to six by using the first and last three characters only. However, the complexity of the system lies in the comparison rules. For encoded strings with a length difference less than three, a matching threshold value is first calculated using a table. Then, using a forward and backward pass over the codes of the strings to be compared, the number of distinct characters between them is obtained. Finally, the encoded strings are considered similar depending on this number being equal or greater than the given threshold value.

A.1.5 Metaphone

The Metaphone algorithm is employed in the popular aspell spell checker (Philips, 1990). Its set of contextual rules maps between n-grams of characters from the source to n-grams of characters from an alphabet of sixteen consonantal symbols. Again, the main focus of this algorithm is on encoding consonantal sounds, while vowels are included only if appearing in the first position of the word. No limit on code length is imposed this time.

³<http://www.markcrocker.com/rexxtipsntricks/rxtt28.2.0482.html>

A.1.6 Double Metaphone

With respect to the original Metaphone, the Double Metaphone algorithm (D. Metaphone) takes into account several spelling peculiarities from different languages, including English, and outputs two alternative encodings of the input word (Philips, 2000).

Although an even newer iteration in the Metaphone family exists, named Metaphone 3, unfortunately its source code is not freely available and for this reason has not been included in this study.

A.1.7 Daitch-Mokotoff Soundex

The so-called Daitch-Mokotoff Soundex algorithm (D-M Soundex) constitutes an improvement on the original Soundex seeking to improve its precision when dealing with Slavic and Yiddish surnames.⁴ The most notable differences with the original algorithm are the length of the codes, up to six characters long; the first character of the word being encoded as the rest, handling some specific character n-grams as a unit; and, lastly, the possibility of outputting multiple possible codes instead of a single one.

A.1.8 Caverphone

Designed by Hood (2002) to match common names from New Zealand, this algorithm (Caverphone 1) performs recursive substitutions and deletions on the original word following a set of rules mostly dealing with character n-grams. In its last steps, it appends a padding of 1's to the code to finally trim its length down to six characters, which is the fixed length for all codes.

Hood (2004) revised his algorithm later (Caverphone 2) by adding a few extra rules and increasing the length of the encodings to ten characters.

A.1.9 Beider-Morse

The main goal of the Beider-Morse algorithm (Beider, 2008) is to reduce the large amount of false positives usually returned by Soundex. This problem is managed by first determining the language of the input text so that a particular set of rules can be used accordingly. If the language cannot be determined, a generic set of rules is applied instead. A set of common rules is also applied after such language-specific or generic rules. These common rules account for final devoicing and regressive assimilation of consonants. Moreover, the alphabet of this system is based on the IPA (Jurafsky and Martin, 2009, Ch. 7 “Phonetics”),

⁴<http://www.avotaynu.com/soundex.html>

which is then simplified in order to merge together symbols with very similar sounds and to make it easier to write the codes using a standard keyboard. Finally, as with other improvements on the Soundex algorithm, it takes into account frequent character n-grams, code length is not limited, has a better support for vowels, and outputs multiple possible encodings.

A.1.10 Fuzzy Soundex

This variation of the Soundex algorithm proposed by Holmes and McCabe (2002) and named Fuzzy Soundex (F. Soundex), employs two mapping tables in two subsequent stages. In the first, a table with mappings between character n-grams is used. Then, for the second stage, another table with mappings between individual characters and numerical digits is used to obtain the final output code, which is not restricted in length.

A.1.11 Lein

A simple variation of the Soundex, the Lein algorithm differs from the original one only in its conversion table (Lynch and Arends, 1977).

A.1.12 Onca

The Onca algorithm (Gill and Baldwin, 1987; Gill et al., 1993) merely consists of a two-step application of the NYSIIS and the Soundex algorithms previously described in Sections A.1.3 and A.1.1, respectively. According to its authors, the new algorithm overcomes the low precision of pure Soundex while retaining its 4-character format.

A.1.13 Phonex

Phonex is a Soundex-like algorithm which aims for a greater coverage of common orthographic variations (Lait and Randell, 1996). It is also influenced by the Metaphone method, hence its name. After removing trailing “s” characters, it uses two sets of rules: one for encoding leading characters and the other for the rest of the word. As in the original Soundex, 4-character codes are obtained as output.

A.1.14 Phonix

Not to be confused with the previous Phonex algorithm, there are two variants or interpretations of the Phonix algorithm. Its original interpretation (Phonix), described by Gadd (1988, 1990), specifies a set of rules to encode the input and target words as well as to

obtain a matching code, called *ending-sound*, for each one of them. It then specifies another set of matching rules regarding the word encodings and ending-sounds previously obtained, which allows for the distinction between three different categories of matches: most-likely, less-likely, and least-likely matches (to be referred to in this work as $\text{Phonix}_{\text{most}}$, $\text{Phonix}_{\text{less}}$, and $\text{Phonix}_{\text{least}}$, respectively). However, the common interpretation of the algorithm (Phonix Comm) skips the part concerning the ending-sound and operates in a similar way to the original Soundex algorithm.

A.1.15 Roger Root

The Soundex-like Roger Root algorithm (Lynch and Arends, 1977) produces 5-numerical codes using two conversion tables: one for the first characters of a word and the second one for the rest.

A.1.16 Census Modified Statistics Canada

Commonly known as StatCan for short, it is a very simple phonetic algorithm which preserves the first characters, deletes the remaining vowels and y's, collapses identical adjacent characters, and truncates the result to four characters (Lynch and Arends, 1977).

A.1.17 Eudex

The Eudex algorithm⁵ encodes words in a way that exposes the differences in their pronunciations, by calculating the Hamming distances between their codes. It returns an 8-byte array as output and makes use of four different conversion tables: two for ASCII and C1 (Latin Supplement) characters at the initial position of a word, and another two similar ones for the remaining characters. These tables were obtained using the IPA classifications of consonants and vowels, encoding the sound articulation features of each symbol into a binary format. This encoding has the property that similarly pronounced symbols correspond to codes with a small Hamming distance, but also highlighting the differences between them in the same way. For this reason, its author suggests using a similarity function that matches codes with a Hamming distance below a given threshold value.

⁵<https://github.com/ticki/eudex>

A.2 Implementation of the phonetic algorithms

In order to avoid unnecessary effort and make their future use by developers and researchers easier, instead of reimplementing from scratch the specifications for the algorithms studied, we downloaded and evaluated implementations of them that are publicly available on the Internet. For each downloaded implementation, its compliance with the original specification of the corresponding algorithm was tested. If the implementation did not comply, alternatives were sought or, when necessary, the source code was modified accordingly.⁶

After having performed this selection process, these are the open source projects from which the algorithm implementations to be used were obtained:

- Apache Commons Codec package:⁷ in the case of the implementations of the Soundex, Refined Soundex, Daitch-Mokotoff Soundex, Beider-Morse, Caverphone 1, Caverphone 2, MRA, and Double Metaphone algorithms.
- talisman package:⁸ Alpha SIS, Eudex, Fuzzy Soundex, Lein, Onca, Phonex, Roger Root, StatCan, Metaphone, and NYSIIS algorithms.
- stringmetric package:⁹ Refined NYSIIS algorithm.
- phonetic_search package:¹⁰ Phonix algorithm.

A.3 Evaluation

As stated before, this work intends to study the behaviour of the different phonetic algorithms available for English in the context of candidate generation for microtext normalization tasks. The criterion assumed for this purpose states that, for each input OOV, the best algorithms should return as output the smallest possible candidate set which still contains the corresponding equivalent normalized word.

A.3.1 Evaluation corpora

Seeking reproducibility, we decided to use the same two lexical normalization dictionaries which were made publicly available in the W-NUT 2015 shared task 2 (Baldwin et al.,

⁶All source code produced during this study is available at <http://www.grupocole.org/software/VCS/phon>.

⁷<http://commons.apache.org/proper/commons-codec/>

⁸<http://github.com/Yomguithereal/talisman>

⁹<http://github.com/rockymadden/stringmetric>

¹⁰http://github.com/olsgaard/phonetic_search

Dictionary	$ chars/word $	$ entries $	$ OOV $
utdallas	5	3,974	47
unimelb	7	41,181	96
canonical	8	165,458	–

Table A.3: Evaluation corpora statistics. The column $|chars/word|$ indicates for the first two rows, corresponding to the lexical normalization dictionaries, the average number of characters per non-standard word; for the last row, corresponding to the canonical dictionary, it indicates the average per standard word. Column $|entries|$ indicates the number of lines in each evaluation dataset. In the case of the first two rows, $|OOV|$ shows the number of standard words not included in the canonical lexicon used.

2015). These dictionaries, utdallas and unimelb, consist of text files conformed by lists of non-standard words obtained from real-world microtexts and their corresponding standard equivalents. It must be noted that non-standard words from these lists are affected by a wide range of texting phenomena, not limited to the phonetic phenomena in which this work is interested. For instance, consider the graphemic substitution in “5o” (“so”) or the abbreviation “2nd” (“second”). Despite this, the full datasets were used in the evaluation, thus resorting to error analysis to account for those instances missed by the phonetic algorithms.

In a similar way, the canonical English dictionary made available by the W-NUT 2015 organization for the task (from now on *canonical*) was used. This dictionary is the list of words from which the normalization candidates are retrieved using the phonetic algorithms. Some simple statistics of these datasets are shown in Table A.3.

A.3.2 Experimental methodology

In order to obtain the normalization candidates for each non-standard word of the evaluation corpora, the following procedure was applied:

1. Before the normalization process, create the phonetic lookup dictionary of key-value pairs (phonetic code, set of words) corresponding to each phonetic algorithm. Each pair groups all the words from the canonical dictionary with the same phonetic code for a particular algorithm.
2. During the normalization process, and for a given phonetic algorithm, match the phonetic codes obtained from the non-standard words of the evaluation corpora with these codes (keys) of the corresponding phonetic dictionaries. If a match is found, retrieve its corresponding set of words (values), which now constitute its

normalization candidates. When multiple alternative codes are available for the same non-standard input word, the final candidate set is formed by the union of those partial sets obtained with each alternative code.

Additionally, in the case of the MRA and Phonix algorithms, their particular lookup procedures (MRA_{custom} , $Phonix_{\text{most}}$, $Phonix_{\text{less}}$, and $Phonix_{\text{least}}$, respectively), previously explained in Section A.1, were also considered. Similarly, multiple results were obtained for the Eudex algorithm by varying the Hamming distance threshold value: $Eudex$, $Eudex_5$, $Eudex_{10}$, and $Eudex_{15}$ for threshold values 0 (i.e., perfect matching), 5, 10, and 15, respectively. In these cases, the results for the general and specialized lookup procedures were obtained.¹¹

Regarding the original Phonix algorithm in particular, three different sets of results were distinguished based on the likelihood of the candidates indicated by the lookup procedure: (1) one set containing the most likely candidates ($Phonix_{\text{most}}$), (2) one including the most and less likely candidates ($Phonix_{\text{less}}$), and (3) one which also adds the least likely candidates ($Phonix_{\text{least}}$). Moreover, results obtained using the alternative interpretation of this algorithm are also included ($Phonix_{\text{Comm}}$).

Precision, recall, and F1 metrics, over a list of words from a particular evaluation dataset, are used to measure the performance of the phonetic algorithms. In the present application context, precision (P) is defined as the mean of the ratio of correct candidates¹² over the total number of candidates retrieved for each word w , over the total number of words:

$$P = \frac{\sum_{w \in \text{words}} \frac{|\text{hits}_w|}{|\text{candidates}_w|}}{|\text{words}|} \quad (\text{A.1})$$

In the case of recall (R), it is calculated as the number of times when the correct candidate was among the set of normalization candidates retrieved for each word over the total number of words:

$$R = \frac{|\text{hits}|}{|\text{words}|} \quad (\text{A.2})$$

Finally, $F1$ score ($F1$) is defined in the usual way by aggregating precision and recall, as already shown in Equation 4.1 in Section 4.2.

Algorithm	$ hits $	$avg cands $	P	R	$F1$
Eudex	1,376	5.6	11.7	34.6	17.5
MRA	1,498	9.2	11.3	37.6	17.4
Metaphone	2,071	26.2	7.8	52.1	13.7
Beider-Morse	1,636	25.5	8.0	41.1	13.4
StatCan	2,225	21.4	7.0	55.9	12.4
Ref. Soundex	1,569	15.4	7.3	39.4	12.3
Eudex ₅	1,800	94.1	4.7	45.2	8.5
Rev. NYSIIS	1,416	27.4	4.7	35.6	8.3
Phonix _{most}	1,681	36.5	4.4	42.2	8.0
F. Soundex	2,160	41.8	4.3	54.3	8.0
NYSIIS	1,410	17.6	4.3	35.4	7.8
Caverphone 2	2,039	69.0	3.9	51.3	7.3
Caverphone 1	2,246	98.2	3.1	56.5	6.0
Eudex ₁₀	2,076	310.5	2.5	52.2	4.8
D-M Soundex	2,154	96.1	2.4	54.2	4.6
Alpha SIS	2,359	241.8	2.2	59.3	4.2
Phonix _{less}	2,191	141.6	2.0	55.1	4.0
Eudex ₁₅	2,199	551.7	1.6	55.3	3.2
Roger Root	2,516	133.6	1.6	63.3	3.2
Soundex	2,469	80.6	1.5	62.1	3.0
D. Metaphone	2,395	84.4	1.4	60.2	2.8
Lein	2,471	101.6	1.3	62.1	2.6
Phonix Comm	2,453	195.0	1.1	61.7	2.3
Onca	2,396	109.2	1.0	60.2	2.0
Phonex	2,656	266.4	0.9	66.8	1.9
Phonix _{least}	2,332	611.1	0.6	58.6	1.3
MRA _{custom}	3,695	15,270.3	0.0	92.9	0.0

Table A.4: Results for the utdallas dictionary, ranked by F1.

A.3.3 Results and discussion

Tables A.4 and A.5 show the results obtained for each algorithm, sorting them by their corresponding F1 scores. For each entry, the second column ($|hits|$) indicates the number of instances from the corpora for which the algorithm could provide the correct answer. The third column ($avg |cands|$) shows the average number of normalization candidates returned for each OOV. The rest of the columns contain the values obtained for the evaluation metrics used; from left to right: precision, recall, and F1 score. As it can be seen, the figures obtained are quite low, mainly due to the low precision achieved in most of the cases.

¹¹For the MRA and Eudex algorithms, it is worth mentioning the relatively high computational cost of their particular lookup procedures. Under their current implementations, they may be suitable for quick checks between pairs of words but not for the extensive dictionary-wide checks required in the current context.

¹²In this case, the number of correct candidates will be either 1 or 0.

Algorithm	hits	avg cands	P	R	F1
MRA	17,485	6.0	17.1	42.4	24.4
Eudex	16,150	3.7	17.4	39.2	24.1
Metaphone	22,448	16.6	13.6	54.5	21.8
Ref. Soundex	18,693	10.5	12.6	45.3	19.7
Beider-Morse	18,307	13.8	12.3	44.4	19.3
Rev. NYSIIS	17,844	16.2	9.1	43.3	15.0
F. Soundex	23,694	28.4	8.3	57.5	14.5
Eudex ₅	18,498	42.2	8.5	44.9	14.3
StatCan	28,632	29.9	7.6	69.5	13.8
Phonix _{most}	19,039	23.4	8.1	46.2	13.7
Caverphone 2	21,580	43.4	7.5	52.4	13.2
NYSIIS	21,094	17.0	6.6	51.2	11.8
Caverphone 1	24,227	61.6	6.2	58.8	11.2
Eudex ₁₀	19,988	151.0	5.4	48.5	9.7
Alpha SIS	26,064	145.9	4.8	63.2	8.9
D-M Soundex	24,470	64.7	4.5	59.4	8.4
Phonix _{less}	23,093	83.4	4.1	56.0	7.7
Eudex ₁₅	20,892	290.4	3.8	50.7	7.1
Roger Root	28,928	98.1	2.8	70.2	5.4
Phonex	28,048	200.5	2.0	68.1	4.0
D. Metaphone	26,253	90.0	2.0	63.7	3.9
Soundex	29,557	95.0	1.8	71.7	3.5
Phonix Comm	26,937	134.8	1.8	65.4	3.5
Phonix _{least}	24,434	387.9	1.7	59.3	3.4
Onca	28,601	109.2	1.4	69.4	2.8
Lein	29,633	134.9	1.2	71.9	2.5
MRA _{custom}	38,768	20,767.4	0.0	94.1	0.0

Table A.5: Results for the unimelb dictionary, ranked by F1.

Accordingly, those algorithms with the highest precision scores end up at the top of this ranking, with Eudex, MRA, and Metaphone outperforming the rest of them.

It is interesting to note that the performance of the MRA algorithm decreases ostensibly when used with its particular lookup procedure (MRA_{custom}). This procedure tries to enlarge the matching window of its otherwise high-precision codes. Thus, it is reasonable to assume that using a large canonical dictionary, as in this case, renders this procedure of little use, as it was designed to work with much smaller lists. Moreover, Phonix_{least}, that is, Phonix using least-likely matches, suffers from the same problem.

At this point, it should be noted that, when developing a microtext normalization system, it may be interesting to gain some recall while sacrificing some precision in exchange. This is due to the fact that a low recall tends to impose an upper limit on the overall performance of the system: there is no way of selecting the right IV unless it appears among the generated

Algorithm	hits	avg cands	P	R	F1
MRA _{custom}	3,695	15,270.3	0.0	92.9	0.0
Phonex	2,656	266.4	0.9	66.8	1.9
Roger Root	2,516	133.6	1.6	63.3	3.2
Lein	2,471	101.6	1.3	62.1	2.6
Soundex	2,469	80.6	1.5	62.1	3.0
Phonix Comm	2,453	195.0	1.1	61.7	2.3
Onca	2,396	109.2	1.0	60.2	2.0
D. Metaphone	2,395	84.4	1.4	60.2	2.8
Alpha SIS	2,359	241.8	2.2	59.3	4.2
Phonix _{least}	2,332	611.1	0.6	58.6	1.3
Caverphone 1	2,246	98.2	3.1	56.5	6.0
StatCan	2,225	21.4	7.0	55.9	12.4
Eudex ₁₅	2,199	551.7	1.6	55.3	3.2
Phonix _{less}	2,191	141.6	2.0	55.1	4.0
F. Soundex	2,160	41.8	4.3	54.3	8.0
D-M Soundex	2,154	96.1	2.4	54.2	4.6
Eudex ₁₀	2,076	310.5	2.5	52.2	4.8
Metaphone	2,071	26.2	7.8	52.1	13.7
Caverphone 2	2,039	69.0	3.9	51.3	7.3
Eudex ₅	1,800	94.1	4.7	45.2	8.5
Phonix _{most}	1,681	36.5	4.4	42.2	8.0
Beider-Morse	1,636	25.5	8.0	41.1	13.4
Ref. Soundex	1,569	15.4	7.3	39.4	12.3
MRA	1,498	9.2	11.3	37.6	17.4
Rev. NYSIIS	1,416	27.4	4.7	35.6	8.3
NYSIIS	1,410	17.6	4.3	35.4	7.8
Eudex	1,376	5.6	11.7	34.6	17.5

Table A.6: Results for the utdallas dictionary, this time ranked by recall.

candidates. Moreover, it is also reasonable to assume that the candidate generation step will be connected to a capable candidate selection step afterwards. Taking this into account, Tables A.6 and A.7 show again the results obtained in our experiments, but this time ranked by their recall figures. The resulting rankings are now very different, providing us with new insights about the performance of the analyzed phonetic algorithms.

Overall, these results indicate that the StatCan, Metaphone, Soundex or Roger Root algorithms would be good choices for candidate generation in a microtext normalization system, depending on the level of compromise sought between precision and recall. This way, the Metaphone algorithm shines when precision is needed, as it is among the top-three algorithms in the F1 classification (see Tables A.4 and A.5) while it does not fall to the bottom of the recall rankings (Tables A.6 and A.7) as in the case of MRA or Eudex, the other two algorithms with the highest F1. On the other hand, Soundex and Roger Root stand

Algorithm	hits	avg cands	P	R	F1
MRA _{custom}	38,768	20,767.4	0.0	94.1	0.0
Lein	29,633	134.9	1.2	71.9	2.5
Soundex	29,557	95.0	1.8	71.7	3.5
Roger Root	28,928	98.1	2.8	70.2	5.4
StatCan	28,632	29.9	7.6	69.5	13.8
Onca	28,601	109.2	1.4	69.4	2.8
Phonex	28,048	200.5	2.0	68.1	4.0
Phonix Comm	26,937	134.8	1.8	65.4	3.5
D. Metaphone	26,253	90.0	2.0	63.7	3.9
Alpha SIS	26,064	145.9	4.8	63.2	8.9
D-M Soundex	24,470	64.7	4.5	59.4	8.4
Phonix _{least}	24,434	387.9	1.7	59.3	3.4
Caverphone 1	24,227	61.6	6.2	58.8	11.2
F. Soundex	23,694	28.4	8.3	57.5	14.5
Phonix _{less}	23,093	83.4	4.1	56.0	7.7
Metaphone	22,448	16.6	13.6	54.5	21.8
Caverphone 2	21,580	43.4	7.5	52.4	13.2
NYSIIS	21,094	17.0	6.6	51.2	11.8
Eudex ₁₅	20,892	290.4	3.8	50.7	7.1
Eudex ₁₀	19,988	151.0	5.4	48.5	9.7
Phonix _{most}	19,039	23.4	8.1	46.2	13.7
Ref. Soundex	18,693	10.5	12.6	45.3	19.7
Eudex ₅	18,498	42.2	8.5	44.9	14.3
Beider-Morse	18,307	13.8	12.3	44.4	19.3
Rev. NYSIIS	17,844	16.2	9.1	43.3	15.0
MRA	17,485	6.0	17.1	42.4	24.4
Eudex	16,150	3.7	17.4	39.2	24.1

Table A.7: Results for the unimelb dictionary, this time ranked by recall.

as good choices if we want to maximize recall while not hurting precision in excess, as in the case of the Lein algorithm, and after having dismissed MRA_{custom} for the reasons given above. Finally, Statcan strikes the best balance in precision and recall, as we can see in the good overall positions obtained in both rankings.

It is interesting to note that a phonetic algorithm can also be considered as a *locality-sensitive hashing algorithm*. These are algorithms that maximize the probability of a *collision* for similar items; this is, that the output for slightly different input elements is the same. In terms of phonetic algorithms, this is equivalent to maximizing the probability of two similar-sounding words having the same phonetic code. From this perspective, the average number of candidates retrieved by a phonetic algorithm is in fact directly proportional to the compression ratio of the locality-sensitive hashing it is performing.

During the error analysis, particular attention was paid to those instances from the

corpora where none or just a few of the algorithms were able to provide the correct answer (in our context, a *hit*). In the case of the zero-hits list, this is, instances for which no algorithm provided a correct answer, it contains examples such as “baddest”-“worst” or “5ayin”-“saying”, variations which do not correspond to phonetic phenomena and are thus outside the scope of this work. A few OOV words such as “thankyou” or “sheesh” also appear in this list. However, other interesting examples not directly supported by any phonetic algorithm can also be found, as is the case of the so-called *number homophones* (Thurlow and Brown, 2003) such as “2morrow”-“tomorrow” or “4got”-“forgot”. This is to be expected since this texting phenomenon, while also being a phonetic substitution, plays with the pronunciation of digits, and phonetic algorithms usually work only with letters. Furthermore, these examples would be easily supported in a microtext normalization system by preprocessing the input and spelling such numbers before applying the phonetic algorithm.

On the remaining lists, it is interesting to note the presence of instances where some phonetic algorithms gave a correct answer despite them not being designed to deal with that particular case. We can mention here those algorithms whose generated codes are shortened to a specific maximum length. Because of this shortening, such algorithms are able to cope with any misspellings occurring in those parts of the original word which are not finally translated into the phonetic code. For example, the MRA algorithm gives the correct answer “performance” for the input word “perfomence” as the second “r” is not encoded in PRFMNC.

Finally, considering the listing of non-standard orthographic forms from (Thurlow and Brown, 2003) and the lists of errors obtained in these experiments, we can conclude the following:¹³

- *Shortenings* (e.g., “dec”-“december”, “epi”-“episode”) are difficult to account for as they usually remove whole chunks of characters from the original standard word, including consonants. The difficulty here is that consonants are the main building blocks for most phonetic codes and, consequently, important information for the algorithms has been stripped from the term to be normalized. This may be solved by adding an extra step in the normalization pipeline which would make use of other word similarity metrics such as the longest common subsequence, overlap coefficient, or cosine distance (Okazaki and Tsujii, 2010).
- *Contractions* (e.g., “frm”-“from”, “lov”-“love”) can be dealt with as they generally include the most meaningful details of the standard word, which mainly consist of its consonants.

¹³These lists are available at <http://www.grupocole.org/software/VCS/phon>

- *g-Clippings* (e.g., “losin”-“losing”, “frikin”-“freaking”) are a simple but problematic texting phenomenon for many of the algorithms studied. Most of them are able to incidentally handle it through their trimming of the output codes. In this way, in sufficiently long words the final “g” consonant is not taken into account for the encoding, hence bypassing the need for specific rules for managing it. On the other hand, it is worth noting that algorithms like Beider-Morse or Phonex, with a wide range of encoding rules, do handle this particular scenario effectively.
- Handling other types of *clippings* is possible for a wide range of algorithms if they perform some kind of clipping themselves during encoding (e.g., “luk”-“luck”, “metallic”-“metallic”) or when the clipping only affects vowels or non-pronounceable characters (e.g., “ther”-“their”, “oclock”-“o’clock”).
- *Acronyms* and *initialisms* (e.g., “omg”-“oh my god”, “lol”-“laughing out loud”) are not included in the evaluation datasets. In any case, they are considered to be outside the scope of phonetic phenomena and, consequently, are not consistently supported by any phonetic algorithm. Nevertheless, they may be normalized using specialized dictionaries (see Section 4.1).
- In the case of *homophony* phenomena, *letter homophones* (e.g., “b”-“be”, “r”-“are”) are generally supported, whereas, as noted earlier, *number homophones* (e.g., “4got”-“forgot”, “in2”-“into”) are not.
- Other misspellings, typos, non-conventional spellings, and accent stylization (e.g., “acount”-“account”, “basterds”-“bastards”, “huni”-“honey”, “dat”-“that”) can be handled by most algorithms as long as the sequence of consonants was preserved in the resulting non-standard word. In the case of examples as “eva”-“ever” or “ova”-“over”, they are only supported by the lowest precision algorithms or by those having specific rules for dealing with such phenomena.

A.4 Related work

As already mentioned in the introductory section of this appendix, phonetic algorithms have been traditionally used for personal name matching. In the literature, the design of a new phonetic algorithm tends to be coupled with a comparative study with the contemporary state of the art in order to highlight its strengths. However, in most cases it is not an exhaustive study, since only a small part of the existing phonetic algorithms are considered, and also because it focuses on the name-matching task, as in the case of (Hood, 2004),

(Beider, 2008), (Holmes and McCabe, 2002), and (Parmar and Kumbharana, 2014). Some exceptions in which a broader comparative study was performed are the studies made by Lynch and Arends (1977), and Lait and Randell (1996).

There are also purely comparative studies of name-matching algorithms where a wider range of techniques were considered, including phonetic algorithms and other types of similarity metrics. This is the case of (Christen, 2006), (Snae, 2007), (Bilenko et al., 2003), (Branting, 2003), or (Gálvez, 2006). In their conclusions, these authors propose a series of recommendations to select the right approach or algorithm for a particular setup or domain, much in the same vein of the present work. However, even in these cases, they only compare a small subset of the phonetic algorithms available and, more importantly, they do so in the context of a different task.

Moving on from the name-matching task, the work of Pinto et al. (2012) enters the domain of microtexts. However, the authors only provide a comparative study between the original Soundex and their proposed improvements, which are not publicly available. Furthermore, their case is not that of microtext normalization either.

Finally, it is worth mentioning the recent work by Fuentes et al. (2016), which compares notably more phonetic algorithms than previous work although in a different scenario: word recognition in Spanish microtext mining. This may be the most similar work to the present contribution, although it focuses on precision and accuracy metrics, Spanish microtexts and, once again, it is not tailored to the microtext normalization task.

In general, these comparative studies mostly use precision and F1 metrics for their quantitative analysis, defining them accordingly for the task at hand. Some of them also give qualitative insights in order to exemplify the behaviour of the phonetic algorithms in each particular use case. On our end, we follow the common trend of using precision and F1, while also explicitly including the recall and other interesting measures such as the average number of normalization candidates retrieved by each algorithm. Likewise, we perform a qualitative study based on a classification of the texting phenomena. Overall, we present a wider comparison of phonetic algorithms than in previous work, focusing on the task of microtext normalization, while using tried and tested methods for performance measurements.

A.5 Conclusions

In this appendix we have evaluated a wide range of English state-of-the-art phonetic algorithms within the context of generating normalization candidates in microtext normalization tasks. This work constitutes, to the best of our knowledge, the only wide-range comparative study of its kind. We perform both qualitative and quantitative analyzes—adapting the

usual performance metrics to our domain—in order to identify the most salient properties of these phonetic algorithms and their appropriateness for the task at hand. We expect that the results obtained will be of help to both developers and researchers of this field when building new intelligent systems for microtext information processing.

Seeking reproducibility and simplicity by using currently existing implementations and publicly available datasets, we have measured the performance of these algorithms, and their strengths and weaknesses were analyzed to identify the best algorithms in terms of a compromise between precision and recall. In the end, we have found that the choice of phonetic algorithm depends heavily on the capabilities of the subsequent candidate selection mechanism to be applied within the microtext normalization pipeline. The faster it can make the right selections among big enough input sets of candidates, the more we can sacrifice in terms of the precision of the phonetic algorithm in favour of coverage. This would be desirable since when obtaining a low number of normalization candidates, the system would run the risk of imposing an upper limit to its overall performance at this early stage.

Appendix B

Learning unsupervised and semi-supervised cross-lingual word embeddings

Despite the promising results reported in the literature, it remains unclear under which conditions the multilingual embedding methods described in Chapter 7 succeed, including our proposed postprocessing step called Meemi. For example, Artetxe et al. (2017) and Conneau et al. (2018a) achieved promising results in the word translation task (i.e., bilingual lexicon induction), but their experiments relied most of the times on the availability of high-quality monolingual source corpora, namely Wikipedia. This is also the case for other analyses on cross-lingual embeddings performance by Søgaard et al. (2018) and Glavaš et al. (2019). As an exception, Artetxe et al. (2018b) used Web crawled data to test their unsupervised framework. Moreover, models trained on social media corpora remain largely untested, even though previous work has shown that these models help obtain better performance in social-media-centered tasks (Tang et al., 2014; Godin et al., 2015; Yang et al., 2018). Similarly, it is still unclear how well existing methods would perform on language pairs with significant differences in their morphology (e.g., English-Finnish, the latter being an agglutinative language) which has not been considered in Conneau et al. (2018a); different word orderings, given the difficulties found by Ahmad et al. (2018) in cross-lingual transfer; or different alphabets (e.g., Latin-Cyrillic).

In this chapter, we broaden the empirical evaluation of state-of-the-art techniques for learning cross-lingual embeddings¹ by using several types of training corpora, various amounts of supervision, languages from different families, and different alignment strategies

¹Although we focus on the *bilingual* case, we refer more generally to the obtained embeddings as *cross-lingual*.

in three different tasks. The results obtained cast some doubt on the view that high-quality cross-lingual embeddings can always be learned without much supervision. Finally, we also outline a number of suggestions on how to train cross-lingual embeddings under different conditions such as the amount of training data or the type of corpora.

B.1 Variables

Our main goal is to explore how the choice of corpora, supervision signals, and languages impacts on the performance of cross-lingual word embedding models. In addition, we also identify some other variables which were not directly studied in this chapter.

B.1.1 Monolingual corpora

It is reasonable to assume that accurate word-level alignments will be easier to obtain from corpora from similar domains with similar vocabularies and register. Wikipedia has become the mainstream monolingual source in cross-lingual word embedding training so far (Artetxe et al., 2017; Conneau et al., 2018a). It provides a particularly reliable bilingual signal because of the highly comparable nature of Wikipedia corpora from different languages. As we will see, this makes finding high-quality alignments considerably easier.

In our analysis we use three different types of corpora: Wikipedia² (as a prototypical example of comparable monolingual corpora), Web corpora from different sources³ (as a prototypical example of non-comparable but generally high-quality corpora) and social media⁴ (as a prototypical example of non-comparable and noisy text). Statistics of these corpora are provided in Table B.1.⁵

B.1.2 Bilingual supervision

Early approaches for learning bilingual embeddings relied on large parallel corpora (Klementiev et al., 2012; Luong et al., 2015), which limited their applicability. More recent approaches instead rely on small bilingual dictionaries as only source of bilingual supervision. In fact, some methods remove the need for a user-supplied bilingual dictionary

²All Wikipedia text dumps were downloaded from the Polyglot project (Al-Rfou et al., 2013): <https://sites.google.com/site/rmyeid/projects/polyglot>

³The sources of the Web corpora were: UMBC (Han et al., 2013b), 1-billion (Cardellino, 2016), itWaC and sdeWaC (Baroni et al., 2009), Hamshahri (AleAhmad et al., 2009), and Common Crawl downloaded from <http://www.statmt.org/wmt16/translation-task.html>.

⁴These corpora were downloaded using a Twitter crawler, at different dates between 2015 and 2018.

⁵Due to some restrictions, we were not able to compile a reliable and large enough Twitter corpus for Russian.

Domain	Corpus	Language	Size	Words
Wikipedia	Wiki _{en}	English	1.7B	12.0M
	Wiki _{es}	Spanish	407M	3.4M
	Wiki _{it}	Italian	338M	3.3M
	Wiki _{de}	German	605M	7.4M
	Wiki _{fi}	Finnish	68M	2.8M
	Wiki _{ru}	Russian	313M	5.4M
	Wiki _{fa}	Farsi	48M	1.0M
Web corpora	UMBC	English	3.5B	8.1M
	1-billion	Spanish	1.9B	5.5M
	itWaC	Italian	1.3B	4.2M
	sdeWaC	German	438M	1.5M
	Comm-crawl	Finnish	2.8B	1.8M
	Comm-crawl	Russian	1.1B	18.8M
	Hamshahri	Farsi	167M	0.8M
Social media	Twitter _{en}	English	294M	5.5M
	Twitter _{es}	Spanish	144M	3.3M
	Twitter _{it}	Italian	63M	1.6M
	Twitter _{de}	German	114M	2.3M
	Twitter _{fi}	Finnish	29M	1.7M
	Twitter _{fa}	Farsi	90M	1.0M

Table B.1: Statistics of the corpora used to train monolingual word embeddings: size (total number of tokens) and words (number of unique tokens).

altogether (Conneau et al., 2018a; Artetxe et al., 2018b), relying instead on synthetic dictionaries that are obtained fully automatically. In our experiments we consider a wide range of signals, including no supervision as well as automatically generated dictionaries of identical words. In the latter case, we rely on the assumption that identical words that occur in both of the monolingual corpora tend to have the same meaning. While this may seem naive, this strategy has been reported in the literature to perform well in practice (Smith et al., 2017; Søgaard et al., 2018).

B.1.3 Languages

In most previous work, the evaluation of cross-lingual embeddings has been limited to a small set of closely-related languages. For instance, Smith et al. (2017) evaluated their model on the English-Italian pair only, Artetxe et al. (2017) performed their evaluation on three languages, all of which share the same alphabet. Moreover, as the considered language pairs vary from one study to another, the relative performance of different methods for particular types of languages remains unclear. More recently, however, Søgaard et al. (2018) have extended the usual evaluation framework by covering additional Eastern European

languages. Similarly, we have extended the range of covered languages by considering Spanish (ES), Italian (IT), German (DE), Finnish (FI), Farsi (FA), and Russian (RU). In all cases, we have used English (EN) as source language. This set of languages represents not only Indo-European languages (Spanish, Italian, German, Farsi, and Russian), but also an agglutinative language (Finnish and Farsi), as well as languages with non-Latin alphabets (Farsi and Russian, with Arabic and Cyrillic alphabets, respectively).

B.1.4 Other variables

It is worth mentioning that there are several other external factors that may affect the quality of cross-lingual embeddings, beyond the ones considered in this study. For instance, in our experiments we use fastText (Bojanowski et al., 2016) with default values, but the impact of other word embedding models such as word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014) could also be analyzed, in the line of the work by Søgaard et al. (2018). Likewise, all cross-lingual models and the post-processing technique we evaluated are used *as-is* with their default configurations.

B.2 Evaluation

We use two standard tasks for evaluating cross-lingual word embeddings: bilingual dictionary induction and cross-lingual word similarity. In addition, we also consider a downstream application: cross-lingual natural language inference. These tasks have been already introduced in Sections 7.3 and 7.4. For the sake of clarity, in this section we only include the tables with the results for unsupervised and supervised systems using the largest dictionary (i.e., with 8K word pairs). In the same vein, we summarize the most important findings in Section B.3, and then leave the fully detailed result tables which cover different dictionary sizes in the intrinsic tasks for Section B.6.

The systems we compare are two well-known cross-lingual embedding methods which can be used in unsupervised and semi-supervised settings, namely the orthogonal version of VecMap⁶ (Artetxe et al., 2018b) and MUSE⁷ (Conneau et al., 2018a). As seed dictionaries we consider three samples of varying sizes, considering 8K, 1K, and 100 word pairs, to test the robustness of the models regarding the amount of supervision available.⁸ Additionally, we also consider synthetic dictionaries, consisting of identical words that are found in the

⁶Note that in Chapter 7 we have used the purely supervised setting, hence the differences in the results. The codebase is still the same and can be obtained at <https://github.com/artetxem/vecmap>

⁷<https://github.com/facebookresearch/MUSE>

⁸These dictionaries were obtained by splitting the training dictionaries provided by Conneau et al. (2018a).

		Wikipedia																	
Model		Spanish			Italian			German			Finnish			Farsi			Russian		
		$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$
unsup	VecMap	39.6	66.1	72.3	42.7	65.7	71.6	28.6	48.3	54.8	19.6	40.4	48.3	20.5	37.0	42.8	19.5	45.3	54.5
	MUSE	39.3	64.7	71.3	41.6	63.2	69.9	28.3	46.5	53.3	0.0	0.0	0.0	0.0	0.0	0.0	14.9	36.0	46.5
	VecMap	39.6	66.2	72.3	42.6	65.9	71.8	28.6	48.3	54.8	22.4	44.5	52.5	22.8	39.7	46.2	20.0	46.3	55.6
8K	MUSE	39.1	65.4	72.3	41.1	63.3	70.1	27.6	45.9	53.2	19.5	40.4	49.5	19.7	35.4	42	21.3	43.7	52.9
	VecMap	39.3	67.4	73.7	41.6	66.5	72.5	28	47.8	54.8	23.8	48.7	57.0	23.4	41.7	47.7	23.0	49.3	58.3
	MUSE	39.3	67.4	73.7	41.3	66.8	72.8	27.1	46.3	53.9	21.7	45.0	53.6	20.7	38.6	45.1	24.4	50.3	59.3
Web corpora																			
Model		Spanish			Italian			German			Finnish			Farsi			Russian		
		$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$
unsup	VecMap	34.8	60.6	67.0	31.4	53.7	60.7	23.2	42.7	50.2	0.0	0.0	0.0	19.7	34.6	40.4	13.8	30.9	38.6
	MUSE	31.4	51.2	57.7	31.4	51.2	57.7	20.8	38.7	46.6	0.0	0.0	0.0	18.1	32.8	37.8	0.0	0.0	0.0
	VecMap	34.6	60.6	66.9	31.9	54.2	60.4	23.1	42.7	50.5	18.9	40.9	48.8	19.6	35.8	41.4	14.6	31.7	39.6
8K	MUSE	32.5	58.2	65.9	32.5	56.0	63.2	22.4	40.9	48.9	20.0	40.1	48.3	17.4	31.6	37.6	15.5	35.6	44.1
	VecMap	34.5	61.6	67.9	33.6	58.3	65.6	23.7	45.4	53.2	22.3	46.7	55.0	21.7	39.0	43.8	18.2	40.0	47.5
	MUSE	33.9	60.7	68.4	33.8	58.4	65.6	23.7	45.3	52.3	23.0	46.1	54.0	19.3	36.0	41.7	18.7	40.5	49.7
Social media																			
Model		Spanish			Italian			German			Finnish			Farsi					
		$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$	$P@1$	$P@5$	$P@10$			
unsup	VecMap	8.1	16.4	20.4	8.8	17.0	22.3	0.1	0.4	0.5	0.0	0.0	0.0	0.0	0.0	0.0			
	MUSE	0.0	0.0	0.0	7.3	14.5	18.3	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.1			
	VecMap	8.7	16.6	21.6	8.9	17.3	22.4	3.2	6.8	9.5	0.2	0.8	1.2	0.4	1.6	2.0			
8K	MUSE	8.1	17.6	22.7	8	16.4	21.1	2.2	6.0	8.4	0.6	2.2	3.2	1.2	4.5	6.3			
	VecMap	9.8	21.3	26.9	10.6	20.0	25.6	3.7	9.6	13.2	1.3	3.6	5.5	1.8	5.1	7.0			
	MUSE	9.5	20.5	26.3	9.5	19.1	24.5	3.0	7.6	11.1	1.5	4.3	6.4	1.6	5.3	8.1			

Table B.2: Bilingual dictionary induction results using English as source language. Performance measured by $P@k$.

corpora for both languages. Lastly, using those same bilingual dictionaries, we use Meemi to refine the cross-lingual embeddings obtained by VecMap and MUSE. We use the same notation as in Section 7.2.3 to refer to these postprocessed vectors: Meemi (M) where M is either VecMap or MUSE.

B.2.1 Bilingual dictionary induction

As already explained in Section 7.3.1, bilingual dictionary induction consists in automatically obtaining the word translations in a target language for words in a source language. We resort again to cosine similarity to select the nearest neighbors in the cross-lingual embedding space, and to precision at k ($P@k$) as our performance metric. Table B.2 summarizes the results obtained by all comparison systems on the test dictionaries published by Conneau et al. (2018a). Note that the test dictionaries do not overlap with the dictionaries used for training.

B.2.2 Cross-lingual semantic word similarity

Another well-known intrinsic task, cross-lingual semantic word similarity consists in measuring the semantic similarity between pairs of words in different languages, as explained in Section 7.3.1. For the evaluation we make use again of the cross-lingual word similarity datasets of the SemEval 2017 task (Camacho Collados et al., 2017), and the results are reported in terms of Pearson and Spearman correlation. The cross-lingual word similarity results for all the systems are displayed in Table B.3. The languages available for this dataset are English, Spanish, Italian, German, and Farsi, hence Finnish and Russian cannot be evaluated in this task. Also note that in this case we are not interested in measuring the monolingual performance of our embeddings, hence we focus on the cross-lingual variant of this task.

B.2.3 Cross-lingual natural language inference

On this occasion, we rely on natural language inference as our only extrinsic (or downstream) task. As we already explained in Section 7.4.2, it consists in detecting entailment, contradiction, and neutral relations between pairs of sentences. We again train a linear classifier⁹ to obtain the predicted label for each pair of sentences. Likewise, we use the full MultiNLI English corpus (Williams et al., 2018) for training and the Spanish, German, and

⁹The codebase for these experiments is that of SentEval (Conneau and Kiela, 2018).

Wikipedia					
	Model	EN-ES	EN-IT	EN-DE	EN-FA
unsup	VecMap	72.1	70.6	69.3	61.3
	MUSE	72.6	71.2	68.9	6.5
8K	VecMap	71.8	70.6	69.3	61.7
	MUSE	72.6	70.9	68.9	58.7
	Meemi (VecMap)	71.9	70.9	70.3	63.4
	Meemi (MUSE)	72.9	71.9	70.1	62.0
Web corpora					
	Model	EN-ES	EN-IT	EN-DE	EN-FA
unsup	VecMap	70.5	68.8	70.4	33.4
	MUSE	71.6	69.4	70.0	23.8
8K	VecMap	70.6	68.8	70.4	33.5
	MUSE	71.9	70.4	70.2	23.9
	Meemi (VecMap)	70.9	70.0	71.8	39.0
	Meemi (MUSE)	72.3	71.1	72.1	33.0
Social media					
	Model	EN-ES	EN-IT	EN-DE	EN-FA
unsup	VecMap	46.9	51.5	31.2	2.4
	MUSE	10.9	49.7	13.0	4.7
8K	VecMap	47.4	51.8	49.5	30.3
	MUSE	47.6	49.3	48.6	42.2
	Meemi (VecMap)	50.1	53.6	53.8	43.1
	Meemi (MUSE)	50.4	52.5	52.0	46.6

Table B.3: Spearman correlation performance of various cross-lingual word embedding models in the cross-lingual word similarity task.

Russian test sets from XNLI (Conneau et al., 2018b) for testing. Accuracy results are shown in Table B.4.¹⁰

B.3 Analysis

Supervision signals. Unsurprisingly, the best alignments of monolingual spaces tend to be obtained with the largest bilingual dictionaries. The unsupervised variants of VecMap (see Figure B.1) and MUSE attain competitive performance in most cases, especially for comparable corpora where alignments are easier to obtain. However, they struggle in the case of noisy social media corpora and unrelated languages (e.g. both VecMap and MUSE obtain inferior results, close to 0, on both Finnish and Farsi), which challenges conclusions from previous work (Conneau et al., 2018a; Artetxe et al., 2018b). Overall, the results

¹⁰For this task we focused on the better performing embeddings learned from Wikipedia and Web corpora.

Wikipedia				
	Model	EN-ES	EN-DE	EN-RU
unsup	VecMap	49.6	46.3	34.1
	MUSE	48.4	47.4	33.3
8K	VecMap	49.2	46.7	33.4
	MUSE	47.7	47.1	33.1
	Meemi (VecMap)	49.5	47.6	33.8
	Meemi (MUSE)	44.2	46.7	33.3
Web corpora				
	Model	EN-ES	EN-DE	EN-RU
unsup	VecMap	48.5	47.9	33.4
	MUSE	47.7	47.1	33.6
8K	VecMap	48.4	47.5	33.2
	MUSE	47.3	48.6	33.1
	Meemi (VecMap)	47.8	48.6	33.8
	Meemi (MUSE)	47.3	48.2	33.2

Table B.4: Accuracy in the cross-lingual natural language inference task (XNLI) using different cross-lingual word embedding models.

obtained when using social media are clearly inferior, suggesting that there is still room for improvement when it comes to dealing with noisy corpora, regardless of the supervision.

VecMap vs. MUSE. One of the main differences between these two models relates to their robustness. The results of VecMap are largely stable across the different types of the supervision. In fact, the best performance for Spanish and Russian on the XNLI task is even obtained in its unsupervised mode. In contrast, MUSE does not perform well with small dictionaries. Figure B.2 illustrates this trend. In addition, MUSE also suffers from some stability issues, as it does not always converge to the optimal solution, which confirms findings from previous work (Artetxe et al., 2018b; Søgaard et al., 2018). In terms of overall results, when given a sufficiently large dictionary training data, the performance of both methods is comparable, which is perhaps unsurprising as they both rely on the solution of the orthogonal Procrustes problem to learn an orthogonal transformation from the given dictionary.

Impact of corpora. As can be observed throughout all the experiments, the more comparable and less noisy the monolingual data is, the better the bilingual alignments. For instance, VecMap goes from an average of 31.2% in $P@1$ on Wikipedia down to 4.3% on social media, considering all language pairs. In word similarity, we observe an analogous performance drop, from 68.4% to 44.8% in Spearman correlation. Additionally, in Figure B.1 we can

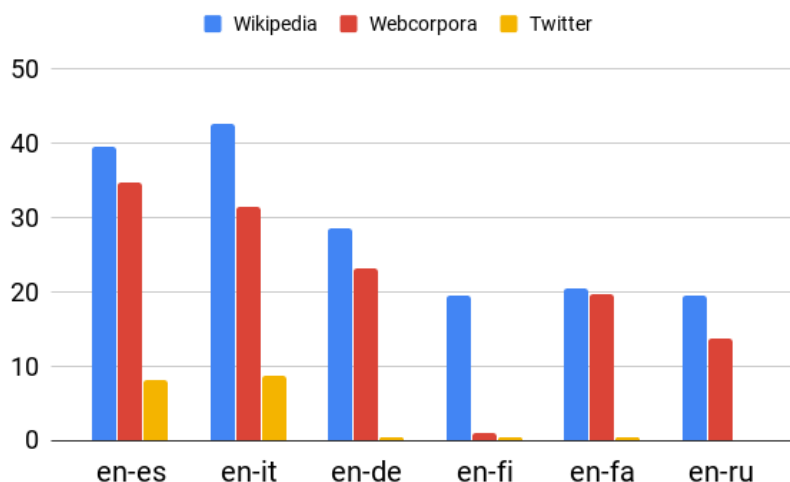


Figure B.1: $P@1$ performance of the unsupervised version of VecMap on dictionary induction across corpus types and language pairs.

observe the negative influence of noisy corpora and distant languages on the performance of the unsupervised version of VecMap on dictionary induction. In terms of error analysis, unsurprisingly we find that the low performance of the models trained on Twitter data is largely due to the noise and the nature of the conversation topics of this domain, which in many cases differ from the more standard language usage that we can find in Wikipedia and, to a lesser extent, Web corpora. For instance, for the word “discover”, instead of “descubren” (one of the correct answers obtained by the models trained on Wikipedia), the translation given by VecMap corresponds to a misspelling of the correct translation: “descubr”. As another example, and clearly due to the specific use of the word on Twitter, “timeline” is not translated to “cronología”, but to “instas”, which refers to the social network Instagram.

Distant languages. As expected, the more different the languages are, the harder it is to obtain a reliable alignment of the monolingual spaces. This is particularly noticeable in the case of Farsi, Russian, and Finnish (and German to a lesser extent). For instance, in the bilingual dictionary task, while most models are over 30.0% in $P@1$ (excluding social media text which causes performance drops in all languages), in the case of Finnish, Farsi, and Russian the results are below 20% in most cases. A similar tendency can be observed for Farsi on the word similarity task, where the differences are even more pronounced. In addition to its idiosyncrasies (Farsi is considered agglutinative and has a noun compounding formation similar to German), the fact that it uses a different alphabet together with

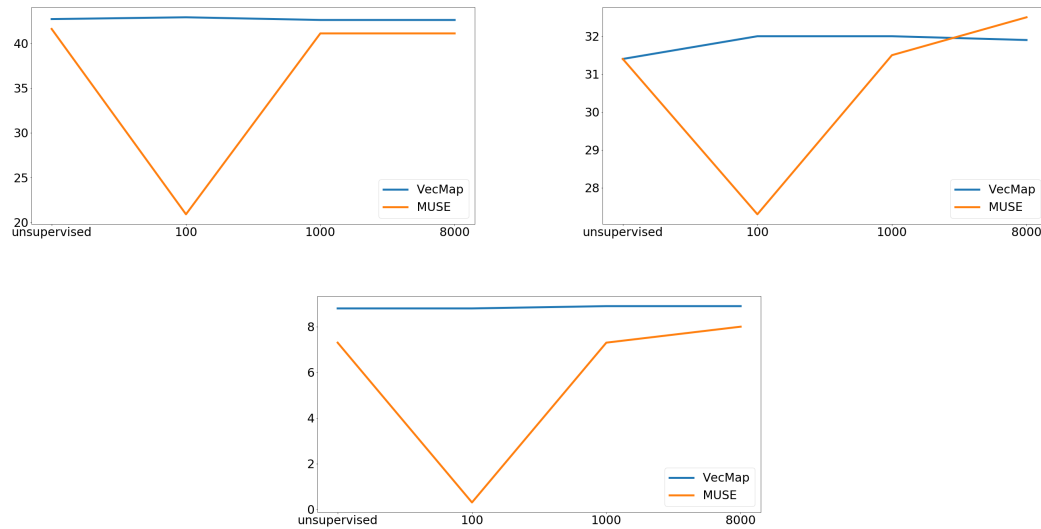


Figure B.2: Comparison between the dictionary induction performance ($P@1$) of VecMap (blue) and MUSE (red) in English-Italian on Wikipedia (left), Web corpora (middle), and social media text (right). The figures clearly show how VecMap produces similar results irrespective of the seed supervision, while the results of MUSE fluctuate depending on the size of the seed dictionary (with its unsupervised variant being superior to using a small dictionary).

the fastText models, which take word morphology into account, may explain this large performance gap. Finally, while the poor performance could be partially explained by the small size of the monolingual training corpora for some languages, it is interesting to see notable performance differences in cases where a distant language has similar or even greater amounts of training data available; e.g., Italian and Russian on Wikipedia, or Italian and Finnish or Russian on Web data.

Distant supervision. As far as the synthetic dictionary of identical words is concerned (see Tables B.6 and B.7), MUSE seems to have more difficulties coping with its noisy nature than VecMap, obtaining an average of 16.9% versus 23.6% in $P@1$ overall in dictionary induction in the Wikipedia and Web corpora domains. In fact, using MUSE in its unsupervised setting or with a small dictionary generally provides better results. However, on social media, using the dictionary of identical words appears to help MUSE considerably in the word similarity task compared to the unsupervised setting, going from 19.6% to 44.2% on average in Spearman correlation overall. This can be attributed to the multilinguality of social

		100	1K	8K
VecMap	EN-ES	-70.4	-2.0	+0.3
	EN-IT	-68.8	-1.0	+1.2
	EN-DE	-70.5	-0.3	+1.4
	EN-FA	-30.5	-32.8	+5.5
MUSE	EN-ES	-71.5	-1.8	+1.7
	EN-IT	-65.1	-0.8	+0.7
	EN-DE	-68.8	-0.1	+1.9
	EN-FA	-7.4	-22.3	+9.1

Table B.5: Absolute improvement (in percentage points) by applying the postprocessing (Meemi) over the two base models VecMap and MUSE on the cross-lingual word similarity task using Web corpora.

media data, where phenomena like code-switching often occur. On the other hand, the consistency of the VecMap semi-supervised algorithm is highlighted again, as using the identical dictionary in this case yields similar results to using external bilingual dictionaries.

Postprocessing. There are two main conclusions regarding Meemi, the postprocessing technique that we introduced in Chapter 7. First, a clean and relatively big bilingual dictionary is needed in order to get improvements over the base methods VecMap and MUSE (for instance, +1.2% $P@1$ and +3.1% Spearman correlation scores on social media on average, using the 8K dictionary), with the performance otherwise ending up significantly lower. In general, the best overall results are achieved when using this postprocessing technique in combination with the largest dictionary (i.e., 8K pairs). Table B.5 shows the performance gains and drops by using Meemi in the cross-lingual word similarity task, clearly showing the need for a reasonably large dictionary. This performance variability depending on the size of the dictionary was not addressed previously. Second, Meemi appears to be particularly useful when the monolingual corpora are not comparable, as shown by the larger improvements attained on Web-based data.

Evaluation tasks. The performance variability in bilingual dictionary induction, cross-lingual word similarity, and cross-lingual inference seems to be very similar across the board, with the main difference being the lower results variability in the XNLI task (which can be expected given that it is a downstream task where additional factors are also involved). The factors with the greatest impact on performance, namely monolingual corpora and language pairs, are clearly reflected in both cases, with analogous drops when going from training on Wikipedia to social media, and also when testing on Finnish, Farsi, or Russian. To test

our intuition, we computed Pearson correlation values from all overlapping results between task pairs. In this case, similarity and dictionary induction attain the highest correlation ($r = 0.78$), with XNLI and dictionary induction also attaining a high correlation score ($r = 0.73$). The lowest correlation score corresponds to cross-lingual similarity and NLI, with a lower figure of $r = 0.28$. Despite being positive, this relatively low correlation may suggest that dictionary induction would be a better proxy to test cross-lingual embedding performance in downstream tasks. We should note, however, that these correlation figures are only indicative and particular to the methods tested in our analysis and, therefore, should not be taken as the global correlation between tasks.

B.4 Related work

Cross-lingual embeddings have become increasingly popular in the past few years (Smith et al., 2017; Artetxe et al., 2017, 2018a; Conneau et al., 2018a). Recent efforts have focused on reducing the need for large amounts of resources (e.g., parallel corpora), which could be difficult to obtain for most languages and language pairs. However, the evaluation of these approaches has tended to be somewhat limited, often using only one type of training corpora, including only similar languages, and considering only one evaluation task. The most similar work to ours is that of Søgaard et al. (2018), which included an in-depth analysis of two of the factors that we have also considered, namely language family and corpora type, but they only considered a single model; i.e., MUSE (Conneau et al., 2018a). Moreover, they studied each factor in isolation. In our case the analysis is also extended to more languages (covering up to 5 language pairs), systems (two unsupervised, two supervised, and a postprocessing technique), evaluation tasks (cross-lingual word similarity), and the impact of external bilingual dictionaries.

Another similar contribution is the analysis by Vulić and Korhonen (2016), which studied the impact of bilingual dictionaries on cross-lingual alignments. However, they only considered closely-related languages using the same alphabet and a single type of corpora (i.e., Wikipedia). Also, given the publication date, this analysis does not account for the important developments in cross-lingual embeddings from recent years, such as the methods we have covered in this chapter. Other empirical comparisons focused mostly on the need for different degrees of supervision, such as (Upadhyay et al., 2016), which has been extended in a more recent survey by Ruder et al. (2017). Here, we complement those studies by analyzing and discussing empirical findings of the most recent state-of-the-art unsupervised and semi-supervised methods in a broader experimental setting, more in line with the recent work by Glavaš et al. (2019). The main differences between this empirical evaluation and the contributions of our work lie in the scope of the survey, since:

1. They only consider Wikipedia data for training.
2. They do not consider postprocessing techniques such as Meemi, which we found to improve the performance of cross-lingual models, especially in the case of distant languages and non-comparable corpora.
3. In our analysis we also consider additional settings with scarce training data such as small seed dictionaries and automatically-constructed dictionaries of identical words.
4. We include a more exhaustive intrinsic evaluation, including semantic similarity.

B.5 Conclusions

We have presented an extensive evaluation of state-of-the-art cross-lingual embedding models in a wide variety of experimental settings. The variables explored in this chapter were: monolingual training corpora, bilingual supervision signals, and language pairs. Likewise, the evaluation procedure included two standard benchmarks for cross-lingual embedding evaluation, namely bilingual dictionary induction and cross-lingual word similarity, as well as cross-lingual natural language inference as an extrinsic task. The set of languages considered included not only related languages such as English, Spanish, Italian, and German, but also languages from different families such as Finnish, Farsi, and Russian.

Our analysis shows a particularly marked variability of the performance concerning the monolingual training corpora used (e.g., comparable corpora such as Wikipedia vs. non-comparable or noisy user-generated corpora) and the specific language pair considered (distant language pairs still constitute a major challenge). We also conclude that bilingual supervision signals constitute a key component for MUSE and Meemi, whereas VecMap obtains more consistent results independently of the case. Finally, Meemi can improve the alignments obtained by VecMap or MUSE as long as a large dictionary is provided.

B.6 Supplementary material: detailed results

In this section we show the full range of results obtained for the dictionary induction (Table B.6) and cross-lingual word similarity (Table B.7) tasks, using all sources of supervision: no supervision, dictionary of identical words, and dictionaries containing 100, 1K, and 8K translation pairs.

Wikipedia																
Model	Dictionary	English-Spanish			English-Italian			English-German			English-Finnish			English-Farsi		
		P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10
VecMap	8K	39.6	66.2	72.3	42.6	65.9	71.8	28.6	48.3	54.8	22.4	44.5	52.5	22.8	39.7	46.2
	1K	39.6	66.2	72.3	42.6	65.7	71.6	28.7	48.3	54.7	22.2	43.9	51.7	23.2	40.2	46.1
	100	39.6	66.2	72.4	42.9	65.7	71.6	28.6	48.3	54.8	21.6	43.4	51.7	22.7	40.6	46.4
	identical	39.5	66.0	72.4	42.7	65.8	71.7	28.6	48.3	54.7	21.6	43.7	51.6	23.4	40.3	46.1
	unsupervised	39.6	66.1	72.3	42.7	65.7	71.6	28.6	48.3	54.8	19.6	40.4	48.3	20.5	37.0	42.8
MUSE	8K	39.1	65.4	72.3	41.1	63.3	70.1	27.6	45.9	53.2	19.5	40.4	49.5	19.7	35.4	42
	1K	39.2	65.4	72.1	41.1	63.3	70.1	27.6	46.0	53.1	18.1	36.8	44.9	19.8	35.3	41.5
	100	24.8	47.5	54.6	20.9	39.2	48.1	0.8	3.4	5.2	0.3	1.3	2.2	6.2	16.1	22.8
	identical	35.9	60.6	67.3	37.8	60.4	68.5	24.8	41.9	49.5	13.4	25.5	32	6.7	16.6	21.3
	unsupervised	39.3	64.7	71.3	41.6	63.2	69.9	28.3	46.5	53.3	0.0	0.0	0.0	0.0	0.0	0.0
Meemi (VecMap)	8K	39.3	67.4	73.7	41.6	66.5	72.5	28	47.8	54.8	23.8	48.7	57.0	0.0	0.0	0.0
	1K	35.5	63.7	69.4	38.6	64.0	70.1	23.1	42.5	49.9	17.8	40.1	48.6	0.0	0.0	0.0
	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	identical	38.7	63.7	70.1	40.6	64.1	70.2	27.5	46.6	53.1	19.3	37.7	45.5	7.1	14.3	17.9
	unsupervised	39.3	67.4	73.7	41.3	66.8	72.8	27.1	46.3	53.9	21.7	45.0	53.6	0.0	0.0	0.0
Meemi (MUSE)	8K	35.4	63.1	69.3	38.2	63.6	70.2	22.4	40.4	47.9	14.7	33.6	41.8	0.0	0.0	0.0
	1K	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	identical	35.4	58.9	65.4	37.0	59.0	65.9	24.0	40.0	47.0	13.0	25.5	32.0	2.5	6.2	8.3
	unsupervised	35.4	58.9	65.4	37.0	59.0	65.9	24.0	40.0	47.0	13.0	25.5	32.0	2.5	6.2	8.3
Web corpora																
Model	Dictionary	English-Spanish			English-Italian			English-German			English-Finnish			English-Farsi		
		P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10
VecMap	8K	34.6	60.6	66.9	31.9	54.2	60.4	23.1	42.7	50.5	18.9	40.9	48.8	19.6	35.8	41.4
	1K	34.6	60.5	67.0	32.0	54.0	60.5	23.1	42.7	50.3	19.4	42.0	49.4	19.5	35.6	41.2
	100	38.5	61.2	67.5	32.0	54.2	60.5	23.0	43.0	50.2	19.3	41.6	49.6	19.7	35.5	41.3
	identical	34.7	60.4	67.0	31.4	54.0	60.7	23.1	42.9	50.5	18.6	41.6	49.3	20.0	35.3	40.3
	unsupervised	34.8	60.6	67.0	31.4	53.7	60.7	23.2	42.7	50.2	0.0	0.0	0.0	19.7	34.6	40.4
MUSE	8K	32.5	58.2	65.9	32.5	56.0	63.2	22.4	40.9	48.9	20.0	40.1	48.3	17.4	31.6	37.6
	1K	32.9	56.8	64.2	31.5	52.7	60.6	22.1	41.0	48.2	18.4	39.1	47.7	16.6	31.1	36.4
	100	32.3	56.1	63.9	27.3	48.0	55.3	17.8	35.0	41.6	2.7	7.9	11.1	0.0	0.5	0.7
	identical	26.1	46.7	53.8	24.7	45.1	52.4	17.4	32.8	40.5	12.6	26.0	33.8	3.0	8.3	5.8
	unsupervised	31.4	51.2	57.7	31.4	51.2	57.7	20.8	38.7	46.6	0.0	0.0	0.0	18.1	32.8	37.8
Meemi (VecMap)	8K	34.5	61.6	67.9	33.6	58.3	65.6	23.7	45.4	53.2	22.3	46.7	55.0	0.0	0.0	0.0
	1K	30.2	55.0	62.7	30.7	54.0	61.1	19.4	38.9	45.9	18.2	39.9	48.1	0.0	0.0	0.0
	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	identical	34.1	58.3	64.8	31.6	54.6	62.5	22.5	42.0	49.0	21.1	43.2	51.3	11.2	23.9	28.6
	unsupervised	33.9	60.7	68.4	33.8	58.4	65.6	23.7	45.3	52.3	23.0	46.1	54.0	0.0	0.0	0.0
Meemi (MUSE)	8K	29.1	54.6	62.3	29.9	52.7	60.3	18.3	37.0	44.1	17.2	37.4	45.6	0.0	0.0	0.0
	1K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	identical	24.3	43.0	49.8	21.8	41.1	48.9	16.4	30.7	37.3	13.1	26.1	33.9	2.0	4.2	5.8
	unsupervised	24.3	43.0	49.8	21.8	41.1	48.9	16.4	30.7	37.3	13.1	26.1	33.9	2.0	4.2	5.8
Social media																
Model	Dictionary	English-Spanish			English-Italian			English-German			English-Finnish			English-Farsi		
		P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10	P@1	P@5	P@10
VecMap	8K	8.7	16.6	21.6	8.9	17.3	22.4	3.2	6.8	9.5	0.2	0.8	1.2	0.4	1.6	2.0
	1K	8.3	17.0	21.3	8.9	17.5	22.0	2.9	6.5	9.3	0.0	0.4	1.1	0.3	1.0	1.4
	100	7.9	15.9	20.2	8.8	17.6	22.3	2.8	6.0	8.6	0.0	0.0	0.1	0.1	0.3	0.4
	identical	8.5	16.9	21.6	9.1	16.8	21.8	2.6	6.7	9.6	0.0	0.0	0.0	0.2	0.5	1.1
	unsupervised	8.1	16.4	20.4	8.8	17.0	22.3	0.1	0.4	0.5	0.0	0.0	0.0	0.0	0.0	0.0
MUSE	8K	8.1	17.6	22.7	8.0	16.4	21.1	2.2	6.0	8.4	0.6	2.2	3.2	1.2	4.5	6.3
	1K	7.2	15.9	20.5	7.3	14.6	18.4	0.9	3.0	4.5	0.6	1.5	2.1	0.9	2.1	3.4
	100	0.4	1.1	1.9	0.3	1.1	1.8	0.1	0.3	0.6	0.0	0.2	0.4	0.1	0.3	0.4
	identical	2.5	5.2	7.1	3.9	10.1	13.7	1.1	2.6	3.7	0.1	0.1	0.2	0.1	0.3	0.8
	unsupervised	0.0	0.0	0.0	7.3	14.5	18.3	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.1
Meemi (VecMap)	8K	9.8	21.3	26.9	10.6	20.0	25.6	3.7	9.6	13.2	1.3	3.6	5.5	0.0	0.1	0.1
	1K	8.3	17.7	22.6	8.6	18.2	23.6	3.0	7.5	10.6	0.5	2.4	3.7	0.0	0.0	0
	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	identical	3.8	9.1	11.8	6.6	14.2	18.2	2.0	4.1	5.9	0.0	0.1	0.2	0.1	0.3	0.5
	unsupervised	9.5	20.5	26.3	9.5	19.1	24.5	3.0	7.6	11.1	1.5	4.3	6.4	0.0	0.1	0.2
Meemi (MUSE)	8K	7.6	16.9	22.3	7.8	15.9	21	1.7	4.1	6.2	0.8	2.3	3.7	0	0	0
	1K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	identical	2.9	5.3	6.7	3.5	9.9	13.2	1.2	2.8	3.6	0.2	0.3	0.3	0.0	0.1	0.2
	unsupervised	2.9	5.3	6.7	3.5	9.9	13.2	1.2	2.8	3.6	0.2	0.3	0.3	0.0	0.1	0.2

Table B.6: Bilingual dictionary induction results in the test sets of Conneau et al. (2018a).

Wikipedia									
Model	Dictionary	English-Spanish		English-Italian		English-German		English-Farsi	
		Pearson	Spearman	Pearson	Spearman	Pearson	Spearman	Pearson	Spearman
VecMap	8K	72.1	71.8	71.2	70.6	70.0	69.3	63.7	61.7
	1K	72.1	71.8	71.2	70.6	70.0	69.3	63.9	61.9
	100	72.1	71.8	71.2	70.6	70.0	69.3	63.9	62.0
	identical	72.1	71.8	71.2	70.6	70.0	69.3	63.8	61.9
	unsupervised	72.1	71.8	71.2	70.6	70.0	69.3	63.4	61.3
MUSE	8K	72.0	72.6	70.7	70.9	68.8	68.9	59.2	58.7
	1K	71.9	72.4	70.6	70.7	68.6	68.7	58.7	58.4
	100	65.1	66.3	63.0	63.6	44.7	49.9	47.7	52.1
	identical	71.0	71.9	69.9	70.5	68.1	68.4	47.9	51.3
	unsupervised	72.2	72.6	71.0	71.2	68.7	68.9	8.0	6.5
Meemi (VecMap)	8K	72.5	71.9	71.8	70.9	70.9	70.3	65.1	63.4
	1K	70.1	69.6	69.7	69.0	67.6	66.8	5.5	5.8
	100	0.0	0.0	5.1	5.0	4.1	3.3	6.8	6.5
	identical	71.0	70.4	69.4	68.7	69.3	68.6	56.1	54.1
	Meemi (MUSE)	8K	73.1	72.9	72.4	71.9	70.7	70.1	64.1
1K	70.4	70.3	69.6	69.6	66.7	66.5	5.0	4.2	
100	2.7	1.4	0.0	0.2	6.3	6.0	0.0	0.0	
identical	70.6	70.9	68.7	68.8	68.0	67.7	46.6	48.2	
Web corpora									
Model	Dictionary	English-Spanish		English-Italian		English-German		English-Farsi	
		Pearson	Spearman	Pearson	Spearman	Pearson	Spearman	Pearson	Spearman
VecMap	8K	71.0	70.6	69.2	68.8	70.9	70.4	35.9	33.5
	1K	71.0	70.6	69.3	68.8	70.9	70.4	35.9	33.5
	100	71.0	70.4	69.2	68.8	71.0	70.5	35.9	33.5
	identical	71.0	70.6	69.3	68.8	70.9	70.4	35.9	33.0
	unsupervised	71.1	70.5	69.2	68.8	70.9	70.4	35.7	33.4
MUSE	8K	71.9	71.9	70.4	70.4	70.5	70.2	29.7	23.9
	1K	71.6	71.5	69.5	69.4	70.3	70.0	28.3	22.3
	100	71.7	71.6	67.4	67.4	68.5	68.8	6.3	7.4
	identical	69.9	70.1	67.3	67.5	70.1	69.7	17.5	14.5
	unsupervised	71.7	71.6	69.4	69.4	70.3	70.0	29.6	23.8
Meemi (VecMap)	8K	71.5	70.9	70.4	70.0	72.3	71.8	40.2	39.0
	1K	69.1	68.6	68.2	67.8	70.9	70.2	1.2	0.7
	100	0.0	0.0	0.0	0.0	0.0	0.0	3.2	3.0
	identical	70.1	69.5	69.2	68.4	71.2	70.6	31.6	28.5
	Meemi (MUSE)	8K	72.5	72.3	71.5	71.1	72.5	72.1	36.4
1K	70.0	69.7	68.9	68.6	70.3	69.9	0.0	0.0	
100	1.7	0.1	1.6	2.3	0.0	0.0	0.3	0.0	
identical	69.2	69.1	67.4	66.9	70.1	69.4	17.3	14.5	
Social media									
Model	Dictionary	English-Spanish		English-Italian		English-German		English-Farsi	
		Pearson	Spearman	Pearson	Spearman	Pearson	Spearman	Pearson	Spearman
VecMap	8K	48.5	47.4	53.9	51.8	51.3	49.5	31.1	30.3
	1K	48.7	47.7	53.7	51.6	51.7	50.2	30.3	29.6
	100	49.8	48.9	54.0	51.7	51.0	49.5	25.7	25.4
	identical	48.4	47.1	54.2	51.9	51.8	50.3	27.8	26.5
	unsupervised	48.0	46.9	53.8	51.5	30.1	31.2	4.2	2.4
MUSE	8K	48.8	47.6	51.0	49.3	48.5	48.6	43.3	42.2
	1K	46.6	45.5	49.7	47.8	44.8	45.7	38.7	38.9
	100	35.8	36.9	29.6	31.3	30.7	34.0	20.8	21.3
	identical	48.1	47.7	50.1	49.8	45.6	46.8	30.5	32.4
	unsupervised	9.9	10.9	50.7	49.7	12.4	13.0	6.9	4.7
Meemi (VecMap)	8K	51.2	50.1	56.1	53.6	55.0	53.8	45.2	43.1
	1K	49.7	48.6	55.4	52.8	52.8	51.3	3.8	3.8
	100	2.4	2.0	2.1	2.8	0.0	0.0	5.6	5.1
	identical	51.7	50.1	56.2	53.4	52.7	51.3	29.9	28.6
	Meemi (MUSE)	8K	51.8	50.4	54.8	52.5	53.1	52.0	48.9
1K	49.5	47.9	53.1	50.7	48.4	47.2	0.0	0.0	
100	5.2	5.4	0.0	0.0	6.2	6.8	0.0	0.0	
identical	49.6	48.2	53.1	51.4	48.4	47.7	30.5	30.6	

Table B.7: Cross-lingual word similarity results in the SemEval-17 dataset (Camacho Colados et al., 2017).

Appendix C

Resumen largo en español *Long summary in Spanish*

En este trabajo de tesis estudiamos dos enfoques para abordar los desafíos en el procesamiento de contenidos textuales no estándar y multilingües generados por los usuarios tal y como se pueden encontrar en la Web a día de hoy. Este tipo de textos son denominados a menudo *textos cortos* o *microtextos*.

En primer lugar, presentamos un enfoque tradicional basado en *pipelines* discretos en el que el texto de entrada es preprocesado para facilitar su ulterior tratamiento por otros sistemas. Esto implica abordar el problema del multilingüismo, primero, identificando el idioma de la entrada para, seguidamente, tratar los fenómenos de escritura no estándar específicos de dicho idioma presentes en la entrada. Para ello se aplicarán técnicas de normalización del texto y (re-)segmentación de palabras.

En segundo lugar, analizamos las limitaciones inherentes a este tipo de modelos discretos, lo cual nos conduce a un enfoque centrado en el empleo de modelos continuos basados en *word embeddings* (i.e., representaciones vectoriales). En este caso, el preprocesamiento explícito de la entrada es sustituido por la codificación de las características lingüísticas y demás matices propios de los textos no estándar en el propio espacio de *embedding* (un espacio vectorial). Nuestro objetivo es obtener modelos continuos que no sólo superen las limitaciones de los modelos discretos, sino que también se alineen con el estado del arte actual del Procesamiento de Lenguaje Natural (PLN), dominado por sistemas basados en redes neuronales.

C.1 Motivación

Los usuarios de Internet producen y comparten todo tipo de contenido escrito en una amplia variedad de servicios y plataformas: páginas Web, correos electrónicos, mensajes de chat, publicaciones en redes sociales, etc. El tipo de textos empleado, sobre todo, en estos últimos casos, tiene dos rasgos específicos que lo diferencian de la mayoría de los textos escritos a mano, a la vez que lo acercan al lenguaje hablado (también referidos como *textos ruidosos*): la *espontaneidad* y la *informalidad*. Esto da como resultado un estilo de escritura fuertemente influenciado por hábitos de habla como, por ejemplo, sonidos prolongados para dar énfasis (p. ej., “nooo”), uso frecuente de preguntas coetilla (p. ej., terminando frases con “¿verdad?”) o el uso general de lenguaje descuidado o inapropiado (p. ej., “k aces” en lugar de “¿qué haces?”), entre otros. Es decir, los usuarios de Internet tienden a *escribir como hablan*.

Asimismo, aún cuando el inglés es el idioma predominante de Internet, este demuestra un claro y creciente *multilingüismo* al acomodar contenidos en prácticamente cualquier idioma. Para ilustrar esto, deberíamos mencionar que las plataformas de redes sociales más populares, como Facebook o Twitter, son utilizadas por una gran cantidad de usuarios en todo el mundo que publican contenidos textuales en su idioma o idiomas particulares de elección. No solo eso, es también habitual el uso del *code-switching*, o la combinación de palabras en distintos idiomas en una misma frase u oración (p. ej., “esta es mi *house*” en lugar de “esta es mi casa”).

C.2 PLN para textos generados por el usuario: adaptación al dominio

Una de las principales preocupaciones del presente trabajo radica en que, por lo general, los sistemas de PLN están pensados para operar sobre textos estándar; por ejemplo, los etiquetadores morfosintácticos, los analizadores sintácticos o los sistemas de clasificación de texto, casi siempre se entrenan y evalúan con textos bien escritos. Por lo tanto, no tienen en cuenta las características lingüísticas del tipo de textos empleados en la Web y en redes sociales que describimos anteriormente, lo que conlleva una penalización en su rendimiento a la hora de operar sobre textos generados por el usuario.

Llegados a este punto, existen dos enfoques generales para abordar este problema (Eisenstein, 2013):

Adaptación del sistema Reimplementar o rediseñar un modelo en particular para que sea compatible con este tipo de textos propios de las redes sociales. La principal ventaja de este enfoque es su alta integración, evitando así la necesidad de cualquier forma de normalización o estandarización de la entrada en un módulo separado y, por lo tanto, evitando posibles fuentes de errores que se propagarían por nuestro sistema. Por contra, su mayor desventaja es que sería necesario adaptar todos los sistemas a los textos de las redes sociales, lo que puede no resultar trivial, ya que estaremos exigiéndole a un sistema ya de por sí complejo dar soporte a una gama mucho más amplia de entradas. Ejemplos de este tipo de enfoque son el etiquetador morfosintáctico (Owoputi et al., 2013) y el analizador sintáctico basado en dependencias (Kong et al., 2014) del *framework* TweetNLP.¹

Adaptación de la entrada Mantener la implementación existente de un modelo que solo acepta textos estándar como entrada e incluir un paso de preprocesamiento en el que adaptamos la entrada para cumplir con los estándares de escritura correspondientes. La ventaja en este caso es la de tener modelos separados para diferentes cometidos, reduciendo así la complejidad de cada uno de ellos y aumentando la modularidad de nuestra solución. Esta forma de disposición secuencial de pasos se denomina generalmente *pipeline* y es muy común en PLN. Por otro lado, su principal desventaja es un bajo nivel de integración, lo que facilita la propagación de errores de un paso al siguiente, a menudo con un efecto acumulativo. Si bien existen métodos para mitigar este problema y que se basan en una reorganización no lineal de los pasos en el *pipeline* o en grafos.

En este trabajo, hemos pasado de una implementación estricta del segundo enfoque a una solución intermedia entre ambos: en lugar de adaptar la entrada, adaptamos la representación interna en los modelos del sistema. El objetivo es mantener la modularidad del segundo enfoque a la vez que la integración del primero; es decir, reducir la posibilidad de propagación de errores y mejorar la integración.

C.3 Enfoque discreto: tareas para la adaptación de la entrada

En primer lugar, hemos considerado un enfoque basado en un *pipeline* discreto de preprocesamiento en el que se tengan en cuenta la mayor cantidad posible de fenómenos de escritura no estándar propios de este tipo de textos.

¹<http://www.cs.cmu.edu/~ark/TweetNLP/>

C.3.1 Identificación del idioma

Esta tarea consiste en determinar el idioma en el que está escrito un texto como primer paso de cara a tratar los fenómenos multilingües de los textos de entrada. Aunque generalmente se asume un escenario monolingüe en el que se usa un solo idioma en un texto dado, en nuestro trabajo pasamos a soportar un escenario donde se puede dar el *code-switching*. Por la misma razón, también consideraremos frases más cortas de lo habitual, ya que nos estamos centrando mayormente en los textos de redes sociales y plataformas que imponen un límite máximo de longitud en caracteres, como en el caso de Twitter. La información sobre el idioma obtenida en este paso nos permitirá elegir los módulos monolingües apropiados posteriormente en nuestro *pipeline* de preprocesamiento sin tener ya que preocuparnos por el posible multilingüismo de estos textos. Con esto pretendemos obtener el soporte requerido para los textos multilingües generados por los usuarios Web.

Nuestra solución propuesta pasa por adaptar y reentrenar las herramientas de identificación automática del idioma existentes utilizando varios corpus de nuestra elección, para que así todos compartan el mismo punto de partida. Con este desarrollo participamos en la tarea competitiva TweetLID (Zubiaga et al., 2014, 2016), donde el dominio de aplicación propuesto era el de tuits escritos tanto en inglés como en los idiomas de uso en la Península Ibérica: español, portugués, gallego, vasco y catalán.

Con respecto a los resultados obtenidos, debemos mencionar cómo las herramientas existentes que muestran un buen rendimiento en otros dominios más tradicionales, o incluso textos reducidos, no son tan precisas en nuestro caso de uso. Podemos pues concluir que el problema de la identificación del idioma no es aún un problema resuelto en PLN, dadas las dificultades mostradas por las herramientas existentes para abordar las particularidades de nuestro entorno experimental. Esta situación se pone especialmente de manifiesto cuando se trata de discernir entre idiomas similares que se usan conjuntamente en un contexto en el que uno de ellos está subrepresentado en comparación con el resto, como es el caso del gallego y el español.

C.3.2 Normalización de microtexto

Este es el paso central que se ocupa de los fenómenos monolingües en nuestro *pipeline* de preprocesamiento.² Idealmente tendremos tantos módulos de normalización como idiomas considerados, y elegiremos entre ellos de acuerdo con la salida del paso anterior de identificación del idioma. El objetivo de esta tarea es obtener una versión más estándar del texto de entrada normalizando cada una de sus palabras. Para ello, entre otros procesos, se

²Se supone que nuestro *pipeline* de preprocesamiento es parte de otro más grande que incluiría una tarea principal para la cual realizamos la adaptación de la entrada.

expanden las abreviaturas (p. ej., “q” se normalizaría como “que” o “qué”), se corrigen las variaciones ortográficas (p. ej., “aber” se reemplazaría por “a ver” o “haber”) y se elimina el énfasis (p. ej., “siii” se normalizaría como “sí”).

Nuestro enfoque para abordar esta tarea se basa en el marco tradicional de dos pasos consistente en: (1) la generación de candidatos de normalización, para el cual utilizamos un corrector ortográfico y un diccionario de normalización, seguida por (2) la selección de candidatos, implementada a través de un modelo de lenguaje a nivel de palabra y un algoritmo de búsqueda. Nuestro sistema depende de este modelo de lenguaje en la última etapa del proceso en un intento de poner menos énfasis en la señal de supervisión humana, y así hacer que el sistema pueda adaptarse más fácilmente a los cambios en un ámbito tan dinámico como el de los textos generados por los usuarios (Bertaglia and Nunes, 2016; Eisenstein, 2013; Han et al., 2013a). Sin embargo, hemos podido también comprobar que un modelo de lenguaje entrenado con grandes cantidades de texto estándar o con una pequeña cantidad de tuits normalizados no es capaz de elegir los candidatos adecuados, aun cuando estos habían sido generados en pasos anteriores.

Asimismo, hemos procurado llegar a una solución modular y adaptable, haciendo para ello uso de los patrones de diseño adecuados, que a la vez nos permitiese iterar fácilmente para su mejora continua así como para su adaptación a diferentes escenarios. En esta línea, en un primer momento desarrollamos una versión para el español que luego fue portada al inglés para participar en la tarea competitiva n° 2 del W-NUT 2015 (Baldwin et al., 2015).

C.3.3 Segmentación de palabras

Esta puede considerarse una subtarea de la anterior que, de hecho, debería situarse antes de ella en el *pipeline*. Puesto que estamos considerando la tarea de normalización a nivel de palabra, sería deseable poder partir de un texto de entrada correctamente segmentado donde las palabras están claramente delimitadas. Sin embargo, este no siempre es el caso en los textos generados por el usuario ya que, por ejemplo, las segmentaciones no estándar de palabras se pueden usar para aportar énfasis (p. ej., “im posible” en lugar de “imposible”) o, simplemente, pueden ser el resultado de errores ortográficos (p. ej., “aver” por “a ver”). Si bien la normalización de la segmentación de palabras se podría integrar propiamente en el módulo principal de normalización, hemos optado por implementar este proceso aparte en su propio paso del *pipeline*, antes del de normalización, para así preservar una mayor modularidad.

Nuestra solución para abordar el problema de la segmentación de palabras consta de dos componentes: un algoritmo de *beam search*, que genera y elige entre los posibles candidatos de segmentación de forma incremental; y un modelo de lenguaje a nivel de

byte o carácter, que permite que el algoritmo pueda clasificar los candidatos, implementado como una red neuronal recurrente o modelo de n-gramas. En la parte experimental hemos comparado el rendimiento de diferentes configuraciones de nuestro sistema, WordSegment (Jenks, 2017) y Microsoft Word Breaker (Wang et al., 2011) sobre textos escritos en español, inglés (tanto textos estándar como microtextos), alemán, turco y finlandés. En general, los mejores modelos neuronales obtienen las mejores cifras de precisión aunque, sorprendentemente, el rendimiento de los modelos más simples de n-gramas es cercano al de sus contrapartidas neuronales siendo notablemente más rápidos. En comparación con otros sistemas disponibles públicamente, como WordSegment y Word Breaker, nuestro enfoque obtuvo, en general, mejores resultados.

C.4 Limitaciones del enfoque discreto y transición a un modelo continuo

Los módulos que resuelven cada una de las tareas de preprocesamiento anteriormente descritas se organizaron en un *pipeline*, una estructura de amplio uso en PLN. Si bien esto nos permite preservar la modularidad deseada, lo hace a cambio de una baja integración entre los módulos en los pasos secuenciales, ya que la información fluye solo hacia adelante. Sin embargo, las dependencias entre tareas en un *pipeline* podrían no ser enteramente 1:1 y unidireccionales; p. ej., el resultado del paso final podría usarse para obtener un mejor resultado en el primer paso, lo que a su vez ayudaría al segundo, y así sucesivamente. Para resolver este problema, Valls-Vargas et al. (2015) propusieron utilizar *pipelines no lineales* y Roth and Yih (2004) fueron más allá al reemplazar esta estructura con grafos de tareas interconectadas. En ambos casos, la integración mejora con respecto al *pipeline* lineal pero no permitirían resolver los problemas particulares presentados por nuestro caso de uso, donde el orden entre el preprocesamiento y las tareas principales debe ser preservado si queremos que dicho preprocesamiento sea de alguna utilidad.

A continuación analizaremos en detalle los problemas que presenta nuestro enfoque discreto actual junto con otras posibles soluciones, también discretas, que no resultan del todo apropiadas o no acaban de resolver estos problemas. Posteriormente, y en vista de tales circunstancias, propondremos la transición hacia un enfoque basado en modelos continuos que sí resulta apropiado para resolver los desafíos encontrados.

C.4.1 Propagación de errores

Esta es una consecuencia inmediata de cualquier proceso en forma de cascada que se base en encadenar las mejores soluciones de cada una de las etapas que lo componen. De este

modo, dado un *pipeline* de varios pasos, el error en el último paso no es simplemente la suma de los errores en los anteriores, sino un múltiplo de esa suma. En los pasos intermedios, hay dos fuentes de errores de salida que interactúan entre sí: (1) entradas incorrectas, que probablemente derivarán a su vez en respuestas incorrectas; y (2) las imprecisiones del modelo utilizado en ese paso, ya que es realista asumir que ningún modelo obtiene una precisión del 100% en ninguna tarea de PLN.

Tomemos, a modo de ejemplo, la oración no estándar en gallego “o meu móbil é bnito” (en español “mi móvil es bonito”). Si el módulo de identificación del idioma lo catalogase erróneamente como texto en español, un posible resultado incorrecto del módulo de normalización posterior sería “o mejor móvil e Benito”, una oración sin sentido pero formada por palabras estándar en español. En este caso, la identificación errónea inicial del idioma provoca la normalización incorrecta de las primeras cuatro palabras, mientras que el error en la última es ya propio del módulo de normalización, siendo la respuesta correcta “bonito” en lugar de “Benito”.

Este problema ha sido estudiado en la literatura y existen varias técnicas y diferentes arquitecturas para minimizar la propagación de errores pero que, sin embargo, no resuelven el problema en nuestro caso:

1. Una alternativa común es considerar un *k-best pipeline* (Wellner et al., 2004; Sutton and McCallum, 2005), donde cada paso genera una clasificación de sus k mejores predicciones en lugar de un único resultado. Este esquema nos permite abordar el problema de la propagación de errores hasta cierto punto, puesto que ya no estamos limitados a un proceso en cascada de una sola predicción donde todas las decisiones son locales para cada paso. Sin embargo, elegir el valor correcto de k podría no ser trivial. Para resolver este problema, Finkel et al. (2006) propone, a su vez, modelar los *pipelines* como *redes bayesianas*. En conjunto, si bien estas soluciones pueden reducir la propagación de errores, siguen siendo construcciones unidireccionales, por lo que no acaban de resolver el problema que nos ocupa.
2. Por contra, los *pipelines no lineales* sí permiten que la información fluya también hacia atrás y afecte los resultados de los pasos anteriores, haciendo uso para ello de bucles de retroalimentación y un proceso iterativo de refinamiento de los resultados. En general, el diseño de estas estructuras depende de las tareas involucradas, conectando entre sí pasos específicos a través de bucles de retroalimentación donde la información de una etapa puede usarse o integrarse efectivamente en otra anterior, generalmente como entrada complementaria. Aún así, sigue habiendo una linealidad estricta dentro de estos bucles, lo que todavía podría contribuir a la propagación de errores. Otro problema es que agregar o reemplazar módulos en el *pipeline* ya no es tan sencillo

(p. ej., en qué paso deben ubicarse), al igual que establecer los propios bucles de retroalimentación.

3. Yendo más allá del diseño en forma de *pipeline*, algunos autores han explorado el uso de grafos para eliminar por completo las restricciones de ordenamiento secuencial de las tareas y, de este modo, eliminar el problema de propagación de errores. El resultado es similar a una combinación entre un *pipeline* no lineal y uno basado en k predicciones donde todos los pasos están conectados entre sí mediante enlaces bidireccionales y donde las mejores soluciones locales se eligen en función de restricciones globales. Desafortunadamente, este enfoque no es válido cuando se considera una configuración de preprocesamiento, dado que en este caso sí se requiere un ordenamiento secuencial entre las tareas de preprocesamiento y las principales.

C.4.2 Fragmentación del contexto

La fragmentación del contexto ocurre cuando tenemos datos de entrada heterogéneos para una tarea específica pero donde, a la vez, cada parte homogénea debe ser procesada por un modelo especializado diferente. Este es precisamente nuestro caso cuando diferentes partes de la entrada están escritas en diferentes idiomas (el denominado *code-switching*). En ese caso, la entrada debe dividirse en los segmentos monolingües que la constituyen para que, seguidamente, cada uno de ellos sea procesado de forma aislada empleando su modelo monolingüe correspondiente. Claramente, cuando se procesa una parte del texto escrito en un idioma determinado, el modelo encargado de ello no tiene acceso a las partes adyacentes escritas en otros idiomas, lo que reduce la cantidad de información contextual disponible. Como ejemplo, considérese la oración “*let’s go to my ksa*” (que podemos traducir directamente como “vamos a mi *ksa*”). Aquí, no tenemos contexto disponible suficiente para normalizar la palabra “*ksa*” (“*casa*”), ya que las palabras circundantes están escritas en un idioma diferente y, por lo tanto, son procesadas aparte por otro modelo. Dos posibles soluciones para evitar este problema serían:

1. Usando solo *modelos multilingües* en nuestros sistemas y eliminando el paso de identificación del idioma, algo que va frontalmente en contra de la modularidad.
2. *Homogeneizando la entrada* al traducir todas las partes a un idioma común, lo que permite centrarnos en un solo idioma en los pasos restantes. Su principal desventaja es que estaríamos introduciendo en nuestro *pipeline* las complejidades de la traducción automática, además de una tarea adicional. Sin embargo, una solución derivada podría ser reemplazar las palabras, sea cual sea el idioma, con sus representa-

ciones vectoriales correspondientes (*embeddings*), lo que serviría como un lenguaje intermedio que homogeneiza el contexto (como se explicará a continuación).

En cualquier caso, y con respecto a las alternativas al *pipeline* lineal vistas anteriormente, es importante resaltar que ninguna de ellas considera siquiera el problema de fragmentación del contexto.

C.4.3 Hacia una aproximación continua

Los sistemas de PLN de última generación basan su funcionamiento principalmente en técnicas de aprendizaje automático que operan sobre *vectores de features* (de valores reales) que describen los elementos de entrada discretos sobre un dominio multidimensional continuo. Como consecuencia, y suponiendo que usemos este tipo de modelos de aprendizaje automático para implementar los diferentes pasos que conforman nuestro *pipeline*, en cada uno de ellos sería necesario realizar a la entrada una transformación discreta a continua (codificación) y luego una transformación continua a discreta (decodificación). Estas continuas codificaciones y decodificaciones conllevarían una pérdida de información que se encuentra en la raíz del problema de propagación de errores. Obsérvese aquí que el procesamiento efectivo de la entrada *no se realiza* sobre su forma discreta, sino sobre su representación continua, y que una salida (o predicción) discreta solo será realmente útil al final del *pipeline* ya que es el usuario quien requiere un resultado fácilmente *interpretable*. Si además consideramos que las tareas de preprocesamiento están, por definición, ubicadas antes de las tareas principales en las que está interesado el usuario, deberíamos cuestionarnos si de verdad vale la pena obtener resultados discretos intermedios en cada paso del *pipeline*.

Más allá de las estructuras vistas hasta ahora, los sistemas modernos de PLN dependen cada vez más de enfoques basados en *redes neuronales* (Plank et al., 2016; Eshel et al., 2017; Devlin et al., 2019). Durante la última década este tipo de modelos de aprendizaje automático han estado obteniendo un rendimiento de estado del arte gracias a la mayor abundancia de datos de entrenamiento, recursos computacionales y nuevos algoritmos de entrenamiento. Dada su gran potencia y flexibilidad, las redes neuronales tienden a ser entrenadas como sistemas de extremo a extremo al proporcionarles solo las entradas sin procesar para obtener las salidas deseadas correspondientes y, por lo tanto, con poco o muy poco preprocesamiento aplicado, eliminando así la necesidad de las tareas propuestas anteriormente.

Pero más interesante es el hecho de que las redes neuronales tienen el potencial de evitar el compromiso entre modularidad e integración que hemos visto hasta ahora al facilitar la *transferencia de conocimiento* a través de mecanismos generales como el *pretraining* y el

fine tuning (Erhan et al., 2010; Howard and Ruder, 2018). Básicamente, esto nos permite entrenar un modelo en una tarea específica, extraer la información codificada en la red al final del proceso (es decir, las *embeddings*) y usarla para entrenar otro modelo para una tarea diferente, que puede hacer uso de esa información para mejorar su rendimiento. De hecho, este proceso puede repetirse sobre más modelos y más tareas codificando más y más conocimiento en las redes resultantes, lo que nos permite obtener sistemas con un grado de integración alto.

En nuestro caso, reemplazamos las tareas de preprocesamiento consideradas anteriormente con el *pretraining* de *embeddings* que codifican información sobre fenómenos propios de los microtextos.³ Estas *embeddings* se utilizarían al entrenar otros modelos enfocados a tareas específicas de PLN como pueden ser el etiquetado morfosintáctico, el análisis de sentimientos o el análisis sintáctico basado en dependencias. Dichos modelos se beneficiarían del conocimiento codificado en las *embeddings*. En particular, reemplazamos el intercambio de información entre tareas por medio de símbolos discretos (p. ej., palabras, etiquetas, oraciones, etc.) que constituyen las entradas y salidas de las diferentes etapas del *pipeline*, con construcciones matemáticas continuas (es decir, las *embeddings*).

Con respecto a los problemas de propagación de errores y fragmentación del contexto, los cuales analizamos en los apartados anteriores, el primero puede abordarse mediante una integración mejorada del sistema, mientras que el segundo puede resolverse traduciendo o transformando las palabras de entrada en *word embeddings* multilingües (Artetxe et al., 2018a; Conneau et al., 2018a; Ammar et al., 2016; Mikolov et al., 2013a) que constituyen de este modo un *lenguaje intermedio* común. En este caso, las traducciones de palabras (p. ej., “casa” - “house”, “gato”-“cat”, etc.) pasarían a estar cercanas entre sí en el espacio de *embedding*, lo que permitiría emplear directamente modelos entrenados en estas representaciones continuas multilingües sin requerir un paso explícito de traducción automática para homogeneizar el contexto.

Con respecto a los pasos incluidos en nuestro *pipeline* de preprocesamiento, las *word embeddings* se pueden utilizar para codificar las particularidades derivadas de los fenómenos propios de los textos generados por los usuarios para reemplazar su procesamiento explícito. De este modo, se podrían usar *word embeddings* multilingües en lugar del paso de identificación del idioma. Además, el paso de normalización, que se ocupa de las variantes no estándar de palabras, también puede evitarse si consideramos que las variantes léxicas de palabras en un idioma particular deberían tener representaciones vectoriales similares cuando se entrenan en grandes cantidades de textos estándar y no estándar (Bertaglia

³Cuando esta codificación se hace a nivel de palabra, obtenemos las denominadas *word embeddings*. Las herramientas más conocidas que permiten obtenerlas son word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) y fastText (Bojanowski et al., 2016)

and Nunes, 2016; van der Goot and van Noord, 2017; Ansari et al., 2017; Sridhar, 2016). Todo esto significa que, por ejemplo, para los términos “amigo”, “amgo”, “frnd” y “friend”, deberían obtenerse *embeddings* cercanas entre sí en el espacio de *embedding*, lo que indica que dichos términos deberían ser tratados de manera similar por otros modelos de PLN, aún sin dejar de ser igualmente capaces de distinguir entre ellas.

C.5 *Embeddings* multilingüe

Para obtener nuestras *embeddings* multilingüe, hemos desarrollado un método de post-procesamiento, que hemos denominado Meemi (por “*Meeting in the Middle*”), que mejora la integración de espacios monolingües inicialmente aislados y posteriormente alineados mediante herramientas del estado del arte como VecMap (Artetxe et al., 2018a) y MUSE (Conneau et al., 2018a).

Para mejorar dicha integración, aplicamos sobre estos alineamientos una transformación lineal no restringida que se aprende haciendo corresponder las traducciones de palabras con sus representaciones promedio. Notablemente, hemos ido más allá de la configuración bilingüe habitual en estas herramientas y hemos mostrado también cómo Meemi puede extenderse naturalmente a un número arbitrario de idiomas que acaban integrados en un único espacio vectorial compartido. En este caso, utilizamos métodos ortogonales en el primer paso de alineación que solo transforman el espacio de *embedding* de uno de los lenguajes (origen) mientras deja intacto el otro espacio (destino), que se convierte en el espacio vectorial multilingüe. Este proceso se repite para los espacios de origen correspondientes a cada idioma restante.

Con respecto a la evaluación, hemos considerado no solo idiomas indoeuropeos habituales, tales como inglés, español, italiano y alemán, sino también otros idiomas más distantes, como el finlandés, el farsi o el ruso. Los resultados obtenidos muestran que Meemi es capaz de mejorar los resultados logrados por los métodos de alineamiento básicos, con ganancias significativas cuando se aplica sobre variantes ortogonales y también cuando se consideran idiomas distantes. Los buenos resultados obtenidos con los modelos multilingües son quizás los más esperanzadores de todos ellos, dado que demuestran que integrar más de dos idiomas en un espacio vectorial compartido es altamente beneficioso en muchas ocasiones.

C.6 *Embeddings* robustas para microtextos

Los modelos de *word embeddings* como word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) o fastText (Bojanowski et al., 2016) ya son capaces de por sí de agrupar variantes estándar y no estándar de palabras (p. ej., “porque” y “pq”) cuando se les proporciona un corpus de entrenamiento lo suficientemente grande que incluya tales variantes (Sumbler et al., 2018). Teniendo esto en cuenta, en este trabajo de tesis se propone ir un paso más allá con una modificación del modelo *skipgram* de fastText destinada a mejorar el rendimiento de las *word embeddings* resultantes en textos ruidosos al tiempo que conserva el rendimiento en los textos estándar. Para conseguirlo, introducimos un nuevo conjunto de palabras en el proceso de entrenamiento, que denominamos *bridge-words* (o *palabras puente*), cuyo objetivo es conectar de una forma más deliberada palabras estándar con sus contrapartidas ruidosas. Hasta donde sabemos, este es el primer intento de tratar explícitamente este tipo de textos ruidosos a nivel de *word embeddings*, yendo más allá del soporte para palabras OOV de modelos como fastText.

Hemos evaluado el rendimiento de nuestra propuesta comparándola con otros modelos de sobra conocidos como word2vec y fastText en una amplia gama de tareas intrínsecas y extrínsecas. Además de la habitual tarea de similitud de palabras, incluimos la detección de valores atípicos (Camacho-Collados and Navigli, 2016), 26 tareas del *benchmark* SentEval (Conneau and Kiela, 2018) y el análisis de sentimientos en Twitter correspondiente a varias ediciones del taller SemEval (Nakov et al., 2013; Rosenthal et al., 2014; Nakov et al., 2016). Los resultados muestran que, si bien el rendimiento de nuestro mejor modelo en textos estándar se mantiene en comparación con los modelos de referencia, generalmente los supera en textos más ruidosos con márgenes más amplios a medida que aumenta el nivel de ruido.

Estas *embeddings* para textos ruidosos encuentran su más clara aplicación dentro de modelos entrenados de extremo a extremo que eliminan la necesidad de pasos de pre-procesamiento que modifiquen la entrada original y, que como se explicó anteriormente, podrían introducir errores que luego se propagarían a otras partes de nuestros sistemas. Por otro lado, cabe destacar que van der Goot et al. (2017) también ha demostrado que la normalización de textos ruidosos no produce una mejora clara sobre el simple uso de las *word embeddings* en ciertos casos.

C.7 Conclusiones y trabajo futuro

Los resultados obtenidos después de una extensa experimentación muestran la capacidad de las *word embeddings* para dar un soporte efectivo por sí mismas a los fenómenos multilingües

y no estándar propios de los textos generados por usuarios. Además, todo esto se logra dentro de un marco conceptual simple y modular que no necesita sacrificar la integración de sistemas. Dichos modelos de *word embeddings* pueden emplearse fácilmente como un elemento fundamental en redes neuronales de última generación que, a su vez, son utilizadas en prácticamente cualquier tarea de PLN.

Dicho esto, cabe señalar que en esta ocasión nuestro objetivo no era obtener una mejora en el rendimiento con respecto a los enfoques discretos tradicionales, sino mostrar que los modelos continuos pueden superar algunas de limitaciones cruciales de aquellos que impondrían un límite superior a su rendimiento y utilidad.

Después de la transición de modelos discretos a continuos en el curso de esta disertación, las futuras líneas de investigación se centrarán principalmente en mejorar nuestros modelos de *word embeddings* robustas y multilingüe.

