UNIVERSIDAD ANDINA
¨NÉSTOR CÁCERES VELÁSQUEZ¨

FACULTAD DE INGENIERÍA DE SISTEMAS
CARRERA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

" PROPUESTA DE UN DICCIONARIO DE PALABRAS STEMMER PARA USO APLICACIONES DE PROCESAMIENTO DE LENGUAJE NATURAL"

TESIS

PRESENTADA POR:

Bach. RIEMAN VALENTINO JEAN BRUNA QUISPE

PARA OPTAR EL TÍTULO PROFESIONAL DE:
"INGENIERO DE SISTEMAS"

JULIACA - PERU

2015

UNIVERSIDAD ANDINA ¨NÉSTOR CÁCERES VELÁSQUEZ¨

FACULTAD DE INGENIERÍA DE SISTEMAS

CARRERA ACADÉMICA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

" PROPUESTA DE UN DICCIONARIO DE PALABRAS STEMMER PARA USO APLICACIONES DE PROCESAMIENTO DE LENGUAJE NATURAL"

PRESENTADA POR:

Bach. RIEMAN VALENTINO JEAN BRUNA QUISPE

PARA OPTAR EL TITIULO PROFESIONAL DE INGENIERO DE SISTEMAS

APROBADO POR:

PRESIDENTE : _____

ING. ALCIDES VELÁSQUEZ ARI

PRIMER MIEMBRO : _____

Dr. RICHARD CONDORI CRUZ

SEGUNDO MIEMBRO : _____

M. Sc. JUAN BENITES NORIEGA

DEDICATORIA

Mi gratitud eterna a ti **DIOS** mío, por la familia que tengo, por esta carrera Profesional maravillosa que me permitiste concluirla con logros, por amarme, protegerme, fortalecerme, por guiar mis pasos y tomarme de tu mano y nunca soltarme, a pesar de mis errores. Gracias Señor por considerarme tu hijo.

A mí querida familia por enseñarme a luchar y salir adelante en la vida a pesar de las dificultades y su apoyo incondicional a lo largo de mi vida personal y profesional gracias.

AGRADECIMIENTO

Agradezco a Dios ser maravilloso, que me dio fuerza y fe para creer lo que me parecía imposible terminar, a mi padre por estar presente no solo en esta etapa tan importante de mi vida, sino en todo momento ofreciéndome lo mejor y buscando lo mejor para mi persona.

No ha sido sencillo el camino hasta ahora, pero gracias a sus aportes, a su amor, a su inmensa bondad y apoyo, lo complicado de lograr esta meta se ha notado menos. Les agradezco, y hago presente mi gran afecto hacia ustedes, mi hermosa familia.

Valentino Bruna Quispe

ÍNDICE

RESUMEN

El recurso más importante que posee la raza humana es conocimiento y la información. En la época actual de información, del manejo eficiente de este conocimiento depende del uso de todos los demás recursos naturales, industriales y humanos.

El Procesamiento del Lenguaje Natural (PLN) es el campo que combina las tecnologías de la ciencia computacional con la lingüística, con el objetivo de hacer posible la comprensión y el procesamiento asistidos por ordenador de información expresada en lenguaje humano para determinadas tareas, como búsquedas de información, la traducción automática, los sistemas de diálogo interactivos, el análisis de opiniones. El PLN no trata de la comunicación por medio de lenguajes naturales de una forma abstracta, sino de diseñar mecanismos para comunicarse y que sean eficaces computacionalmente que se puedan realizar por medio de programas que ejecuten o simulen la comunicación. Los modelos aplicados se enfocan no sólo a la comprensión del lenguaje de por sí, sino a aspectos generales cognitivos humanos y a la organización de la memoria. El lenguaje natural sirve sólo de medio para estudiar estos fenómenos. Hasta la década de 1980, la mayoría de los sistemas de PLN se basaban en un complejo conjunto de reglas diseñadas a mano. A partir de finales de 1980, sin embargo, hubo una revolución en PLN con la introducción de algoritmos de aprendizaje automático, para el procesamiento del lenguaje.

La Lematización es un proceso de eliminación automática de partes no esenciales de las palabras para reducirlas a su parte original (lema). El lema es

la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra. Es decir, el lema de una palabra es la palabra que nos encontraríamos como entrada en un diccionario tradicional. Por ejemplo, decir es el lema de dije, pero también de diré o dijéramos; guapo es el lema de guapas; mesa es el lema de mesas.

La lematización puede realizarse automáticamente mediante programas de análisis morfológico. Hay diversos grados de lematización posible: podemos hacer una lematización puramente morfológica, o bien hacer una lematización sintáctica que tenga en cuenta el contexto en el que aparece la palabra. Por ejemplo, en un análisis morfológico la palabra ama tendría dos lemas: el sustantivo ama y el verbo amar. Sin embargo, en un contexto sintáctico (es decir, en una oración), podemos desambiguarlo y optar por un único lema. Así, en El ama de llaves abrió la puerta, ama es sustantivo, mientras que en María ama a Pedro, ama es del verbo amar. Para poder hacer este tipo de lematización es necesario, por lo tanto, hacer un análisis sintáctico.

Palabras Claves:

Ambigüedad, Lematización, Lenguaje natural, Morfología, Stemmer.

ABSTRACT

The most important resource is the human race possesses knowledge and information. In today's information age, the efficient management of this knowledge depends on the use of all other natural, industrial and human resources.

The Natural Language Processing (NLP) is the field that combines the technologies of computer science to linguistics, in order to enable understanding and computer processing of information expressed in human language assisted for certain tasks, such as searches information, machine translation systems interactive dialogue, analysis of opinions. The PLN is no communication through natural language in an abstract way, but to design mechanisms to communicate and to be effective computationally that can be done through programs that execute or simulate communication. The applied models focus not only to the understanding of language itself, but human cognitive general aspects and memory organization. Natural language serves only as a means to study these phenomena. Until the 1980s, most NLP systems were based on a complex set of rules designed by hand. From the late 1980s, however, there was a revolution in PLN with the introduction of machine learning algorithms for language processing.

Stemming is a process the automatic removal of non-essential parts of words to reduce them to their original party (slogan). The theme is how that agreement is accepted as representative of all inflected forms of a word. That is, the slogan of

a word is the word that we would like input dictionary. For example, say the motto is said, but also to say or we say; is the motto of handsome beautiful; table is the motto of tables.

Stemming can be done automatically by morphological analysis programs. There are varying degrees of stemming possible: we can make a purely morphological stemming either make a syntactic stemming taking into account the context in which the word appears. For example, a morphological analysis on the word love would have two slogans: the noun and the verb to love love. However, in a syntactic context (ie, in a sentence), we can desambiguarlo and opt for a single slogan. Thus, in the housekeeper opened the door, love is a noun, while Mary loves Peter, loves is the verb to love. To make this type of stemming necessary, therefore, to make a parsing.

Keywords:

Ambiguity, Stemming, natural language, Morphology, Stemmer.

INTRODUCCIÓN

"Propuesta de un diccionario de palabras Stemmer para uso aplicaciones de procesamiento de Lenguaje natural"

Un buscador es un sistema o aplicación informática que permite la búsqueda de todo tipo de términos y palabras clave, dándonos como resultado las paginas relacionadas con la palabra ingresada, las búsquedas se realizan mediante la introducción de las denominadas palabras clave, que son simples palabras de referencia que suelen aparecer dentro de un texto guardando correlación con los resultados mostrados, que pueden ser organizados de acuerdo a la temática que estamos buscando.

Esta palabra clave que se ingresa, pasa por un proceso denominado stemmer el cual obtiene el lema o la raíz de dicha palabra, debido que la tecnología de estos buscadores es desconocida o si existe es superficial la información, este método de búsqueda toma mucha importancia para la implementación en buscadores internos de sitios web, debido a esta necesidad se crea y almacena una gran cantidad de palabras lematizadas obteniendo una base de datos de palabras stemmer, utilizando técnicas de lematizacion y de base de datos.

Problema General

¿Es posible el diseño de una propuesta de un diccionario de palabras Stemmer para uso de aplicaciones en procesamiento de Lenguaje natural?

Objetivo General

Diseñar una propuesta de un diccionario de palabras Stemmer para uso de aplicaciones de procesamiento de Lenguaje natural.

Hipótesis General

El diseño de la base de datos con la introducción de más de un millón de palabras nos debe dar una gran cantidad de palabras lematizadas, que sobrepasen los sesenta mil lemas.

Capítulo I: La búsqueda de información se debe realizarse por palabras raíz y derivadas del lexema original. Con esto se mejora la búsqueda porque también busca las palabras en plural/ singular, masculino/femenino, en diferentes tiempos verbales, gerundio, etc. Para realizar ese proceso se reducen las palabras a su léxico básico, el proceso se denomina Stemmer.

Capítulo II: El procesamiento de lenguaje natural se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la comunicación entre personas y máquinas por medio de lenguajes naturales, una de las ramas del PLN se encuentra la búsqueda de información.

Capítulo III:

- Diseño de investigación cuasi experimental.
- Tipo de investigación aplicativo.

Capítulo IV:

- Conclusiones y recomendaciones.

CAPÍTULO I

1. GENERALIDADES

1.1. Título

"Propuesta de un diccionario de palabras Stemmer para uso aplicaciones de procesamiento de Lenguaje natural"

1.2. Descripción del Problema

Al navegar por internet, la accesibilidad a un sitio web es muy sencillo a solo un clic, pero esto se dificulta al realizar búsquedas de información específica ya que en la actualidad existen más de 1000 millones de páginas web, la búsqueda de esta información sería una tarea ardua y, a veces, con resultados no deseados, pues bien para encontrar la información (páginas web) que nos interesa tenemos que usar un motor de búsqueda Google, Yahoo Altavista, etc. estos facilitan la búsqueda de información al ingresar la palabra que deseamos, en pocos segundos, aparecerán en la pantalla del ordenador los resultados encontrados que, en realidad, son diferentes enlaces a otras páginas relacionados con la palabra en cuestión; sin

embargo, algo muy diferente sucede con los buscadores internos de las páginas web o sitios web que al realizar una consulta, si los datos ingresados no son exactamente iguales a la información que se tiene en dicha página, los resultados son nulos, esto se debe a la poca implementación e importancia, que en realidad deberían de tener.

El problema radica en el poca comprensión de las palabras por las diferente variaciones lingüísticas, la búsquedas también deberían realizarse por palabras raíz y derivadas del lexema original. Con esto se mejora la búsqueda porque también busca las palabras en plural/ singular, masculino/femenino, en diferentes tiempos verbales, gerundio, para realizar ese proceso se reducen las palabras a su léxico básico, el proceso se denomina stemmer.

Esto es parte del lenguaje natural, entendido como la herramienta que utilizan las personas para expresarse, posee propiedades que merman la efectividad de los sistemas de recuperación de información textual. Estas propiedades son la variación lingüística y la ambigüedad lingüística. Cuando hablamos de la variación lingüística nos referimos a la posibilidad de utilizar diferentes palabras o expresiones para comunicar una misma idea. En cambio, la ambigüedad lingüística se produce cuando una palabra o frase permite más de una interpretación.

Ambos fenómenos inciden en el proceso de recuperación de información aunque de forma distinta. La variación lingüística provoca el silencio documental, es decir la omisión de documentos relevantes para cubrir la necesidad de información, ya que no se han utilizado los mismos términos que aparecen en el documento. En cambio, la ambigüedad implica el ruido documental, es decir la inclusión de documentos que no son significativos, ya que se recuperan también documentos que utilizan el término pero con significado diferente al requerido. Estas dos características dificultan considerablemente el tratamiento automatizado del lenguaje. A continuación se muestran algunos ejemplos que ilustran la repercusión de estos fenómenos en el proceso de recuperación de información:

A nivel morfológico una misma palabra puede adoptar diferentes roles morfo-sintácticos en función del contexto en el que aparece, ocasionando problemas de ambigüedad (ejemplo 1).

Ejemplo 1. Deja la comida que sobre sobre, la mesa de la cocina, dijo llevando el sobre en la mano.

La palabra sobre es ambigua morfológicamente ya que puede ser un sustantivo masculino singular, una preposición, y también la primera o tercera persona del presente de subjuntivo del verbo sobrar.

A nivel sintáctico, centrado en el estudio de las relaciones establecidas entre las palabras para formar unidades superiores, sintagmas y frases, se produce ambigüedad a consecuencia de la posibilidad de asociar a una frase más de una estructura sintáctica. Por otro lado, esta variación supone la posibilidad de expresar lo mismo pero cambiando el orden de la estructura sintáctica de la frase. (ejemplo 2).

Ejemplo 2. María vio a un niño con un telescopio en la ventana.

La interpretación de la dependencia de los dos sintagmas preposicionales, con un telescopio y en la ventana, otorga diferentes significados a la frase: (i) María vio a un niño que estaba en la ventana y que tenía un telescopio, (ii) María estaba en la ventana, desde donde vio a un niño que tenía un telescopio, y (iii) María estaba en la ventana, desde donde miraba con un telescopio, y vio a un niño.

A nivel semántico, donde se estudia el significado de una palabra y el de una frase a partir de los significados de cada una de las palabras que la componen. La ambigüedad se produce porque una palabra puede tener uno o varios sentidos, es el caso conocido como polisemia (ejemplo 3).

Ejemplo 3. Luís dejó el periódico en el banco.

El término banco puede tener dos significados en esta frase, (i) entidad bancaria y (ii) asiento. La interpretación de esa frase va más allá del análisis de los componentes que forman la frase, se realiza a partir del contexto en que es formulada.

La variación léxica**,** que hace referencia a la posibilidad de utilizar términos distintos a la hora de representar un mismo significado, es decir el fenómeno conocido como sinonimia (ejemplo 4):

Ejemplo 4: Coche / Vehículo / Automóvil.

A nivel pragmático, basado en la relación del lenguaje con el contexto en que es utilizado, en muchos casos no puede realizarse una interpretación literal y automatizada de los términos utilizados. En determinadas circunstancias, el sentido de las palabras que forman una frase tiene que interpretarse a un nivel superior recurriendo al contexto en que es formulada la frase. (ejemplo 5).

Ejemplo 5. Se moría de risa.

En esta frase no puede interpretarse literalmente el verbo morirse si no que debe entenderse en un sentido figurado.

Otra cuestión de gran importancia es la ambigüedad provocada por la anáfora, es decir, por la presencia en la oración de pronombres y adverbios que hacen referencia a algo mencionado con anterioridad (ejemplo 6).

Ejemplo 6. Ella le dijo que los pusiera debajo

La interpretación de esta frase tiene diferentes incógnitas ocasionadas por la utilización de pronombres y adverbio: ¿quién habló?, ¿a quién?, ¿qué pusiera qué?, ¿debajo de dónde? Por tanto, para otorgar un significado a esta frase debe recurrirse nuevamente al contexto en que es formulada.

Con todos los ejemplos expuestos queda patente la complejidad del lenguaje y que su tratamiento automático no resulta fácil ni obvio.

1.3. Planteamiento del Problema

Para el siguiente proyecto de investigación se ha formulado la siguiente interrogante

1.3.1. Problema General

¿Es posible el diseño de una propuesta de un diccionario de palabras stemmer para uso de aplicaciones en procesamiento de Lenguaje natural?

1.3.2. Problemas Específicos

- ¿Por qué es posible almacenar en una base de datos, una gran cantidad de palabras lematizadas?
- ¿Cómo mejorar el rendimiento de la base de datos para realizar consulta en menos de dos segundos de tiempo?

1.4. Justificación del problema

La lematización (stemmer) es un proceso lingüístico que consiste en, dada una forma flexionada (es decir, en plural, en femenino, conjugada, etc), hallar el lema correspondiente.

El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra, en tal sentido se podría decir que el lema es la palabra que nos encontraríamos como entrada en un diccionario tradicional seria la palabra raíz, plural/singular, masculino/femenino, en diferentes tiempos verbales, Por ejemplo:

- Decir, es el lema de dije, pero también de diré o dijéramos.
- Guapo, es el lema de guapas
- Mesa, es el lema de mesas.

La lematización puede realizarse automáticamente mediante programas de análisis morfológico, estos programas los

encontramos en la web como el Grampal. Lematizador, gedl. estos son portales en los cuales hay que ingresar datos y luego de un proceso nos devuelven las palabras ya lematizadas, estos procesos son desconocidos por lo que la implementación en los buscadores de las páginas web seria casi nula.

Por lo cual es necesario la implementar, una base de datos de palabras lematizadas.

La lematización es una tarea propia de la Lingüística Computacional, y es útil en la tecnología aplicada a buscadores, traductores automáticos, extracción de información y demás herramientas vinculadas al Procesamiento del Lenguaje Natural.

1.5. Objetivos

1.5.1. Objetivo General

Diseñar una Propuesta de un diccionario de palabras Stemmer para uso de aplicaciones de procesamiento de Lenguaje natural.

1.5.2. Objetivos Específicos

- Almacenar grandes proporciones de palabras lematizadas, en una base de datos.
- Optimizar los resultados de la consulta en la base de datos a menos de 2 segundos de tiempo.

1.6. Hipótesis

1.6.1. Hipótesis general

El diseño de la base de datos con la introducción de más de un millón de palabras nos debe dar una gran cantidad de palabras stemmer, que sobrepasen los sesenta mil lemas.

1.6.2. Hipótesis especifico

- Con la gran cantidad de documentos digitalizados, se cuenta con recursos para poder lematizar grandes volúmenes de palabras.
- El uso de particiones en la base de datos mejora el tiempo de consulta.

1.7. Variables

1.7.1. Variable Dependiente

- Base de datos

1.7.2. Variable Independiente

- Stemmer (Palabras)

1.7.3. Operación de Variables

Variables	Dimensión	Indicadores
Variable dependiente		
Base de Datos	Gestor de base de datos	□ Tablas □ Registro
Variable independiente		
Palabras	Diccionario	□ Español □ Lema o Raíz

1.8. Técnicas e instrumentos de verificación

Para el estudio de la variable antes mencionada así como sus indicadores se utilizó observación cuasi experimental y Herramientas de registro y control.

CAPÍTULO II

2. Marco Teórico

2.1. Antecedentes:

Autor: Gloria Clavería y Maria Jesús Mancho.

Título: Estudio del léxico y base de datos.

Instituto: Universidad Autónoma de Barcelona.

Año: 2006.

Aplicaciones de las Bases de Datos al Análisis de las Formaciones Compuestas al español.

En la actualidad el investigador cuenta con la enorme ventaja de poder aplicar a su estudio las herramientas que ofrecen las nuevas tecnologías. Gracias a las bases de datos por ejemplo el lingüista puede tener acceso a mayor cantidad de datos, puede establecer relaciones entre éstos, y de este modo puede llegar a efectuar y falsar hipótesis. Así pues este breve estudio tiene el propósito de demostrar alguna de la posibles aplicaciones de las bases de datos al estudio léxico concretamente al de

las palabra creadas a partir de un proceso compositivo tomando como corpus los compuestos que aparecen en el Diccionario Crítico Etimológico Castellano e Hispánico (DCECH) de J. Coromidas con la colaboración de J. A. Pascual como se podrá comprobar a lo largo de este trabajo, la aplicación de una base de datos al estudio de los compuestos se convierten en la mejor manera de llegar a conocer la concepción y tratamiento de los compuestos por parte de los autores del (DCECH), y al mismo tiempo hace posible llegar a establecer o reafirmar cuestiones generales sobre la composición del español.

De este modo esta investigación va estar estructurada en tres apartados el primero de ellos se va a centrar en la concepción y tratamiento de los compuesto del (DCECH), el segundo va a estar dedicado a la metodología y frases de creación de una base de datos teniendo en cuenta los compuestos del diccionario de Coromidas y Pascual y por último, y por ultimo trataremos de extraer resultados del análisis de la base de datos para poder aplicarlos a los estudios delos compuestos del español.

2.2. Procesamiento de lenguaje Natural

El procesamiento de lenguajes naturales, abreviado PLN, o NLP del idioma inglés Natural Language Processing, es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano. El PLN se ocupa de la formulación e investigación de mecanismos eficaces

computacionalmente para la comunicación entre personas y máquinas por medio de lenguajes naturales. El PLN no trata de la comunicación por medio de lenguajes naturales de una forma abstracta, sino de diseñar mecanismos para comunicarse que sean eficaces computacionalmente que se puedan realizar por medio de programas que ejecuten o simulen la comunicación. Los modelos aplicados se enfocan no sólo a la comprensión del lenguaje de por sí, sino a aspectos generales cognitivos humanos y a la organización de la memoria. El lenguaje natural sirve sólo de medio para estudiar estos fenómenos. Hasta la década de 1980, la mayoría de los sistemas de PNL se basaban en un complejo conjunto de reglas diseñadas a mano. A partir de finales de 1980, sin embargo, hubo una revolución en PNL con la introducción de algoritmos de aprendizaje automático para el procesamiento del lenguaje.

2.2.1. Procesamiento de lenguaje Natural

La historia del PLN empieza desde 1950, aunque existe trabajo encontrado desde periodos anteriores. En 1950, Alan Turing publicó Computing machinery and intelligence el cual proponía lo que hoy llamamos test de turing como criterio de inteligencia. El experimento de Georgetown en 1954 involucro traducción automática de más de sesenta oraciones del ruso al inglés. Los autores clamaron que en tres o cinco años la traducción automática sería un problema resuelto. El progreso real en traducción automática fue más lento y después del reporte ALPAC

en 1996, el cual demostró que la investigación había tenido un bajo desempeño. Más tarde investigación a menor escala en traducción automática se llevó a cabo hasta finales de 1980, cuando se desarrollaron los primeros sistemas de traducción automática estadística. Esto se debió tanto al aumento constante del poder de cómputo resultante de la Ley de Moore y la disminución gradual del predominio de las teorías lingüísticas de Noam Chomsky (por ejemplo, la Gramática Transformacional), cuyos fundamentos teóricos desalentaron el tipo de lingüística de corpus, que se basa el enfoque de aprendizaje de máquinas para el procesamiento del lenguaje.

2.2.2. Dificultades en el procesamiento de lenguaje natural

a) **Ambigüedad**

El lenguaje natural es inherentemente ambiguo a diferentes niveles:

- A nivel léxico, una misma palabra puede tener varios significados, y la selección del apropiado se debe deducir a partir del contexto oracional o conocimiento básico. Muchas investigaciones en el campo del procesamiento de lenguajes naturales han estudiado métodos de resolver las ambigüedades léxicas mediante diccionarios, gramáticas, bases de conocimiento y correlaciones estadísticas.

- A nivel referencial, la resolución de anáforas y catáforas implica determinar la entidad lingüística previa o posterior a que hacen referencia.
- A nivel estructural, se requiere de la semántica para desambiguar la dependencia de los sintagmas preposicionales que conducen a la construcción de distintos árboles sintácticos. Por ejemplo, en la frase *Rompió el dibujo de un ataque de nervios*.
- A nivel pragmático, una oración, a menudo, no significa lo que realmente se está diciendo. Elementos tales como la ironía tienen un papel importante en la interpretación del mensaje.

Para resolver estos tipos de ambigüedades y otros, el problema central en el PLN es la traducción de entradas en lenguaje natural a una representación interna sin ambigüedad, como árboles de análisis.

b) Detección de separación entre las palabras

En la lengua hablada no se suelen hacer pausas entre palabra y palabra. El lugar en el que se debe separar las palabras a menudo depende de cuál es la posibilidad que mantenga un sentido lógico tanto gramatical como contextual. En la lengua escrita, idiomas como el chino mandarín tampoco tienen separaciones entre las palabras.

c) Recepción imperfecta de datos

Acentos extranjeros, regionalismos o dificultades en la producción del habla, errores de mecanografiado o expresiones no gramaticales, errores en la lectura de textos mediante OCR.

d) Componentes

- Análisis morfológico. El análisis de las palabras para extraer raíces, rasgos flexivos, unidades léxicas compuestas y otros fenómenos.
- Análisis sintáctico. El análisis de la estructura sintáctica de la frase mediante una gramática de la lengua en cuestión.
- Análisis semántico. La extracción del significado de la frase, y la resolución de ambigüedades léxicas y estructurales.
- Análisis pragmático. El análisis del texto más allá de los límites de la frase, por ejemplo, para determinar los antecedentes referenciales de los pronombres.
- Planificación de la frase. Estructurar cada frase del texto con el fin de expresar el significado adecuado.
- Generación de la frase. La generación de la cadena lineal de palabras a partir de la estructura general de la frase, con sus correspondientes flexiones, concordancias y restantes fenómenos sintácticos y morfológicos.

2.2.3. Aplicaciones

Las principales tareas de trabajo en el PLN son:

- Síntesis del discurso
- Análisis del lenguaje
- Comprensión del lenguaje
- Reconocimiento del habla
- Síntesis de voz
- Generación de lenguajes naturales
- Traducción automática
- Respuesta a preguntas
- Recuperación de la información
- Extracción de la información

2.3. Stemmer

Un lematizador o stemmer es un software o proceso que permite, a partir de un texto etiquetado gramaticalmente, conocer el lema, es decir la raíz o la forma del diccionario de todas las palabras de un texto, en la siguiente figura 01 se muestra el diagrama de flujos del lematizador stemmer.

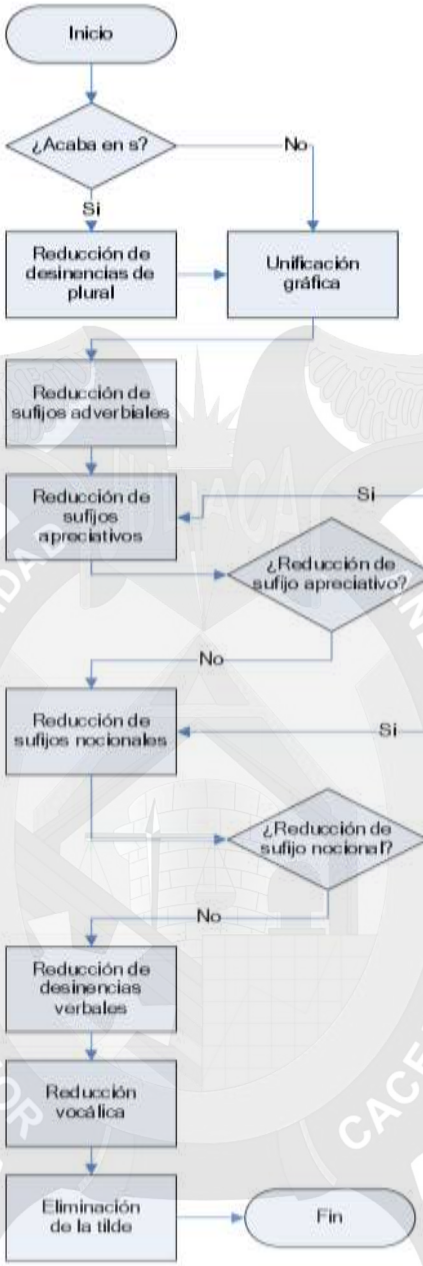


Figura 01

Fuente: http://rua.ua.es/dspace/handle/10045/1310

Al proceso Stemming es un método para reducir una palabra a su raíz o (en inglés) a un stem o lema. Hay algunos algoritmos de stemming que ayudan en sistemas de recuperación de información. Stemming aumenta el recall que es una medida sobre el número de documentos que se pueden

encontrar con una consulta. Por ejemplo una consulta sobre "bibliotecas" también encuentra documentos en los que solo aparezca "bibliotecario" porque el stem de las dos palabras es el mismo ("bibliotec").

2.3.1. Algoritmos

El algoritmo más común para stemming es el algoritmo de Porter. Existen además métodos basados en análisis lexicográfico y otros algoritmos similares (KSTEM, stemming con *cuerpo*, métodos lingüísticos).

2.3.2. Software

Snowball es un pequeño lenguaje de programación para el manejo de strings que permite implementar fácilmente algoritmos de stemming. Se puede generar código en ANSI C y Java. Las páginas de Snowball contienen stemmers para 12 idiomas (incluido el castellano , catalán y euskera). Todas las explicaciones, sin embargo, son dadas en inglés.

2.3.3. Stemming en buscadores comerciales

Desde hace poco tiempo Google utiliza stemming al igual que Yahoo! (donde tiene que activarse explícitamente). En general, los buscadores comerciales no dan muchas explicaciones sobre los algoritmos utilizados.

2.4. Bases de Datos

Una base de datos es un "almacén" que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente.

2.4.1. Particiones

Una partición es una división de una base de datos lógica o sus elementos constituyentes en partes independientes. La partición de bases de datos se hace normalmente por razones de mantenimiento, rendimiento o manejo.

Una aplicación popular y favorable es en un Sistema de Administración de Base de Datos Distribuida. Cada partición puede ser extendida hasta múltiples nodos, y los usuarios en el nodo pueden hacer transacciones locales en la partición. Esto aumenta el rendimiento en sitios que tienen transacciones regularmente involucrando ciertas vistas de datos, y manteniendo la disponibilidad y la seguridad.

Esta partición puede hacerse creando bases de datos más pequeñas separadas (cada una con sus propias tablas, índices, y registros de transacciones) o dividiendo elementos seleccionados, por ejemplo, solo una tabla.

Partición horizontal consiste en poner diferentes filas en diferentes tablas. Por ejemplo, clientes con códigos postales menores que 50000 están almacenados en la tabla ClientesEste, mientras que los clientes con códigos postales mayores o iguales a 50000 están almacenados en la tabla ClientesOeste. Las dos tablas de partición son entonces ClientesEste y ClientesOeste, mientras que una vista con una unión podría ser creada con las dos tablas para poder dar una vista completa de todos los clientes.

Partición vertical consiste en crear miles de tablas con miles de columnas y crear tablas para poner las columnas restantes.

2.4.2. ¿Qué es MySQL?

MySQL es un sistema de gestión de base de datos relacional, esto quiere decir que almacenan datos en tablas separadas en lugar de poner todos los datos en un solo lugar, esto agrega velocidad y flexibilidad. Las tablas son enlazadas al definir relaciones que hacen posible combinar datos de varias tablas cuando se necesitan consultar datos.

MySQL está basado en lenguaje de consulta estructurado (SQL).

2.4.3. ¿Porque utilizar MySQL ?

Existe una variedad de base de administradores de bases de datos MySQL, Microsoft SQL Server, Oracle, Microsoft Access, etc.

MySQL es sumamente fácil de administrar, operar y es famoso por su instalación de 15 minutos, la cual en 15 minutos es posible instalar, configurar y montar una base de datos relacional.

2.5. Marco Conceptual

a) Atomicidad

Cada transacción del usuario debe tratarse de forma atómica. O se ejecuta todo o nada. En todo sistema la información es muy importante y no es posible realizar una transacción a medias. Una transacción se ejecuta exactamente una vez y tiene carácter atómico (de subdivisión), es, decir, el trabajo se realiza en su totalidad o no se realiza en ningún caso.

b) Aislamiento

Una transacción es una unidad de aislamiento, permitiendo que transacciones concurrentes se comporten como si cada una fuera una única transacción que se ejecuta en el sistema. Las transacciones alcanzan el nivel más alto de aislamiento cuando se pueden realizar. En este nivel, los resultados obtenidos de un conjunto de transacciones concurrentes son idénticos a los obtenidos mediante la ejecución en serie de las transacciones.

c) Consistencia

Las transacciones han de cumplir las restricciones definidas dentro la base de datos. Si no las pueden cumplir, se evita su ejecución. De esta forma se conserva la integridad y coherencia de los datos.

d) Durabilidad

Una vez se ha completado la transacción, los resultados de la misma han de ser permanentes y sobrevivir a posibles caídas del sistema o la base de datos.

Debido a que las RDBMS tienen que soportar todas estas propiedades, nunca serían tan rápidas como trabajar directamente sobre archivos, aunque internamente trabajen sobre ellos. La mayoría de desarrolladores prefieren hoy en día sacrificar la velocidad por las funcionalidades.

e) Lema

El lema es la forma que por convención se acepta como representante de todas las formas flexionadas de una misma palabra.

f) Lematizacion

La lematización es un proceso lingüístico que, dada una palabra flexionada, encuentra su lema

Ejemplo: palabra: comiendo, lema: comer

Una palabra está flexionada cuando está en plural (amigos), en femenino (amiga), conjugada (comiendo), en diminutivo (amiguita) o en superlativo (amigota grandota).

g) Stemmer

Es un lematizador, eliminado variaciones lingüísticas como verbos, gerundios, etc. de las palabras, dejando solo la palabra raíz

CAPÍTULO III

3. METODOLOGÍA DE LA INVESTIGACIÓN

3.1. Diseño De La Investigación

- Cuasi experimental

a) Tipo de la investigación

Aplicativo

3.2. Población y Muestra

Población

En el presente trabajo de investigación la población de estudio son las palabras en español, documentos digitalizados que estuvo constituido por más de un millón de palabras

Muestra

La muestra es una porción representativa y adecuada de la población a partir de los cuales se obtuvo datos que son puntos de partida de las generalizaciones.

La técnica que se usó para el presente trabajo de investigación fue, no aleatorio o empírico, la técnica consiste que la muestra es obtenida a criterio del investigador.

Calculo del tamaño de la muestra

$$\boldsymbol{n} = \frac{z^2 * \mathrm{p} * \mathrm{q} * \mathrm{n}}{e^2(n-1) + z^2 * p * q}$$

Donde:

Z : Nivel de Confianza 0.9

P : probabilidad de éxito 0.5

Q : Probabilidad de fracaso 0.5

N : Población 100000.

E : Margen de error de muestra 0.10

Se realiza el cálculo basándonos en la fórmula.

$$\boldsymbol{n} = \frac{z^2 * \mathrm{p} * \mathrm{q} * \mathrm{n}}{e^2(n-1) + z^2 * p * q}$$

$$\boldsymbol{n} = \frac{1.96^2 * 0.5 * 0.5 * 1000000}{0.1^2(1000000-1) + 1.96^2 * 0.5 * 0.5}$$

$$\boldsymbol{n} = 384.01$$

Una vez aplicado la formula, redondeamos al entero inmediato se obtuvo n = 384 palabras la cuales, el cual fue el tamaño para la muestra para el trabajo de investigación de este modo se garantiza la funcionalidad de la base de datos.

a) Instrumentos de Recopilación de los Datos

- La recopilación de los datos para realizar la lematizacion, en el presente trabajo de investigación se realizó atreves del lenguaje de palabras en español.
- Para la evaluación de los datos recopilados se tuvo que realizar filtrados de los datos.

3.3. MATERIALES Y METODOS

3.3.1. Stemming

a) Consideraciones Previas

La primera etapa del procesamiento de texto natural (NLP) es sin duda, el etiquetado de palabras partiendo de un texto plano. Para esto se utilizan numerosas técnicas de búsqueda en diccionario, muchas de las cuales solamente funcionan con palabras exactas. Cuando el texto provee de fuentes con errores, los métodos tradicionales de búsquedas en diccionarios fallan o son muy poco eficientes.

Uno de los motivos de las dificultades en esta área es, precisamente por la enorme cantidad de datos que deben estar almacenados, la ambigüedad de los mismos y sus estructuras, presentes en cada nivel sucesivo de análisis. En el inglés, las flexiones de los verbos son apenas 4 por verbo, los sustantivos y adjetivos solamente existen en dos formas (plural y singular), los artículos no poseen género ni número.

En cambio en el español, solamente la flexión de los verbos involucra 3 personas, 2 números, 4 tiempos, 3 modos y ciertas combinaciones especiales; resultando cerca de70 formas verbales simples. Si a esto anexamos las formas compuestas y los pronombres enclíticos anidables con sus géneros y números, tenemos varias centenas de posibles palabras y grupos a considerar por cada verbo. Con los sustantivos y adjetivos, suceda algo similar si consideramos los "accidentes" como sufijos y prefijos diminutivos, superlativos y peyorativos; además del plural, singular, el neutro y el colectivo (para grupos). Como es de esperar, la dimensión de estas cifras es casi inmanejable sin algoritmos de reducción o bases de datos enormes. Un diccionario completo para el español que contenga las formas más usadas, tendría más de 300 millones de palabras, ocupando del orden de entre 2 y 5 Terabytes, dependiendo del método de almacenamiento y la cantidad de información de etiquetado por palabra. Estos 2-5 Terabytes se calculan asumiendo palabras de 3.5 letras como promedio y etiquetas simples: lema, clasificación gramatical, persona, modo, tiempo, género y número. Esto no involucra información semántica ni palabras relacionadas: sinónimos, antónimos, meronimias e hiperonimias, entre otras. Para etiquetado POS (Part-of- speech), se utilizan diferentes mecanismos y muy a menudo se recurre a técnicas estadísticas con buenos resultados, pero el etiquetado previo sigue siendo un problema complejo.

Una de las técnicas que se utiliza para esto es utilizar un lematizador. Un lematizador se encarga de reducir la duplicidad de contenidos para el aprendizaje, llevando cada palabra a su raíz. De esta forma las palabras "canción" y "canciones" serían lo mismo para el algoritmo de detección de SPAM.

b) **Lematizadores**

Uno de los pasos previos para realizar lo que se viene llamando *minería de texto* es lematizar el texto. Desafortunadamente, no existen buenos lematizadores en español. Al menos, buenos lematizadores libres.

Existen el llamado algoritmo de porter y snowball pero, o son demasiado crudos o están más pensados para un lenguaje con muchas menos variantes morfológicas que el español.

Las reglas de flexión para prefijos y sufijos, se han diseñado basándonos en un estándar de reglas morfológicas llamado compresión AFFIX ampliado especialmente por nosotros para permitir recuperación y expresión gramática y semántica.

El sistema para reconocer las flexiones que ha sufrido un lema es llamado lematizador y se basa en aplicar una a una las reglas de flexión en forma inversa (des-flexión) hallando una presumible raíz o

forma canónica. Luego a ésta forma se la busca entre las palabras del diccionario con sus reglas. Si es hallada se coteja que posea la regla desde la cual se partió y en caso positivo, ésta raíz es la forma canónica buscada.

Durante este proceso se obtiene acumulativamente, un conjunto de etiquetas, tanto la regla aplicada, como la palabra hallada pueden tener etiquetas gramaticales, mientras que las reglas además pueden además contener datos semánticos. Estas reglas se combinan mediante un mecanismo acumulativo con reglas propias de prioridad que aseguran que se construirá la o las etiquetas gramaticales apropiadas y de haberla, también se adicionará la información semántica. El resultado se expresa en una estructura de datos en formato del grupo extendido con el adicionado de un conjunto de etiquetas especiales para uso semántico.

c) Snowball

Snowball es un pequeño lenguaje de programación para el manejo de strings que permite implementar fácilmente algoritmos de stemming. Se puede generar código en ANSI C y Java. Las páginas de Snowball contienen stemmers para 12 idiomas (incluido el castellano , catalán y euskera). Todas las explicaciones, sin embargo, son dadas en inglés.

- Código

```
routines (
        postlude mark_regions
        RV R1 R2
        attached_pronoun
        standard_suffix
        y_verb_suffix
        verb_suffix
        residual_suffix
)

externals ( stem )

integers ( pV p1 p2 )

groupings ( v )

stringescapes {}

/* special characters (in ISO Latin I) */

stringdef a'   hex 'E1'  // a-acute
stringdef e'   hex 'E9'  // e-acute
stringdef i'   hex 'ED'  // i-acute
stringdef o'   hex 'F3'  // o-acute
stringdef u'   hex 'FA'  // u-acute
stringdef u"   hex 'FC'  // u-diaeresis
stringdef n~   hex 'F1'  // n-tilde

define v 'aeiou{a'}{e'}{i'}{o'}{u'}{u"}'

define mark_regions as (

   $pV = limit
   $p1 = limit
   $p2 = limit  // defaults

   do (
      ( v (non-v gopast v) or (v gopast non-v) )
      or
      ( non-v (non-v gopast v) or (v next) )
      setmark pV
   )
   do (
      gopast v gopast non-v setmark p1
      gopast v gopast non-v setmark p2
   )
)

define postlude as repeat (
   [substring] among(
      '{a'}' (<- 'a')
      '{e'}' (<- 'e')
      '{i'}' (<- 'i')
```

```
    '{o'}' (<- 'o')
    '{u'}' (<- 'u')
    // and possibly {u"}->u here, or in prelude
    ''    (next)
  ) //or next
)

backwardmode (

  define RV as $pV <= cursor
  define R1 as $p1 <= cursor
  define R2 as $p2 <= cursor

  define attached_pronoun as (
    [substring] among(
      'me' 'se'  'sela' 'selo' 'selas' 'selos' 'la' 'le' 'lo'
      'las' 'les' 'los' 'nos'
    )
    substring RV among(
      'i{e'}ndo' (] <- 'iendo')
      '{a'}ndo'  (] <- 'ando')
      '{a'}r'    (] <- 'ar')
      '{e'}r'    (] <- 'er')
      '{i'}r'    (] <- 'ir')
      'ando'
      'iendo'
      'ar' 'er' 'ir'
            (delete)
      'yendo'    ('u' delete)
    )
  )

  define standard_suffix as (
    [substring] among(

      'anza' 'anzas'
      'ico' 'ica' 'icos' 'icas'
      'ismo' 'ismos'
      'able' 'ables'
      'ible' 'ibles'
      'ista' 'istas'
      'oso' 'osa' 'osos' 'osas'
      'amiento' 'amientos'
      'imiento' 'imientos'
      (
        R2 delete
      )
      'adora' 'ador' 'aci{o'}n'
      'adoras' 'adores' 'aciones'
      'ante' 'antes' 'ancia' 'ancias'// Note 1
      (
        R2 delete
        try ( ['ic'] R2 delete )
      )
      'log{i'}a'
      'log{i'}as'
```

```
        (
            R2 <- 'log'
        )
        'uci{o'}n' 'uciones'
        (
            R2 <- 'u'
        )
        'encia' 'encias'
        (
            R2 <- 'ente'
        )
        'amente'
        (
            R1 delete
            try (
                [substring] R2 delete among(
                    'iv' (['at'] R2 delete)
                    'os'
                    'ic'
                    'ad'
                )
            )
        )
        'mente'
        (
            R2 delete
            try (
                [substring] among(
                    'ante' // Note 1
                    'able'
                    'ible' (R2 delete)
                )
            )
        )
        'idad'
        'idades'
        (
            R2 delete
            try (
                [substring] among(
                    'abil'
                    'ic'
                    'iv'   (R2 delete)
                )
            )
        )
        'iva' 'ivo'
        'ivas' 'ivos'
        (
            R2 delete
            try (
                ['at'] R2 delete // but not a further   ['ic'] R2 delete
            )
        )
    )
)
```

```
  define y_verb_suffix as (
     setlimit tomark pV for ([substring]) among(
        'ya' 'ye' 'yan' 'yen' 'yeron' 'yendo' 'yo' 'y{o'}'
        'yas' 'yes' 'yais' 'yamos'
           ('u' delete)
     )
  )

  define verb_suffix as (
     setlimit tomark pV for ([substring]) among(

        'en' 'es' '{e'}is' 'emos'
           (try ('u' test 'g') ] delete)

        'ar{i'}an' 'ar{i'}as' 'ar{a'}n' 'ar{a'}s' 'ar{i'}ais'
        'ar{i'}a' 'ar{e'}is' 'ar{i'}amos' 'aremos' 'ar{a'}'
        'ar{e'}'
        'er{i'}an' 'er{i'}as' 'er{a'}n' 'er{a'}s' 'er{i'}ais'
        'er{i'}a' 'er{e'}is' 'er{i'}amos' 'eremos' 'er{a'}'
        'er{e'}'
        'ir{i'}an' 'ir{i'}as' 'ir{a'}n' 'ir{a'}s' 'ir{i'}ais'
        'ir{i'}a' 'ir{e'}is' 'ir{i'}amos' 'iremos' 'ir{a'}'
        'ir{e'}'

        'aba' 'ada' 'ida' '{i'}a' 'ara' 'iera' 'ad' 'ed'
        'id' 'ase' 'iese' 'aste' 'iste' 'an' 'aban' '{i'}an'
        'aran' 'ieran' 'asen' 'iesen' 'aron' 'ieron' 'ado'
        'ido' 'ando' 'iendo' 'i{o'}' 'ar' 'er' 'ir' 'as'
        'abas' 'adas' 'idas' '{i'}as' 'aras' 'ieras' 'ases'
        'ieses' '{i'}s' '{a'}is' 'abais' '{i'}ais' 'arais'
        'ierais'  'aseis' 'ieseis' 'asteis' 'isteis' 'ados'
        'idos' 'amos' '{a'}bamos' '{i'}amos' 'imos'
        '{a'}ramos' 'i{e'}ramos' 'i{e'}semos' '{a'}semos'
           (delete)
     )
  )

  define residual_suffix as (
     [substring] among(
        'os'
        'a' 'o' '{a'}' '{i'}' '{o'}'
           ( RV delete )
        'e' '{e'}'
           ( RV delete try( ['u'] test 'g' RV delete ) )
     )
  )
)

define stem as (
  do mark_regions
  backwards (
     do attached_pronoun
     do ( standard_suffix or
          y_verb_suffix or
          verb_suffix
```

```
        )
      do residual_suffix
   )
   do postlude
)

/*
   Note 1: additions of 15 Jun 2005
*/
```

d) Algoritmo de Porter

Esta es una implementación en PHP de un lematizador de español. Está basado en el algoritmo de Martin Porter. Específicamente, está basado en el Spanish stemming algorithm de Snowball.

El algoritmo de Porter permite hacer stemming, esto es extraer los sufijos y prefijos comunes de palabras literalmente diferentes pero con una raíz común que pueden ser consideradas como un sólo término.

Al realizar una búsqueda, el usuario puede perderse porque las palabras están escritas de otra forma y con poca frecuencia en el documento.

Al aplicar stemming, se asegura que la forma de las palabras no penalice la frecuencia de estas.

Los algoritmos de stemming, o lematización para quienes hablamos español, más conocidos son: Lovins (1968), Porter (1980) y Paice (1990).

Todos eliminan "los finales" de las palabras en forma iterativa, y

requieren de una serie de pasos para llegar a la raíz, pero no requieren "a priori" conocer todas las posibles terminaciones.

Originalmente todos fueron hechos para el inglés, y se diferencian en la eficiencia del código y la elección de sufijos que identifican e eliminan.

Estudios no definitivos, indican que los resultados conseguidos con algoritmo de Lovins no concuerda con los resultados obtenidos con los otros dos que se mencionaron.

La raíz de la lematización es un concepto distinto del de la lingüística (orígen de las palabras) y no aporta al objetivo que persigue la lematización.

No hay razón teórica para que los algoritmos de lematización no puedan quitar también los prefijos (in, ante, anti, etc.), pero la mayor parte de los métodos de stemmer sólo quitan sufijos.

La razón puede ser decidir cuándo es un prefijo y no parte de la raíz (indispensable, introducción, etc.) o porque se puede quitar el significado de la palabra.

El problema se extiende a cuando es un sufijo y no parte de la palabra. Esto se resuelve fijando un mínimo de letras aceptables para la raíz y

con apoyo de una lista de palabras exentas de la aplicación de la regla.

Adicionalmente, hay reglas que indican cuándo un sufijo no debe eliminarse.

Otro problema es el cambio de raíz en algunas palabras, por ejemplo, en plural (repite, repetidos), donde en castellano el problema es mayor.

Los métodos de lematización son dependientes del idioma.

Tienen la ventaja que permiten la reducción de los índices, lo que aumenta la velocidad de procesamiento.

El algoritmo de Porter tiene la ventaja de ir quitando sufijos por etapas, en cambio Lovins requiere de la definición de todas las posibles combinaciones de sufijos.

El algoritmo de Porter se publicó en 1980. Básicamente lee un archivo, toma una serie de caracteres, y de esa serie, una palabra; luego valida que todos los caracteres de la palabra sean letras y finalmente aplica la lematización.

El lematizador hace pasar la palabra por varios conjuntos de reglas, cada conjunto formado por "n" reglas y cada regla está constituida por:

2. un identificador de la regla
3. Un sufijo a identificar
4. El texto por el que se reemplaza el sufijo
5. El tamaño del sufijo
6. El tamaño del texto de reemplazo
7. El tamaño mínimo que debe tener la raíz resultante luego de aplicar la regla (para no procesar palabras demasiado pequeñas).
8. Una función de validación (verifica si se debe aplicar la función una vez encontrado el sufijo)

Cuando ya no queden más conjuntos de reglas por aplicar, se devuelve la palabra resultante y se imprime.

Para traducir el algoritmo de Porter al español, se debe:

1. Ubicar los sufijos que ocurren frecuentemente en español.
2. Identificar los sufijos que ocurren juntos.
3. Establecer el orden en que ocurren

Para la selección de los grupos y orden de procesamiento, se deben tener

en cuenta:

1. Dos sufijos que ocurren juntos no pueden pertenecer al mismo conjunto.
2. Las reglas que quiten sufijos más al final de cada palabra deben ser

   procesados en un paso anterior a los que quitan otros.

3. Si un sufimo aparece siempre que ocurra otro, este sufijo es condicional a la aparición del anterior.

Hay además reglas propias del castellano. Por ejemplo, el sufijo "nos", que NO ES sufijo en palabras como campesinos, casinos, caminos, etc.; pero sí en hacernos, ponernos, presentarnos, etc.

Para depurar el algoritmo hay que considerar 3 pasos:

1. Las palabras terminadas en "r", conceptualmente similares, suelen quedar con distinta raíz, como en los verbos. Por ejemplo, caminar y caminando. Primero se debe eliminar "ndo". Por lo que la eliminación de las "r" es uno de los últimos pasos.
2. Similarmente, las palabras que terminan con vocales, por ejemplo, las palabras terminación y terminal y/o terminó, se dejan para el final.
3. En último término, se aplica una tercera regla que elimina las tildes de la raíz resultante. Por ejemplo, en diálogo y dialogó.

- Código

```
<?php
  /**
  * Copyright (c) 2005 Richard Heyes (http://www.phpguru.org/)
  *
  * All rights reserved.
  *
  * This script is free software.
  */

  /**
  * PHP5 Implementation of the Porter Stemmer algorithm. Certain elements
  * were borrowed from the (broken) implementation by Jon Abernathy.
  *
  * Usage:
  *
  *  $stem = PorterStemmer::Stem($word);
  *
  * How easy is that?
  */

  class PorterStemmer
  {
    /**
    * Regex for matching a consonant
    * @var string
    */
    private static $regex_consonant = '(?:[bcdfghjklmnpqrstvwxz]|(?<=[aeiou])y|^y)';


    /**
    * Regex for matching a vowel
    * @var string
    */
    private static $regex_vowel = '(?:[aeiou]|(?<![aeiou])y)';


    /**
    * Stems a word. Simple huh?
    *
    * @param  string $word Word to stem
    * @return string       Stemmed word
    */
    public static function Stem($word)
    {
      if (strlen($word) <= 2) {
        return $word;
      }

      $word = self::step1ab($word);
      $word = self::step1c($word);
      $word = self::step2($word);
      $word = self::step3($word);
```

```
        $word = self::step4($word);
        $word = self::step5($word);

        return $word;
    }


    /**
    * Step 1
    */
    private static function step1ab($word)
    {
        // Part a
        if (substr($word, -1) == 's') {

            self::replace($word, 'sses', 'ss')
          OR self::replace($word, 'ies', 'i')
          OR self::replace($word, 'ss', 'ss')
          OR self::replace($word, 's', '');
        }

        // Part b
        if (substr($word, -2, 1) != 'e' OR !self::replace($word, 'eed', 'ee', 0)) { // First rule
            $v = self::$regex_vowel;

            // ing and ed
            if (   preg_match("#$v+#", substr($word, 0, -3)) && self::replace($word, 'ing', '')
                OR preg_match("#$v+#", substr($word, 0, -2)) && self::replace($word, 'ed', '')) { // Note use of
&& and OR, for precedence reasons

                // If one of above two test successful
                if (    !self::replace($word, 'at', 'ate')
                    AND !self::replace($word, 'bl', 'ble')
                    AND !self::replace($word, 'iz', 'ize')) {

                    // Double consonant ending
                    if (    self::doubleConsonant($word)
                        AND substr($word, -2) != 'll'
                        AND substr($word, -2) != 'ss'
                        AND substr($word, -2) != 'zz') {

                        $word = substr($word, 0, -1);

                    } else if (self::m($word) == 1 AND self::cvc($word)) {
                        $word .= 'e';
                    }
                }
            }
        }

        return $word;
    }


    /**
    * Step 1c
```

```
     *
     * @param string $word Word to stem
     */
    private static function step1c($word)
    {
        $v = self::$regex_vowel;

        if (substr($word, -1) == 'y' && preg_match("#$v+#", substr($word, 0, -1))) {
            self::replace($word, 'y', 'i');
        }

        return $word;
    }


    /**
     * Step 2
     *
     * @param string $word Word to stem
     */
    private static function step2($word)
    {
        switch (substr($word, -2, 1)) {
            case 'a':
                 self::replace($word, 'ational', 'ate', 0)
              OR self::replace($word, 'tional', 'tion', 0);
                break;

            case 'c':
                 self::replace($word, 'enci', 'ence', 0)
              OR self::replace($word, 'anci', 'ance', 0);
                break;

            case 'e':
                self::replace($word, 'izer', 'ize', 0);
                break;

            case 'g':
                self::replace($word, 'logi', 'log', 0);
                break;

            case 'l':
                 self::replace($word, 'entli', 'ent', 0)
              OR self::replace($word, 'ousli', 'ous', 0)
              OR self::replace($word, 'alli', 'al', 0)
              OR self::replace($word, 'bli', 'ble', 0)
              OR self::replace($word, 'eli', 'e', 0);
                break;

            case 'o':
                 self::replace($word, 'ization', 'ize', 0)
              OR self::replace($word, 'ation', 'ate', 0)
              OR self::replace($word, 'ator', 'ate', 0);
                break;

            case 's':
```

```
            self::replace($word, 'iveness', 'ive', 0)
          OR self::replace($word, 'fulness', 'ful', 0)
          OR self::replace($word, 'ousness', 'ous', 0)
          OR self::replace($word, 'alism', 'al', 0);
          break;

      case 't':
            self::replace($word, 'biliti', 'ble', 0)
          OR self::replace($word, 'aliti', 'al', 0)
          OR self::replace($word, 'iviti', 'ive', 0);
          break;
    }

    return $word;
}


/**
* Step 3
*
* @param string $word String to stem
*/
private static function step3($word)
{
    switch (substr($word, -2, 1)) {
      case 'a':
          self::replace($word, 'ical', 'ic', 0);
          break;

      case 's':
          self::replace($word, 'ness', '', 0);
          break;

      case 't':
            self::replace($word, 'icate', 'ic', 0)
          OR self::replace($word, 'iciti', 'ic', 0);
          break;

      case 'u':
          self::replace($word, 'ful', '', 0);
          break;

      case 'v':
          self::replace($word, 'ative', '', 0);
          break;

      case 'z':
          self::replace($word, 'alize', 'al', 0);
          break;
    }

    return $word;
}


/**
```

```
     * Step 4
     *
     * @param string $word Word to stem
     */
    private static function step4($word)
    {
        switch (substr($word, -2, 1)) {
            case 'a':
                self::replace($word, 'al', '', 1);
                break;

            case 'c':
                 self::replace($word, 'ance', '', 1)
                OR self::replace($word, 'ence', '', 1);
                break;

            case 'e':
                self::replace($word, 'er', '', 1);
                break;

            case 'i':
                self::replace($word, 'ic', '', 1);
                break;

            case 'l':
                 self::replace($word, 'able', '', 1)
                OR self::replace($word, 'ible', '', 1);
                break;

            case 'n':
                 self::replace($word, 'ant', '', 1)
                OR self::replace($word, 'ement', '', 1)
                OR self::replace($word, 'ment', '', 1)
                OR self::replace($word, 'ent', '', 1);
                break;

            case 'o':
                if (substr($word, -4) == 'tion' OR substr($word, -4) == 'sion') {
                    self::replace($word, 'ion', '', 1);
                } else {
                     self::replace($word, 'ou', '', 1);
                }
                break;

            case 's':
                self::replace($word, 'ism', '', 1);
                break;

            case 't':
                 self::replace($word, 'ate', '', 1)
                OR self::replace($word, 'iti', '', 1);
                break;

            case 'u':
                self::replace($word, 'ous', '', 1);
                break;
```

```
            case 'v':
                self::replace($word, 'ive', '', 1);
                break;

            case 'z':
                self::replace($word, 'ize', '', 1);
                break;
        }

        return $word;
    }


    /**
    * Step 5
    *
    * @param string $word Word to stem
    */
    private static function step5($word)
    {
        // Part a
        if (substr($word, -1) == 'e') {
            if (self::m(substr($word, 0, -1)) > 1) {
                self::replace($word, 'e', '');

            } else if (self::m(substr($word, 0, -1)) == 1) {

                if (!self::cvc(substr($word, 0, -1))) {
                    self::replace($word, 'e', '');
                }
            }
        }

        // Part b
        if (self::m($word) > 1 AND self::doubleConsonant($word) AND substr($word, -1) == 'l') {
            $word = substr($word, 0, -1);
        }

        return $word;
    }


    /**
    * Replaces the first string with the second, at the end of the string. If third
    * arg is given, then the preceding string must match that m count at least.
    *
    * @param  string $str   String to check
    * @param  string $check Ending to check for
    * @param  string $repl  Replacement string
    * @param  int    $m     Optional minimum number of m() to meet
    * @return bool          Whether the $check string was at the end
    *                       of the $str string. True does not necessarily mean
    *                       that it was replaced.
    */
    private static function replace(&$str, $check, $repl, $m = null)
```

```
{
    $len = 0 - strlen($check);

    if (substr($str, $len) == $check) {
        $substr = substr($str, 0, $len);
        if (is_null($m) OR self::m($substr) > $m) {
            $str = $substr . $repl;
        }

        return true;
    }

    return false;
}


/**
* What, you mean it's not obvious from the name?
*
* m() measures the number of consonant sequences in $str. if c is
* a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
* presence,
*
* <c><v>       gives 0
* <c>vc<v>     gives 1
* <c>vcvc<v>   gives 2
* <c>vcvcvc<v> gives 3
*
* @param  string $str The string to return the m count for
* @return int         The m count
*/
private static function m($str)
{
    $c = self::$regex_consonant;
    $v = self::$regex_vowel;

    $str = preg_replace("#^$c+#", '', $str);
    $str = preg_replace("#$v+$#", '', $str);

    preg_match_all("#($v+$c+)#", $str, $matches);

    return count($matches[1]);
}


/**
* Returns true/false as to whether the given string contains two
* of the same consonant next to each other at the end of the string.
*
* @param  string $str String to check
* @return bool        Result
*/
private static function doubleConsonant($str)
{
    $c = self::$regex_consonant;
```

```
        return preg_match("#$c{2}$#", $str, $matches) AND $matches[0]{0} == $matches[0]{1};
    }


    /**
    * Checks for ending CVC sequence where second C is not W, X or Y
    *
    * @param  string $str String to check
    * @return bool       Result
    */
    private static function cvc($str)
    {
        $c = self::$regex_consonant;
        $v = self::$regex_vowel;

        return     preg_match("#($c$v$c)$#", $str, $matches)
            AND strlen($matches[1]) == 3
            AND $matches[1]{2} != 'w'
            AND $matches[1]{2} != 'x'
            AND $matches[1]{2} != 'y';
    }
  }
?>
```

- **Cómo utilizarlo**

Descarga la librería de Sourceforge:

http://sourceforge.net/project/showfiles.php?group_id=199822.

Incluye el archivo stemm_es.php en tu php e invoca al método stemm. Este método es estático, por lo que la llamada sería algo así:

```
$result = stemm_es::stemm('canciones');
```

e) Lematizador R

Los lingüistas computacionales de la hispanidad entera: ¿no son capaces de liberar una herramienta de lematización medianamente decente que podamos usar los demás? Lo más parecido a esa herramienta aparentemente inexistente Grampal, que funciona a través de una interfaz *web* y mediante el lenguaje de programación **R.**

El código es el siguiente:

```
require( XML )

lematiza <- function( frase ){
        palabra <- gsub( " ", "+", frase )
        base.url <- paste(
    "http://cartago.lllf.uam.es/grampal/grampal.cgi?m=etiqueta&e=",
    palabra, sep = "" )
        tmp <- readLines( base.url, encoding = 'utf8' )
        tmp <- iconv( tmp, "utf-8" )
        tmp <- gsub( " ", " ", tmp )
        tmp <- readHTMLTable( tmp )
        tmp <- as.character( tmp[[1]]$V3 )
        tmp <- do.call( rbind, strsplit( tmp, " " ) )[,4]
        tmp
}
```

Y las pruebas ejecutadas son:

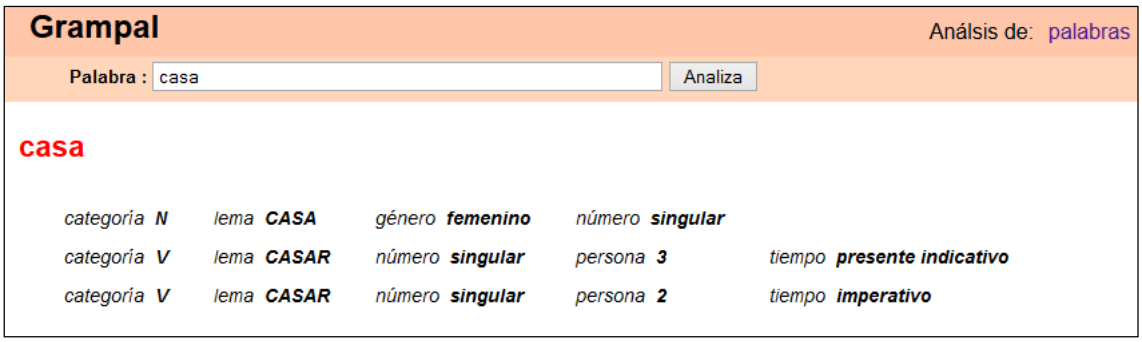
```
> lematiza( "des" )
[1] "DAR"
> lematiza( "anduve" )
[1] "ANDAR"
> lematiza( "casitas" )
[1] "CASITA"
> lematiza( "comimos" )
[1] "COMER"
> lematiza( "queremos comer patatas" )
[1] "QUERER" "COMER"  "PATATA"
```

f) Grampal

Grampal, es un lematizador para el español desarrollado en la Universidad Autónoma de Madrid. Se desconoce la forma de procesamiento de grampal (si es base de datos, o software). Una de las ventajas es que es un recurso online y cuenta con una gran cantidad de palabras, aunque no especifica claramente cuantas palabras lematizadas son. El recurso de grampal, tiene la ventaja que puede ser programado bajo un lenguaje de programación (R por lo general). La dirección web de la página es http://cartago.lllf.uam.es/grampal/grampal.cgi?m=etiquetario.

- **Grampal palabras**

Grampla, no solo lematiza, si no que hace una clasificación morfológica de la palabra, que permitiría hacer una desambiguación en aplicaciones de procesamiento de lenguaje natural. A continuación se ve un ejemplo de una consulta hacia una palabra.

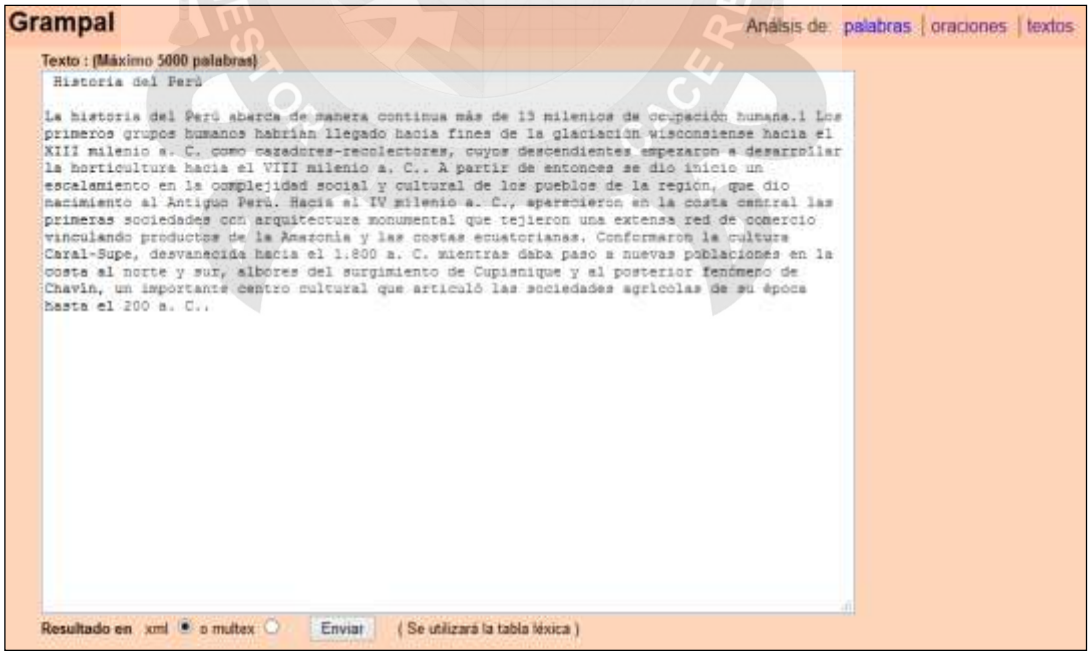


- **Grampal oraciones**

Una de las ventajas de grampal, es que permite hacer lematizacion de oraciones; ahorrando tiempo en las consultas, tarea que resulta más fácil al hacer palabra por palabra.

- **Grampal Textos**

Permite hacer consultas de hasta un máximo de 5000 palabras, el cual facilita si se desea lematizar documentos completos.



El resultado de las consultas puede ser en archivo xql descargable o multex, sin dejar de etiquetar morfológicamente.

- **Grampal etiquetas**

CATEGORÍAS SINTÁCTICAS:				
Sustantivo	N	nombre	*gato*	
	NPR	nombre propio	*María*	
Adjetivo	ADJ		*azul*	
Verbo	V		*cantar*	
Auxiliar	AUX		*habría*	
Pronombre	P		*yo, ahora*	
	REL	relativo	*que*	
	PINT	interrogativo	*qué, cómo*	
Determinante	ART	artículo	*el*	
	POSS	posesivo	*su*	
	DEM	demostrativo	*ese*	
Cuantificador	Q		*uno*	cardinal
			primer	ordinal
			muchos	indefinido
Preposición	PREP		*en*	
Adverbio	ADV		*así*	
Conjunción	C		*y*	
Interjección	INTJ		*ole*	
Marcador discursivo	MD		*oye*	
Desconocida	UNKN		*xx*	
RASGOS MORFOLÓGICOS:				
Genero	masc	masculino	*gato*	
	fem	femenino	*gata*	
Número	sing	singular	*lápiz*	
	plu	plural	*lápices*	
Persona	1	primera	*amo*	
	2	segunda	*amas*	
	3	tercera	*ama*	
Tiempo	inf	infinitivo	*amar*	
	ger	gerundio	*amando*	
	part	participio	*amado*	
	pres_ind	presente indicativo	*amo*	

	indf_ind	pretérito perfecto	*amé*	
	impf_ind	pretérito imperfecto	*amaba*	
	fut_ind	futuro indicativo	*amaré*	
	pres_subj	presente subjuntivo	*ame*	
	impf_subj	imperfecto subjuntivo	*amara*	
	cond	condicional	*amaría*	
	imper	imperativo	*ama*	

g) Base de datos lematizados IPN

Autores y desarrolladores: Grigori Sidorov, Alexander Gelbukh , Francisco Velázquez, Liliana Chanona. Una lista de palabras completa (versión beta) generados con este sistema está disponible. Este es un programa que realiza la lematización y proporciona información gramatical de cada forma de palabra en la oración. Véase la descripción detallada a continuación. Utilizan BDE para el acceso de datos, por lo tanto, en el futuro, se mejorará el rendimiento.

Tamaño del diccionario se encuentra cerca de 25.000 palabras en la cabeza.

El sistema contiene un archivo de texto "complex.dic" con palabras compuestas (a_partir_de, etc.). El mismo archivo se puede utilizar como un diccionario de usuario para palabras sueltas (aunque nosotros no lo recomendamos).

El archive a descargar la lista de palabras es:

http://www.cic.ipn.mx/~sidorov/agme/generate_beta.zip

```
abalanz -son V0SF3S0 abalanzar
abalanz-Aremos V0SF1P0 abalanzar
abalanz-areis V0SF2P0 abalanzar
abalanz-aren V0SF3P0 abalanzar
abalanz -a V0R02S0 abalanzar
abalánz-ame [me] V0R02S0 abalanzar
abalánz -ate [te] V0R02S0 abalanzar
...
abrazadera - NCFS000 abrazadera
abrazadera -s NCFP000 abrazadera
abrazo - NCMS000 abrazo
abrazo -s NCMP000 abrazo
...
aerospacial - AGIS000 aerospacial
aerospacial-es AGIP000 aerospacial
```

h) Otros lematizadores

- **Stemmer Lovins**

El algoritmo derivado Lovins se presentó por primera vez en 1968 por Julie Bet Lovins. Se trata de una sola pasada, Stemmer sensible al contexto, que elimina las terminaciones basadas en el principio de más larga partido. La despalilladora fue el primero en ser publicado y estaba extremadamente bien desarrollado teniendo en cuenta la fecha de su lanzamiento y ha sido la principal influencia en una gran cantidad de la labor futura en la zona

La despalilladora se considera complejo para su edad y utiliza una lista grande (297) de las terminaciones, cada uno de los cuales está asociado con uno de una serie de restricciones contextuales cualitativos que impiden la eliminación de terminaciones en ciertas circunstancias. La despalilladora utiliza una serie de reglas que están diseñados para hacer frente a las excepciones más comunes. Todas las terminaciones se asocian con la excepción predeterminado, que una madre debe ser de al menos dos largas cartas, que está diseñado para evitar la producción de tallos ambigua. Otras reglas afirman una de las siguientes condiciones en la eliminación de la final,

- El aumento de la longitud mínima de una madre después de la eliminación de un final.
- La prevención de la eliminación de terminaciones cuando ciertas cartas están presentes en el tallo restante.
- Las combinaciones de las restricciones anteriores.

En el desarrollo de la despalilladora Lovins describe la forma más deseable de contexto sensible como una regla que se puede generalizar a aplicar en numerosas situaciones. Durante el desarrollo de la despalilladora se descubrió que algunos ejemplos de tales reglas podrían ser encontrados. Para cada uno terminando una serie de casos especiales existen que causa errónea se deriva a ser producido, estos son a menudo únicos para el final y un número de reglas tendrían que ser desarrollado que impida sólo un

pequeño número de errores. Este proceso requeriría grandes cantidades de tiempo y de datos, y ofrecería decrecientes mejoras en el rendimiento con el tiempo. Por esta razón se decidió tratar con las excepciones más obvias y limitar espero que el número de errores que permanecen en paradero desconocido en la lista de excepciones.

El diagrama de flujo que detalla el algoritmo completo propuesto por Lovins e ilustra las dos etapas principales del algoritmo. La fase derivada se ha discutido anteriormente e incluye la eliminación de las terminaciones y la prueba de excepciones asociadas entre otras medidas. La segunda parte del algoritmo es la fase de recodificación.

La excepción ortografía término se utiliza para cubrir todas las circunstancias en las que una madre puede estar escrito en más de una forma. La mayoría de estas excepciones que se producen en Inglés se deben a "derivaciones latinas", tales como la matriz y matrices. Otros tipos de excepciones que se producen se puede atribuir a las diferencias en la ortografía británica y estadounidense, como analiza y analiza, o de las reglas básicas de inflexión que provocan la duplicación de ciertas consonantes cuando se añade un sufijo. Dos formas son propuestos por Lovins para hacer frente a este problema, que se llama la recodificación y la congruencia parcial. El documento original se explica el proceso de decisión que

llevó a la implementación de una fase de recodificación en el algoritmo.

- **Stemmer Paice/Husk**

Stemmer Paice / Husk fue publicado por primera vez en 1990 y fue desarrollado por Chris Paice con la asistencia de Gareth cáscara. Es un stemmer iterativo basado fusión, aunque restante eficiente y fácil de implementar, se sabe que es muy fuerte y agresivo Utiliza una sola tabla de reglas, cada uno de los cuales puede especificar la eliminación o sustitución de un final. Esta técnica de reemplazo se utiliza para evitar las excepciones problema de ortografía como se ha descrito anteriormente, mediante la sustitución de las terminaciones más que la simple eliminación de ellos la despalilladora logra prescindir de una etapa separada en el proceso derivado, es decir, no se requiere recodificación o coincidencia parcial. Esto ayuda a mantener la eficiencia del algoritmo, mientras que sigue siendo eficaz. Las reglas son indexados por la última letra del final para permitir la búsqueda eficiente y son de la siguiente forma;

- Un final de uno o más caracteres, que tuvo lugar en orden inverso
- Una bandera intacta opcional '*'
- Un dígito que especifica el total de eliminación (cero o más)
- Una cadena append opcional de uno o más caracteres

- Un símbolo de la continuación, '>' o '.'

El siguiente ejemplo demuestra cómo las reglas se almacenan y utilizan y cómo el reemplazo se puede utilizar para negar la necesidad de recodificación. La regla nois4j> provoca sión terminaciones para ser sustituidos por j. Esto actúa como un puntero a la sección J de las reglas, lo que lleva a la siguiente *transformación;. Provisión -> provij -> provid* El j-transformación se utiliza para asegurar que el suministro y proporcionar términos se combinó correctamente al provid tallo.

El algoritmo ha cuatro pasos principales se detallan a continuación, y se presenta en el diagrama de flujo:

1. Seleccione la sección correspondiente; Inspeccione la letra final de la palabra y, si está presente, considere la primera regla de la sección correspondiente de la tabla de reglas.
2. Compruebe la aplicabilidad de la regla; Si letras finales de plazo no se ajustan a la regla, o la configuración intactos se violan o las condiciones de aceptabilidad no están satisfechos ir a la etapa 4.
3. Aplica; Eliminar o poner fin a la reforma como se requiere y luego verifique símbolo de terminación, y ya sea terminar o volver a la etapa 1.
4. Mover a la siguiente regla en la tabla, si la letra sección ha cambiado luego terminar, otra cosa ir a la etapa 2.

CAPÍTULO IV

4. SOFTWARE

Se usamos dos:

- Mysql: Gestor de la base de datos.

http://www.mysql.com/

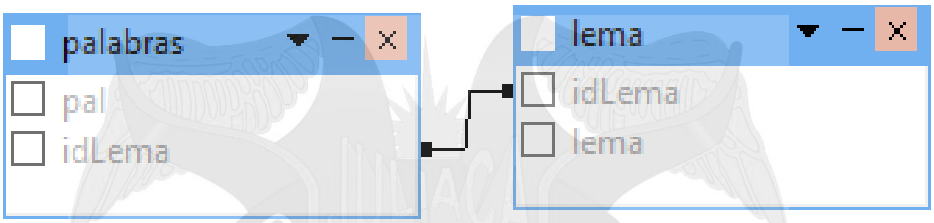
- Navicat: GUI para conexión a la base de datos.

http://navicat.com/download

4.1.1. Creación de la base de datos

La base de datos consta de 2 tablas, inicialmente; durante el proceso se genera tablas temporales y se eliminan de acuerdo a las necesidades del proceso.

- Tabla Palabras: Contiene las palabras (cualquiera que luego se ingresara su lema)
- Tabla Lema: El lema o stemmer de la palabra.



Código SQL de la tabla lema

```
CREATE TABLE `lema` (
`idLema` BIGINT(20) NOT NULL AUTO_INCREMENT,
`lema` VARCHAR(255) NOT NULL COLLATE 'utf8_bin',
PRIMARY KEY (`idLema`),
UNIQUE INDEX `lema` (`lema`)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB;
```

Código SQL de la tabla palabra

```
CREATE TABLE `palabras` (
`pal` VARCHAR(50) NOT NULL DEFAULT '' COLLATE 'utf8_bin',
`idLema` BIGINT(20) NOT NULL,
PRIMARY KEY (`pal`),
INDEX `FK_dicTesauro_lemaTesauro` (`idLema`),
CONSTRAINT `palabras_ibfk_1` FOREIGN KEY (`idLema`) REFERENCES `lema` (`idLema`) ON UPDATE
CASCADE ON DELETE CASCADE
)
COLLATE='utf8_bin'
ENGINE=InnoDB;
```

4.1.2. Creación de las Particiones

Creamos una partición de 20, para la tabla de **palabras**, debido a que buscaremos por la palabra indicada, note que no se pasa ninguna columna en específico, por default toma la columna de la llave primaria.

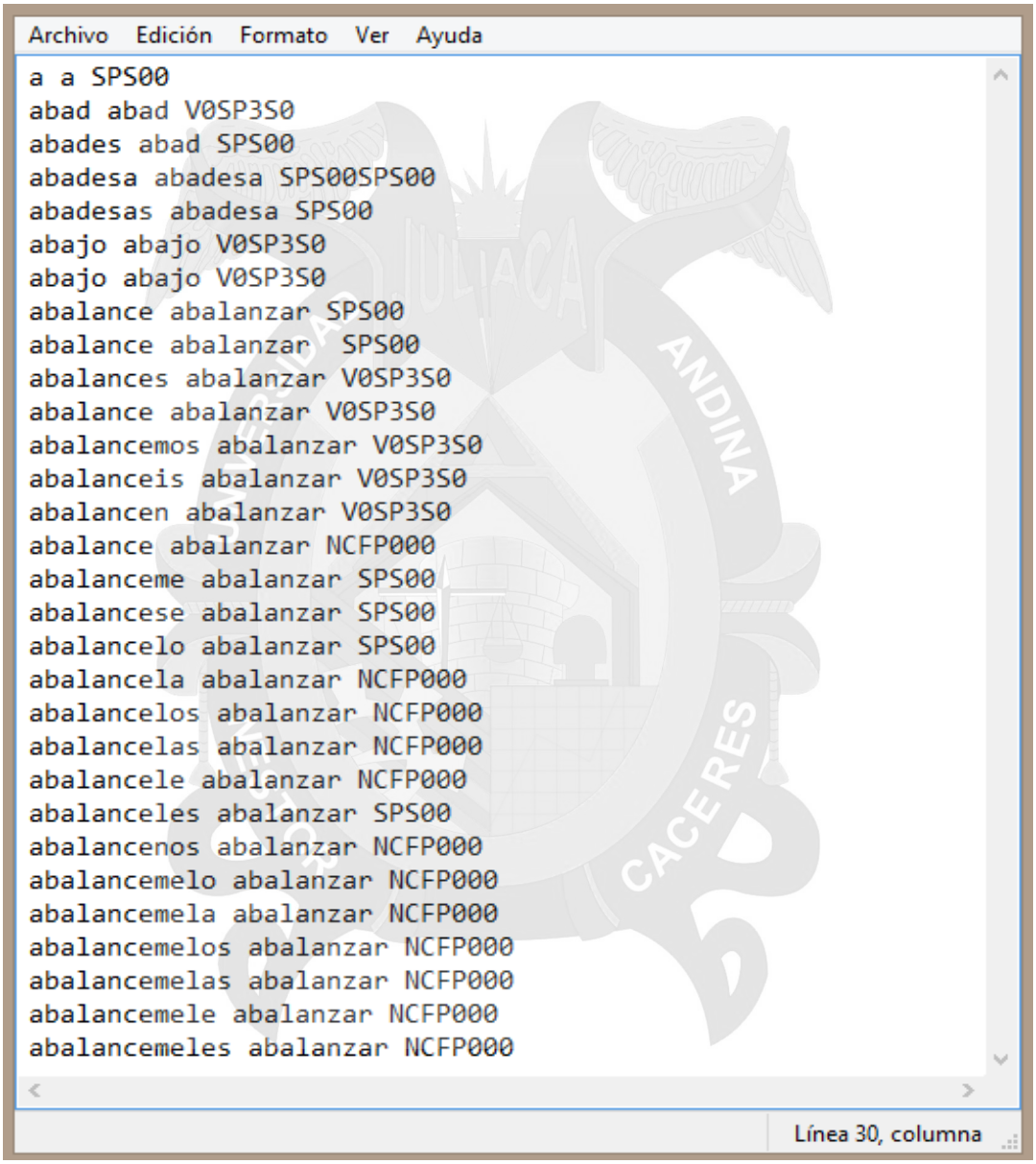
```
ALTER TABLE palabras
PARTITION BY KEY()
PARTITIONS 20;
```

Nota.- Debido a que la tabla lema, es muy considerado pequeño en cantidad de registro, no se implementó particiones.

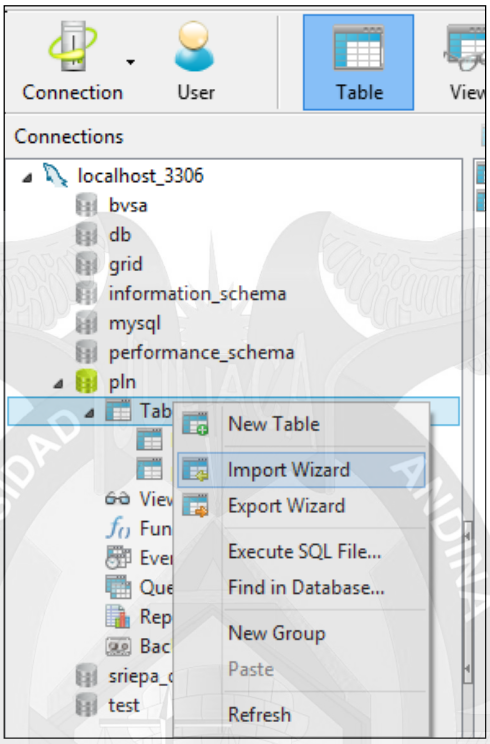
4.1.3. Ingreso de palabras a la base datos

Ingresamos desde la base lematizados del IPN a través de una importación de datos. Seguiremos los siguientes pasos.

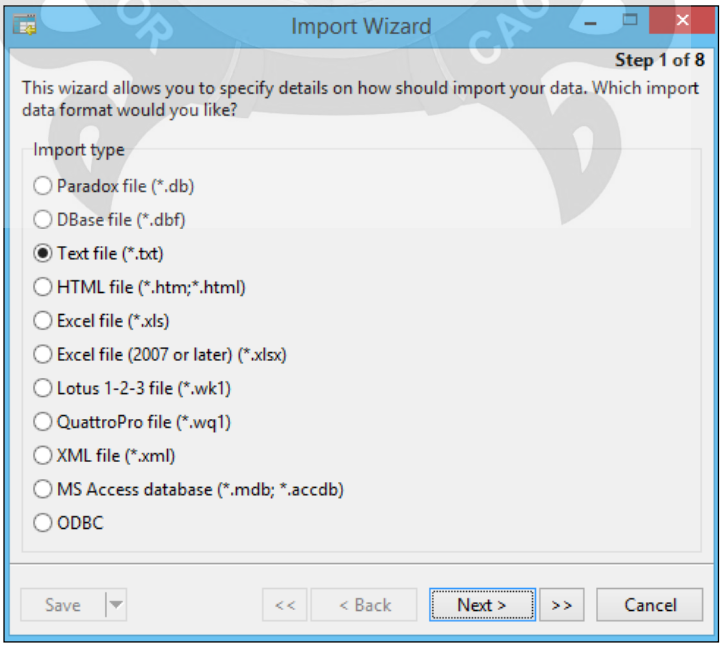
a) Revisamos la estructura del archivo, nótese que tiene 3 campos definidos, de los cuales solo importaremos la primera y la segunda columna:



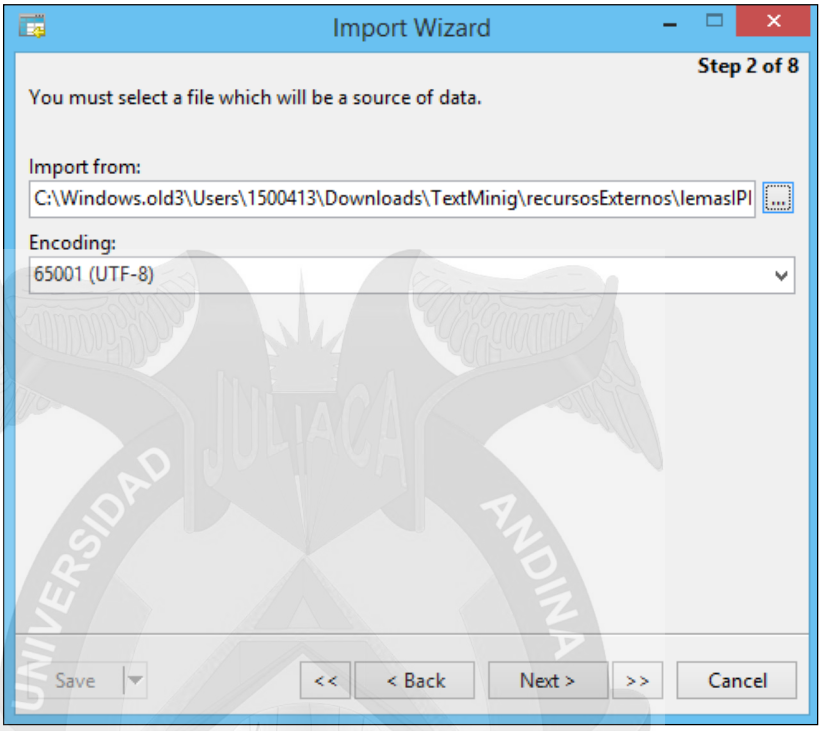
b) Desde navicat, hacemos la tarea de importar.



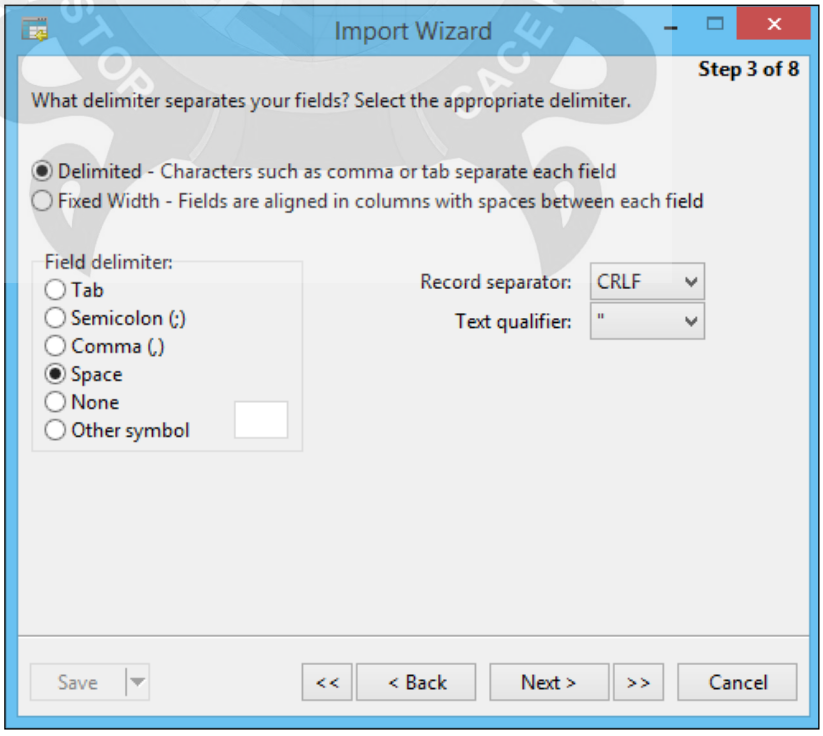
c) Escogemos la opción de archivo de texto.



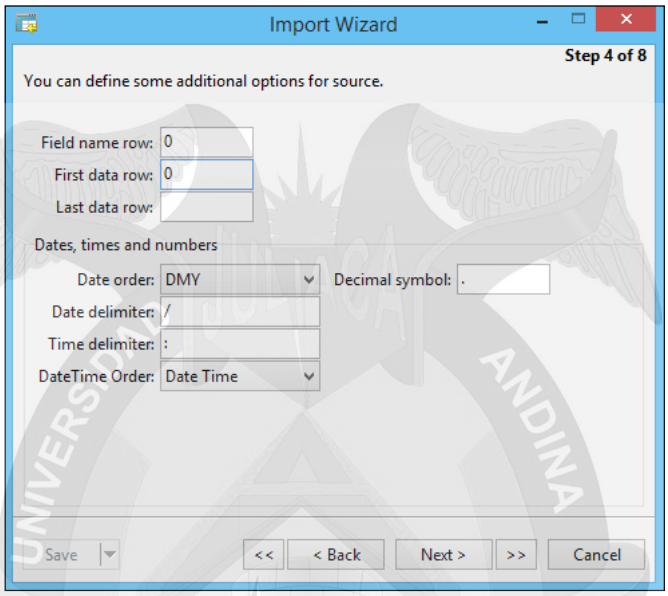
d) Seleccionamos el archivo y la codificación adecuada.



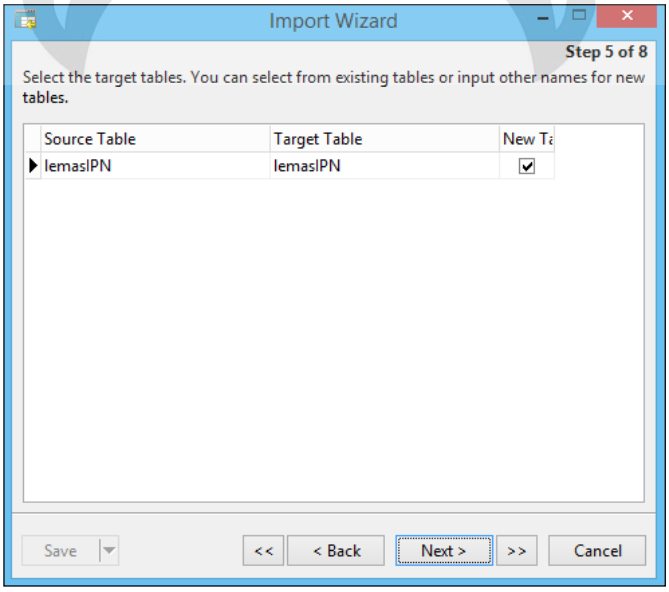
e) Seleccionamos el delimitador de campo (en el caso del archivo es espacio).



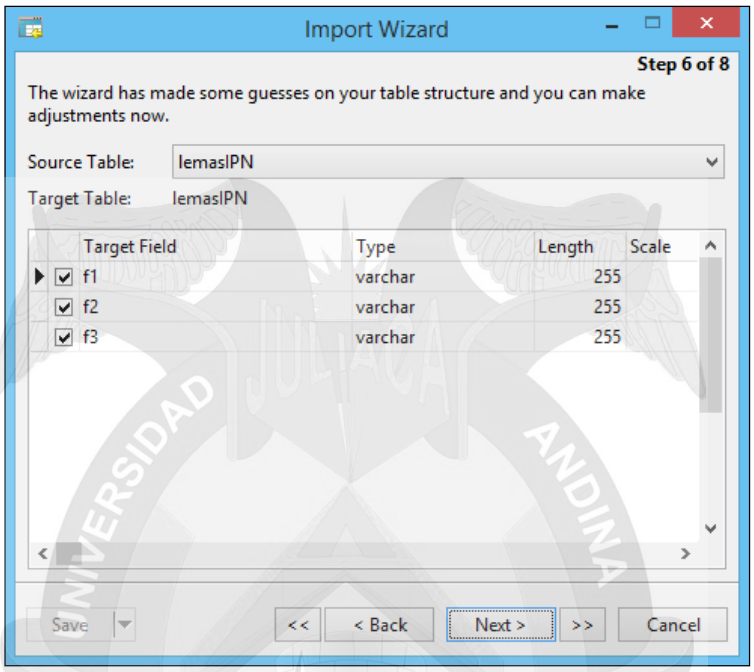
f) Debido a que nuestro archivo no tiene encabezado de columnas, escribimos cero en las opciones de field name y fisrt data row.



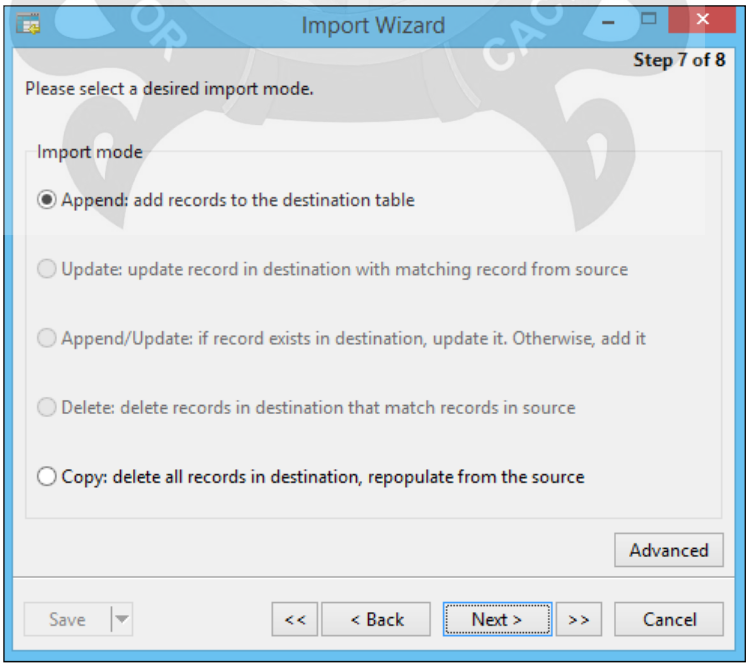
g) Importaremos en una nueva tabla temporal, debido a que posteriormente, seguiremos importando palabras desde el lematizador online Grampal, y necesitaremos identificar las nuevas palabras.



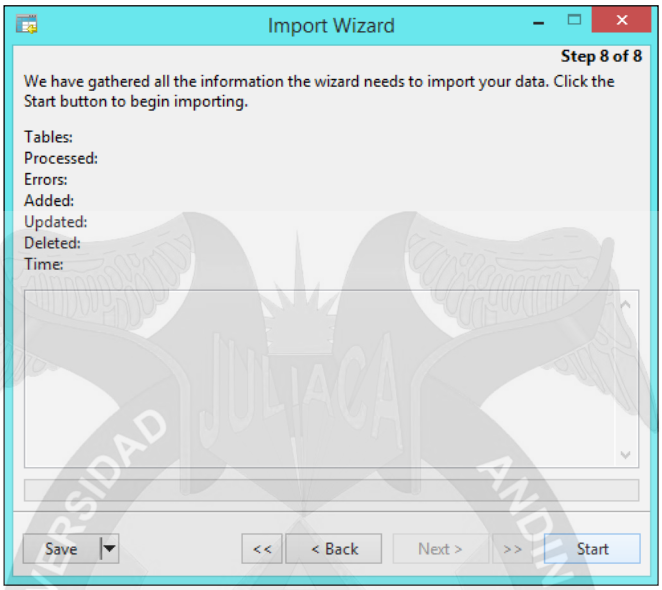
h) Como es una tabla temporal, lo dejaremos con los encabezados de f1,f2 y f3.,



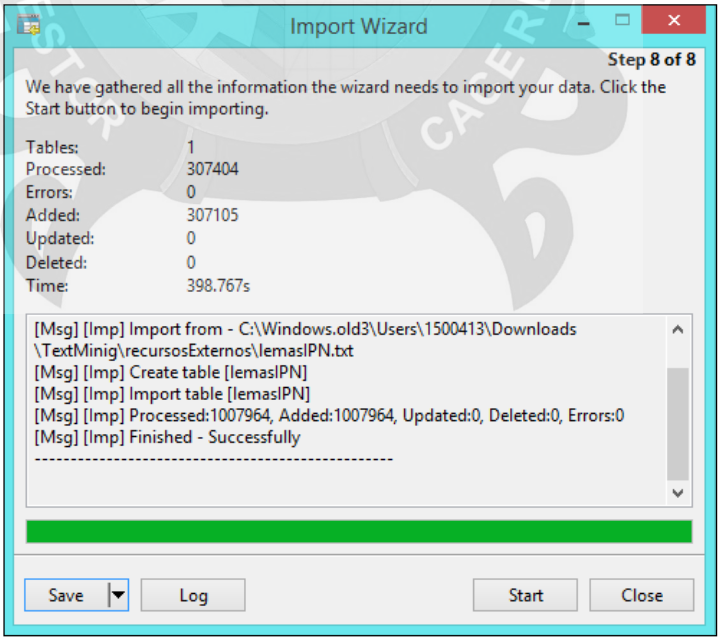
i) Como es una tabla temporal, no es necesario hacer append (anexo de registros), y dejamos por default las opciones.



j) Listo para empezar la importación, y procederemos a comenzar.



k) El progressbar nos dará indicación de cuando finalizo la importación, cabe .mencionar que el tiempo depende de las característica de la computadora.



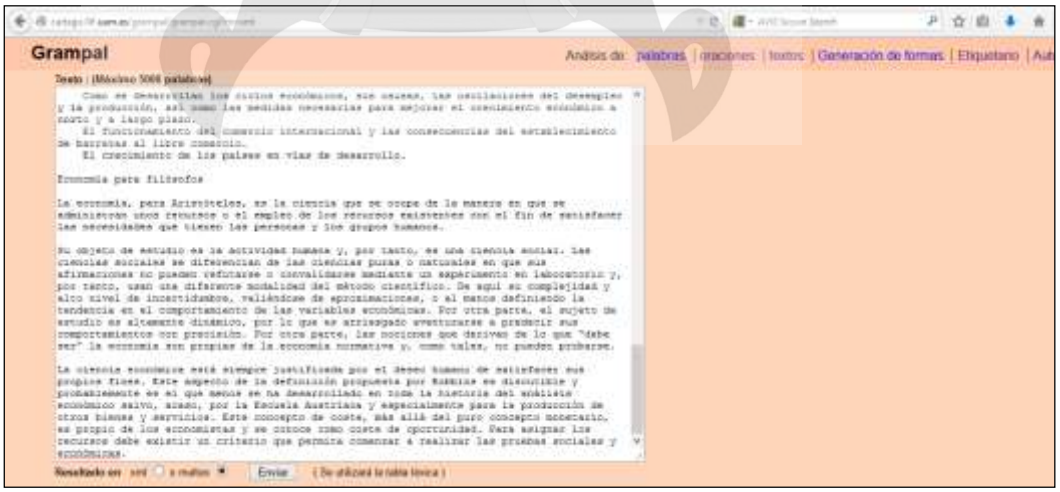
Nota.- Se obtuvieron 307105 registros, casi el 50% de uno de nuestros objetivos. Queda pendiente el ingreso hacia las tablas palabras y lema.

4.2. Desde lematizador online Grampal

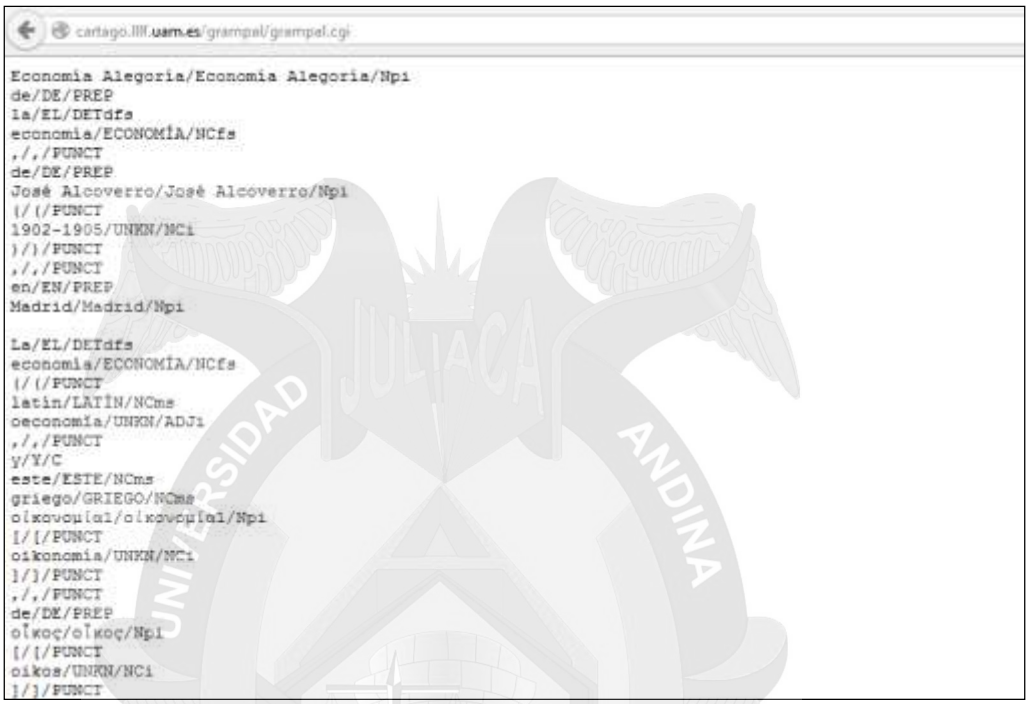
En este proceso, se extrae desde los recursos de internet, para el caso de uso hemos ingresado a archivos de wikipedia con diferentes temas (tales como economía, negocios, registros, entre otros).

a) Seleccionamos la información del recurso digital de internet

b) Copiamos, y pegamos dentro del lematizador de Grampal, en modo de textos, el modo de regreso será en la opción de multitex.



c) Una vez procesado la tematización, se guarda en un archivo de texto, y procedomos con los pasos del capítulo y subsección anterior.



4.3. Estadísticas de lematización Grampal

Debido a uno de los objetivos planteados, fue superar 60000 palabras de español, y por la base de datos IPN solo se obtuvo, se tiene 1006980. Cabe mencionar que la importación se hizo de un archivo general, del lematizador Grampal; debido a que los resultados se guardaban en un archivo de texto donde se anexaba las nuevas consultas.

Se realizo un total de 183 consultas con un límite de 5500 palabras (aunque el lemitazdor indica 5000, se puede superar esta hasta más de 6000, el dato exacto no se pudo ubicar.

N° Consultas	Promedio de palabras Consulta	Total Palabras Procesadas
183	5500	1006500

N° de Palabras Grampal	Palabras Totales en Base datos	Porcentaje de palabras Repetidas
69546	1006980	58.56%

a) Script de procesamiento en la Base de datos

Tomaremos como ejemplo base el archivo de la tabla temporal lema IPN.

1. Vaciamos los lemas hacia la tabla lema.

```
INSERT IGNORE INTO lema (lema, tipo ) (SELECT f3,f2 FROM lemasIPN);
```

Usamos la palabra ignore, para evitar si existe un lema repetida, la ejecución continúe, y no se detenga por alguna restricción adicionada.

2. Vaciamos las palabras a la tabla palabras.

```
INSERT IGNORE INTO palabras (idLema, pal) (SELECT idLema,f1 FROM lemasIPN, lema WHERE lemasipn.f3 = lema.lema );
```

Hacemos una unión de las tablas lemasIPN y lema, para vaciar a al tabla palabras, debido a que en la tabla palabras neceistamos el IdLema, y

lapalabra; Idlema lo tenemos en la tabla lema y la palabra lo tenemos en la tabla lemasIPN.

3. Borramos las tablas temporales

```
DROP TABLE lemasIPN;
```

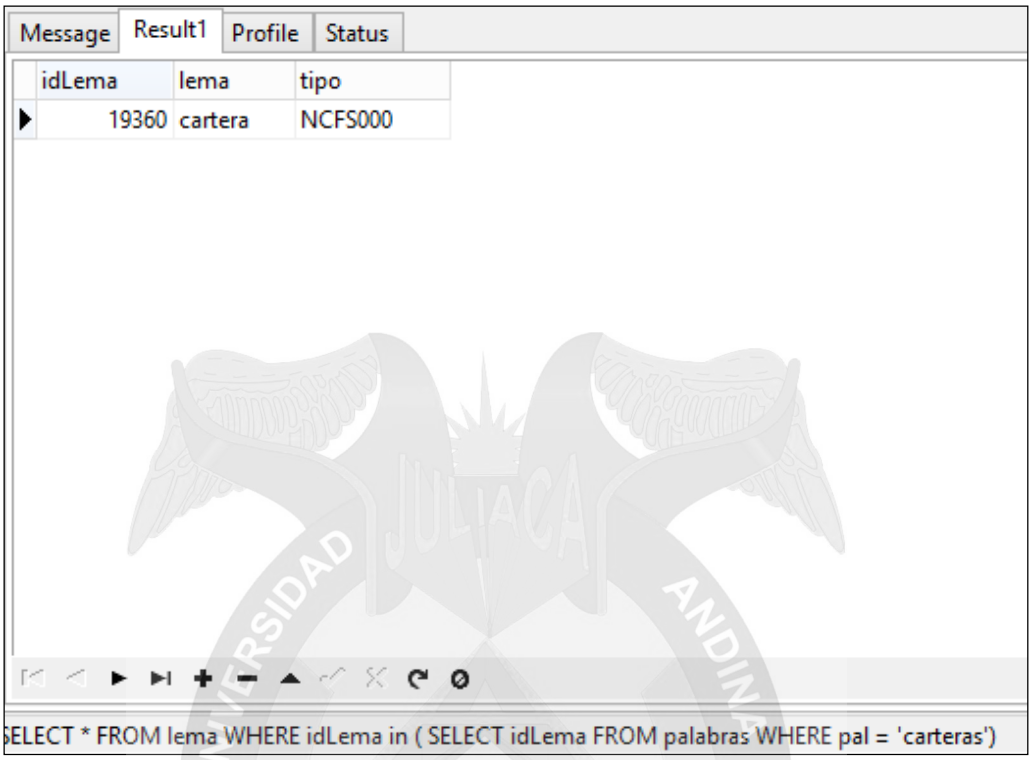
Recordemos, que lemasIPN fue creado como una tabla temporal, y para los palabras provenientes del lematizador grampal se creó con el nombre de lemaGrampal como tabla temporal.

4.4. Uso del Diccionario

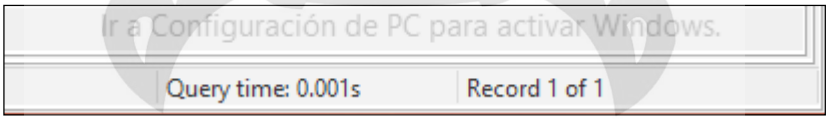
El uso del diccionario, es hacer la consulta de la palabra, y tiene que regresar el lema respectivo, se hace con la siguiente consulta SQL:

```
SELECT * FROM lema WHERE idLema in ( SELECT idLema FROM palabras WHERE pal = 'carteras');
```

En la consulta se puede reemplazar la palabra carteras, por la deseada en la búsqueda. El resultado obtenido es cartera, determinado a que carteras esta en plural y debería estar en singular la lematización. Una vez ejecutada la consulta, el resultado obtenido es:



Por otro lado, otro de los objetivos era la optimización de la consulta, teniendo en cuenta que contamos con más de 1000000 de registros, y se combina con la tabla de lemas.



Como se aprecia en la consulta, el tiempo estimado esta muy por debajo del objetivo secundario planteado, el cual es un indicativo; de la técnica de particiones (PARTITIONS), es efectivo para implementación de bases de datos de grandes cantidades de información.

a) Resumen de Base de datos

Tabla	N° Registros
Lema	69546
palabras	1006980

b) Pruebas Realizadas

N° de Pruebas	N° Palabras Respondidas
384	438

Las palabras que no fueron respondidas, pudiera ser motivo de ingreso manual a la base de datos, aunque la clasificación morfología, depende de los sistemas con una estructura ordenada. Las palabras no escritas correctamente, tales los que tienen acentos y/o diéresis, dieron un resultado nulo, puesto que el diccionario no reconoce palabras mal escritas.

CONCLUSIONES

PRIMERA: Se implementó un diccionario stemmer (lematizado), en español con una gran cantidad de palabras para el procesamiento de lenguaje natural.

SEGUNDA: Se llegó a lemtizar una gran cantidad de palabras llegando a más de 60000 lemas, se tuvo que ingresar una gran cantidad de palabras esto debido a las duplicidades que existía.

TERCERA: La optimización de la base de datos, utilizando el método de partición optimizo las consultas de manera extraordinaria, siendo el tiempo de respuesta muy por debajo del propuesto en los objetivos, y teniendo tolerancia a poder crecer la base de datos.

RECOMENDACIONES

PRIMERA: Al realizar la recolección de los datos, se debe tomar distintos temas para la variedad de las palabras lematizadas, y que exista la menor cantidad de duplicidades.

SEGUNDA: Se debe de tener en cuenta de que la lematizacion, se realiza solo para palabra no para símbolos por lo que se debe de realizar los filtrados de estos caracteres, caso contrario ocuparían espacio innecesario en nuestra base de datos.

TERCERA: Se puede usar el diccionario de la real academia de la lengua española para insertar las palabras lematizadas al 100%, aunque el trabajo seria extremadamente exigente.

CUARTA: Se debería continuar con la investigación pero orientado a implementar esta base de datos en los buscadores de las páginas web.

BIBLIOGRAFÍA

- [01] Alexander Gelbukh y Grigori Sidorov, PROCESAMIENTO AUTOMÁTICO DEL ESPAÑOL con Enfoque en Recursos Léxicos Grandes, Registro en la Biblioteca del Congreso de EE.UU.
- [02] Alejandro Peña Ayala, Lenguaje Natural: Descripción de las Etapas para su Tratamiento, Instituto Politécnico Nacional.
- [03] Analista de Sistemas Sergio Kourchenko Barrena, José Manuel Alarcón, Fundamentos de la Base de Datos SQL.
- [04] Arli Juan A. Martínez López, Algunos Problemas en la Lematización de las Unidades Fraseológicas, Universidad de Bergen y Universidad Noruega de Ciencias Empresariales.
- [05] Arturo Mora Rioja, Bases de Datos. Diseño y Gestion, Editorial Sintesis 2009.
- [06] David M. Kroenke. Procesamiento de Base de Datos, Premtice-Hall, Hispanoamerica, 1996.
- [07] Eva Gómez Ballester y Patricio Martínez Barco, Bases de Datos 1, Escuela Politécnica Superior Universidad de Alicante.
- [08] Henry F. Korth y S. Sudarsan Abraham Silberschatz, Fundamentos de Bases de Datos, Mc Graw Hill, 2002.
- [09] Introducción a los Sistemas de Bases de Datos, Instituto Tecnológico Autónomo de México.

- [10] Laudon Keneth C. y Laudon Jane P., Sistemas de Información Gerencial (8ª ed.) México, Pearson Educación 2004.
- [11] Maria Teresa Martin Valdivia y Luis Alfonso Ureña Lopez , Ejercicios de Bases de Datos Relacionales, Universidad de Jaen. Servicio de Publicaciones e Intercambio.
- [12] Mary E.S. Loomis, Estructuras de datos y Organizacion de archivos, Mc Graw Hill 2000.
- [13] Michael J. Folk, Estructuras de archivos. Un conjunto de herramientas conceptuales, Mc Graw Hill, 2002.
- [13] Ralph Kimball, Laua Reeves, Margy Ross, Warren, The Data WareHouse Lifecycle Toolkit Wiley Edition.
- [14] Raquel Gómez Díaz , La Lematizacion en Español: Una Aplicación para La Recuperación de Información, Editorial Trea 2005.
- [15] http://www.dwinfocenter.org/casefor.html
- [16] http://datawarehouse.ittoolbox.com/

ANEXO

1. REQUISITOS PARA LA BASE DE DATOS

El sistema de base de datos operacional MySQL es hoy en día uno de los más importantes en lo que hace al diseño y programación de base de datos de tipo relacional.

Requisitos mínimos de hardware y software para la instalación de MySQL Y Navicat

Requerimientos mínimos de instalación MySQL server versión 5.1	
Requerimiento	MySQL
RAM	512 MB
Memoria virtual	1024 MB
Espacio disco duro para instalación	1 GB
Tamaño máximo de la base de datos	Sin limite
Sistema Operativo: Windows , Linux	

Navicat es una herramienta de diseño de base de datos poderosos y rentables que ayuda a los usuarios a crear modelos de datos lógicos y físicos de alta-calidad. Es compatible con varios sistemas de bases de datos, como MySQL, Oracle, SQL Server, PostgreSQL y SQLite.

Requerimientos mímimos de instalación Navicat	
Requerimiento	Navicat
RAM	512 MB
Memoria virtual	1024 MB
Espacio disco duro para instalación	65 MB
Sistema Operativo: Windows	

2. Lematizador Grampal

Es un lematizador online, lematiza palabras y textos enteros con una capacidad de 5000 palabras por consultas.

Diseñadores de la pagina.

Para la lematizacion se elige la opción deseada si es por palaba o texto.