Prototipado Rápido de un Sistema de Normalización de Tuits: Una Aproximación Léxica*

Rapid prototyping of a Tweet Normalization System: A Lexical Approach

Jesús Vilares y **Miguel A. Alonso** y **David Vilares**
Departamento de Computación, Universidade da Coruña
Campus de Elviña s/n, 15071 – A Coruña
{jesus.vilares, miguel.alonso, david.vilares}@udc.es – www.grupolys.org

Resumen: Este trabajo describe el sistema de normalización de tuits en español desarrollado por el Grupo de Lengua Y Sociedad de la Información (LYS) de la Universidade da Coruña para el Tweet-Norm 2013. Se trata de un sistema conceptualmente sencillo y flexible que emplea pocos recursos y que aborda el problema desde un punto de vista léxico.
Palabras clave: Twitter en español; normalización de tuits; variación léxica; procesamiento léxico; arquitectura en *pipeline*

Abstract: This work describes the system for the normalization of tweets in Spanish designed by the Language in the Information Society (LYS) Group of the University of A Coruña for Tweet-Norm 2013. It is a conceptually simple and flexible system, which uses few resources and that faces the problem from a lexical point of view.
Keywords: Twitter in Spanish; tweet normalization; lexical variation; lexical processing; pipeline architecture

1. Introducción y objetivos

El uso del lenguaje en *microtextos* como los de Twitter y los SMS, denominado *texting*, se aleja mucho del estándar (López Rúa, 2007; Oliva et al., 2013): empleo de ciertas convenciones en la comunicación (p.ej. usar *emoticonos* para mostrar emociones), ignorar la ortografía (p.ej. falta de tildes, intercambio de consonantes homófonas como `b`/`v` o `c`/`q`/`k`, etc.), ajustar el mensaje a la longitud máxima permitida (p.ej. 140 letras en un tuit) mediante acortamientos, contracciones, transformaciones, etc. Hablaremos entonces de *variantes léxicas* para referirnos a los términos resultantes (Han, Cook, y Baldwin, 2013), siendo tales fenómenos, mayormente de base fonética, específicos del idioma. Trabajos como los de Thurlow (2003) para el inglés y López Rúa (2007) para el español describen su fenomenología particular.

Asimismo, aplicar técnicas de Procesamiento del Lenguaje Natural a los millones de tuits generados a diario sería de gran interés para la inteligencia empresarial, pero las alteraciones del lenguaje antes descritas lo dificultan. Una solución es transformar dicho texto a lenguaje estándar, es decir, *normalizarlo*. Si bien existen precedentes para otros idiomas (predominando el inglés) (Xue, Yin, y Davison, 2011; Costa-Jussà y Banchs, 2013; Beaufort, 2011; Han, Cook, y Baldwin, 2013; Liu, Weng, y Jiang, 2012), apenas existen trabajos para el español (Oliva et al., 2013). Se trata, además, de propuestas complejas, fruto de un largo desarrollo. En contraste, para nuestra participación en Tweet-Norm hemos optado por una propuesta sencilla aunque flexible, que emplease pocos recursos. Nuestro objetivo no ha sido tanto buscar un alto rendimiento como mostrar hasta dónde es posible avanzar empleando técnicas sencillas.

La estructura del artículo es como sigue. La Sección 2 aborda la arquitectura del sistema, su funcionamiento y los recursos empleados. La Sección 3 detalla los resultados obtenidos. Finalmente, la Sección 4 presenta nuestras conclusiones y trabajo futuro.

2. Arquitectura y recursos

Buscando simplicidad y flexibilidad, optamos por una arquitectura en *pipeline* que nos permitía integrar, eliminar e intercambiar módulos de forma sencilla. Dichos módulos se comunican empleando un formato de representación intermedia codificado como texto y de

* Trabajo parcialmente subvencionado por el Ministerio de Economía y Competitividad y FEDER (proyecto TIN2010-18552-C03-02) y por la Xunta de Galicia (ayudas CN 2012/008 y CN 2012/319).

naturaleza estructurada y jerarquizada. En este esquema un *tuit* está formado por *términos* y para cada término existe una serie de *candidatos* para su normalización.

Dado que la tarea de *normalización* consiste en etiquetar una *palabra fuera-de-vocabulario* (OOV, *out-of-vocabulary word*) como correcta o bien proponer su forma correcta, dicha tarea puede verse como un proceso en dos fases. Primero, identificar las *palabras dentro-del-vocabulario* (IV, *in-vocabulary word*) del tuit, y así saber cuáles son las OOVs. Segundo, proponer la forma correcta de las OOVs detectadas, lo que a su vez implica: *(a)* identificar si se trata de una palabra correcta pero desconocida (que denominaremos *OOVs propias*) o bien de una *variante léxica*; *(b)* en este último caso, obtener su forma estándar normalizada.

Esto nos llevó a plantear la arquitectura general del sistema en tres etapas: *(1)* los tuits se tokenizan y preprocesan; *(2)* identificamos las IVs en base al *vocabulario del sistema*, obteniendo el *conjunto inicial de OOVs*; *(3)* clasificamos las OOVs en *propias* y *variantes léxicas*, y proponemos para éstas su forma estándar normalizada, aplicando para ello una serie de procesos de normalización sobre el conjunto inicial de OOVs, obteniendo así sus *normalizaciones candidatas* para luego elegir la más adecuada.

Sobre esa arquitectura general se plantearon dos versiones del sistema (véase Figura 1), las cuales describimos a continuación.

2.1. Planteamiento inicial

En un principio el sistema se concibió en torno a dos herramientas: el preprocesador multilingüe `Twokenize` (Owoputi et al., 2013), diseñado para tokenizar tuits así como identificar ciertas entidades de interés tales como cifras, emoticonos, URLs, etc.; y la herramienta de corrección ortográfica GNU Aspell[1] (rel. 0.60) junto con su diccionario. Debemos señalar que si bien el mecanismo de búsqueda de correcciones de Aspell combina una búsqueda por distancia de edición con una búsqueda fonética basada en el *algoritmo del metáfono* (Philips, 1990), en el caso del español ésta última no está disponible. Como muestra la parte izquierda de la Figura 1, ambas herramientas nos permitirían abordar de forma sencilla las fases de preprocesamiento e identificación de OOVs.

[1] `http://aspell.net`

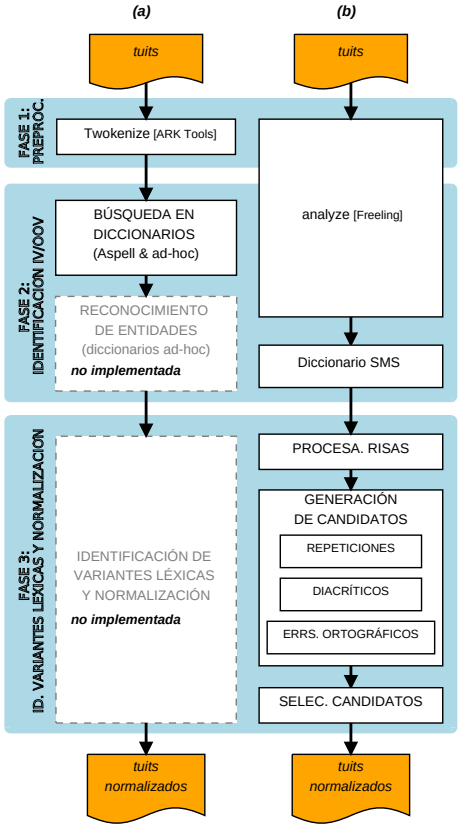

Figura 1: Arquitectura del sistema: *(a)* planteada inicialmente; *(b)* implementada

Otras opciones barajadas fueron el tokenizador genérico del corpus Europarl (Koehn, 2005), así como otros diccionarios del español a los que teníamos acceso: el del AnCora (Taulé, Martí, y Recasens, 2008), el del proyecto ERIAL (Barcala et al., 2002) y el MULTEXT (Véronis, 1999). Sin embargo elegimos aquéllos por su libre disponibilidad, lo que facilita la replicabilidad de los experimentos, así como por sus buenas prestaciones. Asimismo, tras detectar ciertas carencias del diccionario de Aspell para ciertos tipos de palabras frecuentes en este tipo de textos (jerga del dominio, interjecciones, onomatopeyas, etc.), nos preparamos para ampliar el vocabulario del sistema empleando una serie de diccionarios y *gazetteers ad-hoc* creados a partir de diversas fuentes web libremente disponibles y de nuestra propia experiencia. También se había previsto realizar un proceso de reconocimiento de entidades en base a los *gazetteers* antes referidos.

Finalmente, en la tercera fase del proceso clasificaríamos y normalizaríamos las OOVs.

2.2. Sistema final

Sin embargo, tal arquitectura no fue la que finalmente se implementó, siendo ésta la que se muestra en la parte derecha de la Figura 1. La razón fue que, por un error de interpretación, creímos que no se permitía emplear el mismo tipo de preprocesamiento llevado a cabo por la organización para la creación del corpus de entrenamiento y que empleaba la herramienta `analyze` de FREELING (Padró y Stanilovsky, 2012).[2] Fue después, con el sistema ya en un estado avanzado de implementación, cuando nos percatamos del error. Decidimos entonces cambiar a `analyze` para una mejor comparabilidad de nuestros resultados con los del resto de participantes, que supusimos emplearían mayormente FREELING. Esto supuso un nuevo retraso que, dado lo ajustado de los plazos, hizo que entre otras cosas no pudiéramos integrar el reconocedor de entidades previsto. Por contra, la fase de identificación de OOVs se simplificó bastante respecto a la planteada inicialmente, de tal forma que el único diccionario empleado, aparte del de FREELING, fue uno de jerga SMS.[3]

Llegados a este punto, con el conjunto inicial de OOVs identificado, se pasa a la fase de normalización. Primero se detectan y normalizan las onomatopeyas de risas (p.ej. `jajaa`) usando patrones. Seguidamente se generan, de forma acumulativa, posibles normalizaciones para las OOVs remanentes en base a los mecanismos de variación considerados, que describimos en la Sección 2.3. Dicho proceso es iterativo y se recoge en el Algoritmo 1.

Usar una *cola de prioridad* (Q_{gen}) como estructura de almacenamiento nos permite primar los candidatos según sean o no IVs, lo complejo de su generación, y lo frecuente y factible del mecanismo de variación. Es más fácil, por ejemplo, que una OOV sea una variante por repetición de vocales que lo sea por repetición de vocales, eliminación de diacríticos y errores ortográficos todo a la vez. De este modo el supuesto mejor candidato será siempre el primero.

Finalmente, una vez generados las posibles normalizaciones de cada OOV, éstas son normalizadas al primero, y mejor, de sus candidatos. Aunque no lo parezca, implícitamente se está realizando la mencionada clasificación de las OOVs en *OOVs propias*, aquéllas normalizadas a ellas mismas por no tener candidatos mejores, y en *variantes léxicas*, aquéllas que cuentan con otros candidatos mejores y que son normalizadas a dichos términos.

2.3. Mecanismos de variación

Actualmente consideramos únicamente tres posibles fuentes de variación, que son abordadas desde una perspectiva léxica para así limitar la complejidad del sistema.

En primer lugar, la *repetición de caracteres*. Si una OOV contiene dos o más letras iguales seguidas podría ser una variante de este tipo (p.ej. `besooos` vs. `besos`). Las normalizaciones consideradas serán, por orden: la palabra sin repeticiones, con repeticiones de máximo longitud dos, y las palabras resultantes de eliminar todas sus repeticiones excepto una, reducida a longitud dos.[4]

A continuación, los *errores en los diacríticos*. Cualquier OOV con vocales podría ser una variante de este tipo (p.ej. `camion` vs. `camión`). Se eliminan sus diacríticos (de tenerlos) y se comprueba si el término resultante es una IV. Si lo es, se toma como único candidato. En los otros casos se generan con ASPELL sus correcciones candidatas,[5] quedándonos sólo con aquéllas que difieren únicamente en los diacríticos.

Finalmente, *otros errores ortográficos*. Cualquier OOV podría ser una forma mal escrita de otra palabra (p.ej. `palabar` vs. `palabra`). Los candidatos serán las correcciones devueltas por ASPELL.

Asimismo se intentó incorporar, sin éxito, un cuarto mecanismo de normalización basado en búsqueda fonética empleando el *algoritmo del metáfono* (Philips, 1990), el cual, como se indicaba en la Sección 2.1, no está disponible para el español en ASPELL. Para ello creamos un índice invertido del vocabulario en base a sus transcripciones fonéticas usando una adaptación al español del algoritmo (Mosquera, 2011). A la hora de la búsqueda dicha transcripción se combinaba con un algoritmo de correspondencia vocálica pues las vocales eran obviadas por el algoritmo.

[2] Ficheros de configuración en `http://devel.cpl.upc.edu/freeling/svn/trunk/src/main/twitter/`.

[3] Descubrimos después que, por algún error, el volcado del diccionario contenía únicamente entradas hasta la *K*, mientras que su web (`http://www.diccionariosms.com`) parece haber sido ya cerrada.

[4] Asimismo, se priorizan las repeticiones de ciertas letras en base a su naturaleza y frecuencia en el diccionario. Por orden: `e`, `o`, `r`, `l`, `c` y `n`.

[5] Se emplean siempre simultáneamente los *suggestion modes* `ultra` y `normal` de ASPELL.

Algoritmo 1 Generación de normalizaciones candidatas.

Entrada:

t_{oov}: término OOV a normalizar.

$g_i()$: función generadora de normalizaciones candidatas para el fenómeno v_i, donde $\mathcal{V} = (v_1, \ldots, v_N)$ son los fenómenos de variación abordados, ordenados por precedencia y frecuencia.

$apl_i()$: función booleana que detecta si un término podría ser una variante léxica de tipo v_i.

Salida:

Q_{gen}: *lista de candidatos para la normalización*. Actúa a modo de *cola de prioridad* donde los candidatos han sido almacenados por orden de inserción priorizando aquéllos que son IVs sobre los OOVs. De este modo los candidatos IVs se situarán en la parte anterior de la cola y los OOVs en la posterior, y a su vez, dentro de cada clase, los candidatos estarán por orden de creación, habiendo sido generados de forma acumulativa aplicando los diferentes mecanismos de normalización asociados a $\mathcal{V}$. En resumen: primero, los candidatos IVs generados a partir de t_{oov} empleando g_1; luego los generados empleando g_2; los generados aplicando, secuencialmente, g_1 y g_2; los generados con g_1 y g_3; con g_2 y g_3; con g_1, g_2 y g_3; etc.; y a continuación lo mismo para los candidatos OOVs, con el propio t_{oov} en primer lugar.

```
1: function GENERAR_CANDIDATOS (t_oov, ⟨g_i, apl_i⟩_{v_i∈V})
2:     Q_gen ← (t_oov)        (* Inicializamos la lista de candidatos al propio t_oov. *)
3:     for all v_i ∈ V do     (* Para cada fenómeno de variación v_i a tratar ... *)
4:         Q_aux ← Q_gen          (*... tomamos los candidatos generados hasta entonces ...*)
5:         for all t_j ∈ Q_aux do   (*... y para cada uno de dichos candidatos t_j ...*)
6:             if (apl_i(t_j) = true) then   (*... comprobamos si podría ser una variante de tipo v_i. Si lo es ...*)
7:                 C ← g_i(t_j)              (*... generamos sus normalizaciones candidatas para ese fenómeno...*)
8:                 for all c_k ∈ C do        (*... y para cada posible normalización c_k ...*)
9:                     if c_k ∈ Q then           (*... comprobamos si ya había sido generada antes: ...*)
10:                        actualizar(Q_gen, c_k, v_i)   (*... de ser así, actualizamos el conjunto de operaciones por
                                                          las que dicho candidato c_k es generado ...*)
11:                    else
12:                        encolar(Q_gen, c_k, v_i)      (*... sino, encolamos el nuevo candidato. *)
13:                    end if
14:                end for
15:            end if
16:        end for
17:    end for
18:    return Q_gen    (* Una vez tratados todos los fenómenos, devolvemos los candidatos obtenidos. *)
19: end function
```

corpus	*dev100*		*dev500*		*test600*	
	base	***norm***	***base***	***norm***	***base***	***norm***
err	8	8	46	46	134	134
pos	3	41	62	224	47	219
neg	85	47	406	244	448	276
$accu_{py}$	2.46	33.61	9.49	34.30	7.10	33.08
$accu_{php}$	–	–	–	–	**7.10**	**33.53**

Cuadro 1: Resultados obtenidos.

3. Evaluación

Por limitaciones de tiempo sólo tenemos dos tipos de *runs*: *(a)* un *baseline* sin normalizar que toma directamente la salida de `analyze` (*base*); y *(b)* el resultado del proceso de normalización descrito anteriormente (*norm*). Los resultados obtenidos se muestran en el Cuadro 1, indicando: el número de errores de alineamiento (*err*), el número de OOVs normalizadas de forma correcta (*pos*) e incorrecta (*neg*), y el *accuracy*, tanto el calculado mediante el *script* PYTHON inicial ($accu_{py}$) como con el *script* PHP final ($accu_{php}$).

Su rendimiento, del 33-34 %, nos sitúa a la cola de los participantes, si bien muy cerca de otros, por lo que dada la simplicidad de nuestra aproximación y los pocos fenómenos de variación tratados, no es un mal resultado.

4. Conclusiones y trabajo futuro

Ésta ha sido nuestra primera incursión formal en la normalización de tuits en español. Se trata de un sistema sencillo, que opera a nivel léxico usando pocos recursos, y que tiene una arquitectura en *pipeline* que lo hace muy flexible. Su rendimiento ha sido satisfactorio a pesar de su simplicidad.

En el futuro, además de abordar el resto de fuentes de variación más comunes, pretendemos incluir diversas mejoras: un formato de representación intermedia entre módulos en XML; integrar un detector del idioma y un reconocedor de entidades; emplear puntuaciones; y usar información contextual para mejorar el filtrado de candidatos.

Bibliografía

Barcala, F. M., E. M. Domínguez, M. A. Alonso, D. Cabrero, J. Graña, J. Vilares, M. Vilares, G. Rojo, M. P. Santalla, y S. Sotelo. 2002. Una aplicación de RI basada en PLN: el proyecto ERIAL. En *Actas de las I Jornadas de Tratamiento y Recuperación de Información (JOTRI 2002)*, págs. 165–172.

Beaufort, R. 2011. From SMS gathering to SMS normalization: finite-state algorithms. TCTS Lab's seminars, University of Mons. Disponible en `http://cental.fltr.ucl.ac.be/team/beaufort/file/TCTS2011_beaufort.pdf`.

Costa-Jussà, M. R. y R. E. Banchs. 2013. Automatic normalization of short texts by combining statistical and rule-based techniques. *Language Resources and Evaluation*, 47(1):179–193.

Han, B., P. Cook, y T. Baldwin. 2013. Lexical normalization for social media text. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(1).

Koehn, P. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation. En *Proc. of the 10th Machine Translation Summit (MT Summit X)*, págs. 79–86. Corpus disponible en `http://www.statmt.org/europarl/`.

Liu, F., F. Weng, y X. Jiang. 2012. A broad-coverage normalization system for social media language. En *Proc. of the 50th Annual Meeting of the Association for Computational Linguistics (ACL'12): Long Papers - Vol. 1*, págs. 1035–1044. ACL.

López Rúa, P. 2007. Teaching L2 vocabulary through SMS language: Some didactic guidelines. *Estudios de lingüística inglesa aplicada*, 7:165–188.

Mosquera, A. 2011. The Spanish metaphone algorithm (Algoritmo del metáfono para el español). Código disponible en `https://github.com/amsqr/Spanish-Metaphone`.

Oliva, J., J. I. Serrano, M. D. del Castillo, y A. Iglesias. 2013. A SMS normalization system integrating multiple grammatical resources. *Natural Language Engineering*, 19(1):121–141.

Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, y N. A. Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. En *Proc. of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2013)*, págs. 380–390. ACL. *Toolkit* disponible en `http://www.ark.cs.cmu.edu/TweetNLP/`.

Padró, L. y E. Stanilovsky. 2012. Freeling 3.0: Towards wider multilinguality. En *Proc. of the 8th Int. Conference on Language Resources and Evaluation (LREC'12)*. ELRA. *Toolkit* disponible en `http://nlp.lsi.upc.edu/freeling/`.

Philips, L. 1990. Hanging on the metaphone. *Computer Language*, 7(12):39–43.

Taulé, M., M. A. Martí, y M. Recasens. 2008. AnCora: Multilevel annotated corpora for Catalan and Spanish. En *Proc. of the 6th Int. Conference on Language Resources and Evaluation (LREC'08)*. ELRA.

Thurlow, C. 2003. Generation Txt? the sociolinguistics of young people's text-messaging. *Discourse Analysis Online*, 1(1).

Véronis, J. 1999. MULTEXT-corpora. An annotated corpus for five European languages. CD-ROM. Distribido por ELRA/ELDA.

Xue, Z., D. Yin, y B. D. Davison. 2011. Normalizing microtext. En *Analyzing Microtext, Papers from the 2011 AAAI Workshop*, vol. WS-11-05 de *AAAI Workshops*. AAAI.