ALGORITMOS

DE STEMMING

Procesamiento avanzado del Lenguaje Natural

Doctorado en Computación

Curso 2007/2008

Susana Ladra González

Índice

Introducción

En los últimos años ha aumentado de forma considerable la cantidad de información disponible digitalmente. Principalmente desde la aparición de Internet y de la Web, se ha multiplicado la cantidad de bibliotecas digitales y bases de datos documentales. La mayor parte de esta información está formada por texto, entendiendo como tal una secuencia de símbolos que representan no sólo lenguaje natural, sino también música, código de programas, señales de flujo multimedia, secuencias biológicas, series temporales, etc. Sin embargo, dentro de toda esta información textual, la parte más importante está constituida por texto en lenguaje natural.

Toda esta cantidad de información textual sólo resulta útil si se disponen de los medios adecuados para acceder y recuperar los datos concretos que se necesitan en un momento dado. Para ello resulta fundamental el desarrollo de técnicas de búsqueda eficientes, siendo éste el objetivo de la disciplina conocida como Recuperación de Información. Los sistemas de recuperación de información intentan determinar cuáles son los documentos de la colección que satisfacen una necesidad del usuario.

La Recuperación de Información trata de la representación, almacenamiento, organización y acceso a la información presente en colecciones de documentos contextuales. La idea base es que la semántica de los documentos y las necesidades de los usuarios pueden expresarse mediante un conjunto de términos de indexación. El proceso de recuperación de información depende en gran medida de la correcta representación de estos términos. El principal problema consiste en que los conceptos que representan los términos se pueden manifestar de distintas formas, conocidas como variantes lingüísticas.

Cuando se habla de variación lingüística, se refiere a la posibilidad de utilizar diferentes palabras o expresiones para comunicar una misma idea. Este fenómeno incide en el proceso de recuperación de información provocando el silencio documental, es decir, la omisión de documentos relevantes para cubrir la necesidad de información, debido a que no se están utilizando los mismos términos en la consulta que los que aparecen el documento.

Con el objetivo de que no se pierdan documentos relevantes, los sistemas de recuperación de información reconocen y agrupan las variantes mediante algoritmos de *conflación.* El proceso de conflación se puede desarrollar con distintos métodos, como eliminación de afijos, segmentación de palabras, bigramas de letras, técnicas lingüísticas, búsqueda léxica a través de un tesauro o normalización de sintagmas. Las técnicas más usadas son la eliminación de afijos y la búsqueda léxica a partir de un tesauro. Estas técnicas tienen como objetivo que las distintas variantes léxicas se consideren una única unidad equivalente en un sistema de recuperación de información.

Los algoritmos de *stemming* son una de las técnicas de conflación más extendidas, por medio de los cuales las variantes flexionales y derivacionales se reducen a una forma común, o forma canónica. De esta forma, el stemming permite un poco de flexibilidad a la hora de realizar una consulta de términos en un índice, ya que no es necesario que los términos de la consulta coincidan exactamente con los términos del documento, sino que coincidan sus formas canónicas.

El stemming consiste por tanto en quitar los sufijos a una palabra para quedarse con la raíz, obviando modos y tiempos verbales, personas y géneros. Es un proceso no invertible, pero esto no es ningún problema, ya que el término transformado sólo se usa en el índice, mientras que se almacena el texto en su forma original.

El stemming es un proceso que no siempre es deseado, ya que a veces se requiere una búsqueda exacta de los términos, por ejemplo, al hacer búsqueda por autor en un catálogo de una biblioteca, por lo que el proceso de stemming debe restringirse a ciertas partes de los documentos.

El proceso de stemming es diferente al proceso de *lematización*. El objetivo de ambas transformaciones consiste en reducir las formas flexionales y derivacionales a una forma base común (*stem*). Sin embargo, mientras que el stemming consiste únicamente en un proceso heurístico que corta las palabras por el final intentando alcanzar el objetivo la mayoría de las veces, la lematización intenta realizar el proceso de forma correcta, requiriendo el uso de un vocabulario y un análisis morfológico de las palabras, hasta obtener el lema de las palabras. Un ejemplo de la diferencia de ambos procesos sería el resultado final cuando se aplican a una palabra inglesa como "saw". En el primer caso, el stemming devolvería "s" o "saw" (dependiendo del algoritmo utilizado y si tiene en cuenta la longitud de la palabra), mientras que la lematización devolvería "see" o "saw" dependiendo de si el uso de la palabra era como sustantivo o como forma verbal.

Los stemmers usan generalmente reglas específicas del lenguaje, por lo que para su implementación se requiere de un conocimiento detallado del lenguaje en cuestión. Sin embargo, requieren menos conocimiento que los lematizadores, que requieren un vocabulario completo y un análisis morfológico para lematizar correctamente las palabras.

La aproximación basada en reglas presenta dos problemas básicos: el over-stemming (reducir demasiado, representando con un mismo stem formas que deberían ser representadas con varios), y el under-stemming (reducir poco, obteniendo distintos stems para formas que se corresponderían únicamente con uno).

La eficacia del stemming ha sido objeto de discusión. Muchos estudios apuntan que el proceso es eficaz en función de la complejidad morfológica de la lengua en que se opere. La mayoría de los algoritmos son implementados para la lengua inglesa, por lo que sólo nos centraremos en los más importantes de ellos para esta lengua.

Algoritmos de *stemming*

Existen diferentes algoritmos de stemming o de eliminación de afijos. Los principales algoritmos se aplican a la lengua inglesa y son los siguientes:

- Algoritmo de Lovins (1968)
- Algoritmo de Porter (1980)
- Algoritmo de Paice/Husk (1990)

Existen otros muchos algoritmos de stemming que siguen la misma filosofía que los mencionados (como el algoritmo de Dawson, de 1974), además de otros con una base más lingüística y que usan validaciones léxicas (como el algoritmo de Krovetz, de 1993), en el se comprueba la validez de la raíz obtenida en un diccionario.

En este trabajo nos centraremos en los tres algoritmos basados en reglas enumerados anteriormente. A continuación se explican en detalle cada uno de estos algoritmos, definiendo sus reglas, etapas y detallando gráficamente cada uno de los pasos seguidos en un diagrama de flujo.

Posteriormente se compararán estas tres técnicas aplicadas a un sistema real implementado, viendo los efectos producidos en las búsquedas de palabras sobre un texto en inglés.

Algoritmo de Lovins:

En 1968, Julie Beth Lovins presentó el primer algoritmo de stemming conocido, base principal sobre la que se realizó todo el trabajo posterior en esta área. Es un algoritmo de una única pasada y sensible al contexto. En este ámbito, el contexto se refiere a cualquier propiedad de la raíz que queda una vez eliminado un sufijo. Un algoritmo libre de contexto no tiene ninguna restricción a la hora de obtener una nueva raíz, simplemente se aplicaría la primera regla encontrada aplicable.

Para saber qué regla aplicar en cada caso, este stemmer se basa en el principio de la equiparación más larga tomada de una lista bastante amplia de sufijos. Este principio establece que dada cualquier clase de finales de palabra, si emparejan varias reglas simultáneamente, se debe usar aquélla con la que se obtiene emparejamiento mayor. Para implementar este principio se deben recorrer todos los finales en orden decreciente de longitud.

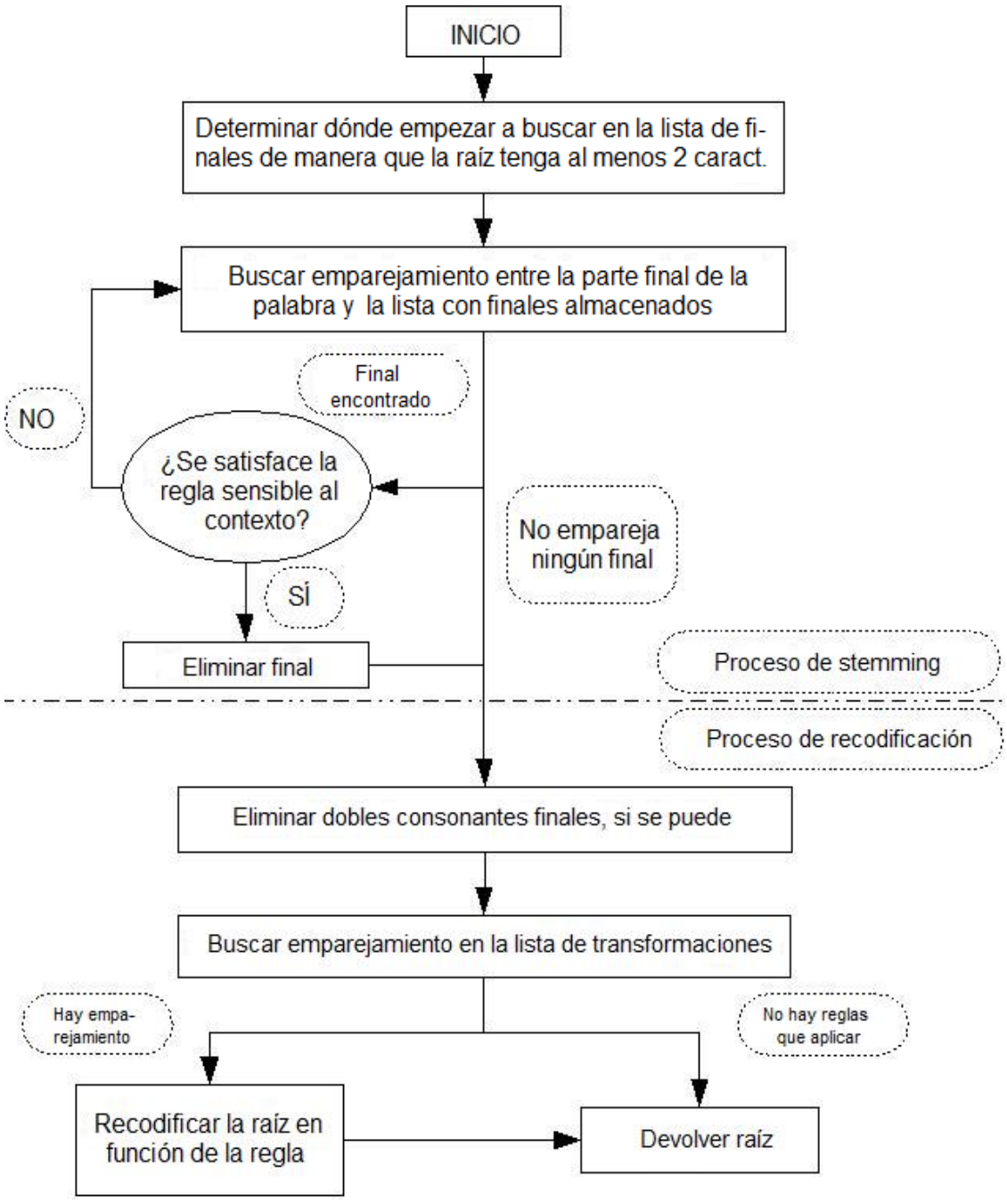
El algoritmo utiliza una larga lista de terminaciones que se asocian a diferentes restricciones contextuales, que previenen la eliminación de dichos finales en algunos casos no deseados. Las reglas intentan cubrir todas las excepciones comunes del lenguaje, e intentan respetar algunas restricciones como que la raíz obtenida debe tener un mínimo de dos letras o evitar eliminar algunos sufijos dependiendo de las letras contenidas en la raíz.

Una excepción ortográfica se refiere al caso en que una raíz se podría escribir de diferentes formas. Esta situación ocurre principalmente debido a la existencia de palabras inglesas derivadas del latín (como “matrix/matrices”), a la diferencia de escritura entre el inglés británico y el inglés americano (por ejemplo “analyzed/analised”) o a singularidades propias del idioma, como doblar consonantes cuando se añade algún sufijo.

El algoritmo tiene dos etapas diferenciadas. En primer lugar hay una fase de stemming, que incluye la eliminación de las partes finales de las palabras y comprobación de excepciones. En la segunda fase del algoritmo se recodifica la raíz obtenida anteriormente.

En la recodificación se utiliza conjunto diferente de reglas que el conjunto utilizado para eliminar los sufijos. En estas reglas se tienen en cuenta todas las excepciones ortográficas comentadas, de forma que las reglas son sensibles al contexto. También es importante ver que estas reglas están ordenadas, de manera que si tuviesen otro orden se obtendrían raíces incorrectas.

A continuación se especifica un diagrama de flujo con los distintos pasos del algoritmo:



Algoritmo de Porter:

El algoritmo de Porter fue presentado en 1980. Desarrollado por Martin Porter en la Universidad de Cambridge, este stemmer es también un algoritmo de eliminación de sufijos sensible al contexto.

Consiste en 5 ó 6 fases secuenciales en las que se procede a la reducción de las palabras. Existen convenciones para elegir qué reglas aplicar en cada fase, como por ejemplo, elegir aquella regla que se aplica al sufijo de mayor longitud.

Muchas reglas usan el concepto de medida de una palabra. De esta forma se comprueba si la palabra tiene la longitud suficiente para aplicar una regla en la que le quitará el sufijo.

Se realizan algunas definiciones previas:

- Una consonante es cualquier letra excepto A, E, I, O, U, además de Y cuando es precedida de una consonante.
  Por ejemplo, en la palabra "BOY" las consonantes son B, Y, mientras que en la palabra "TRY" las consonantes son T, R.
- Una vocal es cualquier letra que no sea consonante.
- Una vocal se denota como *v* y una consonante como *c*.
- Una lista de consonantes seguidas se denota por *C* y una secuencia de vocales se representa como *V*.

Con estas definiciones una palabra puede entenderse como una secuencia $[C](VC)^m[V]$, donde el superíndice m indica m repeticiones de VC, y los corchetes denotan la presencia opcional.

El valor m se llama medida de una palabra, y puede ser igual o mayor que cero. Esta medida, como especificábamos con anterioridad, se usa para decidir si se va a eliminar o no un sufijo.

Ejemplos:

$m = 0$ TR, EE, TREE, Y, BY.

$m = 1$ TROUBLE, OATS, TREES, IVY.

$m = 2$ TROUBLES, PRIVATE, OATEN, ORRERY.

Las reglas del algoritmo tiene la forma (condición) S1 → S2 , que significa que el sufijo S1 se reemplazará por S2 (otro sufijo o la cadena vacía) si las letras restantes una vez reemplazado satisfacen la condición indicada.

Por ejemplo, en la regla (m > 1) EMENT -> , eliminaría el sufijo de REPLACEMENT a REPLAC, ya que REPLAC tiene medida m = 2.

La condición podría seguir los siguientes formatos:

*S - la raíz acaba en S (análogo para el resto de las letras).

v - la raíz contiene una vocal.

*d - la raíz acaba con doble vocal (por ejemplo -TT, -SS).

*o - la raíz acaba en *cvc*, donde la segunda *c* no es W, X ó Y (por ejemplo -WIL, -HOP).

Además de expresiones "and", "or", "not" para formar condiciones más complejas.

Pasos del algoritmo:

- En el primer paso se eliminan los plurales y participios pasados. Este paso es el más complejo y se subdivide en tres partes

  a) Se encarga de eliminar los plurales. A continuación se enumeran las reglas y algunos ejemplos:

```
SSES -> SS      caresses -> caress
IES  -> I       ponies -> poni
                Ties -> ti
SS   -> SS      caress -> caress
S    ->         cats -> cat
```

  b) Elimina las terminaciones "ing", "ed"

```
(m>0) EED -> EE      feed -> feed
                     agreed -> agree
(*v*) ED  ->         plastered -> plaster
                     bled -> bled
(*v*) ING ->         motoring -> motor
                     sing -> sing
```

Si se han ejecutado alguna de las dos últimas reglas del apartado anterior, se ejecutan las siguientes reglas de forma que se recodifica la raíz para que sea una forma reconocible para los siguientes pasos:

```
AT -> ATE              conflat(ed) -> conflate
BL -> BLE              troubl(ed) -> trouble
IZ -> IZE              siz(ed) -> size
(*d and not (*L or *S or *Z)) -> single letter
                       hopp(ing) -> hop
                       tann(ed) -> tan
                       fall(ing) -> fall
                       hiss(ing) -> hiss
                       fizz(ed) -> fizz
(m=1 and *o) -> E      fail(ing) -> fail
                       fil(ing) -> file
```

c) Simplemente transforma las 'y' por 'i' si existe una vocal en la raíz.

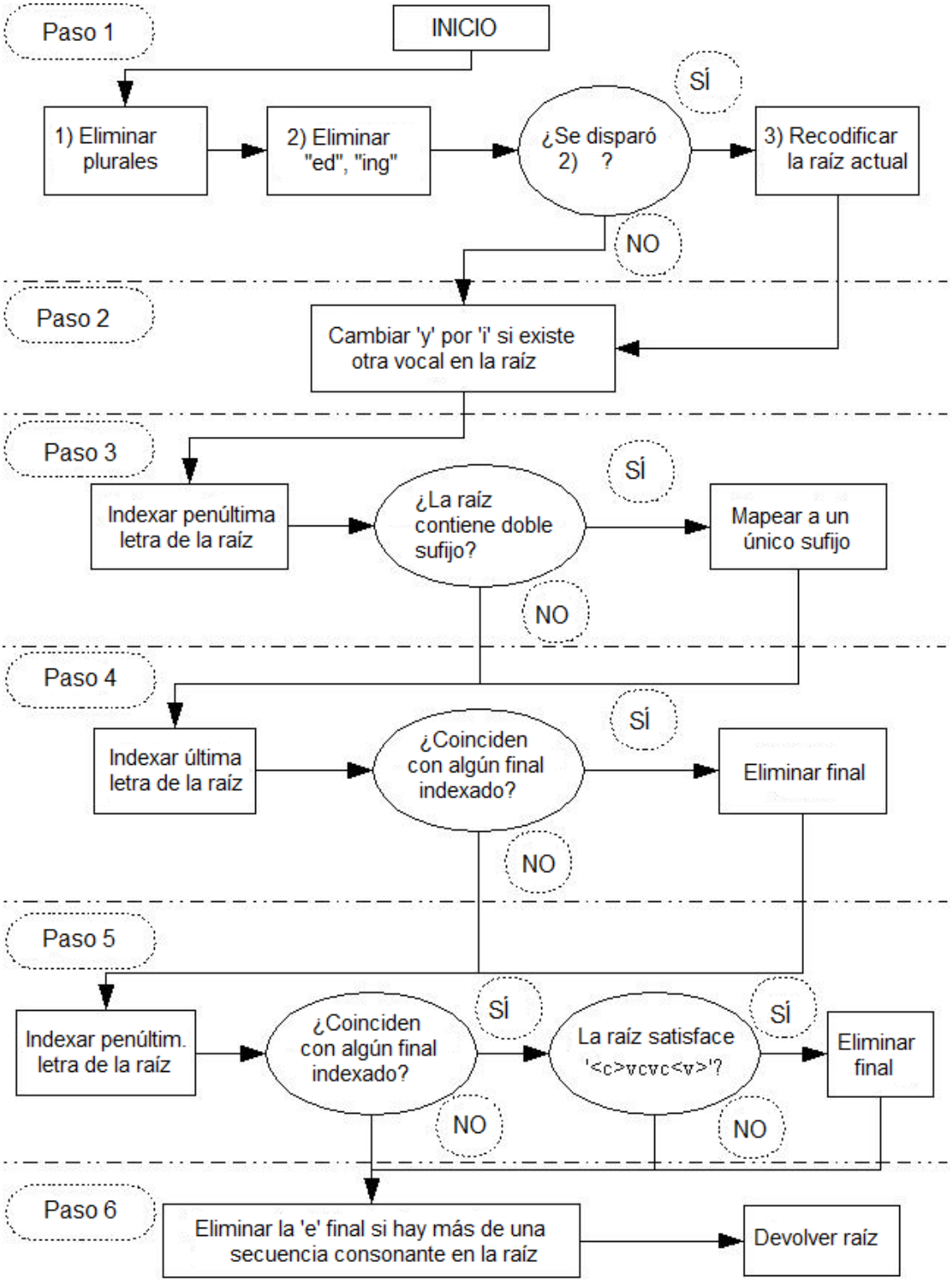
```
(*v*) Y -> I           happy -> happi
                       sky -> sky
```

- Los siguientes pasos son más directos. En ellos se comprueban si las raíces contienen sufijos que se puedan eliminar y, si la palabra tiene suficiente longitud, se lleva a cabo dicha eliminación.

No hay ninguna base lingüística que respalde la decisión de no eliminar el sufijo cuando la palabra es demasiado corta. Simplemente se respalda en la observación del autor del algoritmo en que el uso de esa medida *m* puede ser muy eficiente a la hora de decidir cuándo eliminar sufijos. Enumera los siguientes ejemplos, donde las palabras de la lista A no eliminan su sufijo –ATE, mientras que las palabras de la lista B sí.

list A	list B
RELATE	DERIVATE
PROBATE	ACTIVATE
CONFLATE	DEMONSTRATE
PIRATE	NECESSITATE
PRELATE	RENOVATE

Todos los pasos del algoritmo de Porter se detallan de forma gráfica en el siguiente diagrama de flujo:



Algoritmo de Paice/Husk:

El algoritmo de Paice/Husk, o conocido simplemente como algoritmo de Paice, fue presentado en 1990 y desarrollado por Chris Paice, con la colaboración de Gareth Husk.

Es un algoritmo iterativo que utiliza una única tabla de reglas que especifican si hay que eliminar o reemplazar un final de palabra. Al diferenciar las reglas que eliminan los finales de palabra y aquéllas que los reemplazan se intenta evitar el problema de las excepciones de ortografía. De esta forma no hace falta recodificación de la raíz o buscar emparejamientos parciales.

Las reglas se indexan por las últimas letras del sufijo para ganar eficiencia en los emparejamientos. Cada regla está compuesta por:

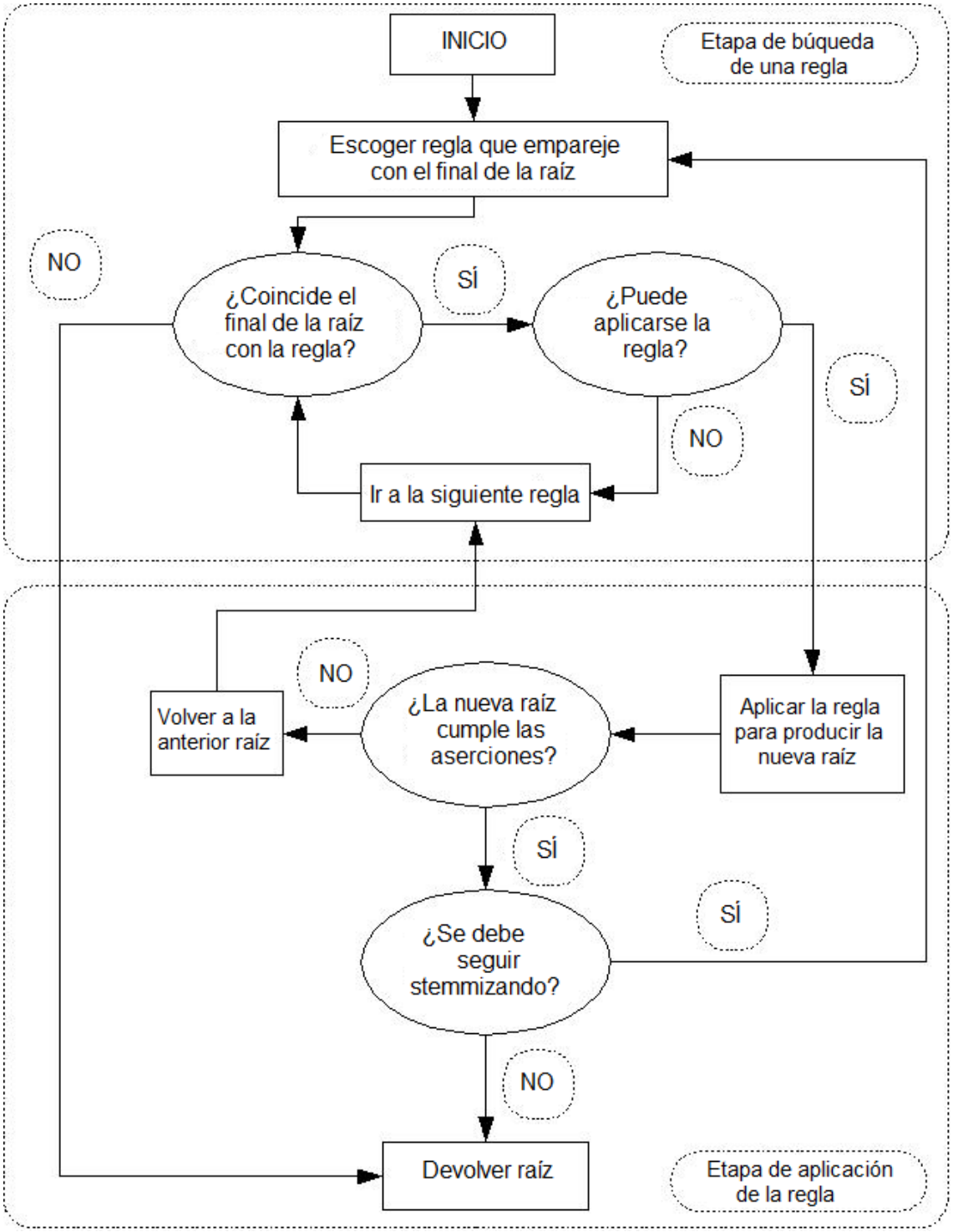
- Un sufijo de uno o más caracteres almacenados al revés
- Un flag opcional * indicando que la palabra no ha sido modificado aún
- Un número indicando el total de letras a eliminar (mayor o igual que cero)
- Un cadena opcional de uno o más caracteres para concatenar a la raíz
- Un símbolo de continuación '>' o de terminación '.' del algoritmo

Por ejemplo, la regla `sei3y>` hace que se elimine el sufijo "ies" (está almacenado en orden inverso), se reemplace por "y" y se vuelva a aplicar el algoritmo de stemming (se indica con el símbolo '>').

La regla `ylp0.` significa que si la palabra termina en "ply" no se hace ningún cambio en la palabra (se especifica esto con el `0`), y se termina el proceso (la terminación del algoritmo se indica con '.').

El algoritmo consiste en cuatro pasos en los que se buscan las reglas a utilizar, y se mira la aplicabilidad de cada regla. Si la parte final de la palabra no coincide con la de la regla o se viola cualquier otra restricción se busca una nueva regla para poder aplicarla a la palabra. En caso contrario, si se puede aplicar la regla encontrada, se elimina el sufijo correspondiente de la forma indicada en la regla y se comprueba el símbolo de terminación para saber si se sigue iterando o se para el proceso y se devuelve la raíz producida en el proceso.

Estos pasos se pueden observar en el diagrama de flujo que aparece a continuación:



Evaluación experimental

Para estudiar más en detalle los distintos algoritmos de stemming y realizar una comparativa entre los mismos se ha implementado un índice invertido comprimido sobre texto comprimido en inglés y se le ha aplicado a la lista de términos cada uno de los distintos algoritmos (Lovins, Porter, Paice).

El índice invertido está orientado a buscar palabras dentro de un único documento, no en una colección de documentos, de forma que las operaciones básicas que debe soportar el índice son encontrar la posición en el texto de las ocurrencias de las palabras buscadas, así como extraer snippets (las frases que contienen la ocurrencia de la palabra).

Para incorporar el proceso de normalización de las palabras usando cada uno de los algoritmos mencionados, se han introducido una serie de estructuras comprimidas que permitan el acceso a las variantes de cada raíz. De esta forma, el índice tiene una lista con la posición de las apariciones de todas las palabras asociadas a una raíz, y para cada raíz se tiene una estructura donde se almacenan todas sus variantes de forma compacta. Esto es necesario ya que el texto se almacena comprimido con respecto a su forma original, es decir, en el texto comprimido aparecerán las palabras con sus variantes, para poder extraer y mostrar al usuario la frase original que rodea a cada ocurrencia de una palabra. Como se trata de un índice invertido a bloques, no se almacenan todas las posiciones de palabras asociadas a una raíz determinada, sino que se almacenan los fragmentos del texto donde aparece alguna de sus variantes, y para localizar la posición exacta hay que buscar todas las variantes en ese fragmento de texto.

Como ejemplo de prueba se ha utilizado un texto Calgary, corpus utilizado ampliamente para la evaluación experimental en el área de compresión de texto. Es un texto pequeño, de unos 2 MB, que se comprime usando el método de compresión End-Tagged Dense Code (un método de compresión orientado a palabras y a bytes, de forma que cada palabra se codifica con una secuencia de byes) obteniendo un texto de un tamaño de aproximadamente 700 KB. Este texto contiene un total de 30.995 palabras distintas obtenidas tras el proceso de compresión (incluidos separadores de palabras y palabras válidas).

Los algoritmos analizados se han implementado utilizando los códigos suministrados por los propios autores, en el caso del algoritmo de Porter y Paice, y por una versión no oficial pero ampliamente reconocida para el algoritmo de Lovins. Se han adaptado para que siguiesen una misma interfaz, de forma que se puedan acoplar al código del índice invertido implementado.

Una vez realizada la implementación, procedemos a la evaluación y comparación de los algoritmos de stemming con respecto a una serie de características que detallaremos a continuación.

Fuerza del stemmer:

Un stemmer se considera "débil" si reduce a una misma raíz sólo los casos en los que las palabras están fuertemente relacionadas, por ejemplo agrupando las formas singulares y plurales, o eliminando ciertos sufijos comunes como "ing" o "ed". Por otra parte, un stemmer se considera "fuerte" si agrupa en una única raíz común una mayor cantidad de variantes.

Esta propiedad, sin embargo, no representa la precisión del algoritmo a la hora de formar los grupos de palabras asociadas, ya que es independiente de la exactitud del método al devolver la raíz de una palabra.

Para evaluar esta propiedad de los algoritmos se utilizan las siguientes medidas:

- Número de palabras por clase de conflación

  Es el tamaño medio de cada clase de conflación, entendiendo por una clase de conflación todas aquellas palabras que se reducen a la misma raíz.

  Es una medida dependiente del número de palabras de entrada. Dado un conjunto de palabras, un mayor valor del parámetro indica un stemmer más fuerte.

  La medida se calcula siguiendo la fórmula: $MWC = \frac{N}{S}$, donde MWC es la media de palabras en las clases de conflación (mean of words per conflaction class), N es el número de palabras distintas antes de realizar el stemming y S el número de raíces distintas después del proceso.

  Para el caso de estudio detallado obtenemos los siguientes valores:
  El número de palabras del vocabulario antes del stemming es: 30995

  El número de raíces distintas después del proceso de stemming es:

  Algoritmo de Lovins: 14819 raíces distintas → $MWC = 2{,}09$

  Algoritmo de Porter: 16599 raíces distintas → $MWC = 1{,}87$

  Algoritmo de Paice: 15083 raíces distintas → $MWC = 2{,}05$

  De esta forma, obtenemos que los stemmings más fuertes son los algoritmos de Lovins y de Paice, mientras que el algoritmo de Porter es el más débil. Estos resultados coinciden con los publicados por Frakes y Fox en un estudio realizado en el año 2003, si bien ellos obtenían el mayor valor para el algoritmo de Paice (aún así obtenían valores similares entre Lovins y Paice como es nuestro caso).

- Índice de compresión

El índice de compresión representa cuánto se reduce la colección de palabras tras el proceso de stemming. La idea es que un stemmer más fuerte obtiene un índice de compresión más alto.

Para calcularlo utilizamos la siguiente fórmula: $IC = \frac{(N-S)}{N}$, donde IC es el índice de compresión, N y S siguen denotando el número de palabras distintas antes del stemming y el número de raíces distintas después del proceso, como se ha explicado anteriormente.

Para el caso de estudio detallado obtenemos los siguientes valores:

Algoritmo de Lovins: 0,52

Algoritmo de Porter: 0,47

Algoritmo de Paice: 0,51

Los valores obtenidos con esta medida indican una relación de orden análoga a la obtenida con la medida anterior, ya que simplemente es una forma distinta de medir la misma propiedad usando los mismos parámetros.

Es necesario detallar que los resultados obtenidos en este experimento son mayores que los obtenidos en el mencionado estudio de Frakes y Fox (su índice de compresión no sobrepasaba no llegaba ni al 0.34) ya que el vocabulario extraído del texto comprimido contiene palabras en mayúsculas y minúsculas, por lo que simplemente transformando todo el vocabulario a minúsculas se obtendría una compresión no despreciable (ya que muchas palabras que contienen su primera letra o su totalidad en mayúsculas puede coincidir con alguna que está toda en minúsculas, agrupando todas estas variantes directamente, sin ni siquiera el uso de un stemming), mientras que los casos de prueba utilizados en el estudio de Frakes y Fox consisten en un conjunto de palabras distintas escritas todas en minúsculas. De hecho, para comprobar este razonamiento hemos calculado el número de palabras distintas después de pasar todo a minúsculas y el resultado es 23348, por lo que los factores de compresión serían 0,36 para el algoritmo de Lovins, 0,29 para el algoritmo de Porter y 0,35 para el algoritmo de Paice, lo que se acercarían a los resultados obtenidos en el estudio.

Existen otras medidas para evaluar la fuerza del stemmer, como:

- Factor de modificación de las palabras: es el porcentaje de palabras que se han modificado en el proceso de stemming, de forma que un alto porcentaje indica que el stemmer es fuerte.

- Número de caracteres eliminado: se mide el número de letras que se eliminan de las palabras con respecto a la raíz. El problema es que esta medida es difícil de aplicar en el caso de stemmers que no sólo eliminan los sufijos de las palabras, sino que también reemplazan unos sufijos por otras terminaciones. Para ello sería mejor utilizar otra medida, como la detallada a continuación.

- Distancia Hamming: se mide como el número de posiciones de dos cadenas que contienen caracteres diferentes. Si tienen diferente longitud, se usa la distancia Hamming modificada, que calcula la distancia Hamming de las dos cadenas hasta la última posición de la cadena más corta y le suma la diferencia de las dos longitudes. Para evaluar la fuerza del stemmer se puede calcular la media de la distancia entre cada palabra y la raíz obtenida con el stemmer.

Similitud entre stemmers

Se puede comparar cómo se parecen los algoritmos de stemmer. Para ello miramos cuántas raíces poseen en común:

Lovins – Porter:	11.402 raíces iguales, un 69% de similitud
Lovins – Paice:	10.405 raíces iguales, un 69% de similitud
Porter – Paice:	11.738 raíces iguales, un 70% de similitud

Se obtienen un 70% de coincidencias sobre el número total de raíces, un valor relativamente alto. Es necesario detallar que existen cerca de 2000 palabras que contienen algún valor numérico, por lo que la mayoría de estas palabras obtienen la misma raíz en todos los algoritmos (la mayoría, números, no sufren ningún proceso de reducción de afijos, como es natural).

Cantidad de errores

Se puede evaluar la eficacia de un algoritmo de stemming contando el número de errores que ocurren en el proceso. Estos errores son de dos tipos:

- Under-Stemming
  Se produce este error cuando dos palabras relacionadas no son reducidas a la misma raíz. Esto reduce la capacidad de recuperación de los sistemas de recuperación de información, de forma que no devuelven todos los documentos deseados al realizar una consulta, ya que el mismo concepto está siendo representado por dos raíces distintas.

- Over-Stemming
  Se produce este error cuando dos palabras no relacionadas son reducidas a la misma raíz. Esto reduce la precisión de los sistemas de recuperación de información, ya se recuperan más documentos de los deseados, al estar representado en la misma raíz otro concepto no relacionado con el de la consulta.

Estos errores se pueden contabilizar si se aplican los stemmers a conjuntos de prueba seleccionados y se comprueba el número de veces que palabras relacionadas han sido reducidas correctamente y erróneamente a la misma raíz y cuántas veces dos palabras relacionadas han sido o no reducidas a raíces distintas.

El objetivo de medir estos errores es que, a pesar de que es positivo comprimir lo máximo posible el índice de términos, comprimirlo al máximo no siempre es óptimo. Cuando la conflación es más fuerte es muy probable que dos palabras que representen el mismo concepto se reduzcan a la misma raíz. De esta forma se puede ganar aumentar la recuperación, pero a costa de la pérdida de precisión en los sistemas de recuperación de información.

En los experimentos realizados observamos que es más frecuente obtener errores de under-stemming en el algoritmo de Porter, ya que es un algoritmo más débil y reduce menos las palabras, pudiendo quedar dos relacionadas sin agrupar, mientras que es más frecuente que los algoritmos de Paice y Lovins sufran errores de over-stemming. Aún así, los dos tipos de errores se dan en los tres algoritmos.

Otros aspectos a tener en cuenta:

Compresión del vocabulario:

Otra forma de medir el índice de compresión es comparar el espacio usado por el índice en tres apartados diferentes: el vocabulario obtenido tras la aplicación del algoritmo (es decir, cuánto ocupa el conjunto de raíces distintas), la lista de variantes obtenidas y cuánto ocupan las listas invertidas que indican las ocurrencias de los términos.

Para el caso de estudio detallado obtenemos los siguientes valores (en bytes):

	Sin stemmer	Lovins	Porter	Paice
Lista de raíces	185.462	96.889	112.827	95.909
Lista de variantes	158.361	124.065	131.302	125.029
Listas invertidas	384.026	323.572	330.768	313.911
Total	727.849	544.526	574.897	534.849

En la tabla se puede apreciar la diferencia de tamaño en bytes de las distintas estructuras utilizadas en el índice (al margen del propio texto comprimido, cuya longitud es independiente del algoritmo de stemming utilizado, ya que se almacena la forma comprimida original).

Con esta tabla observamos que el algoritmo de Paice es el que obtiene una mayor compresión de las estructuras, ya que es más agresivo, haciendo que la lista de raíces ocupe menos. Como vimos anteriormente en el apartado de fuerza del stemmer, Paice obtenía más formas canónicas que Lovins, sin embargo, dados los resultados obtenidos en este apartado, podemos deducir que reduce en mayor medida las palabras, eliminando una mayor cantidad de letras de las mismas. Por este motivo la lista de variantes ocupa un mayor espacio que la de Lovins. Con respecto a Porter, observamos que es el más débil de los tres, de manera que el vocabulario de raíces ocupa un mayor número de bytes que el resto, así como sus variantes.

Tiempo de procesamiento

En este apartado se evaluará la complejidad temporal de los algoritmos midiendo el tiempo que tarda cada stemmer en procesar el vocabulario y crear las estructuras asociadas a las raíces y sus variantes en memoria. El tiempo detallado es el que tarda cada algoritmo en obtener la raíz de cada palabra, para todas las palabras distintas existentes en el texto.

Para el caso de estudio detallado obtenemos los siguientes valores:

Algoritmo de Lovins: 0,24 segundos

Algoritmo de Porter: 0,18 segundos

Algoritmo de Paice: 0,37 segundos

Podemos observar que el algoritmo de Paice es el que más tiempo necesita para procesar todo el vocabulario. Esto se debe a la mayor complejidad del algoritmo, que obtiene, como se ha visto en el apartado anterior, una mayor compresión de las estructuras utilizadas para almacenar el vocabulario de raíces y sus variantes. Además este algoritmo necesita cargar una tabla de reglas en memoria antes de iniciar el procesado de las palabras y reducirlas a raíces.

Por otro lado, observamos que el algoritmo de Porter es el que menos tiempo necesita para realizar el mismo proceso. Esto es coherente con el resultado visto en el apartado anterior, ya que realmente es el que menos modifica las palabras existentes.

Conclusiones

El principal objetivo del proceso de stemming es simplificar el proceso de construcción de las consultas, de manera que se pueda responder a las necesidades del usuario fácilmente, sin tener que expandir la consulta o mediante técnicas más avanzadas. De esta forma, un concepto representado en los documentos por un amplio número de palabras (debido a la variedad propia del lenguaje, y la existencia de formas flexionales y derivacionales de las palabras), se pueda representar como un único término que permita responder a la consulta adecuadamente.

Aunque el objetivo está claro, es difícil realizarlo en la práctica sin más herramienta que unas pocas reglas que intentan determinar cuál es la raíz de una palabra. Además, estas reglas dependen del lenguaje utilizado, por lo que no es fácil adaptar unos algoritmos desarrollados principalmente al inglés a otros idiomas. Una vez efectuado el stemming se obtienen una serie de raíces que representan a un conjunto de palabras que deberían estar relacionadas lingüísticamente. En la práctica, sin embargo, es muy difícil que los algoritmos no produzcan errores, siendo frecuente la reducción de dos palabras no relacionadas a una misma raíz (over-stemming) o el caso opuesto, en el que dos palabras sí relacionadas entre ellas se ven representadas por dos raíces distintas (under-stemming). Estos errores hacen que el sistema de recuperación de información pueda obtener una menor precisión (en el caso del over-stemming, ya que se devuelven documentos que no tienen que ver con las necesidades del usuario) o que baje el número de documentos recuperados (en el caso del under-stemming). En general, por lo visto en el caso de estudio, su uso mejora ampliamente la capacidad de recuperación dada una consulta, además de que no perjudica gravemente la precisión del sistema.

El uso de un algoritmo de stemming produce además una ventaja colateral: se obtiene una significativa reducción del tamaño del índice. Esto ocurre debido a que al tener menos términos en el índice (ya que es una técnica de conflación, varias palabras se representan por una única raíz) hay menos listas invertidas que almacenar, y las que se almacenan son más densas, de manera que se ahorra en punteros.

Además, en este trabajo se ha comparado propiedades de los algoritmos de stemming más conocidos: Lovins, Porter y Paice. Al medir la fuerza del stemmer, se ha obtenido que los algoritmos de Lovins y Paice obtienen resultados mayores que Porter, de manera que se puede concluir que estos algoritmos son más agresivos que el algoritmo de Porter. Estos resultados se ven respaldados por los obtenidos al medir la compresión de las estructuras del índice y los tiempos de procesamiento de los algoritmos.

Que un algoritmo sea más fuerte o más agresivo no implica directamente que sea mejor, ya que puede estar cometiendo una mayor cantidad de errores de over-stemming, de manera que se reducen a una misma raíz conceptos que no tienen una relación lingüística. Un algoritmo más débil genera sin embargo más errores de under-stemming, ya que deja algunas palabras sin agrupar. En general, cuanto más fuerte sea un stemmer, mayor será su capacidad de recuperación de documentos relevantes, perdiendo algo de precisión y obteniendo un mayor índice de compresión.

Dependiendo del contexto en el que se use el sistema de recuperación, el uso de stemmers puede ser beneficioso. Además, según sea la naturaleza de las consultas, el sistema de recuperación debe ofrecer una mayor precisión o le interesará intentar recuperar la mayor cantidad de documentos posibles. Analizando estas características se podrá decidir si es recomendable el uso de un algoritmo de stemming y, en caso afirmativo, la fuerza del algoritmo. Si es más importante la precisión, es recomendable el uso de un algoritmo de stemming más conservador, más débil, como el algoritmo de Porter. Si por el contrario, se desea intentar obtener todos los documentos posibles, sin importar que se devuelva alguno que no sea relevante, se pueden utilizar algoritmos más agresivos como los algoritmos de Paice y Lovins, que son algoritmos más fuertes.

Bibliografía

http://www.tartarus.org/~martin/PorterStemmer/

http://www.cs.waikato.ac.nz/~eibe/stemmers/

http://www.comp.lancs.ac.uk/computing/research/stemming/

http://www.data-compression.info/Corpora/CalgaryCorpus/

Frakes, W.B. & Baeza-Yates, R (1992) *Information Retrieval: Data Structures and Algorithms*, Englewood Cliffs, NJ, Prentice Hall

Frakes, W.B. & Fox, C.J. (2003) *Strength and Similarity of Affix Removal Stemming Algorithms*, SIGIR Forum, 37: 26-30

Harman, D., (1991) *How effective is suffixing?* Journal of the American Society for Information Science 42 (1), 7-15.

Lovins, J. (1968) *Development of a Stemming Algorithm*, Mechanical Translation and Computational Linguistics, 11: 22-31

Paice, C.D. (1990) *Another Stemmer*, SIGIR Forum, 24: 56-61

Porter, M.F. (1980) *An Algorithm for Suffix Stripping*, Program, 14(3): 130-137