PROCESAMIENTO DE VARIANTES MORFOLÓGICAS EN BÚSQUEDAS DE TEXTOS EN CASTELLANO*

Fernando Raúl Alfredo Bordignon
Walter Panessi **

RESUMEN

El tratamiento de los morfemas dependientes es una de las estrategias utilizadas con la finalidad de incrementar la proporción de documentos relevantes recuperados en los sistemas de búsqueda de información. La técnica de stemming permite extraer sufijos y prefijos comunes, de tal forma que palabras que literalmente son diferentes, pero tienen una raíz común, pueden ser consideradas como un sólo término con base en su raíz.
El presente trabajo consiste en una adaptación del algoritmo de Porter, para procesar corpus de documentos en castellano.

PALABRAS CLAVE: Stemming, Procesamiento de sufijos, Incremento de relevancia.

BORDIGNON, Fernando Raúl Alfredo y PANESSI, Walter. Procesamiento de variantes morfológicas en búsquedas de textos en castellanos. En: Revista Interamericana de Bibliotecología. Vol. 24, No. 1 (ene.-jun., 2001); p. 69-88.

ABSTRACT

Processing dependent morphemes is one of the strategies used to increase the proportion of relevant retrieved documents in information retrieval systems. The stemming technique allows the extraction of common suffixes, in such a way that literally different words with a common root can be considered as a single term, as referred to their root. The present work is an adaptation of Porter's algorithm to process Spanish corpora.

KEY WORDS: Stemming, Suffixes processing, Relevance increase.

BORDIGNON, Fernando Raúl Alfredo *and* PANESSI, Walter. *Processing Morphological in Searches of Spanish Texts. In: Revista Interamericana de Bibliotecología. Vol. 24, No. 1. (jan.-jun., 2001); p. 69-88.*

* Artículo de investigación que versa sobre un tema derivado de las tecnologías de recuperación de la información, recibido en el mes de abril de 2000 y aceptado en marzo de 2001.

** Licenciados que laboran en la División Estadística y Sistemas del Departamento de Ciencias Básicas de la Universidad Nacional de Luján, Argentina. E-mails: wpanessi@unlu.edu.ar y bordi@unlu.edu.ar, respectivamente.

INTRODUCCIÓN

En un documento, una palabra dada puede ser encontrada de muchas formas diferentes. Por ejemplo las palabras, "computadora", "computador", "computacional", "computando", "computadoras", "computadores", junto con algunas otras palabras, tienen diferente forma pero todas hacen referencia cercana a un mismo concepto. Si algún usuario requiriera información sobre *computadoras,* podría perderse de encontrar documentos relevantes, sólo porque estas palabras están escritas de otra forma y tienen poca frecuencia en el documento, por ende no se tomaron en cuenta como palabras índices. Esta situación es claramente indeseable.

Una manera de solucionar este problema es introducir algoritmos de stemming, los cuales eliminan las terminaciones de las palabras, reduciéndolas a un término común o raíz (stem). En el ejemplo presentado, la raíz sería "comput". Para un documento dado, se podrían unir varias palabras con la misma raíz y, de esta manera, elevar su frecuencia en el texto, haciéndolas candidatas a términos índices. En una consulta, la aplicación de stemming asegura que el usuario no sea penalizado por el uso de una palabra con una forma específica que no ha tenido una frecuencia elevada en el documento.

Numerosos algoritmos de stemming se vienen desarrollando desde hace años. Tres de los más conocidos son los construidos por Lovins en 1968 [Lov68], Porter en 1980 [Por80] y Paice en 1990 [Pai90]. Todos estos algoritmos van eliminando consecutivamente los finales de las palabras para arribar a su raíz. Por ejemplo, "computationally" podría ser procesado de la siguiente manera, (el ejemplo esta dado por un término en inglés, dado que los tres algoritmos funcionan sólo con este idioma), primero se le quitaría "ly" quedando "computational", luego "al", quedando "computation", después "tion" resultando "computa" y finalmente la "a" quedando "comput", que sería la raíz de este término en inglés, que casualmente coincide con la raíz en castellano. Estos algoritmos interactivos, requieren un conjunto de múltiples pasos para llevar una palabra dada hasta su raíz, pero no requieren conocimiento *a priori* de todas las posibles combinaciones de terminaciones de las palabras. La diferencia principal entre estos tres algoritmos, está dada por la eficiencia de su código y en la elección de los sufijos que identifican y eliminan. Un estudio realizado indica que la performance del método de stemmer

de Lovins, puede no concordar con los otros dos pero sus resultados no fueron completamente decisivos [Pai96].

No se debe confundir el significado de una raíz para el método de stemmer con lo que sería la raíz de una palabra, (el origen, lingüísticamente hablando). Se podría esperar equivocadamente que, un método tal llevara la palabra a su forma general e impersonal (masculino singular o verbos en infinitivo) , pero un trabajo semejante daría como resultado tener que emplear una base de datos para contrastar un término. El realizar stemming utilizando una gran base de datos de textos para contrastar, consume demasiado tiempo de procesamiento. Estudios realizados indican que el esfuerzo realmente no vale la pena [Har90]. Por consiguiente, los métodos de stemmer, quitan los sufijos sin importar si la raíz resultante queda o no con significado idiomático.

No hay ninguna razón teórica por la cual los métodos de stemmer no deban también quitar los prefijos. Prefijos tales como "in" (inestable) , "ante" (antesala), "anti" (antiadherente), son candidatos a ser retirados, pero de hecho, la mayoría de los métodos de stemmer utilizados hoy en día quitan sólo los sufijos. Una razón probable sería, la dificultad que conlleva la decisión de cuándo una secuencia de letras es un prefijo, por ejemplo, mientras que "in" es un prefijo negativo en "indeseable", no lo es en "introducción". Otro motivo por el que quizás no se quiten los prefijos, es que al quitarlos podría cambiar sustancialmente el significado de la palabra buscada.

Hay dos problemas que el algoritmo de stemming debe resolver. Uno, es el quitar terminaciones de palabras que falsamente parecen ser sufijos. Por ejemplo, mientras que la terminación "ed" sería retirada normalmete de cualquier palabra, en otras como "bed" no debería quitarse. Este problema puede ser resuelto fijando un mínimo de letras aceptables para una raíz y con una pequeña lista de palabras que deben ser exentas de la aplicabilidad de esta regla. Si se fijara un mínimo de tres letras para una raíz, la palabra "bed" no sería tocada, pero entonces qué pasaría con "breed" que se esta manera, podría convertirse en candidata para ser excluida de la aplicación de esta regla. Alternativamente se podrían aplicar algunas reglas que determinaran cuándo una raíz debe ser quitada y cuando no. Por ejemplo, incluir una regla que diga que cuándo la palabra termina con "eed", el sufijo "ed" no se quite.

El otro problema que el algoritmo de stemmer debe resolver, es el cambio de la raíz en determinadas formas de una palabra. Para clarificar este problema, observe qué pasa con el plural de algunas palabras como "Knife", que cambiaría en plural por "Knives". Lo mismo ocurre en el castellano por ejemplo, con la palabra "repite" y "repetido". Estos cambios morfológicos son excepciones a las reglas, y en inglés hay relativamente pocos casos donde se presenta este problema que en castellano, es mayor.

MODELO PROPUESTO

Las variaciones morfológicas de las palabras, pueden traer problemas para la recuperación de documentos relevantes debido a que, palabras que se escriben diferente (con muy poca variación) pero que se refieren a un mismo concepto, pueden tener poca frecuencia de ocurrencia separadamente, pero si se las unifica, su frecuencia puede aumentar significativamente. También se ha expresado que la forma de resolver este tipo de inconvenientes es aplicando algún método de stemming.

Los métodos de stemming son enteramente dependientes del idioma, es decir que cualquier stemmer desarrollado sirve únicamente para el idioma para el que fue creado. Esta última observación se deduce fácilmente si se piensa que los stemmer reconocen terminaciones o sufijos de las palabras, y cada idioma tiene sus propios sufijos.

Adicionalmente este método trae aparejada otra ventaja, la reducción del tamaño de los índices y el consiguiente aumento de velocidad del procesamiento.

Los métodos de stemming han sido ampliamente explorados, de tal manera que se han desarrollado stemmer para el inglés, para el latín [Sch96], para el portugués y para el turco [Ekm95], pero no se ha encontrado ninguno para el castellano. Este trabajo tiende a cubrir ese espacio, presentando un método de stemmer en castellano mediante una adaptación del algoritmo de Porter.

STEMMER EN CASTELLANO

La elección del algoritmo de Porter está fundamentada en que, a diferencia de otros algoritmos, Porter va quitando los sufijos por etapas, en cambio el de Lovins, por ejemplo, requiere de la definición de todas las posibles combinaciones de sufijos.

En el caso de idiomas tan ricos como el castellano una descripción tal, podría ser, no sólo ardua sino también difícil de realizar. Considéresen, por ejemplo, las diferentes combinaciones de sufijos que podrían aplicarse a la palabra "presentar". En la tabla 2 se expresan 25 formas morfológicas diferentes.

Tabla 2:
Algunas de las variaciones morfológicas posibles para el término "Presentar"

Palabra	Combinaciones de Sufijos
Presentable	able
Presentables	ables
Presentación	ación
Presentaciones	aciones
Presentado	ado
Presentador	ador
Presentadores	adores
Presentándonos	ándonos
Presentar	ar
Presentáramos	áramos
Presentaríamos	aríamos
Presentarla	arla
Presentarlas	arlas
Presentarle	arle
Presentarles	arles
Presentarlo	arlo
Presentarlos	arlos
Presentarse	arse
Presentase	ase
Presentásemos	ásemos
Presente	e
Presentémonos	émonos
Presentismo	ismo
Presento	o
Presentó	ó

Como se ve, las variaciones morfológicas en el castellano son muchas, y la especificación de cada una de las posibles combinaciones de sufijos, no es una tarea trivial. Por consiguiente, se ha optado por el algoritmo de Porter.

El Algoritmo de Porter

Porter publicó en 1980 un algoritmo para el método de stemming que fue tomado como base por muchos investigadores. El algoritmo lee un archivo, toma una serie de caracteres y de esa serie, una palabra; luego la valida verificando que todos los caracteres involucrados sean letras, de ser así, aplica stemming sobre ella.

La aplicación de stemmer consiste en hacer pasar esta palabra a través de varios conjuntos de reglas, cada conjunto de reglas está formada por n reglas, y cada regla por:

1. Un identificador de regla
2. El sufijo a identificar
3. El texto por el cual debe ser reemplazado al encontrar el sufijo
4. El tamaño del sufijo
5. El tamaño del texto de reemplazo
6. El tamaño mínimo que debe tener la raíz resultante luego de aplicar la regla (ésto es a los efectos de no procesar palabras demasiado pequeñas)
7. Una función de validación (una función que verifica si se debe aplicar la regla una vez encontrado el sufijo)

Considere, a modo de ejemplo, la siguiente regla perteneciente al algoritmo original de Porter :

106, "ed", LAMBDA, 1, -1, -1, ContainsVowel

Analizándola elemento a elemento se tiene que :

1. *106* es el identificador de la regla
2. *"ed"* es el sufijo que debe localizar al final de la palabra
3. *"LAMBDA"* es el texto por el cual se debe reemplazar el sufijo una vez encontrado (en este caso LAMBDA es una constante definida como cadena vacía)
4. *"1"* es el tamaño del sufijo (Tamaño de "ed" - 1)

5. *"-1"* es el tamaño del texto que se debe reemplazar por el sufijo. Como se trata en este caso de una cadena vacía, el tamaño sería 0, que restándole 1, sería -1

6. El siguiente *"-1"* es el tamaño mínimo que debe tener la raíz una vez que le quitemos "ed" (-1 significa que lo quite cuando al menos queden tres caracteres)

7. *"ContainsVowels"* es la función de validación. En el caso particular de ContainsVowels, verifica que la palabra sin "ed" contenga vocales

Así, esta regla se aplicaría a las palabras lowed quedando low, shared quedando shar, pero no se aplicaría a la palabra shed, ya que si le sacáramos "ed" no quedarían vocales y además, la raíz quedaría con dos letras y se pide que tenga al menos tres.

Un conjunto de reglas está formado por un grupo de estas reglas y siempre termina con una con identificador **"000".**

Cuando alguna regla del conjunto puede ser aplicada, no se revisa ninguna regla más de ese conjunto, se sustituye el sufijo por el texto de reemplazo y se devuelve el número de identificador de la regla aplicada. Esto último es usado para aplicar conjuntos que son condicionales a la aplicación de una regla en particular, por ejemplo, *si se aplicó la regla 106 revise el conjunto de reglas RuleList2b, de lo contrario continúe con RuleList3*. Obviamente si no se pudo aplicar ninguna regla del conjunto, devuelve 000.

Cuando ya no queden más conjuntos de reglas para aplicar, se retorna la palabra resultante y se imprime, tomando otra palabra y comenzando el ciclo nuevamente hasta que ya no queden más palabras por procesar en el archivo de texto.

Estudio Preliminar

Se ha visto que en el algoritmo de Porter, cuando una regla es aplicada dentro de un conjunto, termina la secuencia de búsqueda dentro de él. En otras palabras, la aplicación de una regla, inhibe la aplicación de cualquier otra regla posterior dentro del mismo conjunto. Ésto trae aparejado un problema que debe ser resuelto.

Considere el conjunto de reglas siguiente :

```
static RuleList step00_rules[] =
  {
003, "mente", LAMBDA, 4, -1, 0, NULL,
  004, "al", "a", 1, 0, 0, NULL,
  000, NULL, NULL, 0, 0, 0, NULL,
  };
```

Considere también el procesamiento de las palabras "Personalmente" y "Personal".

Cuando se procese el conjunto de reglas *step00_rules* con la primera palabra, la regla 003 se aplicará, dejando como resultado "personal", pero la regla 004, no lo hará debido a que la 003 se aplicó con anterioridad. En el caso de la segunda palabra, la regla que se aplicará será la 004 dejando como resultado "persona". El sufijo "al", en la primer palabra, queda "atrapado" por la aplicación de una regla anterior del mismo conjunto. De esta manera, las raíces obtenidas serán diferentes ("personal" versus "persona"), y ésto es indeseable para el método de stemming.

Dada esta característica del algoritmo de Porter, debió realizarse un estudio preliminar de los sufijos en castellano, para lograr una primera aproximación a las formas morfológicas de los términos en este idioma. Los objetivos son :

- Ubicar los sufijos que ocurren frecuentemente en castellano
- Identificar los sufijos que ocurren juntos
- Establecer el orden en el que ocurren

Para la realización de este estudio, se desarrolló un programa sobre el que se procesaron 52 documentos (1.623 Kb), y se obtuvieron 17.278 palabras diferentes (para más detalles, ver "Resultados de la Experiencia").

La lista de palabras obtenidas, se contrasta contra una lista de sufijos, creando una tabla relación "sufijo/palabra", que contiene el identificador de cada elemento (sufijo y palabra) junto con un número que representa el peso de la posición que ocupa el sufijo, en razón del tamaño de la palabra. Por ejemplo, en el caso de la palabra "personalmente", el sufijo "al" tendrá el valor 0.5385 dado que la palabra

cuenta con 13 letras y el sufijo se encuentra en la séptima posición, por lo tanto 7 / 13 = 0.5385, para el sufijo "mente", el valor de esta relación es 0.6923.

Ordenando el resultado, se obtienen dos informes. Uno, de palabras por sufijos, que contiene un sufijo y las palabras que lo contienen, y otro, de sufijos por palabras, una palabra con los sufijos que contiene.

Un análisis posterior de estos informes ayudó al establecimiento del orden en que los sufijos aparecen en las palabras, y, de esta manera, se pudo comenzar a armar los grupos de reglas.

Dada la situación expresada al comienzo, las reglas que conforman los grupos debieron ser cuidadosamente seleccionadas de la misma manera que el orden de procesamiento de cada conjunto. Para la selección de los grupos y el orden de procesamiento, se debe tener en cuenta que :

- Dos sufijos que ocurren juntos no pueden pertenecer a un mismo conjunto.
- Las reglas que quitan sufijos que ocurren más al final de cada palabra deben ser procesadas en un paso anterior a las que quitan otros sufijos.
- Si un sufijo aparece siempre que ocurra otro, este sufijo es condicional a la aparición del anterior.

Un análisis adicional de estos informes revela que, algunas terminaciones de palabras que parecen ser sufijos, no lo son. Tal es el caso del sufijo "nos" (a nosotros) para las palabras "campesinos", "casinos", "caninos", etc; en contrapartida con palabras tales como "hacernos", "ponernos", "presentarnos".

Este último caso lleva a la construcción de reglas de validación propias para el castellano.

Finalmente, la adaptación del algoritmo de Porter es realizada paso a paso, en forma interactiva, colocando una regla a la vez y evaluando los resultados obtenidos en cada caso.

Tres pasos se tienen en cuenta para la depuración final del algoritmo. En primer lugar, las palabras que terminan con "r" suelen dar como resultado que algunas palabras conceptualmente similares, queden con diferente raíz. Tal es el caso de los verbos. Cuando un verbo está en infinitivo, generalmente termina con "r", pero si por el contrario, el verbo está conjugado, la aplicación de stemming dejará las

raíces diferentes. Considerese, por ejemplo, el verbo "caminar" y su conjugación "caminando". Cuando se le quita el sufijo "ndo" al verbo conjugado, la raíz queda "camina". Por tal motivo, las "r" finales son quitadas en uno de los últimos pasos.

En segundo lugar, un caso similar se presenta con las palabras que terminan con vocales. Por ejemplo, a la palabra "terminación", cuando se le quite el sufijo "ción", quedará como "termina", lo cual sería aceptable si no existiese la palabra "terminal", que al quitarle el sufijo "al" queda como "termin", u otras palabras como "terminó" que quedarían sin variación por no tener sufijo alguno. Por ende, también se quitan las vocales terminales a las palabras en uno de los últimos pasos.

En último término, se aplica una tercera regla que elimina todas las tildes que están en la raíz resultante, a efectos de evitar casos como, por ejemplo, "diálogo" y "dialogó".

Documentos Utilizados

Para la realización del estudio preliminar, se consideró la construcción de un documento base en forma manual. Esta tarea podría traer ciertos inconvenientes. En primer lugar, la arbitrariedad de la elección de los términos estaría afectada, y en segundo lugar, la imaginación se vería agotada rápidamente. Por estos motivos se recurrió a algoritmos que recorren documentos, extraen las palabras, verifican que éstas sean válidas (que estén formadas en su totalidad por letras), y chequean que no estén ya incluidas en el documento base.

Para lograr un mayor número de palabras diferentes, se trata de que estos textos se refieran a temas totalmente diferentes. De esta forma, se utilizaron textos que versaban sobre una variada temática.

La lista de sufijos propios del castellano se obtiene desde una página de lingüística en Internet. Esta lista se actualiza con algunos otros sufijos no existentes en ella que se presentan frecuentemente en el idioma.

Resultados de la Experiencia

Resultados de la construcción de la Base de Datos de Documentos

En la etapa previa a la adaptación del algoritmo, se recolectaron 52 documentos. La tabla 3 muestra los datos obtenidos de esa recuperación.

Uno de los datos más interesantes de esta tabla es la relación que existe entre las palabras agregadas y las palabras útiles del documento. Las palabras útiles son aquellas que quedaron luego de la validación que realiza el algoritmo (que las palabras contengan sólo letras). Las palabras agregadas son, de las útiles, sólo aquellas que todavía no están en la base de datos.

La rápida caída que sufre esta relación, es parecida a la anunciada por la ley de Zipf [Zip49]. En las figuras 4 y 5 se grafica la relación Agregadas/Útiles (triángulo), comparada con la curva que daría la aplicación de la ley de Zipf (cuadrado) en escalas lineal y logarítmica respectivamente. El diseño de un experimento con distribución F de Snedecor, con una H0 = "Ambas curvas son iguales", con un alfa de 0.05, demuestra que las dos curvas son iguales con una probabilidad de error de 0,25. Por lo cual se concluye que la obtención de palabras sin repetición desde documentos nuevos, tiene una distribución como la anunciada por Zipf.

Tabla 3 :

Resultado obtenido sobre el procesamiento de documentos durante el estudio preliminar

Número de Orden	Tamaño en Bytes	Cantidad de Palabras	Palabras Útiles	Palabras Agregadas	Relación Agregadas/ Útiles
1	105887	20982	15913	3269	0.20542952
2	57166	11063	8529	1194	0.13999297
3	54415	9802	8007	881	0.11002872
4	39424	7451	6084	649	0.10667324
5	39701	6632	5483	533	0.09720956
6	29749	5223	4474	425	0.09499329
7	28584	5283	4387	407	0.09277411
8	51001	10362	6740	606	0.08991098
9	43332	7992	6882	604	0.08776518
10	51099	9731	7801	670	0.08588642
11	24987	4545	3903	334	0.0855752
12	29247	5300	4423	350	0.07913181
13	28273	5658	4100	302	0.07365854
14	39014	7306	5902	401	0.06794307
15	53038	10233	8575	571	0.06658892
16	40351	7096	5903	393	0.06657632
17	38702	7077	5754	377	0.06551964
18	24802	4470	3809	235	0.06169598

Número de Orden	Tamaño en Bytes	Cantidad de Palabras	Palabras Útiles	Palabras Agregadas	Relación Agregadas/ Útiles
19	20492	5232	2787	171	0.0613563
20	22198	3975	3345	191	0.05710015
21	24718	4530	3736	213	0.05701285
22	25406	4793	3912	217	0.05547035
23	24881	4441	3781	197	0.05210262
24	26191	4960	4103	208	0.05069461
25	28771	5021	4201	212	0.05046418
26	20549	3895	3198	158	0.04940588
27	22408	4170	3478	165	0.04744106
28	26691	4809	4011	190	0.04736973
29	31455	5497	4652	208	0.04471195
30	30121	5452	4700	207	0.04404255
31	22813	4300	3443	151	0.0438571
32	22301	4062	3162	137	0.04332701
33	21704	3789	3160	136	0.04303797
34	38178	6685	5770	239	0.04142114
35	34278	6160	5177	186	0.03592814
36	20759	3587	3032	108	0.03562005
37	25822	5183	4005	138	0.03445693
38	50705	10468	6614	227	0.03432114
39	31451	5692	4519	152	0.03363576
40	27173	4912	4147	138	0.03327707
41	26403	4974	3933	127	0.03229087
42	25119	4510	3995	125	0.03128911
43	25525	4655	3893	120	0.03082456
44	29717	5395	4540	129	0.0284141
45	24698	4672	3787	107	0.02825456
46	20765	3559	2969	81	0.02728191
47	21833	4819	2429	66	0.02717168
48	20080	3920	3067	82	0.02673622
49	19918	3691	3134	80	0.02552648
50	22213	4294	3464	78	0.02251732
51	27928	4900	4141	86	0.02076793
52	20070	3430	2818	47	0.0166785
Totales					

Figura 4 : Relación Agregadas/Útiles
Vs Ley de Zip (Escala Lineal)

Figura 5 : Relación Agregadas/Útiles
Vs Ley de Zip Escala Logarítmica

Resultado del Análisis de Palabras Comunes

Para comprobar si la repetición de palabras en castellano tienen una distribución como la anunciada por la ley de Zipf, se hizo otro análisis.

Lo que se desea averiguar es, básicamente, si las palabras comunes del castellano, al igual que en el inglés, disminuyen tan rápidamente que permitan que se cumpla:

f * r = C (La frecuencia por el rango es igual a una constante)

En la tabla 4 se muestran las primeras 30 palabras más frecuentes del castellano con base al corpus de prueba. Según el estudio realizado, la correspondencia de esta distribución, coincide con la expuesta por la ley de Zipf, mucho más que la anterior. Según los datos obtenidos, usando el mismo método con los mismos parámetros, la H0 es verdadera con una probabilidad de error de 0.000000069.

En la figura 6 y 7, se grafican las frecuencias de las primeras 300 palabras más comunes (triángulo) y la curva de la ley de Zipf ideal (cuadrado), en escala lineal y logarítmica respectivamente. En estas gráficas, se puede observar que las curvas difieren en muy poco.

Tabla 4 :
Primeras 30 palabras más frecuentes del castellano.

Palabra	Frecuencia	Palabra	Frecuencia	Palabra	Frecuencia
de	20383	las	3089	más	1418
la	9141	una	3045	al	1271
en	7187	por	2637	su	1124
que	6547	con	2531	lo	864
el	6449	para	2478	puede	738
y	6088	es	2305	este	690
los	4634	del	2226	esta	664
a	4625	no	1575	entre	659
se	3441	como	1515	si	658
un	3379	o	1480	ser	605

Figura 6 : Frecuencia de palabras comunes
Vs Ley de Zip (Escala Lineal)

Figura 7 : Frecuencia de palabras comunes
Vs Ley de Zip Escala Logarítmica

En relación a las stopwords ó palabras vacías, dichos estudios han mostrado que de las 250 a 300 palabras más comunes en el inglés, pueden representar el 50 % o más de un documento dado [Wat92]. En un estudio realizado en 1967 [Kus67] sobre un millón de palabras obtenidas de textos generales en inglés, se encontró que dos de las palabras más comunes, "the" y "of", representaban el 10 % de las palabras contenidas; las cuatro siguientes (en orden de frecuencias), "and", "to", "a" e "in", representaban otros 10 %. Finalmente, el 30% del texto estaba compuesto por sólo 80 palabras. ¿Qué ocurre entonces con el castellano? El presente análisis sobre la proporción que ocupan las palabras más comunes en castellano muestra que las dos primeras palabras forman el 12% del corpus; las cuatro siguientes, en orden de frecuencia de ocurrencias, un 10%; las 8 próximas un 11%, las 16 que continúan un 8%; y finalmente, se necesitaron ochenta palabras más para formar un 11%. El resto del corpus forman un 48%, mientras que sólo 100 palabras representan el 52% de éste. La figura 8, muestra estas proporciones gráficamente.

Figura 8 :
Proporción de las palabras más frecuentes

Por otra parte, 7,996 palabras de las 17,763, ocurrieron solo una vez, 2,808 dos veces, 1,495, tres veces, 853 cuatro veces y 628 cinco veces. Esta distribución también sigue un patrón similar al enunciado por ley de Zipf.

Resultado del Algoritmo de Stemming

Estudios realizados [Ekm95], muestran que los algoritmos de stemming, funcionan bien para idiomas tales como el inglés, y en general, para idiomas no muy ricos en cuanto a variaciones morfológicas. Este argumento está basado en una medida de compresión calculada a partir de la cantidad de términos diferentes obtenidos y las distintas raíces que quedan luego de la aplicación del algoritmo de stemming.

La tabla 5 [Ekm95], muestra un extracto de los resultados obtenidos por este estudio, junto con el resultado del análisis de los datos que se obtuvieron aplicando la adaptación del algoritmo de Porter al castellano.

Tabla 5 :
Resultado comparativo de los algoritmos de Porter para inglés y castellano

Algoritmo	Cantidad de términos distintos	Cantidad de raíces distintas	Porcentaje de compresión
Porter en Inglés	18.348	11.671	36,39 %
Porter en Castellano	17.763	4.284	75,88 %

Esta prueba de compresión se ha realizado sobre diferentes bases de datos, con diferentes palabras. En todas las pruebas realizadas se han obtenido aproximadamente la misma cantidad de raíces. Este último dato revela la consistencia del modelo para el castellano.

En el anexo I se presenta la porción de código del programa de Porter que contiene las reglas adaptadas al lenguaje castellano y que han sido utilizadas en la experiencia.

Si se considera que realizar stemming tiene dos propósitos :

1. Aumentar la frecuencia de las palabras
2. Disminuir el tamaño de los índices

El factor de compresión sólo atiende el segundo objetivo.

Otra experiencia pudo ser realizada para probar si el primer objetivo también se cumple. Esta experiencia involucra a aquellas palabras con baja frecuencia en el corpus. La hipótesis dice que, si se aplica stemming a un corpus, es probable que las raíces de aquellas palabras con baja frecuencia de ocurrencia, aumenten

considerablemente su presencia, de tal manera que su peso en el corpus pueda llevarlas a ser consideradas como términos índices.

En el corpus se encontraron 13,780 palabras con frecuencias entre 1 y 5. Se extrajeron estas palabras del corpus y se le aplicó stemming. Se contrastaron las raíces obtenidas con el corpus y se calculó la frecuencia. La tabla 6 muestra el cambio en la frecuencia de estas palabras.

Tabla 6 :
Cambio de frecuencias de ocurrencias luego de stemming

Frecuencia	Sin Stemming	Con Stemming
1	7,996	4,215
2	2,808	1,209
3	1,495	486
4	853	353
5	628	206
6	0	140
7	0	91
8	0	81
9	0	70
10	0	60
11	0	43
12	0	41
13	0	29
14	0	20
15	0	17
16	0	11
17	0	13
18	0	12
19	0	6
20	0	8
21	0	3
22	0	1

Como se puede apreciar, el resultado demuestra un aumento considerable en la frecuencia de las palabras. De esta manera, se comprueba que el primer objetivo también se cumple.

CONCLUSIONES

La adaptación del algoritmo de Porter para el castellano, ha mostrado, mediante los resultados obtenidos, el cumplimiento de los objetivos propuestos para un modelo de stemmer. Por un lado un factor de compresión del 76% aproximadamente, y por el otro, el aumento de la frecuencia de ocurrencia de las palabras con poca presencia en el corpus.

Se ha mostrado también que el castellano, al igual que otros idiomas, cumple con la ley de Zipf en la distribución de las palabras más comunes (conectores léxicos) en un documento, ésto es útil para la aplicación de umbrales que determinen cuando una palabra puede ser tomada como término índice y cuando no, o, por otra parte, para la creación de la lista de stopwords o palabras vacías, lista que, puede ser empleada como complemento del algoritmo de stemmer, quitando aquellas palabras con mayor frecuencia en un documento o colección de documentos antes de la aplicación de stemming.

ANEXO:

Adaptación del algoritmo de Porter al castellano.
Juego de reglas utilizadas

```
/**********************************stem.c*********************************
Propósito : Implementación del algoritmo de Stemming de Porter documentado
en: Porter, M.F., "An Algorithm For Suffix Stripping,"
Program 14 (3), July 1980, pp. 130-137.
Proveniencia : Escrito por B. Frakes and C. Cox, 1986.
Adaptado por Walter F. Panessi , 1999.
- Adaptación al lenguaje castellano **/
/**************************Juego de reglas*****************************/
static RuleList step00_rules[] =
        {       001, "es", LAMBDA, 1, -1, 0, EndsWithDLR,
                002, "mente", LAMBDA, 4, -1, 0, NULL,
                000, NULL, NULL, 0, 0, 0, NULL, };
static RuleList step01_rules[] =
        {       004, "al", "a", 1, 0, 0, EndsWithVowels,
                005, "al", LAMBDA, 1, -1, 0, NULL,
                000, NULL, NULL, 0, 0, 0, NULL, };
```

```
static RuleList step1a_rules[] =
        {       101, "ces", "z", 2, -1, -1, NULL,
                102, "r", LAMBDA, 0, -1, 0, NULL,
                103, "nos", LAMBDA, 2, -1, 1, EndsWithOR,
                104, "s", LAMBDA, 0, -1, 0, NULL,
                105, "cion", "r", 3, 0, 0, NULL,
                106, "ción", "r", 3, 0, 0, NULL,
                107, "n", LAMBDA, 0, -1, 0, NULL,
                000, NULL, NULL, 0, 0, 0, NULL, };
static RuleList step1b_rules[] =
        {       110, "ente", LAMBDA, 3, -1, 0, NULL,
                111, "encia", LAMBDA, 4, -1, 0, NULL,
                112, "dr¡a", LAMBDA, 3, -1, 0, NULL,
                113, "mo", LAMBDA, 1, -1, 0, NULL,
                114, "ro", LAMBDA, 1, -1, 0, NULL,
                115, "se", LAMBDA, 1, -1, 0, NULL,
                116, "la", LAMBDA, 1, -1, 0, EndsWithOR,
                117, "le", LAMBDA, 1, -1, 0, EndsWithOR,
                118, "lo", LAMBDA, 1, -1, 0, EndsWithOR,
                119, "me", LAMBDA, 1, -1, 0, EndsWithOR,
                120, "do", LAMBDA, 1, -1, -1, NULL,
                121, "da", LAMBDA, 1, -1, -1, EndsWithVowels,
                000, NULL, NULL, 0, 0, 0, NULL, };
                static RuleList step1b1_rules[] =
        {       125, "n", LAMBDA, 0, -1, 0, NULL,
                000, NULL, NULL, 0, 0, 0, NULL,};
static RuleList step1b2_rules[] =
        {       130, "re", "r", 1, 0, 0, EndsWithVowels,
                000, NULL, NULL, 0, 0, 0, NULL, };
static RuleList step2_rules[] =
        {       220, " ", LAMBDA, 0, -1, 0, NULL,
                221, "iza", LAMBDA, 2, -1, 0, NULL,
                222, "iz¢", LAMBDA, 2, -1, 0, NULL,
                223, "izar", LAMBDA, 3, -1, 0, NULL,
                224, "¡a", LAMBDA, 1, -1, 0, NULL,
                225, "imiento", "e", 6, 0, 0, NULL,
                226, "miento", LAMBDA, 5, -1, 0, NULL,
```

```
227, "tivo", LAMBDA, 3, -1, 0, NULL,
228, "tiva", LAMBDA, 3, -1, 0, NULL,
228, "tiv", LAMBDA, 2, -1, 0, NULL,
229, "ble", LAMBDA, 2, -1, 0, NULL,
230, "bl", LAMBDA, 1, -1, 0, NULL,
231, "dad", LAMBDA, 2, -1, 0, EndsWithVowels,
232, "ize", LAMBDA, 2, -1, 0, NULL,
233, "iz", LAMBDA, 1, -1, 0, NULL,
000, NULL, NULL, 0, 0, 0, NULL, };
static RuleList step8_rules[] =
    {   800, "r", LAMBDA, 0, -1, 1, NULL,
        000, NULL, NULL, 0, 0, 0, NULL,};
static RuleList step9_rules[] =
    {   990, "a", LAMBDA, 0, -1, -1, NULL,
        991, "á", LAMBDA, 0, -1, -1, NULL,
        992, "e", LAMBDA, 0, -1, -1, NULL,
        993, "é", LAMBDA, 0, -1, -1, NULL,
        994, "i", LAMBDA, 0, -1, -1, NULL,
        995, "í", LAMBDA, 0, -1, -1, NULL,
        996, "o", LAMBDA, 0, -1, -1, NULL,
        997, "ó", LAMBDA, 0, -1, -1, NULL,
        998, "u", LAMBDA, 0, -1, -1, NULL,
        999, "ú", LAMBDA, 0, -1, -1, NULL,
        000, NULL, NULL, 0, 0, 0, NULL, };
static RuleList step99_rules[] =
    {   990, "a", LAMBDA, 0, -1, -1, NULL,
        991, "á", LAMBDA, 0, -1, -1, NULL,
        992, "e", LAMBDA, 0, -1, -1, NULL,
        993, "é", LAMBDA, 0, -1, -1, NULL,
        994, "i", LAMBDA, 0, -1, -1, NULL,
        995, "í", LAMBDA, 0, -1, -1, NULL,
        996, "o", LAMBDA, 0, -1, -1, NULL,
        997, "ó", LAMBDA, 0, -1, -1, NULL,
        998, "u", LAMBDA, 0, -1, -1, NULL,
        999, "ú", LAMBDA, 0, -1, -1, NULL,
        000, NULL, NULL, 0, 0, 0, NULL, };
/*****************************************************************/
```

BIBLIOGRAFÍA

EKMEKCIOGLU, F. C.; LYNCH, M. F. y WILLETT, P. Development and evaluation of conflation techniques for the implementation of a document retrieval system for Turkish text databases. En: The New Review of Document and Text Management. Vol. 1 (1995) ; p. 131-146.

HARMAN, Donna K. y CANDELA, G. Retrieving records from a gigabyte of text on a minicomputer using statical ranking. En: ASIS. Vol. 41, No 8 (1990); p. 581-589.

KUCERA, H. y FRANCIS, W. N. Computational analysis of present-day american english. Rhode Island : Brown University Press, 1967

LOVINS, Julie Beth. Development of a stemming algorithm Mechanical. En: Translation and Computational Linguistics. Vol. 11, No. 1-22 (1968); p. 22-31.

PAICE, Chris. Another stemmer. En: SIGIR Forum. Vol. 24, No. 3; p. 56-61.

PAICE, Chris. Method for evaluation of stemming algorithms based on error counting. En: JASIS. Vol. 47, No. 8 (1996); p. 632-649.

PORTER, M. F. An algorithm for suffix stripping. En: Program. Vol. 14 (1980); p. 130-137.

SCHINKE, R. *et al.* A stemming algorithm for Latin text databases. En: Journal of Documentation. Vol. 52 (1996); p.172-187.

WATTERS, Carolyn. Dictionary of information science and technology. San Diego, California : Academic Press, 1992.

ZIPF, George Kinglsey. Human behavior and the principle of least effort. Cambridge, Massachusetts: Addison-Wesley , 1949.