Procesamiento del Lenguaje Natural, Revista nº 57, septiembre de 2016, págs. 75-82 recibido 29-03-2016 revisado 28-04-2016 aceptado 03-05-2016

Segmentación de palabras en español mediante modelos del lenguaje basados en redes neuronales*

Spanish word segmentation through neural language models

Yerai Doval
Grupo COLE, Dpto. de Informática
E.S. de Enxeñaría Informática
Universidade de Vigo
Campus de As Lagoas
32004 – Ourense (España)
yerai.doval@uvigo.es

Carlos Gómez-Rodríguez, Jesús Vilares
Grupo LYS, Dpto. de Computación
Facultade de Informática
Universidade da Coruña
Campus de A Coruña
15071 – A Coruña (España)
{cgomezr, jvilares}@udc.es

Resumen: En las plataformas de *microblogging* abundan ciertos *tokens* especiales como los *hashtags* o las menciones en los que un grupo de palabras se escriben juntas sin espaciado entre ellas; p.ej.: `#añobisiesto` o `@ryanreynoldsnet`. Debido a la forma en que se escriben este tipo de textos, este fenómeno de ensamblado de palabras puede aparecer junto a su opuesto, la segmentación de palabras, afectando a cualquier elemento del texto y dificultando su análisis. En este trabajo se muestra un enfoque algorítmico que utiliza como base un modelo del lenguaje —en nuestro caso concreto uno basado en redes neuronales— para resolver el problema de la segmentación y ensamblado de palabras, en el que se trata de recuperar el espaciado estándar de las palabras que han sufrido alguna de estas transformaciones añadiendo o quitando espacios donde corresponda. Los resultados obtenidos son prometedores e indican que tras un mayor refinamiento del modelo del lenguaje se podrá sobrepasar al estado del arte.
Palabras clave: segmentación de palabras, ensamblado de palabras, español, modelos del lenguaje basados en redes neuronales

Abstract: In social media platforms special tokens abound such as hashtags and mentions in which multiple words are written together without spacing between them; e.g. `#leapyear` or `@ryanreynoldsnet`. Due to the way this kind of texts are written, this word assembly phenomenon can appear with its opposite, word segmentation, affecting any token of the text and making it more difficult to perform analysis on them. In this work we show an algorithmic approach based on a language model —in this case a neural model— to solve the problem of the segmentation and assembly of words, in which we try to recover the standard spacing of the words that have suffered one of these transformations by adding or deleting spaces when necessary. The promising results indicate that after some further refinement of the language model it will be possible to surpass the state of the art.
Keywords: word segmentation, word assembly, spanish, neural language models

1 Introducción

La popularización de la Web 2.0 y las redes sociales en los últimos años ha dado como resultado una fuente de datos prácticamente inagotable. Millones de usuarios crean contenidos cada minuto sobre temáticas muy diversas y en formatos de lo más variado: desde textos manifestando la opinión de un usuario ante un producto determinado hasta vídeos en los que se relata la vida cotidiana de un usuario. Tal y como ya se ha demostrado, el procesamiento y análisis de esta información puede resultar de gran utilidad en un amplio abanico de ámbitos: inteligencia de negocio (Gallinucci, Golfarelli, y Rizzi, 2013), lucha antiterrorista y contra el acoso (Alonso et al., 2015), predicción de la valoración de

* Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad español a través de los proyectos FFI2014-51978-C2-1-R y FFI2014-51978-C2-2-R, y por la Xunta de Galicia a través del programa Oportunius. We gratefully acknowledge NVIDIA Corporation for the donation of a GTX Titan X GPU used for this research.

ISSN 1135-5948

líderes políticos por la ciudadanía (Alonso y Vilares, 2016) o incluso predicción de brotes epidémicos (Cebrian, 2012). Este hecho pone de manifiesto la importancia del desarrollo de sistemas que permitan explotar de forma efectiva este tipo de fuentes.

Con respecto a las fuentes de datos en forma de texto, las plataformas de *microblogging* como Twitter gozan de gran popularidad actualmente. En éstas, abundan ciertos fenómenos de escritura no estándar, englobados en lo que se conoce como *texting*[1], así como *tokens*[2] especiales como los *hashtags*, muy empleados por los usuarios para etiquetar una entrada (p.ej.: Feliz 29 #añobisiesto) o usados en lugar de una o varias palabras del cuerpo del mensaje en que aparecen (p.ej.: Este año es #añobisiesto). En diversas aplicaciones, sin embargo, los *hashtags* que contienen varias palabras son tratados como una sola, reduciendo así la profundidad del análisis efectuado sobre estos *tokens* (p.ej.: en el caso de #añobisiesto, no se tendrían en cuenta las palabras `año` y `bisiesto` que conforman el *hashtag*). De igual forma, bien debido a errores de escritura o bien a la influencia del *texting*, cualquier conjunto consecutivo de palabras podrían aparecer reunidas en un mismo *token* (p.ej.: `nopuedeser!`) o incluso fragmentadas algunas de ellas en múltiples *tokens* (p.ej.: `im posible!`). En cualquier caso, estos fenómenos plantean serios problemas para todos aquellos sistemas que en algún momento trabajen a nivel de palabra, como todos los que se apoyan en diccionarios y lexicones, ya que no dispondrán de las entradas correctas; p.ej.: el sistema puede reconocer las palabras `año` y `bisiesto` pero no `añobisiesto`, ni tampoco `bi` y `siesto` por separado.

En este trabajo haremos frente a la incorrecta segmentación de palabras de un texto a través de un sistema formado por dos componentes. En primer lugar, un algoritmo de búsqueda basado en el *beam search* que tratará de encontrar la mejor segmentación posible para un texto de entrada. En segundo lugar, un modelo del lenguaje basado en redes neuronales que guiará al algoritmo en la toma de decisiones. El principal motivo de elegir una aproximación basada en redes neuronales es su probada eficacia en la tarea de modelado del lenguaje (Mikolov y Zweig, 2012; Józefowicz, Zaremba, y Sutskever, 2015). En cualquier caso, se ha probado también un modelo del lenguaje basado en *n-gramas* para confirmar este punto y se ha enfrentado nuestro sistema contra el conocido Word Breaker del Oxford Project de Microsoft[3], el cual también posee como base un modelo de *n-gramas*, aunque mucho más maduro.

Durante la evaluación se ha estudiado el impacto en el rendimiento general del sistema de los parámetros ajustables del algoritmo así como del modelo del lenguaje empleado. Los resultados obtenidos muestran el gran potencial de nuestra aproximación, debido en gran medida al modelo del lenguaje basado en redes neuronales. Así, con el refinamiento de este componente en el trabajo futuro se espera superar el excelente rendimiento del sistema Word Breaker, al cual nos hemos aproximado en gran medida a pesar de disponer de un corpus de entrenamiento mucho más reducido (Wikipedia v. *Web crawl*).

El resto del documento se estructura como sigue. En la Sección 2 se repasa el estado del arte. La Sección 3 muestra la aproximación propuesta para lidiar con el problema introducido en la presente sección. En la Sección 4 mostramos los resultados obtenidos al evaluar dicha aproximación. Por último, la Sección 5 expone las conclusiones obtenidas así como algunas líneas de trabajo futuro.

2 Trabajo relacionado

La segmentación de palabras es un paso previo muy importante en diversos tipos de sistemas: traducción automática (Koehn y Knight, 2003), recuperación de información (Alfonseca, Bilac, y Pharies, 2008) o reconocimiento del habla (Adda-decker, Adda, y Lamel, 2000) son algunos ejemplos. La segmentación de palabras para idiomas asiáticos cuenta con gran atención en la comunidad investigadora, debido principalmente a la inexistencia de caracteres de separación de palabras en estos idiomas (Suzuki, Brockett, y Kacmarcik, 2000; Huang y Zhao, 2007; Wu y Jiang, 1998). Por otra parte, los idiomas con una morfología compleja como el alemán, el noruego o el griego, se benefician también en gran medida de este tipo de sistemas de segmentación de palabras (Alfonseca, Bilac, y Pharies, 2008; Koehn y Knight, 2003).

[1] Modo de escritura en el que prima la tendencia a escribir como se habla en entornos informales.

[2] En este trabajo, secuencia de caracteres exceptuando los de espaciado delimitada por éstos.

[3] `https://www.projectoxford.ai/demo/weblm`

Con respecto a la problemática específica de la Web, en primer lugar aparecieron los sistemas que trataban de analizar URLs. Dado que estos elementos tampoco permiten el uso de espacios, las palabras que pudieran contener deberían ser obtenidas mediante un proceso de segmentación (Chi, Ding, y Lim, 1999; Wang, Thrasher, y Hsu, 2011). Más tarde, con la llegada de la Web 2.0, el uso de los *hashtags* se hizo muy popular, requiriendo de nuevo un proceso de segmentación para poder ser analizados convenientemente (Srinivasan, Bhattacharya, y Chakraborty, 2012; Maynard y Greenwood, 2014).

En general, el proceso seguido por la mayor parte de estos trabajos puede resumirse en (1) obtención de todas las posibles segmentaciones, para lo que pueden emplearse conjuntos de reglas (Koehn y Knight, 2003) o (también en conjunción con) recursos como diccionarios y lexicones de palabras o morfemas (Kacmarcik, Brockett, y Suzuki, 2000), y (2) selección de la segmentación más probable según alguna función de puntuación, para lo que puede usarse el análisis sintáctico de las posibles segmentaciones (Wu y Jiang, 1998) o la secuencia de palabras más probable según algún modelo del lenguaje (Wang, Thrasher, y Hsu, 2011). Otras aproximaciones, muy empleadas por ejemplo para el idioma chino, consideran el problema de la segmentación de palabras como un problema de etiquetación. Bajo este enfoque, el sistema ha de asociar a cada carácter una etiqueta de entre las siguientes: comienzo de palabra, mitad de palabra, fin de palabra o palabra de carácter único. Recientemente, a los métodos tradicionales de entrenamiento supervisado como *Maximum Entropy* (Berger, Pietra, y Pietra, 1996) o *Conditional Random Fields* (Lafferty, McCallum, y Pereira, 2001) se han unido los basados en redes neuronales artificales debido al auge del denominado *aprendizaje profundo* (*deep learning*) (Bengio, 2009).

De los trabajos vistos en este apartado cabe destacar tres puntos que diferencian a nuestra propuesta: (1) No tienen en cuenta el ensamblado de palabras, al menos de forma explícita, generalmente porque los casos de uso considerados no lo requieren. (2) Los experimentos iniciales en este trabajo se han realizado sobre textos en castellano. (3) Los sistemas que basan buena parte de su procesamiento en el uso de lexicones y diccionarios pueden verse limitados en contextos en los que abunde el ruido —como la propia Web 2.0—, el cual no es nuestro caso.

3 Aproximación propuesta

El sistema que proponemos está formado por dos componentes bien diferenciados y que trabajan de forma conjunta: el modelo del lenguaje basado en redes neuronales y el algoritmo de segmentación/ensamblado. El primero sirve como proveedor de información sobre el lenguaje al segundo, que empleará este conocimiento para realizar las segmentaciones y ensamblados que correspondan para recuperar el espaciado entre palabras en un texto de entrada.

3.1 Modelo del lenguaje basado en redes neuronales

Tradicionalmente, el enfoque empleado para el modelado del lenguaje era el conteo de *n-gramas* (Brown et al., 1992) (secuencia de n caracteres o palabras) junto con el uso de técnicas de suavizado que permitieran considerar secuencias desconocidas durante el entrenamiento (Chen, 1998). Sin embargo, posteriormente se demostró que mediante el uso de técnicas basadas en redes neuronales y representaciones continuas de los *tokens* de entrenamiento es posible el desarrollo de modelos del lenguaje que superan en rendimiento a los tradicionales (Mikolov y Zweig, 2012).

Uno de los tipos de redes neuronales más empleados para el modelado del lenguaje son las redes neuronales recurrentes (Mikolov y Zweig, 2012), en las cuales los elementos de procesado (neuronas) no sólo poseen información sobre su entrada actual (en un instante de tiempo t) sino también información sobre su operación con su entrada anterior (en el instante de tiempo $t-1$). Esta propiedad convierte a este tipo de redes en una herramienta muy adecuada para el modelado de series temporales, en general, y de secuencias de unidades de texto en particular, donde dada una secuencia de datos (caracteres) de entrada se obtenga una predicción para el dato (carácter) siguiente en la secuencia. De entre todas las redes de la familia de las recurrentes, concretamente las LSTMs (*Long Short Term Memory*) (Hochreiter y Schmidhuber, 1997) han demostrado un buen rendimiento en la tarea de modelado del lenguaje natural (Józefowicz, Zaremba, y Sutskever, 2015). Esto es debido en gran parte a su capacidad para retener la información de las

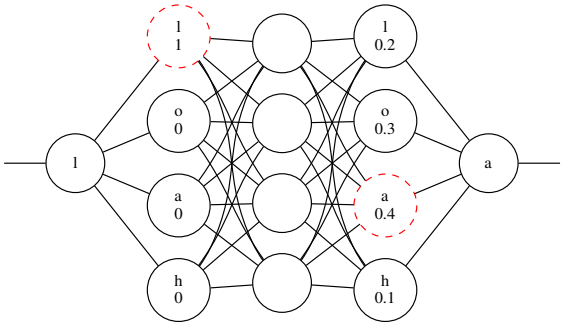

Figura 1: Predicción del siguiente carácter usando una red neuronal. El conjunto de caracteres considerados es $C = \{a, h, l, o\}$.

entradas más alejadas en el pasado, considerando así un contexto más amplio a la hora de procesar la entrada actual. Los mecanismos que facilitan dicha capacidad se encuentran ubicados en el diseño de cada uno de los elementos de procesamiento de este tipo de redes: las denominadas *puertas* (*gates*) de entrada, salida y olvido, junto con la *celda* de memoria. Estos elementos se encargan de regular el flujo de información dentro de cada unidad de procesamiento de la red, determinando así cuándo un dato es importante y debe ser recordado y cuándo debería ser olvidado por no ser relevante.

Por los motivos anteriormente expuestos, en nuestro trabajo hemos hecho uso de modelos del lenguaje basados en redes neuronales obtenidos mediante el entrenamiento de LSTMs a nivel de carácter. El objetivo de estas redes es el de estimar la distribución de probabilidad del carácter siguiente a uno dado. En la Figura 1 se muestra una representación gráfica de una red neuronal realizando el proceso de predicción. A nivel de implementación hemos empleado el *framework* de aprendizaje profundo `Torch`[4] junto con la implementación de LSTM realizada por Andrej Karpathy,[5] configurada en 3 capas ocultas de 1 000 neuronas cada una.

3.2 Algoritmo de segmentación y ensamblado de palabras

Asumiendo que se dispone de un modelo del lenguaje a nivel de caracteres entrenado sobre textos similares a los que nuestro sistema recibirá como entrada, el algoritmo de segmentación y ensamblado de palabras se muestra

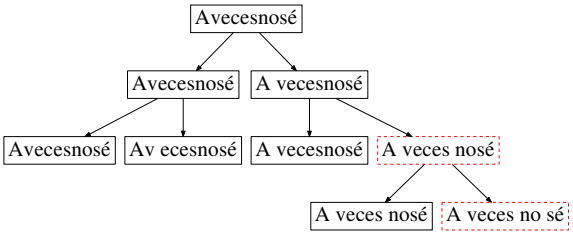

Figura 2: Ejecución del algoritmo con nivel máximo de recursión 3 y selección de un solo candidato al alcanzar resultados parciales.

en el Algoritmo 1.[6] La idea fundamental es la siguiente: primero, se eliminan los caracteres de espaciado del texto de entrada (primera sentencia de la línea 18). Luego, para cada carácter del texto resultante se obtiene mediante el modelo del lenguaje la distribución de probabilidad para el siguiente carácter, teniendo en cuenta además los caracteres anteriores al actual (línea 4). Se introducirá entonces un carácter de espacio justo después del actual siempre que aquél posea una mayor probabilidad que el carácter siguiente del texto o que su probabilidad sea alta de acuerdo con el parámetro *threshold* (condición en línea 6, inserción en línea 11) explicado más abajo. En cualquier caso, se ramifica la ejecución mediante una llamada recursiva que considere el no añadir o eliminar el carácter de espacio para una mayor exhaustividad en la búsqueda de la solución óptima (línea 8). Con el fin de mantener el tiempo de ejecución del algoritmo dentro de unos límites aceptables así como dirigir este proceso de búsqueda, la profundidad de recursión se limita terminando la ejecución al alcanzar un determinado nivel en las llamadas recursivas (línea 13). En este momento se dispone de un conjunto de resultados parciales de entre los que se escogen los mejores según la media de sus probabilidades logarítmicas (línea 24) y con los que se reinicia el algoritmo (bucle en la línea 19). Este proceso se repite hasta que no quede más texto de entrada que procesar (línea 25). Una representación gráfica de este proceso puede observarse en la Figura 2.

Varios aspectos de este algoritmo son ajustables por el usuario por medio de tres parámetros de entrada: *threshold*, *stop* y *prun*. El primero es empleado para permitir los casos de recursión siempre que el carácter actual considerado se encuentre entre los

[4] `http://torch.ch/`

[5] `https://github.com/karpathy/char-rnn`

[6] Código y recursos disponibles en `https://cloud.wyffy.com/index.php/s/YFgcHU1inp2WBMs`.

Algoritmo 1 Algoritmo de segmentación y ensamblado de palabras. *text* contiene el texto de entrada, *acctext* acumula el texto de salida y $\top_n(C)$ representa el conjunto de los n elementos más probables del conjunto C.

```
1: function ESPACIAR(text, model, threshold, stop, level, acctext)
2:     resultado ← ∅ ; b ← stop − level                    ▷ Límite para la recursión
3:     for i ∈ [1, |text|] do
4:         pred ← predecir(text_i, acctext, model)          ▷ Lista de tuplas (carácter, probabilidad)
5:         acctext ← acctext + text_i
6:         if pred_espc > pred_text_{i+1} ∨ espc ∈ ⊤_threshold(pred) then
7:             if pred_text_{i+1} ∈ ⊤_threshold(pred) then
8:                 recur ← espaciar(text_{i+1..|text|}, model, threshold, stop, level + 1, acctext)
9:                 resultado ← resultado ∪ recur ; b ← b − 1
10:            end if
11:            acctext ← acctext + espc
12:        end if
13:        if b =< 0 then break end if
14:    end for
15:    return resultado ∪ {(acctext, P(acctext, model))}   ▷ Lista de tuplas (cadena, probabilidad)
16: end function
17: function SEGMENTAR(text, model, threshold, stop, prun)
18:    text ← remove(text, espc) ; partials ← {([], 0)}
19:    repeat
20:        tmp ← ∅
21:        for part ∈ partials do
22:            tmp ← tmp ∪ espaciar(text_{l+1..|text|}, model, threshold, stop, 1, part_cadena)
23:        end for
24:        partials ← ⊤_prun(tmp)
25:    until ∀part_cadena, l ← |remove(part_cadena, espc)|, text_{l+1..|text|} = []
26:    return partials
27: end function
28: segmentar(text, model, threshold, stop, prun)        ▷ Llamada inicial a la función
```

threshold primeros más probables de la predicción realizada para el siguiente carácter. *stop* determina el nivel máximo de profundidad a la que puede llegar el algoritmo con las llamadas recursivas. Por último, *prun* determina cuántos de los mejores resultados parciales se escogen tras una parada del algoritmo (por llegar a la profundidad máxima *stop*) y con los que se reiniciará la ejecución. El caso particular de $stop = 2$ es equivalente a una búsqueda tipo *beam search*, ya que se seleccionan los mejores nodos después de expandir cada nivel.

La ejecución del algoritmo descrito dará como resultado una lista con las posibles segmentaciones para una entrada *texto* y un modelo del lenguaje *model*.

4 Evaluación

Siendo el sistema descrito nuestra primera propuesta para resolver el problema de la segmentación y ensamblado de palabras, era de nuestro interés el realizar una evaluación que nos permitiese observar sus puntos fuertes y débiles. Así, el análisis del rendimiento realizado trata de considerar todos aquellos factores que podrían tener un mayor impacto sobre el rendimiento final del sistema, cuantificado en términos de *precisión*, medida como el número de ocasiones en las que el sistema da el resultado correcto sobre el número total de instancias del conjunto de evaluación, y *tiempo de ejecución* en segundos.

Con respecto al modelo del lenguaje, se ha comparado la puntuación de validación (*negative log-likelihood*, *NLL*) obtenida en el entrenamiento de cada modelo con la precisión y tiempo de ejecución final del sistema. Cabe esperar que a menor puntuación de validación mayor precisión, quedando como incógnita el comportamiento del tiempo de ejecución. Durante el entrenamiento se han empleado los artículos de parte del volcado de la Wikipedia en castellano del 28/02/2015, filtrado mediante el programa `wikiextractor`[7] y con las expresiones propias del lenguaje de marcado de la Wikipedia eliminadas. Del resultado se extrajo el primer 25 % de líneas de texto, totalizando 4 000 000 de líneas, 5 666 491 de oraciones, 134 360 571 de palabras, 1 245 884 de palabras distintas, 815 525 127 de caracteres y 5 128 caracteres distintos. Para entrenar el mejor modelo del lenguaje aquí presentado fueron necesarios aproximadamente dos días.

En el caso del algoritmo propuesto, se ha estudiado el impacto de los parámetros de entrada *threshold*, *stop* y *prun*. Dado que a ma-

[7] `https://github.com/attardi/wikiextractor`

	NLL	*P*	*t* (s)
M. neuronas	3.1178	0.0	3799
	2.9974	0.0	8259
	1.5485	0.52	6888
	1.3198	0.63	6442
	0.9898	**0.71**	**5913**
		P	*t* (s)
threshold	10	0.66	2258
	20	**0.70**	**4401**
	50	0.71	6396
prun	1	0.41	3669
	2	0.70	5471
	3	0.77	7160
	4	**0.79**	**8794**
	5	0.82	21478
stop	2	0.65	3655
	3	**0.71**	**5913**
	4	0.75	12714
	5	0.77	36594
M. *n-gramas*	–	0.51	15722

Tabla 1: Resultados de precisión y tiempo de ejecución para cada modelo del lenguaje y valor de los parámetros del algoritmo probados.

yor valor asignado a estos parámetros mayor sería la exhaustividad de la búsqueda, cabría esperar encontrar curvas ascendentes de precisión y tiempo de ejecución.

Para la evaluación se ha tomado el último 25 % de líneas de texto del volcado procesado de la Wikipedia. Además, se ha dividido el texto resultante en secuencias de caracteres de longitud mínima 20, realizando los cortes en el siguiente carácter de espaciado, y para cada secuencia se han eliminado los caracteres de espaciado. Del resultado se escogen de forma aleatoria 1 000 líneas de texto (5 065 palabras, 2 722 palabras distintas, 42 562 caracteres y 94 caracteres distintos).

Los datos reflejados en la Tabla 1 se muestran coherentes con nuestra intuición inicial. Los mejores modelos del lenguaje permiten obtener los mejores resultados además de reducir el tiempo de ejecución. Este último dato surge como consecuencia de una menor ramificación en la ejecución del algoritmo con estos modelos, los cuales son capaces de descartar con mayor seguridad opciones de segmentación poco probables. Únicamente nos encontramos con un dato anómalo para el peor modelo en el que el tiempo de ejecución alcanza un valor mínimo. En este caso, de nuevo el modelo descarta con mucha seguridad la mayoría de las opciones de segmentación aunque, en vista de los resultados de precisión, lo hace de forma incorrecta.

En cuanto a los parámetros del algoritmo, cuanto mayor sean los valores asignados mayor será el tiempo de ejecución, con ciertos valores estableciendo límites entre lo que sería una ejecución más o menos rápida y otra mucho más lenta. Así, pueden considerarse valores adecuados $threshold = 20$, $prun = 4$ y $stop = 3$, considerando incrementar ligeramente cada uno de ellos si se desea maximizar el rendimiento a costa del tiempo de ejecución.

En conjunto, asignando a los parámetros del algoritmo los valores $threshold = 30$, $prun = 5$ y $stop = 4$, la precisión alcanzada es de 0.82. En estos momentos, el factor limitante en cuanto a rendimiento y tiempo de ejecución parece ser el modelo del lenguaje, tal y como cabría esperar. La dificultad en este punto es la mejora de este componente, lo cual requiere en cualquier caso de mayores cantidades de tiempo para el proceso de entrenamiento. Sin embargo, los datos obtenidos nos indican que invirtiendo más recursos en este proceso se podrá obtener un sistema muy competitivo, dado que no hemos encontrado todavía evidencias de que la tendencia creciente del rendimiento del sistema con respecto al ajuste del modelo del lenguaje vaya a detenerse próximamente.

Para contrastar los resultados obtenidos, en primer lugar hemos probado a sustituir el modelo del lenguaje empleado por uno basado en *n-gramas*, con el objetivo de validar el enfoque neuronal. Se construyó pues, mediante el paquete *software* `kenlm`,[8] un modelo de orden 5 en el que se dejó el resto de parámetros del paquete en su valor por defecto. Empleando de nuevo los valores $threshold = 30$, $prun = 5$ y $stop = 4$ en el algoritmo, la precisión obtenida es de 0.51, claramente inferior a la conseguida por nuestro modelo basado en redes neuronales.

En segundo lugar, hemos comparado el rendimiento de nuestro sistema contra el de uno bien conocido en el ámbito de la segmentación: el Word Breaker de Microsoft. Realizando una ejecución con el *Bing body model* de orden 5, este sistema obtiene una precisión de 0.85, posicionándose ligeramente por encima de nuestra propuesta. Llegados a este punto, es conveniente resaltar que la ejecución del Word Breaker no se realizó sobre el mismo conjunto de evaluación empleado hasta ahora, ya que debido a limitaciones de este sistema fue necesario procesar las instan-

[8] `http://kheafield.com/code/kenlm.tar.gz`

cias de evaluación disponibles de modo que sólo contuvieran caracteres alfanuméricos del conjunto de minúsculas de ASCII. Estas limitaciones surgen muy probablemente de la necesidad de reducir efectos indeseados como la dispersión en el modelo del lenguaje, basado en *n-gramas*. Además, el sistema de Microsoft cuenta con la ventaja añadida de haber sido entrenado con un *Web crawl*, un corpus mucho más extenso y completo que el empleado para entrenar nuestras redes neuronales. En definitiva, aun operando en un dominio mucho más restringido y alimentado por mejores recursos lingüísticos, el Word Breaker no consigue resultados notablemente mejores, lo cual indica la viabilidad y prometedor futuro de nuestra propuesta. De cualquier modo, las razones para usar este sistema para la comparación han sido su nivel de madurez, fácil accesibilidad a través de un API REST y su disponibilidad para trabajar con textos en castellano.

5 Conclusiones y trabajo futuro

En este trabajo hemos presentado nuestra primera propuesta para la segmentación y ensamblado de palabras en textos escritos en español. Nuestra aproximación hace uso de un algoritmo parametrizable que aprovecha la información condensada en un modelo del lenguaje a nivel de caracteres basado en redes neuronales, implementado como una red neuronal recurrente tipo LSTM. Los experimentos realizados nos han permitido observar el rendimiento y tiempo de ejecución del sistema en función de los valores asignados a los parámetros del algoritmo y el modelo del lenguaje empleado. Se han observado así los valores más convenientes para dichos parámetros de modo que se contenga el tiempo de ejecución del sistema y no se obstaculice su rendimiento. Por otra parte, los resultados obtenidos muestran claramente la importancia de continuar entrenando modelos del lenguaje que posean un mayor ajuste con los textos sobre los que se emplee el sistema para mantener la tendiencia ascendente del rendimiento, lo cual en estos momentos constituye el siguiente desafío. Por último, el modelo del lenguaje basado en redes neuronales se ha mostrado muy superior al de *n-gramas* empleado como *baseline*, y un sistema bien conocido como el Word Breaker, a pesar de trabajar en un ámbito más restringido, no ha destacado contra nuestra propuesta.

Como línea futura de trabajo nos centraremos en la parte de modelado del lenguaje, pieza fundamental del sistema y que presenta múltiples oportunidades de mejora. Por una parte consideramos la inclusión de un modelo del lenguaje entrenado en orden inverso. De esta forma no sólo poseeríamos información sobre los caracteres anteriores para realizar las predicciones de los siguientes, sino también sobre los posteriores. Por otra parte, la mejora del modelo del lenguaje empleado: bien mediante una búsqueda más exhaustiva del mejor conjunto de hiperparámetros para la arquitectura actual (LSTM) o bien a través de otros tipos de arquitectura como podrían ser las redes convolucionales. En cualquier caso, el proceso de entrenamiento con la arquitectura actual continúa su curso y se espera seguir obteniendo resultados cada vez mejores. También se considerará el uso de otros corpus de entrenamiento más extensos que el utilizado actualmente, como por ejemplo un mayor porcentaje del volcado de la Wikipedia o un *Web crawl* obtenido del proyecto Common Crawl.[9] Finalmente, tenemos intención de entrenar modelos para los idiomas en los que la segmentación es un problema bien conocido, como el chino o el alemán.

Bibliografía

Adda-decker, M., G. Adda, y L. Lamel. 2000. Investigating text normalization and pronunciation variants for german broadcast transcription. En *ICSLP'2000*, páginas 266–269.

Alfonseca, E., S. Bilac, y S. Pharies. 2008. Decompounding query keywords from compounding languages. En *Proc. of the 46th Annual Meeting of the ACL: Short Papers*, HLT-Short '08, páginas 253–256, Stroudsburg, PA, USA. ACL.

Alonso, M. A., C. Gómez-Rodríguez, D. Vilares, Y. Doval, y J. Vilares. 2015. Seguimiento y análisis automático de contenidos en redes sociales. En *Actas: III Congreso Nacional de i+d en Defensa y Seguridad, DESEi+d 2015*, páginas 899–906.

Alonso, M. A. y D. Vilares. 2016. A review on political analysis and social media. *Procesamiento del Lenguaje Natural*, 56:13–24.

[9] `https://commoncrawl.org/`

Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Berger, A. L., S. D. Pietra, y V. J. D. Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Brown, P. F., P. V. deSouza, R. L. Mercer, V. J. D. Pietra, y J. C. Lai. 1992. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, Diciembre.

Cebrian, M. 2012. Using friends as sensors to detect planetary-scale contagious outbreaks. En *Proc. of the 1st International Workshop on Multimodal Crowd Sensing*, CrowdSens '12, páginas 15–16, New York, NY, USA. ACM.

Chen, S. F. 1998. An empirical study of smoothing techniques for language modeling. Informe técnico.

Chi, C.-H., C. Ding, y A. Lim. 1999. Word segmentation and recognition for web document framework. En *Proc. of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, páginas 458–465, New York, NY, USA. ACM.

Gallinucci, E., M. Golfarelli, y S. Rizzi. 2013. Meta-stars: Multidimensional modeling for social business intelligence. En *Proc. of the Sixteenth International Workshop on Data Warehousing and OLAP*, DOLAP '13, páginas 11–18, New York, NY, USA. ACM.

Hochreiter, S. y J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Huang, C. y H. Zhao. 2007. Chinese word segmentation: A decade review. *Journal of Chinese Information Processing*, 21(3):8–20.

Józefowicz, R., W. Zaremba, y I. Sutskever. 2015. An empirical exploration of recurrent network architectures. En *Proc. of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France*, páginas 2342–2350.

Kacmarcik, G., C. Brockett, y H. Suzuki. 2000. Robust segmentation of japanese text into a lattice for parsing. En *Proc. of the 18th Conference on Computational Linguistics - Volume 1*, COLING '00, páginas 390–396, Stroudsburg, PA, USA. ACL.

Koehn, P. y K. Knight. 2003. Empirical methods for compound splitting. En *Proc. of the Tenth Conference on European Chapter of the ACL - Volume 1*, EACL '03, páginas 187–193, Stroudsburg, PA, USA. ACL.

Lafferty, J. D., A. McCallum, y F. C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. En *Proc. of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA*, páginas 282–289.

Maynard, D. y M. A. Greenwood. 2014. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. En *LREC*, páginas 4238–4243.

Mikolov, T. y G. Zweig. 2012. Context dependent recurrent neural network language model. En *2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA*, páginas 234–239.

Srinivasan, S., S. Bhattacharya, y R. Chakraborty. 2012. Segmenting web-domains and hashtags using length specific models. En *Proc. of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, páginas 1113–1122, New York, NY, USA. ACM.

Suzuki, H., C. Brockett, y G. Kacmarcik. 2000. Using a broad-coverage parser for word-breaking in japanese. En *Proc. of the 18th Conference on Computational Linguistics - Volume 2*, COLING '00, páginas 822–828, Stroudsburg, PA, USA. ACL.

Wang, K., C. Thrasher, y B.-J. P. Hsu. 2011. Web scale nlp: A case study on url word breaking. En *Proc. of the 20th International Conference on World Wide Web*, WWW '11, páginas 357–366, New York, NY, USA. ACM.

Wu, A. y Z. Jiang. 1998. Word segmentation in sentence analysis. En *Proc. of the 1998 International Conference on Chinese Information Processing*, páginas 169–180.