FACULTADE DE INFORMÁTICA

Departamento de Computación

Proxecto de Fin de Carreira de Enxeñería Informática

Sistema de minería de opiniones para el español utilizando métodos de análisis sintáctico de dependencias

Autor:

David Vilares Calvo

Directores:

Miguel Ángel Alonso Pardo

Carlos Gómez Rodríguez

Septiembre 2012

Título del proyecto:	Sistema de minería de opiniones para el español utilizando métodos de análisis sintáctico de dependencias
Clasificación del proyecto:	Investigación y Desarrollo
Autor:	David Vilares Calvo
Director:	Miguel Ángel Alonso Pardo
Director:	Carlos Gómez Rodríguez
Fecha:	

Tribunal

Presidente:

Vocal 1º:

Vocal 2º:

Vocal 3º:

Secretario:

Calificación:

A mis padres y a mi hermana

Agradecimientos

Son muchas las personas que, con su ayuda, han hecho posible que haya conseguido llegar hasta aquí.

En primer lugar quiero expresar mi agradecimiento a mis dos directores, Miguel y Carlos, por la atención que me han prestado durante estos meses, haciendo mi trabajo mucho más fácil.

Por supuesto, gracias también a toda mi familia. Gracias de corazón a mis padres, por su apoyo en los momentos difíciles, sin el que no hubiera podido llegar hasta el final. Gracias a Silvia, por sus ánimos; y gracias a Suso y a Mónica, por toda su ayuda durante estos años.

Le estoy también muy agradecido a todos mis amigos, tanto a los que conocía antes de llegar aquí como a con los que he compartido estos cinco años; con quienes he pasado muchos buenos momentos.

A todos vosotros, gracias.

Resumen

La aparición de *blogs*, foros y redes sociales, donde multitud de usuarios expresan opiniones sobre una variedad de temas; ha despertado el interés del sector empresarial. Se ve en este tipo de recursos una oportunidad de conocer cómo sus productos son percibidos por los consumidores. Por ello, han empezado a demandarse soluciones que puedan analizar y monitorizar estas críticas.

La minería de opiniones, o análisis del sentimiento, permite realizar un tratamiento automático de la subjetividad de un texto, determinar su polaridad para reflejar si es positivo, negativo, neutro o mixto; e indicar la fuerza con la que se expresan las opiniones en un documento dado. Muchos de los sistemas actuales relacionados con este área atienden, casi exclusivamente, a criterios léxicos y a un conocimiento superficial del enunciado para resolver su polaridad, lo que lleva en ocasiones a interpretaciones equivocadas ya que no son capaces de capturar adecuadamente el sentido expresado por el autor.

El sistema propuesto en este proyecto permite medir la polaridad de textos subjetivos escritos en castellano basándose en una aproximación semántica. Esta técnica emplea una serie de diccionarios donde para cada palabra subjetiva se dispone de una valoración genérica que mide en qué grado ese término es positivo o negativo. Para tratar de reducir el porcentaje de interpretaciones erróneas, se dispone de un analizador sintáctico que permite extraer el significado de las frases a través de relaciones de dependencia entre sus componentes. Para ello, el sistema realiza tareas previas de segmentación, *tokenización* y etiquetado morfológico que proporcionan toda la información necesaria para la obtención de dichas relaciones. A partir de esta estructura se realiza una evaluación atendiendo tanto a criterios léxicos como sintácticos que devuelve la orientación semántica del texto correspondiente.

Este trabajo se centra en el tratamiento de adjetivos, nombres, verbos, adverbios, y en resolver tres construcciones lingüísticas significativas en un entorno de minería de opiniones que no pueden solucionarse a nivel puramente léxico: la intensificación, que permite potenciar o reducir la orientación semántica de los términos, las oraciones subordinadas adversativas y la negación.

El tratamiento de la intensificación y las subordinadas adversativas se basa en una estrategia de ponderación sobre ramas del árbol sintáctico que posibilita conocer los fragmentos de las oraciones donde se expresa un mayor énfasis. Para resolver el fenómeno de la negación, se introduce el concepto de alcance de la negación y se establece un procedimiento que permite

modificar únicamente la polaridad de los términos afectados por la negación.

El rendimiento de las soluciones propuestas ha sido puesto a prueba en diversos *corpus* estándar de minería de opiniones en castellano, obteniendo resultados que superan a los de sistemas que no hacen uso de información sintáctica.

Palabras clave: Minería de Opiniones, Análisis del Sentimiento, Análisis Sintáctico de Dependencias

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Conocer lo que los demás piensan ha sido desde siempre un factor esencial en la toma de decisiones. Pedir o dar una opinión a alguien cercano es algo que la mayoría de nosotros hemos hecho en algún momento. Gracias a internet y a la web, ahora es posible encontrar opiniones y experiencias más allá de nuestro entorno. La aparición en los últimos años de los *blogs*, los foros o las redes sociales ha provocado que multitud de usuarios utilicen estos recursos para expresar sus opiniones sobre toda una variedad de temas.

Estos recursos permiten que cualquier consumidor pueda tener una visión más o menos detallada sobre un producto o servicio que desee adquirir. La diversidad y cantidad de opiniones expresadas por millones de usuarios potenciales resultan de gran utilidad para fabricantes y vendedores, que ven en ellas un mecanismo para conocer de primera mano cómo sus artículos son percibidos por los consumidores. Estas críticas permiten a su vez a las compañías comerciales conocer cuáles son los aspectos de sus productos que funcionan, y cuáles los que no. Además es muy común que los usuarios establezcan relaciones y comparaciones con los artículos ofertados por la competencia, lo que permite a las empresas conocer cuáles son los puntos en los que necesitan mejorar y en qué sentido. Los beneficios económicos que se pueden derivar de este conocimiento son evidentes y por eso el mercado ha empezado a reclamar soluciones que permitan analizar y monitorizar todo este flujo ingente de opiniones.

Todos estos factores han contribuido a que el área de la *minería de opiniones* (MO), también conocida como *análisis del sentimiento*, juegue un papel relevante como ámbito de investigación en los últimos años.

1.1. Contextualización

La MO trata de determinar automáticamente la *subjetividad* de un texto (si en él se opina o no), la *polaridad* o *sentimiento* (si la opinión que se expresa es positiva o negativa, neutra o mixta) y la fuerza o convencimiento con la que se refleja la crítica expuesta [1, 2]. Las actividades relacionadas con este ámbito han crecido exponencialmente en los últimos años gracias a la disponibilidad de abundantes recursos web donde se expresan opiniones[1] y a la propia demanda del mercado. Sin embargo, desarrollar sistemas capaces de analizar contenidos de opinión de forma efectiva requiere afrontar toda una serie de retos y problemas, cuyo estudio da pie y justifica el presente trabajo.

1.1.1. Retos y problemas

Son tres los principales retos que debe ser capaz de resolver un sistema general de análisis del sentimiento:

1. Dada la consulta de un usuario, determinar en qué partes de un documento se expresa opinión. Esta tarea puede ser relativamente sencilla en webs especializadas en críticas, que muestran las opiniones en un determinado lugar, pero puede complicarse en sitios como los *blogs* personales según las características de estilo, formato y presentación de cada autor.

2. Obtener el sentimiento general expresado en los textos seleccionados para una consulta dada. Aparte de la polaridad del enunciado, podría ser también necesario extraer la opinión condensada en determinados fragmentos del texto, conocer a fondo las características de un artículo o comparar los aspectos de distintos ítems de una misma categoría.

3. Presentar los resultados del análisis de los textos subjetivos de una forma clara y sencilla para los usuarios. Al clasificar los textos, lo normal será tener pocas clases que deben poder generalizarse para múltiples dominios y usuarios. Por ejemplo, la clasificación de los enunciados en positivos y negativos, o la popular categorización en varias estrellas representando una escala de valoración.

El sistema propuesto en el proyecto se ha centrado en resolver el punto 2, esto es, en extraer el sentimiento de un texto según la opinión del autor. Una tarea más complicada de lo

[1] Es el caso, por ejemplo de las redes sociales, los foros *online* o los *blogs* personales.

que *a priori* puede parecer. Uno de los principales problemas radica en conocer un conjunto fiable de palabras que sean buenas indicadoras de sentimiento. La polaridad puede, además, ser dependiente en muchos casos del contexto y del dominio en el que se encuentre. Así, la palabra *"impredecible"* podría tener una valoración negativa en algunos casos, como en:

"El comportamiento de esta aplicación es impredecible"

Y sin embargo en otro dominio su sentimiento podría ser positivo o incluso depender de los propios gustos del lector, por ejemplo en:

"La película tuvo un final impredecible"

Otro de los problemas típicos es el empleo de figuras literarias, caso de la ironía o del sarcasmo, como formas de expresar una crítica. La utilización de éstas hace que el sentimiento expresado en el enunciado se muestre en ocasiones de una forma mucho más sutil y difícil de detectar. También es importante considerar aspectos de la estructura del texto. El orden en que se expresan las ideas sirve para reflejar en parte el sentimiento general. Según [1], es común que las frases finales de un discurso tengan una mayor importancia en la valoración dado que es en esta parte del texto donde el autor indica sus conclusiones y emplea un mayor énfasis. Pero sobre todo, una de las mayores dificultades a las que se enfrentan este tipo de sistemas son las construcciones lingüísticas propias del lenguaje. Así, identificar y tratar fenómenos como la negación o la intensificación son claves para lograr un análisis completo de la subjetividad para un enunciado dado.

1.1.2. Tratamiento de la polaridad

Actualmente, existen dos aproximaciones principales para tratar de resolver automáticamente la polaridad de un texto [1, 3]:

1. *Aprendizaje automático* (AA): Se construye un clasificador a partir de un conjunto de entrenamiento formado por una colección de textos etiquetados, donde se expresa una opinión favorable o desfavorable. Se trata de un entrenamiento supervisado en el que el clasificador aprenderá a reconocer en base a los ejemplos que se le presenten.

2. *Orientación Semántica* (OS) : Se emplean diccionarios donde cada palabra se encuentra etiquetada con su orientación semántica, permitiendo medir en qué grado esa palabra es positiva o negativa. El sentimiento del texto se obtiene agregando los valores de

los términos del texto que aparezcan en alguno de los diccionarios. Existen dos formas de generar estos diccionarios: crearlos de forma manual o construirlos de forma semi-automática, utilizando *palabras semilla*. En este último caso, el diccionario se va ampliando en un proceso iterativo atendiendo a relaciones de proximidad física. Se parte de un grupo reducido de palabras de polaridad extrema, como *"felicidad"* o *"asesino"* y, posteriormente, en cada iteración palabras cercanas a términos positivos o negativos pasarán a considerarse también positivas o negativas, respectivamente.

Los clasificadores obtenidos a partir de la primera alternativa se caracterizan por conseguir un buen rendimiento base para el dominio en el que son entrenados. Sin embargo, presenta complicaciones para mejorar su precisión. Además, las soluciones desarrolladas empeoran drásticamente su rendimiento cuando se utilizan para analizar textos de un dominio diferente al del *corpus* con el que se entrenaron [3].

La segunda alternativa permite una mejor adaptación a los diferentes dominios y además contempla más aspectos del texto[2]. El principal inconveniente radica en que los recursos deben construirse cada vez que se quiere aplicar a una nueva lengua, mientras que empleando AA es suficiente con obtener un conjunto de entrenamiento y clasificar los documentos.

Para la implementación del sistema, se ha optado por la segunda opción dado que se ha podido disponer de un diccionario semántico para el castellano, el SODictionariesV1.11Spa, desarrollado por la Universidad Simon Fraser y cuyo contenido será explicado más adelante.

1.2. Objetivos y contribuciones

Aunque ya existen herramientas de MO[3], la mayoría están pensadas para el procesamiento de textos en inglés, prestando escasa atención a otros idiomas, en particular el castellano. Además suelen aplicar un conocimiento casi exclusivamente léxico [2, 3, 4] y muy superficial de los enunciados, atendiendo a heurísticas y patrones, lo que lleva a interpretaciones equivocadas en no pocas ocasiones.

El sistema que se propone en este proyecto tiene como finalidad analizar el sentimiento de textos subjetivos escritos en español, indicando si en ellos se está dando una opinión negativa o positiva. Para reducir el porcentaje de interpretaciones equivocadas se hace uso de métodos de análisis sintáctico de dependencias. Ello supone una aportación original respecto

[2]Como la capacidad de procesar elementos del contexto que rodea a un determinado término.

[3]En el ámbito comercial destaca Radian6 (www.radian6.com) que permite monitorizar y extraer información de contenidos en inglés.

al estado del arte actual del problema y constituye una de las contribuciones más importantes del trabajo. El objetivo es disponer de una estructura sintáctica profunda, conocida como *grafo de dependencias*, que permita tratar aspectos del lenguaje que no pueden resolverse a nivel puramente léxico.

El sistema desarrollado procede a clasificar la polaridad de los textos, utilizando tanto análisis léxico como una evaluación sobre la estructura sintáctica. Se tiene muy en cuenta el empleo de los intensificadores en los textos, analizando el árbol sintáctico de las oraciones para tratar de determinar con exactitud cuáles son las palabras que se ven afectadas por este tipo de modificador. También se presta especial atención a cómo determinar la polaridad cuando existen una o más ocurrencias de términos negativos en el enunciado. Para ello, es necesario introducir el concepto de *alcance de la negación* y establecer un procedimiento que permita, utilizando el grafo de dependencias y una serie de reglas específicas, cambiar únicamente la polaridad de los términos realmente afectados por este tipo de construcción.

La incorporación de las distintas funcionalidades propuestas en el trabajo es evaluada con pruebas de precisión para así conocer los efectos que producen, y el comportamiento del sistema final se comprueba con distintos *corpus*[4] (en castellano). Los resultados son prometedores ya que muestran que la utilización de la estructura sintáctica produce una mejora efectiva del rendimiento frente a sistemas que no hacen uso de ella, y servirán de base para futuros trabajos de investigación.

1.3. Ámbitos de aplicación

El análisis del sentimiento sirve, de forma genérica, para construir sistemas capaces de puntuar opiniones sobre productos o servicios automáticamente, pero son muchos los campos y las aplicaciones concretas que utilizan los principios de la MO [1]:

- *Aplicaciones de inteligencia de negocio*: Es probablemente uno de los campos donde este área puede tener una mayor proyección. El análisis del sentimiento permite extraer fragmentos de un texto que representen críticas de características de los productos, dando a conocer cuáles son los aspectos concretos del artículo que la empresa debe mejorar. La MO también es de utilidad para realizar informes completos sobre la valoración del producto de una compañía. Un sistema de este tipo puede buscar en los distintos recursos web opiniones sobre dicho producto y elaborar resúmenes acerca de

[4] Colección de documentos.

su percepción.

- *Integración en aplicaciones relacionadas con* PLN: Este área puede ser de utilidad en sistemas de búsqueda de respuestas, donde las preguntas clasificadas como subjetivas requieren un tratamiento distinto al de cuestiones factuales. El análisis del sentimiento también ha demostrado ser de utilidad en sistemas de extracción de información, que ven mejorado su rendimiento cuando la información subjetiva es descartada.

- *Aplicaciones de valoración en política*: Las técnicas utilizadas para analizar las opiniones sobre productos o servicios son extrapolables al mundo de la política. Ya existen investigaciones y trabajos dirigidos a conocer lo que los votantes piensan acerca de sus representantes. Entre las utilidades de este tipo está la identificación de cuáles son las propuestas políticas que más gustan a los ciudadanos, las que provocan más rechazo o conocer la valoración sobre los partidos o cargos públicos.

- *Aplicaciones de inteligencia gubernamental*: El análisis del sentimiento puede ser de utilidad a la hora de filtrar, analizar y monitorizar comunicaciones sospechosas vía internet.

1.4. Estructura de la memoria

El contenido de la memoria se estructura de la siguiente forma. El segundo capítulo realiza una introducción de las etapas relacionadas con el *procesamiento del lenguaje natural* (PLN) que ha sido necesario realizar para desarrollar un sistema de MO fiable y eficiente. En el capítulo 3 se hace una descripción general del sistema, se esboza su arquitectura general y se detallan los aspectos en los que se ha centrado el proyecto. Seguidamente, en el capítulo 4 se presenta brevemente la planificación de este trabajo. Los tres siguientes capítulos se centran en la descripción de las etapas previas al análisis de la orientación semántica. Así, el capítulo 5 trata la segmentación de frases y palabras, el sexto se centra en el proceso de etiquetación del sistema que permite obtener la información morfológica de las palabras y el séptimo explica cómo se ha realizado el análisis sintáctico de las oraciones para que el sistema sea capaz de interpretar los textos. El capítulo 8 introduce el núcleo del analizador semántico. Los tres capítulos posteriores explican cómo se han desarrollado nuevas funcionalidades que permiten tratar los aspectos sintácticos: el 9 se centra en las construcciones lingüísticas capaces de potenciar la subjetividad de las palabras, esto es la intensificación, el 10 trata las implicaciones

de las oraciones subordinadas adversativas y el 11 estudia el fenómeno de la negación, un aspecto crítico en un entorno de análisis del sentimiento. En el capítulo 12 se muestran y explican los distintos resultados obtenidos para el sistema. En el último capítulo se presentan las conclusiones y se comentan, brevemente, algunas ideas que podrían desarrollarse en el futuro.

Capítulo 2

Procesamiento del lenguaje natural para minería de opiniones

Desarrollar sistemas de MO requiere realizar actividades previas relacionadas con el procesamiento del lenguaje natural (PLN), que ahora introducimos. Más concretamente, en la sección 2.1 se describen las actividades previas necesarias para realizar cualquier aplicación real relacionada con el PLN. En la sección 2.3 se explica el análisis sintáctico de dependencias, que ha permitido extraer las relaciones entre los componentes de una frase y que ha servido de base para la interpretación de su significado.

2.1. Preprocesamiento

Trabajar con aplicaciones de PLN implica realizar en un primer paso actividades de segmentación y etiquetado, que representan el punto de partida para la fase de análisis léxico de los documentos sobre los que se pretende trabajar.

Un *segmentador* es el encargado de identificar y separar las distintas oraciones de un texto. En un primer momento podría pensarse en reglas generales que ayuden a separar las frases. Así, un punto seguido de una letra mayúscula podría indicar el comienzo de una oración. Sin embargo, existen varias situaciones que invalidan esta regla como por ejemplo la utilización de siglas o abreviaturas. Este tipo de fenómenos unidos a otros donde los signos ortográficos puedan tener una función distinta de la habitual, hace que sea necesario considerar reglas heurísticas y patrones de no poca complejidad que permitan al segmentador resolver un caso concreto en un dominio concreto.

Separadas las frases, el siguiente paso es identificar los *tokens*[1] de una oración, tarea de la que se ocupará el *tokenizador*.

En conjunto, estas dos tareas permiten abordar la etiquetación morfológica del texto. En términos de lingüística computacional, se define *etiqueta* como la información léxica que identifica a un *token*. Tal información puede asociar diferentes grados de complejidad, dependiendo del tipo de aplicación que se quiera construir. Típicamente incluye la categoría gramatical, amén de información asociada a la misma como es por ejemplo el género o el número. Por ejemplo, a la palabra *"perro"* se le podría asignar la etiqueta *sustantivo* o *sustantivo masculino singular*, según la cantidad de información léxica que se quiera incorporar.

2.2. Etiquetación

Una vez preprocesado el texto, es posible afrontar la fase de análisis léxico propiamente dicha.

Definición 2.2.1. Sea $S = [w_0, w_1, ...w_n]$ una frase del texto y sea $T = \{t_0, t_1, ..., t_m\}$ el conjunto de todas las etiquetas posibles; se denomina *etiquetación* al proceso de asignar a cada término de una oración su etiqueta correspondiente, obteniendo una lista de tuplas (s,t), donde $s \in S$ y $t \in T$.

■

Etiquetar palabras es una tarea compleja. El problema radica en que un mismo término puede tener etiquetas diferentes según el contexto de la oración en el que se emplee. A ello hay que sumar el problema de la ambigüedad léxica, común en PLN. Por ejemplo, la palabra *"lava"* actuaría en la siguiente frase como un sustantivo:

"El volcán entró en erupción expulsando mucha lava."

Pero también podría actuar como un verbo en otro contexto:

"Marcos siempre lava su coche antes de salir de viaje."

El etiquetador debe resolver esta ambigüedad, al menos a nivel léxico[2]. Existen diferentes técnicas de resolución, cuyo rendimiento está ligado al *corpus* de entrenamiento que se emplee y al tipo de documentos que se traten, aunque no suele haber grandes diferencias de precisión.

[1] Un *token* representa una cadena de caracteres. Cada signo ortográfico y cada palabra se considerarán como un *token*.

[2] La ambigüedad de tipo sintáctico y semántico ha de resolverse a esos niveles, no siendo posible su tratamiento en esta fase.

Es frecuente emplear procesos de etiquetación estadísticos basados en *n-gramas*[3]. Markov fue el primero en proponer este tipo de técnica, utilizando bigramas y trigramas para tratar de determinar si la siguiente letra de una palabra sería vocal o consonante. Sus estudios permitieron desarrollar toda una rama de etiquetadores estadísticos capaces de calcular la secuencia de etiquetas más probable dentro de una oración [5, 6], lo que otorga cierta capacidad contextual al proceso. En los últimos años, un enfoque más popular consiste en interpretar la etiquetación como un proceso genérico de clasificación, donde a cada término dentro de su contexto se le asigna una de las distintas clases de etiquetas posibles. La palabra se clasifica en base a una serie de características de su entorno[4], permitiendo tener en cuenta tanto a los términos que la preceden como a los que la siguen. En documentos correctamente escritos el rendimiento de este tipo de métodos es realmente bueno, pero su precisión empeora de forma sensible cuando trabajan con textos donde abundan las palabras desconocidas.

Un enfoque alternativo a los modelos de n-gramas lo constituye el basado en transformaciones, comúnmente conocido como etiquetación de Brill [7]. Esta propuesta se basa en el aprendizaje basado en transformaciones y dirigido por error para tratar de asignar la etiqueta correcta a cada palabra. En la implementación original del algoritmo, se distinguen dos fases. La primera es de inicialización. Aquí, a cada palabra se le asigna en un primer momento su etiqueta más probable sin considerar el contexto en el que aparece. En caso de que se trate de una palabra desconocida, se tratará como un nombre propio si comienza por una letra mayúscula y sino se etiquetará como un sustantivo común. A continuación se lleva a cabo la fase de aprendizaje. Partiendo de una serie de reglas contextuales y reglas heurísticas de etiquetación de palabras desconocidas, se lleva a cabo un proceso iterativo en el que se seleccionan las reglas candidatas que mejoren la precisión del etiquetador y que se resume en los siguientes pasos:

1. Medir el número de errores antes y después de aplicar de forma separada cada una de las reglas.

2. Seleccionar la regla que mejore más el rendimiento.

3. Añadir la regla candidata al conjunto de reglas, y etiquetar de nuevo el texto.

4. Repetir el proceso hasta que no haya reglas que mejoren la precisión actual.

[3]Un n-grama es una secuencia de n elementos de una misma frase. Estos pueden ser palabras sílabas o simplemente caracteres.

[4]Cómo características tipográficas o las propias etiquetas.

Esta aproximación ha sido en la que se ha basado este trabajo dado su buen rendimiento en entornos en los que muchas palabras son desconocidas, como es el caso de la web, donde las normas ortográficas son frecuentemente ignoradas y el uso de vocablos agramaticales o no disponibles en el diccionario es habitual.

Otra técnica con la que se han obtenido etiquetadores con un buen rendimiento es la basada en el formalismo reduccionista no cuantitativo denominado *English Constraint Grammar* (EngCG), desarrollado por la Universidad de Helsinki. Esta aproximación utiliza un conjunto de reglas tanto de contexto global como de contexto local. En vez de considerar una gramática formal hace uso de restricciones, por lo general negativas, que van eliminando los análisis imposibles del contexto.

En cualquier caso, los etiquetadores actuales están muy perfeccionados. En los últimos años la mejora se ha medido en décimas de punto porcentual como se refleja[5] en el cuadro 2.1. Los resultados representan la evaluación de varios etiquetadores sobre el *corpus* Penn Treebank, formado por documentos periodísticos en inglés extraídos del *Wall Street Journal*. Para el castellano, la tendencia de la precisión tiende a ser ligeramente menor, dada la mayor libertad de disposición de las palabras en una oración, lo que complica su etiquetación.

Etiquetador	**Fecha de publicación**	**Precisión**
TnT	2000	96,46 %
Melt	2009	96,96 %
GeNiA Tagger	2005	97,05 %
SVMTool	2004	97,16 %
Stanford Tagger 1.0	2003	97,24 %
Stanford Tagger 2.0	2011	97,32 %
SCCN	2011	97.5 %

Cuadro 2.1: Precisión teórica de distintos etiquetadores

[5]Resultados extraídos de http://aclweb.org/aclwiki/index.php?title=POS_Tagging_%28State_of_the_art%29 Visitado por última vez en agosto de 2012.

2.3. Análisis sintáctico de dependencias

El análisis sintáctico es el proceso que permite extraer las relaciones entre los componentes de una oración, lo que contribuye a comprender e interpretar eficazmente un texto, completando el análisis léxico precedente y sirviendo de partida para el análisis semántico posterior. Se pueden distinguir dos estrategias para tratar de obtener la estructura sintáctica de una frase: el *análisis basado en constituyentes* y el *análisis de dependencias*. En el primer caso se emplea una gramática formal para extraer la estructura de las frases, cuya representación es una interpretación literal de las reglas gramaticales. En la figura 2.3 se muestra un ejemplo de gramática y análisis realizado con esta aproximación. El análisis de dependencias sigue un enfoque distinto, de tipo lingüístico, donde el objetivo es determinar la relaciones sintácticas existentes entre los elementos de una frase. Existen varios motivos que han motivado que este tipo de análisis haya cobrado importancia en los últimos años:

- Su utilidad para aplicaciones relacionadas con el PLN, dada la forma en que codifican las relaciones entre elementos.
- Una mejor adaptación que la ofrecida por una interpretación literal de las reglas gramaticales, para desarrollar herramientas que permitan trabajar con varios idiomas, sobre todo en lenguajes de gran riqueza sintáctica, como el castellano.
- El buen rendimiento que presentan los analizadores de dependencias y la disponibilidad de *corpus* en distintos idiomas, así como de formatos estándar de representación.

La idea básica del análisis de dependencias consiste en generar una estructura sintáctica formada por una serie de palabras conectadas por relaciones binarias, llamadas *relaciones de dependencia*. A la palabra sintácticamente subordinada en una relación de dependencia se le denomina *dependiente* y a la palabra de la que depende, *padre*. El *tipo de dependencia* es la etiqueta que se le asocia a cada relación y que resume la información sintáctica que liga a la palabra subordinada con la subordinante. Con el fin de simplificar las definiciones formales e implementaciones suele incluirse un nodo artificial, denominado ROOT, que actúa como raíz en todas las oraciones. Así, para la oración *"El niño rompió el muñeco rojo"* un posible análisis sería el que se muestra en la figura 2.1, del que directamente se podría extraer la estructura sintáctica representada en la figura 2.2, similar a la utilizada en nuestro sistema. Con un análisis basado en constituyentes la estructura del árbol sintáctico para la misma oración sería muy distinta, tal y como se refleja en la figura 2.3.

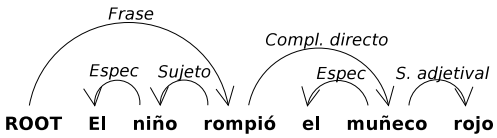


Figura 2.1: Ejemplo de relaciones de dependencias.

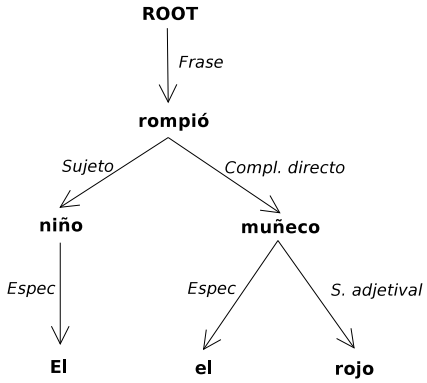


Figura 2.2: Ejemplo de un análisis sintáctico de dependencias.

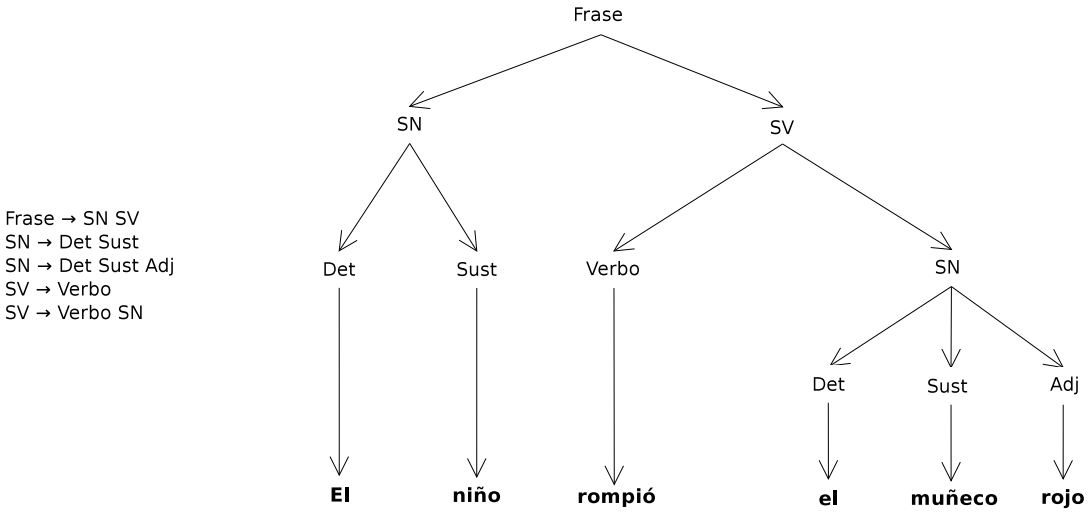


Figura 2.3: Ejemplo de un análisis sintáctico basado en constituyentes.

Como ya se comentó y como se aprecia en las figuras 2.2 y 2.3, el análisis sintáctico de dependencias representa la oración de forma que las relaciones entre componentes se identifican fácilmente, algo que no siempre es posible con el análisis basado en constituyentes. Es importante destacar que con este tipo de análisis la información más sensible acerca de la oración queda representada en los niveles superiores del árbol, algo muy útil en aplicaciones como la que nos ocupa porque permite extraer los sintagmas involucrados en las opiniones. A la estructura sintáctica creada a partir de una oración, como la de la figura 2.2, se le denomina *grafo de dependencias* [8].

Definición 2.3.1. Sea S = $[w_0 w_1 ... w_n]$ una secuencia del texto donde w_0 es el nodo ROOT, y sea el tipo de relaciones de dependencia un conjunto finito $R = \{r_1, ... r_m\}$ que representa todas las posibles relaciones que pueden establecerse entre dos palabras en una oración, un *grafo de dependencias* $G = (V, A)$ es un grafo dirigido donde:

1. $V = \{w_0, w_1, ..., w_n\}$ es el *conjunto de los nodos del grafo*, constituido por cada uno de los *tokens* de la oración, más w_0 que representa al nodo artificial.

2. $A \subseteq V \times R \times V$ representa el *conjunto de relaciones de dependencias* para el análisis de una oración en particular.

3. Cada uno de los elementos de A es de la forma $w_i \rightarrow w_j$, que es una *relación* donde w_i es el padre y w_j es el dependiente, que se anota con un tipo de dependencia $r \in R$.

■

Como se observa, el grafo de dependencias de la figura 2.2, es un árbol, como lo son todas las estructuras sintácticas obtenidas por el analizador que emplea nuestro sistema. Así, en el resto del trabajo se usará indistintamente la notación de *grafo de dependencias* o *árbol de dependencias*.

Aunque existe toda una variedad de métodos de análisis sintáctico de dependencias, ninguno hace hipótesis alguna sobre las dependencias utilizadas, las funciones gramaticales u otras características que puedan variar según el idioma, sino que dicha información se obtiene mediante aprendizaje automático. Ello supone una ventaja importante sobre las estrategias de análisis basadas en constituyentes.

2.3.1. Métodos dirigidos por los datos

En este trabajo se consideran estrategias *dirigidas por los datos*. En este contexto, a partir de un conjunto de datos debe generarse un analizador que asuma como válida cualquier oración, para devolver un grafo de dependencias. Destacan las que emplean un entrenamiento supervisado, lo que implica resolver dos grandes cuestiones:

- *Aprendizaje*: Debe ser posible generar un modelo de análisis a partir de los grafos de dependencias que conforman el conjunto de entrenamiento.
- *Análisis*: El modelo obtenido en la fase de entrenamiento deberá devolver el grafo de dependencias correcto para las oraciones.

En el estado del arte, dos son las estrategias más utilizadas, que pasamos a describir.

Métodos basados en grafos

Se caracterizan por definir un espacio de posibles grafos de dependencias para la oración de entrada. Aquí, el objetivo de la etapa de aprendizaje es entrenar un modelo que sea capaz de asignar una puntuación a cada uno de los grafos candidatos para una oración de entrada. El modelo se define a partir de un algoritmo de análisis y de una serie de restricciones sobre la estructura de los grafos. La puntuación asignada representa en cada caso cuál es la probabilidad de que ese análisis sea el correcto. Existen diferentes formar de puntuar; que varían según los algoritmos que emplea cada método. Con el modelo ya generado, la fase de análisis se reduce a encontrar el grafo con mejor puntuación para cada oración.

Métodos basados en transiciones

Un *sistema de transiciones* consiste en un conjunto de estados o configuraciones conectados por transiciones. El ejemplo más conocido y simple son los autómatas finitos que permiten cambiar de estado según la entrada que reciben en cada momento.

Los sistemas de transiciones empleados en el análisis sintáctico de dependencias son más complejos. Los estados no son atómicos sino que cada configuración dispone de su propia estructura interna, y cambiar de estado es un proceso más complejo que analizar la entrada en cada momento. A diferencia de los métodos basados en grafos, el objetivo de la fase de aprendizaje es entrenar un modelo que para cada oración sea capaz de obtener el grafo de dependencias correcto, en vez de manejar y puntuar un conjunto de árboles candidatos.

El modelo debe ser capaz de predecir la siguiente transición basándose en el historial de transiciones tomadas. El cometido de la fase de análisis es obtener para cada oración el conjunto de transiciones que devuelve el grafo de dependencias correcto. Esta es la estrategia utilizada en este trabajo, lo que exigirá una descripción completa más adelante. En particular se detallará la naturaleza de las transiciones, así como su proceso de selección.

2.3.2. El formato CoNLL 2006

Cada año, en el *Conference on Computational Natural Language Learning* (CONLL) se propone una tarea donde los participantes prueban sus sistemas bajo el mismo conjunto de datos. Amén de comparar los resultados de las distintas propuestas, esta iniciativa resulta muy útil porque el formato de datos propuesto es adoptado como estándar para el desarrollo de sistemas posteriores o para la evaluación de los mismos. En concreto, en el año 2006, la tarea propuesta fue el análisis sintáctico de dependencias multilingüe y comúnmente es conocida como la *CoNLL-X shared task* [9]. El formato propuesto es conocido como CONLL 2006 y se ha convertido en el estándar para el desarrollo de sistemas de análisis de dependencias. Se trata de un formato de 8 columnas donde para cada *token* se indica [8]:

1. ID: La posición dentro de la oración. Se reserva la posición 0 para el *token* extra correspondiente al ROOT.
2. FORM: El propio *token*.
3. LEMMA: La forma canónica de la palabra. Por ejemplo, para la palabra *"rompió"*, el lema sería *romper*.
4. CPOSTAG: Etiqueta de grano grueso con información léxica muy general, comúnmente su categoría gramatical. Así, *Verbo* sería la etiqueta de grano grueso de *"rompió"*.
5. POSTAG: Etiqueta de grano fino. Suele añadirse información léxica que complementa a CPOSTAG. *Verbo_preteritoIndicatico* podría ser una etiqueta de grano fino para el ejemplo anterior.
6. FEATS: Conjunto de información sintáctica y morfológica. Por ejemplo, para el caso de *"rompió"*, unas FEATS válidas serían *3persona_singular*.
7. HEAD: Indica el ID del *token* del que depende, esto es, su padre.
8. DEPREL: Indica el tipo de dependencia que se mantiene con HEAD.

En los años 2007 y 2008 se han propuesto evoluciones de este formato[6] que incluyen alguna información a mayores. Sin embargo, para este proyecto se ha optado por CoNLL 2006 por ser el más utilizado y conocido.

En el cuadro 2.2 se muestra una representación en formato CoNLL 2006 para el ejemplo de la figura 2.2. Los valores de POSTAG, FEATS o DEPREL no son los que se emplearon en este trabajo, sino que se trata de una codificación simplificada que permite comprender el ejemplo. En el cuadro 3.3 se mostrará un ejemplo real de las representaciones de los árboles de dependencias utilizadas en el proyecto.

ID	FORM	LEMMA	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL
1	El	el	d	d_articulo	masc_sing	2	Espec
2	niño	niño	n	n_comun	masc_sing	3	Sujeto
3	rompió	romper	v	v_preterito	3pers_sing	0	Sentencia
4	el	el	d	d_articulo	masc_sing	5	Espec
5	muñeco	muñeco	n	n_comun	masc_sing	3	Compl. directo
6	rojo	rojo	a	a_calificativo	masc_sing	5	S. adj

Cuadro 2.2: Ejemplo de un árbol de dependencias en formato CoNLL 2006

[6]En términos generales esta clase de formatos reciben el nombre de CoNLL-X.

Capítulo 3

El sistema de minería de opiniones

En este capítulo se introduce a grandes rasgos el funcionamiento general del sistema. Se presentan los recursos lingüísticos empleados y una breve explicación de las herramientas que han sido utilizadas en el proyecto.

3.1. Introducción

Como ya se ha expuesto, el desarrollo de un sistema de MO requiere abordar no pocos retos en el ámbito del PLN. En este proyecto se ha estudiado cómo tratar esos problemas, siempre desde una óptica basada en la información lingüística de los textos.

La mayoría de los sistemas actuales aplican soluciones exclusivamente léxicas para inferir la polaridad de un texto [2, 3, 4]. Ello permite resolver la orientación semántica de una palabra considerándola aisladamente, pero no consigue ir más allá, lo que supone una seria limitación práctica. Lo cierto es que sin una mínima carga de análisis sintáctico no es posible interpretar ni comprender los textos, ya que no se dispone de información de cómo los elementos de una oración se relacionan entre sí. En consecuencia, los acercamientos léxicos de MO integran patrones y heurísticas para estudiar elementos del lenguaje más complejos, lo que lleva en no pocas ocasiones a una clasificación incorrecta del texto.

Para tratar de resolver este problema, proponemos una solución basada en el análisis sintáctico de dependencias. La estructura obtenida se ha utilizado concretamente para analizar construcciones lingüísticas determinantes en el análisis del sentimiento de los textos. Tres han sido los fenómenos para los que se ha desarrollado una solución:

- La *intensificación*: Hay determinadas palabras y expresiones que pueden variar la orientación semántica de un *token*. Se habla de *amplificadores* si potencian la polaridad del

elemento al que afectan; o *decrementadores* si la reducen. Por ejemplo, *"muy bonito"* potencia el valor semántico de la palabra *"bonito"*, mientras que *"ligeramente bonito"* lo disminuye.

- Las *oraciones subordinadas adversativas*: Estas oraciones se caracterizan por la contraposición de ideas que se establece entre la oración subordinada y subordinante. En términos de análisis de la subjetividad implica el contraste entre dos opiniones. En trabajos como [2, 3], el tratamiento de este tipo de oraciones es considerado como un caso especial de intensificación, pero en nuestra propuesta ha sido estudiado como un caso aparte por sus implicaciones sintácticas.

- La *negación*: Las construcciones negativas son uno de los recursos más empleados a la hora de expresar una opinión y uno de los lugares donde el empleo del análisis sintáctico resulta de mayor utilidad. El objetivo es identificar con precisión cuáles son los elementos que se ven afectados por la negación, modificando de este modo únicamente la polaridad de los mismos.

Aun así, hay algunos aspectos del discurso que sólo pueden ser resueltos a través del empleo de heurísticas. Ejemplo de ello son el empleo de mayúsculas o la mayor importancia de las frases finales.

Otro aspecto a tener en cuenta son las fuentes de las que se recogen las opiniones. Normalmente éstas procederán bien de foros *online*, de *blogs* personales o de redes sociales. Es común que en estos lugares la gente ignore o no respete las normas ortográficas, o incluso utilice términos fuera del diccionario disponible. Esto puede complicar tareas como la segmentación, la *tokenización* o el análisis morfológico, sobre todo en lenguajes de gran riqueza flexiva como es el caso del castellano. Para ello se ha incorporado un preprocesador que se encarga de formatear el texto, facilitando las tareas posteriores de la separación de frases y palabras.

A grandes rasgos, el esquema general de la figura 3.1 ilustra las etapas que se han llevado a cabo para resolver la polaridad de un texto.

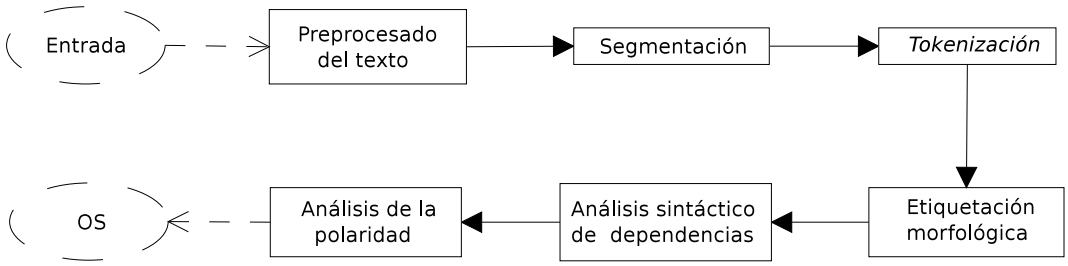


Figura 3.1: Esquema general del sistema de MO.

3.2. Tecnologías y herramientas empleadas

Se pasa ahora a describir brevemente las tecnologías que han servido de punto de partida para el desarrollo del trabajo.

3.2.1. NLTK

NLTK[1] (*Natural Language Toolkit*) es una plataforma de código abierto pensada para desarrollar programas Python relacionados con el PLN. NLTK nació con fines didácticos, pero ha sido adoptado muy rápidamente tanto en la industria como en el ámbito de la investigación, tanto por su eficiencia como por su sencillez de utilización y aprendizaje. Para la parte de la implementación del sistema en la que se integra el NLTK, se han empleado como referencias [10, 11] y la API[2] correspondiente.

Esta herramienta dispone de un directorio de datos, llamado *nltk_data*, donde se almacenan distintos *corpus*, modelos y libros que pueden ser útiles para el desarrollo de una aplicación PLN, aunque en su mayoría se trata de recursos pensados para el inglés.

3.2.2. Python

Python[3] [12] es un lenguaje de programación interpretado, con un fuerte tipado dinámico. Se caracteriza por tener una sintaxis simple, codificando de una manera que pretende ser cercana al lenguaje natural. Python permite utilizar peculiaridades de distintos paradigmas de programación, como el funcional, e incorpora los fundamentos básicos de la orientación a objetos (OO). Además es posible codificar *scripts* de una forma rápida y sencilla. Su implementación se encuentra sujeta a una licencia libre por lo que se puede distribuir de forma

[1] http://nltk.org Visitado por última vez en agosto de 2012.
[2] http://nltk.org/api/nltk.html Visitado por última vez en agosto de 2012.
[3] http://www.python.org/ Visitado por última vez en agosto de 2012.

gratuita.

El sistema de MO propuesto en este proyecto ha sido íntegramente implementado en Python, siendo la existencia de NLTK como plataforma asociada un factor muy importante para la elección de este lenguaje. Aunque existían también otros entornos de PLN para otros entornos como Java[4], eran más reducidos, menos documentados y con menos desarrolladores activos.

3.2.3. MaltParser

MaltParser[5] [13] es un generador de analizadores sintácticos de dependencias, implementado en Java y desarrollado por la Universidad de Växjö y la Universidad de Uppsala. A partir de un conjunto de entrenamiento, donde las oraciones estén etiquetadas como grafos de dependencias, se entrena un modelo de forma supervisada que permita analizar posteriormente nuevas frases. Se caracteriza por implementar distintos algoritmos de análisis, pudiendo configurarse para escoger cualquiera de ellos.

Los modelos generados por MaltParser presentan un buen rendimiento en una variedad de lenguajes como el inglés, francés, sueco o castellano. Sin embargo, es necesario configurar y modificar distintos parámetros si se quiere optimizar la precisión para un idioma en concreto.

3.2.4. Epydoc

Epydoc[6] es un generador automático de documentación API para programas Python, con un formato de presentación muy similar a la popular javadoc[7]. Para ello, se hace uso de *epytext*, un lenguaje de marcado utilizado para documentar las funciones dentro de un módulo Python, de forma que es posible añadir información sobre varios campos, como los parámetros, los valores de retorno o las precondiciones. Así, a partir de las anotaciones realizadas con este lenguaje, Epydoc procesa los módulos que se le indiquen y obtiene documentación en html, aunque también puede configurarse para otros formatos de salida, como pdf. En este proyecto, junto con el código fuente, también se proporciona una API generada con esta herramienta.

[4]http://opennlp.apache.org/ Visitado por última vez en agosto de 2012.

[5]http://www.maltparser.org/ Visitado por última vez en agosto de 2012.

[6]http://epydoc.sourceforge.net/ Visitado por última vez en agosto de 2012.

[7]http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html Visitado por última vez en agosto de 2012.

3.3. Recursos

A continuación se describen, también brevemente, los recursos lingüísticos utilizados en el desarrollo del presente proyecto.

3.3.1. SFU Spanish Review Corpus

Se trata de un *corpus* con 400 opiniones en castellano desarrollado por la Universidad Simon Fraser (SFU) [2], provenientes de la web ciao.es. Contiene reseñas sobre hoteles, coches, música, libros, películas, lavadoras, ordenadores y móviles. Cada categoría está constituida por un total de 50 textos, 25 de los cuales expresan una opinión positiva, mientras los otros 25 expresan una negativa. Este conjunto de críticas es el que se ha empleado para probar las distintas versiones del sistema, ya que asociado a este *corpus* se ha desarrollado un sofisticado sistema de análisis del sentimiento, *The Spanish* SO *Calculator*[8], que permite disponer de una referencia fiable de comparación de resultados, lo que le convierte en el *corpus* de referencia para MO en español.

Son textos planos con faltas de ortografía y abreviaciones sin preprocesar, lo que permite probar nuestra propuesta en un entorno muy realista, aunque ello complica tareas como la etiquetación, el análisis sintáctico o el propio análisis del sentimiento. La aparición tanto de opiniones a favor como en contra dentro de una misma crítica es otro factor que dificulta la obtención de la polaridad, pero garantiza un entorno de experimentación completo y fiable. Véase un caso que refleja muchos de estos problemas en el siguiente ejemplo[9]:

"Soy fan,super fan de estos chicos,pero hasta yo que los adoro,reconozco que salvo alguna cancion,este disco se lo podian haber ahorrado.Puede que algun fan que lea esto no este de acuerdo,lo respeto,claro que si,pero a mi me dejo fria. My friends o aeroplane,son temas que me gustan mucho lo reconozco, pero de ahi a decir algo bueno de este disco,me lo pone dificil. La ausencia de John Frusciante,el guitarrista,lo convierte en un disco olvidable, ya que Dave Navarro,con el que no acabaron muy bien,no consiguio intagrase en el grupo ,su estilo de musica era diferente a esta mezcla de punk y rock que los hacia unicos. Por suerte solo hubo uno con el,que no digo que sea mal guitarra,es que a mi me es indiferente. Mi recomendacion es que busqueis cualquiera de sus discos antes que este y luego comparais.Un saludo"

[8]Sistema desarrollado por la SFU que obtiene la polaridad de los textos en castellano basándose en aproximaciones léxicas.

[9]Ejemplo extraído del *corpus* de la SFU sin realizar ningún tipo de corrección gramatical ni sintáctica.

3.3.2. SODictionariesV1.11Spa

Es un conjunto de diccionarios de orientación semántica para adjetivos, adverbios, sustantivos, verbos e intensificadores creado por la SFU [2]. Las palabras con sentimiento que conforman estos diccionarios han sido extraídas de las críticas del *corpus* SFU y de la traducción de los diccionarios de orientación semántica empleados por *The English SO Calculator*[10]. Cada término se encuentra anotado con un valor dentro de una escala que abarca valores desde -5 para las palabras más negativas, hasta 5 para las más positivas. Todos los términos disponibles son lemas[11] y siguen el mismo formato de representación. En el cuadro 3.1 se indica un resumen del contenido del diccionario y en el 3.2 se muestra un pequeño fragmento del diccionario de adjetivos.

Diccionario	**Nº elementos**
adjetivos	2,049
sustantivos	1,324
verbos	739
adverbios	548
intensificadores	157

Cuadro 3.1: Resumen del contenido del SODictionariesV1.11Spa

Palabra	**OS**
prohibido	-1
evocador	2
resultón	4
traumatizado	-4
deplorable	-5
ingenuo	-2
nuevecito	3

Cuadro 3.2: Fragmento del diccionario de orientación semántica de adjetivos

[10]Sistema de minería de opiniones implementado por la SFU que obtiene el sentimiento de texto en inglés, empleando una aproximación léxica.

[11]Forma canónica de una palabra. Por ejemplo, el lema de *"cantaste"* es *"cantar"*.

3.3.3. Ancora Corpus Dependencies

Corpus formado por más de 500.000 palabras disponible tanto para el catalán como para el castellano [14]. Desarrollado por el CLiC[12], está formado fundamentalmente por textos periodísticos. Puede descargarse vía web desde la página del propio grupo[13], pero es necesario registrarse previamente. Cada una de las oraciones del texto se encuentra anotada como un árbol de dependencias en formato CoNLL-X. Una de las características de este *corpus* es que ninguna de sus palabras dispone de información en el campo POSTAG, la única información morfológica se encuentra en los campos CPOSTAG y FEATS. En el cuadro 3.3 se muestra un ejemplo en formato CoNLL 2006 para una oración de este *corpus*, representada como árbol de dependencias en la figura 3.2.

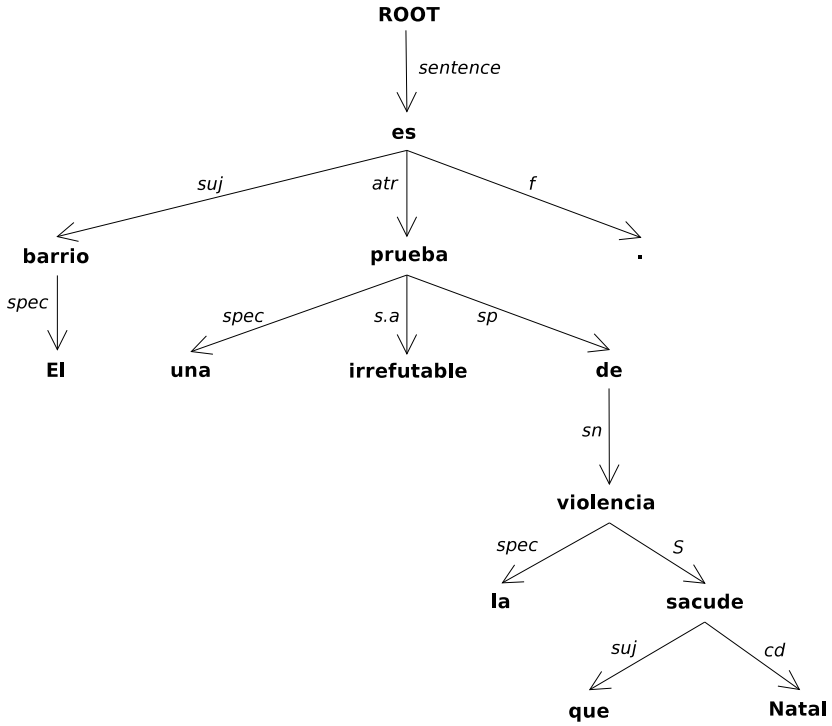


Figura 3.2: Ejemplo de un análisis sintáctico de dependencias, según Ancora.

[12]Centre de Llenguatge i Computació.

[13]http://clic.ub.edu/corpus/ Visitado por última vez en agosto de 2012.

ID	FORM	LEMMA	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL
1	El	el	d	_	postype=article|gen=m|num=s	2	spec
2	barrio	barrio	n	_	postype=common|gen=m|num=s	3	suj
3	es	ser	v	_	postype=semiauxiliary| num=s|person=3|mood=indicative|tense=present	0	sentence
4	una	uno	d	_	postype=indefinite|gen=f|num=s	5	spec
5	prueba	prueba	n	_	postype=common|gen=f|num=s	3	atr
6	irrefutable	irrefutable	a	_	postype=qualificative|gen=c|num=s	5	s.a
7	de	de	s	_	postype=preposition	5	sp
8	la	el	d	_	postype=article|gen=f|num=s	9	spec
9	violencia	violencia	n	_	postype=common|gen=f|num=s	7	sn
10	que	que	p	_	postype=relative|gen=c|num=c	11	suj
11	sacude	sacudir	v	_	postype=main| num=s|person=3|mood=indicative|tense=present	9	S
12	Natal	Natal	n	_	postype=proper	11	cd
13	.	.	f	_	punct=period	3	f

Cuadro 3.3: Ejemplo de un árbol de dependencias en formato CoNLL, según Ancora

3.3.4. Spanish Movie Reviews

Corpus formado por 3,878 opiniones sobre películas españolas extraídas de la página web de críticas de cine www.muchocine.net. Para cada documento, el autor indica su valoración en una escala de 1 a 5, donde 1 es lo más negativo y 5 lo más positivo. Creado por el grupo ITALICA de la Universidad de Sevilla, es posible descargarlo vía web[14].

Todos los textos están en formato XML[15] y todos disponen de un campo resumen escrito por el autor del texto que pretende reflejar la idea general de la crítica. Cada fichero XML tiene asociados otros cuatro archivos:

- *review.pos* y *summary.pos*, donde se muestra información lingüística de las palabras de la crítica y del resumen, respectivamente.
- *review.dep* y *summary.dep*, donde se representan los árboles de dependencias tanto para las oraciones de la crítica como las del resumen, respectivamente.

Ni los ficheros *.pos* ni los *.dep* se emplearon para desarrollar este sistema, puesto que el *corpus* seleccionado para el entrenamiento del etiquetador y del analizador sintáctico fue el Ancora Corpus Dependencies, técnicamente más completo, amplio y general, lo que ha permitido generar un etiquetador y un analizador más fiables.

3.3.5. HOpinion

Se trata de un *corpus* de aproximadamente 18,000 reseñas de hoteles extraídas de la página www.tripadvisor.es que ha sido desarrollado por el CLiC y que como el Ancora Corpus Dependencies también es posible descargarlo gratuitamente previo registro. De igual forma que en el *corpus* de Spanish Movie Reviews, cada texto se encuentra etiquetado con una valoración entre 1 y 5 que refleja el sentimiento percibido.

Tanto el Spanish Movie Reviews como el *corpus* de HOpinion se han empleado en este trabajo para comprobar como el sistema desarrollado se comporta sobre textos distintos a aquellos que han servido para su desarrollo. Ello proporciona validez al entorno de evaluación.

3.4. Metodología de desarrollo

La metodología de desarrollo seguida en el proyecto ha sido una evolutiva incremental. Para cada incremento se ha realizado su correspondiente análisis y especificación de requisitos,

[14] http://www.lsi.us.es/~fermin/index.php/Datasets. Visitado por última vez en agosto de 2012.
[15] http://www.w3.org/XML/ Visitado por última vez en agosto de 2012.

diseño, implementación, así como las pruebas de precisión necesarias para conocer el efecto de las funcionalidades que se han ido incorporando. Las iteraciones en las que se dividió el desarrollo del sistema son:

1. *Desarrollo del núcleo del sistema.* Incluye el preprocesado, etiquetado y análisis sintáctico. Se implementó una versión funcional en la que la polaridad del texto se obtenía únicamente atendiendo a criterios léxicos, sumando las orientaciones semánticas de los *tokens*.

2. *Tratamiento de la intensificación.* Para ello se hace uso del árbol de dependencias.

3. *Tratamiento de las subordinadas adversativas*, donde de nuevo se utiliza el árbol de dependencias para capturar el sentimiento expresado en este tipo de oraciones.

4. *Tratamiento de la negación*, a partir de la información proporcionada por las dependencias del árbol.

3.5. Arquitectura general del sistema

El sistema de análisis del sentimiento propuesto en este trabajo se descompone en subsistemas que se encargan de resolver las etapas ilustradas en la figura 3.1:

- Subsistema *sentimentanalyzer*: Es el elemento central de este proyecto. Se encarga de obtener la orientación semántica para un texto de entrada a partir de su representación como un conjunto de árboles de dependencias. Sus dos clases fundamentales son *SentimentAnalyzer* y *Dictionary*. La clase *SentimentAnalyzer* se ocupa de evaluar los árboles de sintácticos basándose en un sistema de funciones de visita, explicado en capítulos posteriores. La clase *Dictionary* se encarga de proporcionar información léxica o semántica sobre un nodo del grafo como, por ejemplo, resolver su orientación semántica u obtener su lema.

- Subsistema *parser*: Tiene como cometido interactuar con el analizador de dependencias para conseguir la representación de las opiniones en formato CoNLL 2006. También es el encargado de transformar los ficheros CoNLL en una estructura de datos que represente los grafos de dependencias, de manera que el analizador de la orientación semántica pueda procesarlos de forma eficiente.

- Subsistema *preparator*: Se encarga de realizar todas las tareas previas al análisis sintáctico propiamente dicho. En particular, segmenta, *tokeniza* y preprocesa las oraciones del texto e interactúa con el etiquetador del sistema para obtener toda la información morfológica del enunciado.

- Subsistema *tagger*: Tiene como objetivo configurar todo el proceso de entrenamiento para obtener una cadena de etiquetado adaptada a un entorno de MO.

El diagrama de secuencia de la figura 3.3 muestra las interacciones entre las principales clases involucradas al obtener la polaridad de un texto.

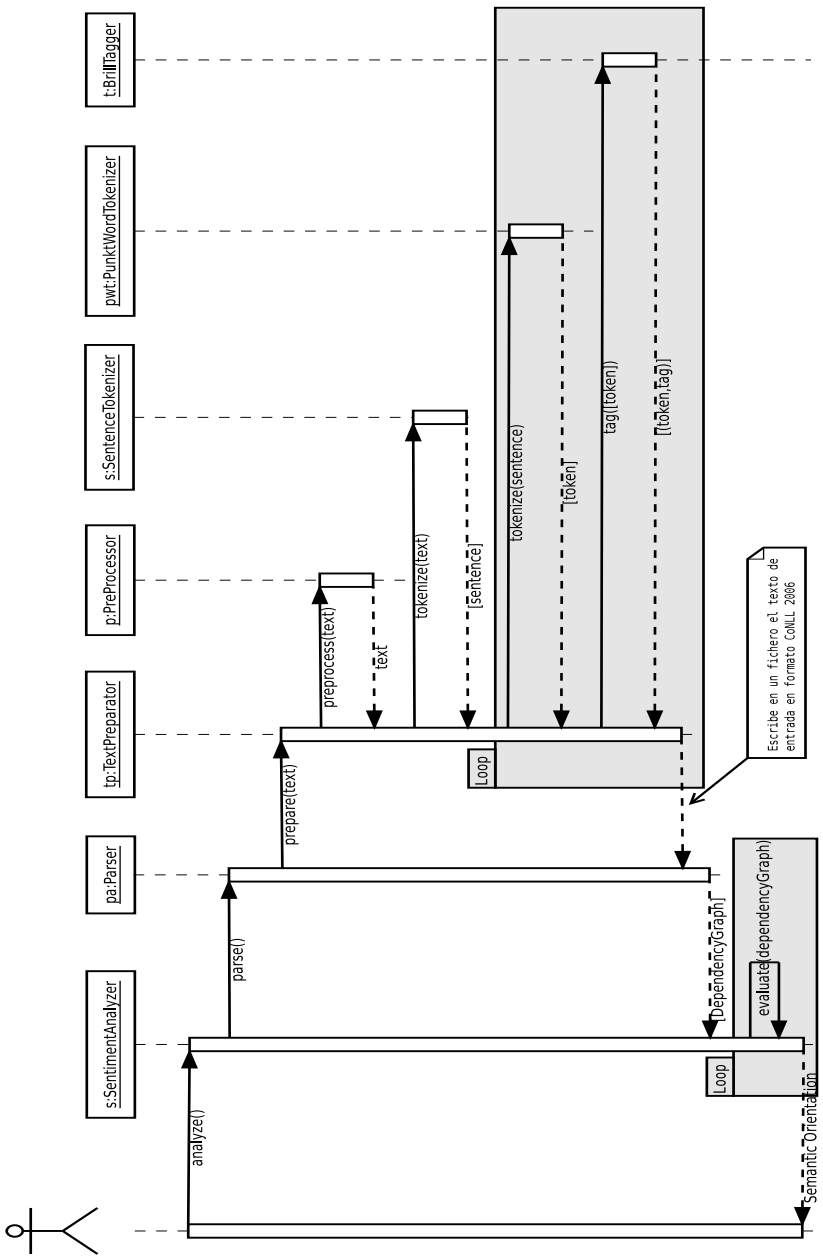


Figura 3.3: Diagrama de secuencia para obtener la orientación semántica de un texto

Capítulo 4

Planificación

En este capítulo se ilustra la planificación elaborada para este proyecto, así como los recursos humanos que han participado en el mismo. También se presenta la línea base establecida, y un seguimiento en el momento de la finalización del trabajo que permite comparar costes y desviaciones.

4.1. Planificación de las actividades

En un trabajo principalmente investigador, como el propuesto, es complicado planificar de una forma precisa; dada la obligación de estudiar y profundizar en nuevas técnicas y aproximaciones que mejoren las soluciones actuales. En cualquier caso, se realizó una planificación de las actividades que permitió establecer una serie de metas y analizar la viabilidad de este proyecto. Para llevar a cabo esta tarea, se ha empleado la herramienta Microsoft Project 2007, por ser una de las más completas y por estar familiarizado con ella.

En la figura 4.1 se ilustra la planificación inicial de este trabajo cuyo comienzo se fecha el 24 de agosto de 2011 y donde se distinguen cuatro grandes tareas, comentadas ya con anterioridad:

- Desarrollo del núcleo del analizador del sentimiento.
- Incorporación del tratamiento de la intensificación.
- Integración del tratamiento de las oraciones subordinadas adversativas.
- Inclusión del fenómeno de la negación.

El análisis, diseño e implementación de estas actividades fue responsabilidad del alumno, que representa el rol de *Analista-Diseñador-Programador* en la hoja de recursos de Project,

como se observa en la figura 4.2. Mientras, los dos codirectores fueron los encargados de supervisar el trabajo, actuando como *Jefe de proyecto* y participaron en la fase de análisis del sistema para cada una de las etapas. Estos recursos sólo han podido estar disponibles a un 50 % de su capacidad máxima dadas sus otras ocupaciones.

Id	Nombre de tarea	Duración
1	Inicio	0 días
2	Investigación y análisis previo	15 días
3	Instalación del software necesario	2 días
4	**Desarrollo del núcleo del sistema**	**112 días**
5	**Preprocesado**	**16 días**
6	Análisis	3 días
7	Diseño	5 días
8	Implementación	7 días
9	Pruebas	1 día
10	Segmentación	2 días
11	Tokenización	2 días
12	**Etiquetación**	**23 días**
13	Análisis	3 días
14	Diseño	5 días
15	Implementación	13 días
16	Evaluación	2 días
17	**Análisis sintáctico de dependencias**	**33 días**
18	Análisis	3 días
19	Diseño	5 días
20	Implementación	15 días
21	Evaluación	10 días
22	**Núcleo del analizador de la orientación semántica**	**36 días**
23	Análisis	7 días
24	Diseño	10 días
25	Implementación	18 días
26	Evaluación	1 día
27	**Incorporación de la intensificación**	**27 días**
28	Análisis	4 días
29	Diseño	7 días
30	Implementación	15 días
31	Evaluación	1 día
32	**Incorporación de las oraciones subordinadas adversativas**	**29 días**
33	Análisis	4 días
34	Diseño	7 días
35	Implementación	17 días
36	Evaluación	1 día
37	**Incorporación de la negación**	**37 días**
38	Análisis	5 días
39	Diseño	10 días
40	Implementación	21 días
41	Evaluación	1 día
42	Fin	0 días
43	Elaboración de la memoria	30 días
44	Fin del proyecto	0 días

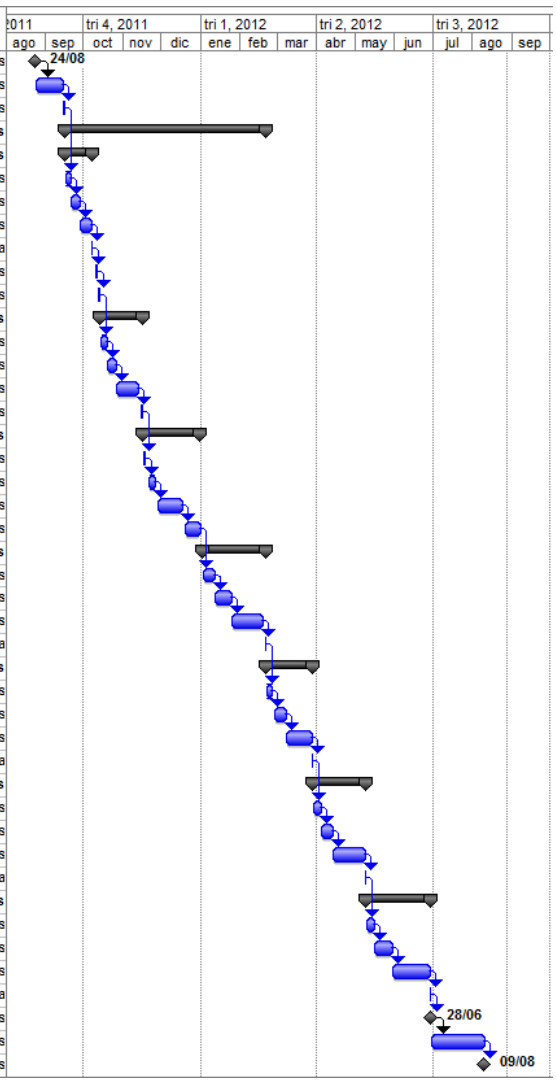


Figura 4.1: Diagrama de Gantt de las actividades del proyecto

Id	Nombre del recurso	Tipo	Iniciales	Capacidad máxima	Tasa estándar	Tasa horas extra	Costo/Uso
1	Analista-Diseñador-Programador	Trabajo	A	50%	20,00 €/hora	0,00 €/hora	0,00 €
2	Jefe de proyecto	Trabajo	J	50%	35,00 €/hora	0,00 €/hora	0,00 €

Figura 4.2: Recursos humanos del proyecto

4.2. Línea base y estimaciones del coste

Tras descartar posibles sobrecargas, el siguiente pasó fue establecer la línea de base, ilustrada en la figura 4.4. Se observa que el camino crítico[1] es único, ya que no hay tareas que puedan ser realizadas en paralelo; y que la finalización estaría programada para el 9 de agosto de 2012. Este punto indica cómo debería transcurrir el proyecto si no hubiera ningún inconveniente ni retardo en ninguna de las actividades. Así pues, es posible calcular el coste previsto, lo cuál queda reflejado en la figura 4.3; indicando la duración (252 días), el trabajo (1.154 horas) y el coste (25.270,00€).

	Comienzo	Fin
Actual	mié 24/08/11	jue 09/08/12
Previsto	mié 24/08/11	jue 09/08/12
Real	NOD	NOD
Variación	0d	0d

	Duración	Trabajo	Costo
Actual	252d	1.154h	25.270,00 €
Previsto	252d	1.154h	25.270,00 €
Real	0d	0h	0,00 €
Restante	252d	1.154h	25.270,00 €

Porcentaje completado:
Duración: 0% Trabajo: 0%
Cerrar

Figura 4.3: Costes previstos del proyecto

[1] En color rojo.

Id	Nombre de tarea	Duración
1	Inicio	0 días
2	Investigación y análisis previo	15 días
3	Instalación del software necesario	2 días
4	**Desarrollo del núcleo del sistema**	**112 días**
5	**Preprocesado**	**16 días**
6	Análisis	3 días
7	Diseño	5 días
8	Implementación	7 días
9	Pruebas	1 día
10	Segmentación	2 días
11	Tokenización	2 días
12	**Etiquetación**	**23 días**
13	Análisis	3 días
14	Diseño	5 días
15	Implementación	13 días
16	Evaluación	2 días
17	**Análisis sintáctico de dependencias**	**33 días**
18	Análisis	3 días
19	Diseño	5 días
20	Implementación	15 días
21	Evaluación	10 días
22	**Núcleo del analizador de la orientación semántica**	**36 días**
23	Análisis	7 días
24	Diseño	10 días
25	Implementación	18 días
26	Evaluación	1 día
27	**Incorporación de la intensificación**	**27 días**
28	Análisis	4 días
29	Diseño	7 días
30	Implementación	15 días
31	Evaluación	1 día
32	**Incorporación de las oraciones subordinadas adversativas**	**29 días**
33	Análisis	4 días
34	Diseño	7 días
35	Implementación	17 días
36	Evaluación	1 día
37	**Incorporación de la negación**	**37 días**
38	Análisis	5 días
39	Diseño	10 días
40	Implementación	21 días
41	Evaluación	1 día
42	Fin	0 días
43	Elaboración de la memoria	30 días
44	Fin del proyecto	0 días

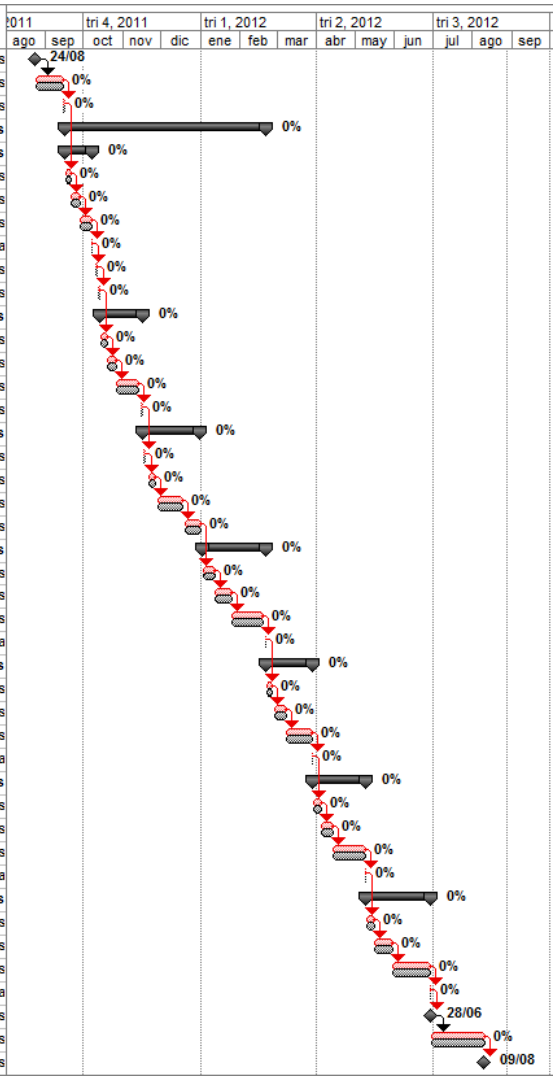
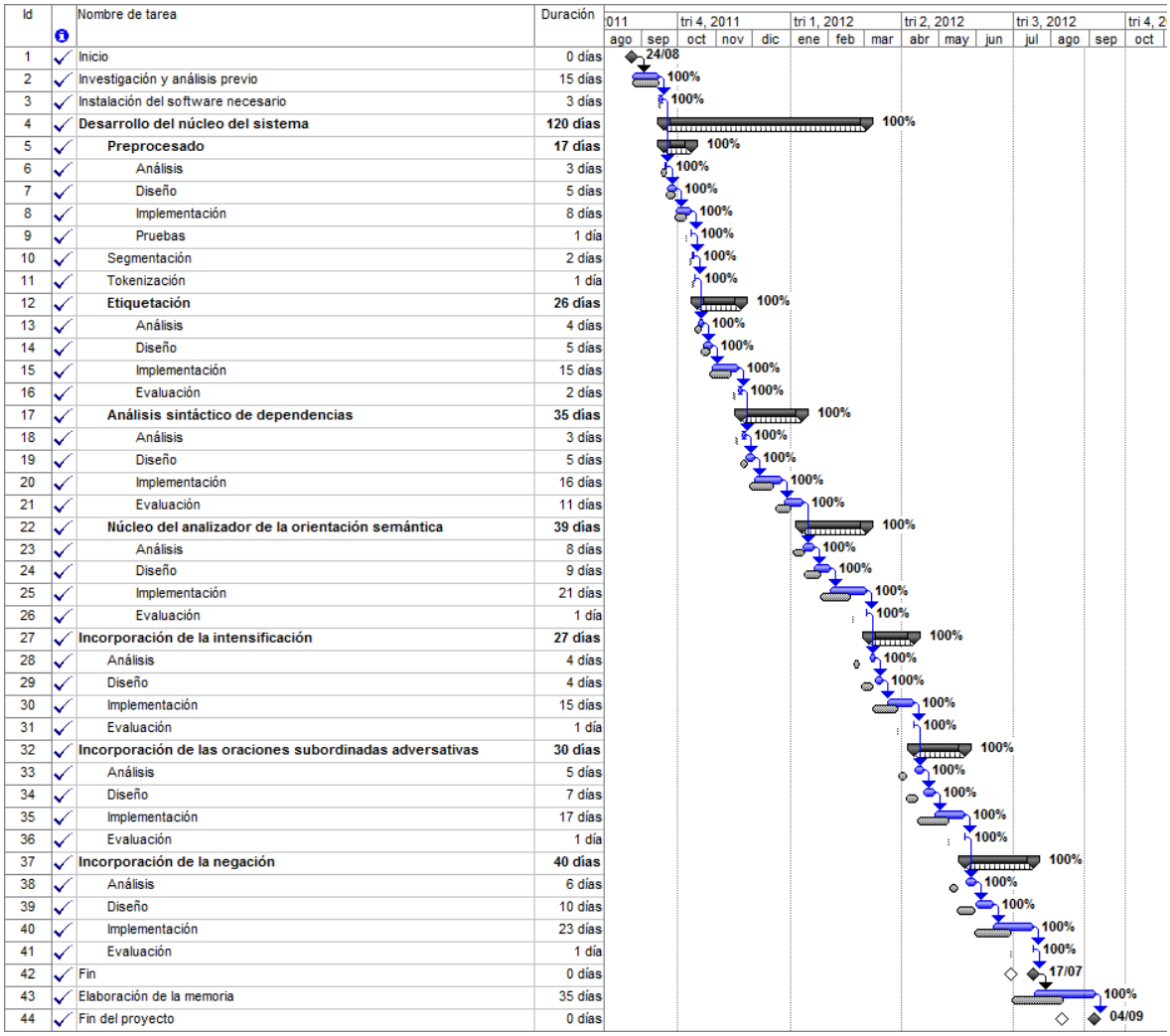


Figura 4.4: Línea base del proyecto

4.3. Seguimiento

Únicamente se ha realizado un seguimiento al final de proyecto para conocer cuáles han sido las desviaciones, tanto en tiempo como en coste, respecto a lo inicialmente planificado. En la figura 4.5 se muestra el diagrama de Gantt una vez finalizado el proyecto. El retraso ha sido de 26 días, con un sobrecoste de 1.460€, lo que elevaría el coste total hasta los 26.730,00€ y retardaría la entrega hasta el 4 de septiembre de 2012. El principal motivo de esta demora han sido las complicaciones relacionadas con el desarrollo del análisis de la orientación semántica.



Id	Nombre de tarea	Duración
1	Inicio	0 días
2	Investigación y análisis previo	15 días
3	Instalación del software necesario	3 días
4	**Desarrollo del núcleo del sistema**	**120 días**
5	**Preprocesado**	**17 días**
6	Análisis	3 días
7	Diseño	5 días
8	Implementación	8 días
9	Pruebas	1 día
10	Segmentación	2 días
11	Tokenización	1 día
12	**Etiquetación**	**26 días**
13	Análisis	4 días
14	Diseño	5 días
15	Implementación	15 días
16	Evaluación	2 días
17	**Análisis sintáctico de dependencias**	**35 días**
18	Análisis	3 días
19	Diseño	5 días
20	Implementación	16 días
21	Evaluación	11 días
22	**Núcleo del analizador de la orientación semántica**	**39 días**
23	Análisis	8 días
24	Diseño	9 días
25	Implementación	21 días
26	Evaluación	1 día
27	**Incorporación de la intensificación**	**27 días**
28	Análisis	4 días
29	Diseño	4 días
30	Implementación	15 días
31	Evaluación	1 día
32	**Incorporación de las oraciones subordinadas adversativas**	**30 días**
33	Análisis	5 días
34	Diseño	7 días
35	Implementación	17 días
36	Evaluación	1 día
37	**Incorporación de la negación**	**40 días**
38	Análisis	6 días
39	Diseño	10 días
40	Implementación	23 días
41	Evaluación	1 día
42	Fin	0 días
43	Elaboración de la memoria	35 días
44	Fin del proyecto	0 días

Figura 4.5: Diagrama de Gantt para el seguimiento al 100 % del proyecto

Capítulo 5

Preprocesado del texto

En este capítulo se indica en detalle cómo se han llevado a cabo el formateado y la segmentación del texto, de manera que éste quede preparado para que las tareas de etiquetado y análisis sintáctico aseguren en lo posible un tratamiento robusto del análisis semántico. Dado que los errores del usuario son impredecibles, no se puede llevar a cabo una reparación exhaustiva del texto, pero si es posible prever y corregir ciertos patrones.

5.1. Formateado del texto

Desgraciadamente, la forma en la que los usuarios expresan sus opiniones en internet no respeta en muchas ocasiones las normas ortográficas. Las faltas de ortografía, la abreviación de palabras, la ausencia de tildes o la incorrecta colocación de los signos de puntuación son algunos de los ejemplos usuales. Ante una mala calidad del texto de entrada, tareas como la segmentación, el etiquetado o el análisis sintáctico ven disminuido sensiblemente su rendimiento.

En el caso del Spanish Review Corpus de la SFU, son muchas las reseñas que presentan estas deficiencias. Para tratar de evitar que un texto no gramatical influya en exceso en la precisión del sistema, se ha definido un preprocesador *ad-hoc* que se encarga de encontrar y corregir algunos de los problemas que más pueden afectar en las etapas posteriores. Amén de cuestiones de menor impacto, dos son los fenómenos de mayor complejidad que se han tratado:

- *Separación de los signos de puntuación.* Uno de los factores más importantes para un buen funcionamiento del segmentador es la correcta colocación de los signos de

puntuación. En el caso del implementado en el proyecto, se encarga de formatearlos debidamente, siempre que no formen, por ejemplo, parte de un número.

- *Formateo de expresiones compuestas.* En castellano es habitual la existencia de grupos de palabras que actúan como si fueran sólo una. Así, sería un error considerar sistemáticamente por separado los *tokens* de expresiones como *"sin embargo"*, *"mientras que"* o *"a menos que"*, ya que actúan como si fueran una sola unidad, al menos en algunas ocasiones.

  Como se comentará más adelante, en el *corpus* de Ancora utilizado para entrenar tanto al etiquetador como al analizador sintáctico, este tipo de expresiones han sido formateadas por sus autores para considerarse como un único *token*, cuando es necesario. De este modo, por ejemplo, *"a menos que"* pasaría a representarse como *"a_menos_que"*.

  Sin embargo, es obvio que cuando un usuario expresa una opinión en la web este formateo no se realiza *a priori*, lo que obliga a que el sistema asuma la tarea de su tratamiento.

  Para resolver este contratiempo se optó por construir un diccionario de expresiones compuestas extraídas del propio *corpus* utilizado para el entrenamiento del analizador. De esta manera, antes de segmentar las palabras de una oración, el preprocesador del sistema de MO analiza el texto y formatea expresiones que aparezcan en el diccionario de formas compuestas.

Así, el Ejemplo 5.1.1 representa un pequeño texto de entrada con las situaciones comentadas sobre estas líneas y cómo quedaría éste tras haber sido preprocesado.

Ejemplo 5.1.1. El texto bajo estas líneas presenta expresiones compuestas como *"Sin duda"* que representan un solo *token*, y signos de puntuación que no respetan las convenciones del castellano, como es el caso de *",pero"*:

"Ese ordenador es rápido y fiable,pero no recomiendo comprarlo. Sin duda es muy caro,2.700€."

Tras haber sido preprocesado el enunciado, estos problemas son corregidos obteniendo como resultado:

"Ese ordenador es rápido y fiable , pero no recomiendo comprarlo . Sin_duda es muy caro , 2.700€ ."

■

Hay otros problemas, como por ejemplo la falta de tildes, que no han sido resueltos aquí. No obstante, se ha propuesto una solución que se ha llevado a cabo en el propio etiquetador, y que se explicará en el capítulo correspondiente al desarrollo del mismo.

5.2. Segmentación de frases y palabras

Una vez formateados los signos de puntuación, es más fácil para el segmentador separar las oraciones de un texto. Dentro del directorio de recursos *nltk_data* existen mecanismos que proporcionan ayuda para realizar diversas tareas, entre ellas la segmentación. En concreto se dispone de una serie de ficheros que almacenan instancias serializadas[1] de la clase *nltk.tokenize.punkt.PunktSentenceTokenizer* ya configuradas para distintos idiomas. Estos objetos se caracterizan por utilizar un algoritmo de entrenamiento no supervisado que construye un modelo en el que se tienen en cuenta abreviaturas, siglas, colocaciones[2] y otras palabras que representan el inicio de una oración. Con ese modelo, se intenta separar las distintas oraciones de un texto buscando delimitadores.

Ejemplo 5.2.1. Segmentación del texto del Ejemplo 5.1.1. El segmentador configurado con el fichero específico para el castellano devolvería en este caso dos oraciones:

$$O_1 = Ese\ ordenador\ es\ rápido\ y\ fiable\ ,\ pero\ no\ recomiendo\ comprarlo\ . \tag{5.1}$$

$$O_2 = Sin\_duda\ es\ muy\ caro\ ,\ 2{,}700€\ . \tag{5.2}$$

■

El siguiente paso, tras la segmentación de frases, es la separación de palabras. Son muchos los enfoques seguidos para realizar esta tarea; algunos de ellos muy básicos como la separación atendiendo a espacios en blanco o alguna expresión. Sin embargo, para el sistema propuesto en este proyecto fue necesario emplear una estrategia más elaborada, dado que los enfoques simples no cubren todos los casos posibles que permiten distinguir los términos de una oración.

Uno de los principales problemas de la *tokenización* es determinar la estrategia a seguir con símbolos como el "."o la ",". Dado que en la etapa de preprocesado ya se formatearon

[1] Se denomina serializar al proceso de codificar un objeto como un array de bytes en un medio de almacenamiento, por ejemplo un fichero.

[2] Combinaciones típicas de palabras caracterizadas por presentar un ciero grado de fijación o idiomaticidad. Así, es correcto decir *"le entraron ganas de [..]"* pero no *"le introdujeron ganas de [..]"*, aun cuando *"introducir"* y *"entrar"* son sinónimos si se consideran separadamente.

los signos de puntuación para separarse de las palabras, los únicos puntos que aparecen siguiendo o precediendo a caracteres se corresponden bien con números decimales o bien con siglas. Así, se ha optado por una aproximación que mantenga ligados estos signos de puntuación a los símbolos junto a los que aparecen. El sistema se apoya en un *tokenizador* de la clase *nltk.tokenize.PunktWordTokenizer*, que permite segmentar las palabras de una oración siguiendo la técnica comentada. El Ejemplo 5.2.2 ilustra el funcionamiento de este segmentador de palabras.

Ejemplo 5.2.2. *Tokenización* para las oraciones 5.1 y 5.2:

$O_1 tokens =$ [Ese, ordenador, es, rápido, y, fiable, ,, pero, no, recomiendo, comprarlo, .]

$O_2 tokens =$ [Sin_duda, es, muy, caro, ,, 2.700€, .]

■

Capítulo 6

Etiquetación

Tras las etapas de preparación del texto y de segmentación de oraciones y palabras, ya es posible comenzar con el proceso de etiquetación. En este capítulo se describe como se ha desarrollado esta tarea, se explican aspectos que ha sido necesario considerar para incrementar el rendimiento del etiquetador y se presentan resultados de la precisión de las alternativas probadas.

6.1. Desarrollo del etiquetador

Como ya se ha indicado anteriormente, para el desarrollo del sistema se optó por el etiquetador de Brill [7], que necesita ser entrenado. Para ello se construyó un conjunto de datos de aprendizaje a partir del Ancora Corpus Dependencies [14]. Los documentos de este *corpus* se encuentran anotados en formato CoNLL, incluyendo información en los campos CPOSTAG y FEATS, que han sido combinados en una única etiqueta para crear un conjunto de entrenamiento donde cada *token* tiene asociada mucha más información que la simple categoría gramatical[1].

Para evitar un sobreaprendizaje del corpus, el 90 % del mismo se utilizó como conjunto de entrenamiento, mientras que el 10 % restante sirvió como conjunto de test. En busca de la mejor alternativa para la siguiente etapa de análisis sintáctico, se decidió entrenar con etiquetas de distinta granularidad. Los distintos conjuntos de etiquetas completos se describen en el apéndice B.

Se realizó una evaluación del sistema con etiquetas de grano grueso, que sólo tienen en

[1] Género, número, o tiempo en caso de los verbos, es alguna de la información que se considera y que puede resultar de utilidad para el análisis sintáctico posterior.

cuenta la información del campo CPOSTAG, lo que permite distinguir entre un total de 12 etiquetas[2]. También se evaluó la precisión con una granularidad que incluye el total de la información disponible en las etiquetas. En este caso, con cada una de las distintas categorías gramaticales aparecerá información morfológica complementaria. Así, a la forma verbal *"es"* le corresponde *v* como etiqueta de grano grueso mientras que su etiqueta de grano fino sería *v:postype_semiauxiliary-num_s-person_3-mood_indicative-tense_present*.

Antes de comenzar a explicar cómo se ha implementado el proceso de etiquetación sobre la base del algoritmo de Brill, es necesario introducir dos conceptos que permiten comprender mejor como se ha llevado a cabo esta tarea:

- *Etiquetado en cadena*. Al procesar un texto, habrá palabras a las que no sea posible asignar una etiqueta. Este tipo de etiquetado funciona de forma similar a una cadena de responsabilidad. De esta manera, cuando un etiquetador no pueda anotar una palabra, es posible delegar en el siguiente integrante de la cadena, que volverá a intentarlo. Este proceso se repetirá hasta que el *token* sea anotado o hasta que la cadena se agote.

- *Etiquetado por defecto*. Este tipo de etiquetado asigna siempre la misma etiqueta a cualquier término que procese. Así, es posible crear etiquetadores que anoten todas las palabras con la etiqueta *verbo*, por ejemplo. La precisión de esta clase de etiquetadores es reducida, pero son de utilidad cuando actúan como último eslabón en las cadenas de etiquetado para asegurarse de que ningún término quede sin recibir una etiqueta.

En su implementación original, el algoritmo de Brill [7] se caracterizaba por comenzar asignando a cada *token* su etiqueta más probable. Sin embargo, en el presente trabajo se ha optado por otra aproximación, que aplicase correcciones a partir de los resultados de un etiquetador base. El reto estuvo, pues, en conseguir un etiquetador que tuviese un rendimiento aceptable por sí solo y que no consumiese ni demasiado tiempo de entrenamiento ni tampoco de etiquetación. La solución propuesta consistió en crear un sistema que para asignar la etiqueta correcta a cada palabra se basase en el contexto de la oración a la que pertenece. Para ello se han utilizado las clases del paquete *nltk.tag*, en particular las clases relacionadas con etiquetadores basadas en contexto, de modo que el etiquetador base propuesto anota los términos utilizando n-gramas tal y como se refleja en la figura 6.1. Así, una instancia de *UnigramTagger* utilizaría como contexto solamente la propia palabra para resolver la etiqueta. Un objeto de la clase *BigramTagger* utilizaría, además de la propia palabra, la etiqueta

[2]Nombre, verbo o adjetivo son algunas de las etiquetas más frecuentes.

asignada al *token* anterior; mientras que para una instancia de *TrigramTagger* el contexto estaría formado por la propia palabra, además de las dos etiquetas previas. Si se quiere utilizar un mayor número de n-gramas entonces habría que recurrir a la clase *NgramTagger*, que permite aumentar el número de etiquetas previas.

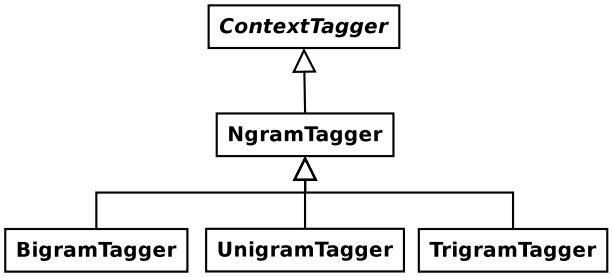


Figura 6.1: Organización jerárquica de los etiquetadores de contexto del NLTK.

Uno de los principios básicos que hay que tener en cuenta en este tipo de etiquetación es que un mayor contexto no implica una mejor precisión. La razón es que la etiqueta de un *token* es probable que no se vea condicionada por palabras relativamente lejanas, sino que seguramente sólo exista algún tipo de relación con los elementos más cercanos. Por tanto, si se emplea un contexto demasiado amplio se corre el riesgo de establecer dependencias entre etiquetas cuando realmente no las hay.

Ejemplo 6.1.1. Para reconocer en la oración bajo estas lineas que la octava palabra, *"lava"*, actúa como sustantivo y no como verbo, influye decisivamente que la séptima palabra *"mucha"* sea un determinante. Sin embargo, por ejemplo, nada tiene que ver que la primera palabra, *"El"*, sea un determinante o la tercera, *"entró"*, un verbo.

$$"_1 El\ _2 volcán\ _3 entró\ _4 en\ _5 erupción\ _6 expulsando\ _7 mucha\ _8 lava"$$

■

Utilizar un contexto amplio de manera eficaz requeriría entrenar los etiquetadores con conjuntos extremadamente grandes, de modo a considerar todas combinaciones de etiquetas posibles en un contexto de tamaño n, lo que es inviable en la práctica por la carga computacional y espacial asociada. En particular, los etiquetadores que analizan un contexto con más elementos que la propia palabra presentan un grave problema. Cuando se disponen a anotar una oración, no existe ninguna información en la que basarse al principio de la misma, lo que provoca que su precisión disminuya drásticamente; como se puede observar en el cuadro 6.1.

No obstante, gracias al etiquetado en cadena, es posible combinar todos estos etiquetadores basados en contexto para generar uno que si pueda ser utilizado en la arquitectura propuesta por Brill como etiquetador base. En busca de éste, se probaron distintas cadenas, cuya precisión queda reflejada en el cuadro 6.2.

Etiquetador	**Grano grueso**	**Grano fino**
Unigrama	**90,08 %**	**89,34 %**
Bigrama	23,37 %	15,63 %
Trigrama	13,38 %	8,64 %
Quadgrama	8,82 %	6,68 %

Cuadro 6.1: Precisión de etiquetadores basados en n-gramas

.

Cadena	**Grano grueso**	**Grano fino**
Unigrama, Brigrama	91,96 %	91,33 %
Unigrama, Brigrama, Trigrama	**92,02 %**	**91,35 %**
Unigrama, Brigrama, Trigrama, Quadgrama	91,99 %	91,30 %

Cuadro 6.2: Resultados para la búsqueda del etiquetador base de Brill

Con los resultados obtenidos, la base que se escogió para que actuase como etiquetador inicial de Brill fue la cadena formada por un *UnigramTagger*, un *BigramTagger* y un *TrigramTagger*. Se observa en el cuadro 6.2 como la cadena que utiliza un *Quadgrama* empieza a ver afectado negativamente su rendimiento, lo que confirma que un contexto demasiado amplio puede empeorar la precisión.

Como ya se explicó, tras la fase de inicialización, que en este caso correspondería al etiquetador inicial, tiene lugar una fase de aprendizaje. En este punto, se proponen una serie de reglas candidatas, siendo necesario identificar aquellas susceptibles de mejorar la precisión. Existen reglas base que suelen emplearse en el algoritmo de Brill y que examinan propiedades de los términos cercanos, como la etiqueta o la propia morfología del *token*. Para configurar estas reglas se ha utilizado el módulo *nltk.tag.brill*, y de la misma forma que en la implementación original de Brill, en la fase de entrenamiento se van seleccionando de forma iterativa cada una de las reglas. Si al aplicar un patrón el rendimiento del etiquetador mejora, entonces esa regla es incorporada.

Para conseguir mejorar aún más el rendimiento, al etiquetador inicial se le añadió otro por defecto al final de la cadena de etiquetado. En el caso del por defecto de grano grueso

se optó por etiquetar las palabras desconocidas como sustantivos, dado que era la categoría gramatical dominante en el texto y además fue con la que se obtuvo una mayor mejora. Para el de grano fino se siguió la misma estrategia, asignando en este caso la etiqueta de nombre propio a todas las palabras que llegasen a él.

Etiquetador	**Etiquetador por defecto**	**Grano grueso**	**Grano fino**
Etiquetador Base		92,02 %	91,35 %
	✓	95,39 %	93,72 %
Brill		92,35 %	91,71 %
	✓	**95,46 %**	**94,10 %**

Cuadro 6.3: Precisión de la cadena básica de etiquetado

6.2. Optimización

Los resultados obtenidos para la cadena propuesta eran ya aceptables para un idioma como el castellano, como se muestra en el cuadro 6.3, pero en busca de mejorar el rendimiento para lograr un análisis morfológico más robusto que afectase lo menos posible al análisis sintáctico y semántico posterior, dos optimizaciones fueron objeto de estudio.

6.2.1. Ampliación de la cadena de etiquetado

La implementación de Brill [7] descrita antes delegaba en el etiquetador por defecto cualquier palabra que no fuese capaz de anotar, asignándole una etiqueta de nombre o nombre propio según el conjunto de etiquetas considerado fuese de grano grueso o de grano fino. Sin embargo, la precisión del proceso no se incrementó en exceso, ya que muchos de los *tokens* desconocidos no se correspondían en realidad con sustantivos.

La solución propuesta para evitar que tantas palabras desconocidas se anotasen incorrectamente consistió en ampliar la cadena de etiquetado, incluyendo un nuevo eslabón, que actuase entre el etiquetador de Brill propiamente dicho y el utilizado por defecto. Para este etiquetador se optó por un enfoque que siguiese la misma estrategia que la base de Brill, salvo que ahora las etiquetas se asignaban atendiendo a prefijos y sufijos de las palabras, en vez de emplear n-gramas. Se trata de una aproximación que suele funcionar bien para la anotación de palabras desconocidas en castellano. En este idioma es común que los sufijos

ayuden en muchas ocasiones a determinar la categoría gramatical de una palabra. Algunos ejemplos típicos se muestran en el cuadro 6.4.

Categoría gramatical	**Sufijos**
Sustantivos	-ada, -algia, -amen, -anza, -fobía, etc
Adjetivos	-aco, -aje, -ego, etc
Verbos	-ar, -er, -ir, -ando, -endo, etc

Cuadro 6.4: Algunos sufijos que determinan la categoría gramatical

En este contexto se optó por crear un etiquetador que aprendiese a anotar atendiendo a las subcadenas iniciales y finales de cada palabra, siendo el principal problema de esta estrategia determinar la longitud adecuada de los prefijos y sufijos a analizar. En el cuadro 6.5 se muestra la combinación que con la que se consiguió el mejor etiquetador basado en afijos, refiriéndonos con *preN* a un etiquetador que analiza los N primeros caracteres de una palabra y con *sufN* a uno que tiene en cuenta las N últimos.

Cadena	**Grano grueso**	**Grano fino**
pre1	28,19 %	18,30 %
pre1,pre2	32,75 %	23,08 %
pre1,pre2,pre3	38,27 %	28,37 %
pre1,pre2,pre3,pre4	41,7 %	32,64 %
pre1,pre2,pre3,pre4,pre5	43,19 %	34,90 %
pre1,pre2,pre3,pre4,pre5,suf3	41,58 %	37,78 %
pre1,pre2,pre3,pre4,pre5,suf3,suf4	44,21 %	41,49 %
pre1,pre2,pre3,pre4,pre5,suf3,suf4,suf5	46,12 %	44,20 %
pre1,pre2,pre3,pre4,pre5,suf3,suf4,suf5 + *tagger* por defecto	**46,12 %**	**44,37 %**

Cuadro 6.5: Precisión de etiquetadores en cadena basados en prefijos y sufijos

El nuevo eslabón en la cadena provoca que el etiquetador por defecto apenas tenga influencia en el sistema. De hecho, sólo mejora ligeramente si se está considerando el conjunto de etiquetas de grano fino. Aún así, se incluyó en la implementación para asegurarse de que no quede ninguna palabra sin etiqueta.

6.2.2. Conjunto de entrenamiento adaptado a un entorno web

Como se comentó anteriormente, la mala calidad de entrada de los textos y en concreto la falta de tildes, es un factor que influye en el rendimiento real de un etiquetador. Se ob-

servó que aunque la precisión teórica de los algoritmos probados era buena, ésta disminuía drásticamente en textos sin acentuar. En un entorno de MO es habitual que muchos usuarios no respeten las normas ortográficas, lo que para un sistema de análisis del sentimiento supone una dificultad añadida, y aunque ya se ha comentado como se corrigieron y procesaron algunos de los errores típicos, nada se ha dicho en relación a cómo tratar la ausencia de tildes.

Al respecto, nuestra opción se basó en resolver el problema desde el etiquetador. La solución propuesta consistió en modificar el *corpus* de entrenamiento de forma que cada oración tenga su equivalente con todas las palabras sin acentuar gráficamente, lo que parece no afectar en exceso a la precisión del sistema, una consecuencia de que en castellano es relativamente rara la aparición de palabras con tildes diacríticas[3]. Es decir, en muchos de los casos no existirá una mayor posibilidad de confusión para el etiquetador, aunque la forma correcta de la palabra llevase tilde y aún existiendo dicha confusión; el etiquetador base de Brill basado en n-gramas es capaz de resolver la ambigüedad léxica en la mayoría de las ocasiones.

Etiquetador	**Etiquetador de afijos**	**Corpus$_{modificado}$**	**Grano grueso**	**Grano fino**
Brill			92,36 %	91,71 %
	✓		**97,23 %**	**95,86 %**
		✓	92,30 %	91,63 %
	✓	✓	97,12 %	95,71 %

Cuadro 6.6: Precisión de la cadena de etiquetado optimizada

Los resultados del cuadro 6.6 indican que para el conjunto de etiquetas de grano grueso el rendimiento es mejor, pero aun así en el sistema se utilizó un etiquetador de grano fino por la mayor información morfológica que proporciona al analizador sintáctico. Conocer el género, número o el tiempo verbal concreto permite obtener una mayor precisión en el análisis de dependencias porque las relaciones entre componentes resultan más claras.

Referente a la cadena de etiquetado usada en el proyecto, se empleó tanto el etiquetador inspirado en prefijos y sufijos como el conjunto de entrenamiento ampliado con las palabras desacentuadas. Aunque no fuese la opción de configuración con la que se obtuviese una mejor precisión teórica, su rendimiento demostró ser superior por las características del entorno web en las que trabaja el sistema[4]. A modo de resumen, en la figura 6.2.2 se muestra la estructura global de la cadena de etiquetado en la que se apoyó el analizador sintáctico para obtener los

[3]Tildes colocadas para diferenciar palabras que aun escribiéndose igual tienen significados distintos.
[4]Como las palabras sin acentuar.

árboles de dependencias de las oraciones.

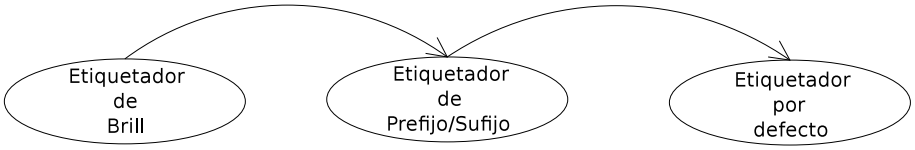


Figura 6.2: Arquitectura del etiquetador del sistema

La cadena de etiquetado actúa siempre tras las etapas de segmentación y *tokenización*, que como ya se comentó, formatean las oraciones del texto para que el etiquetador pueda procesarlas. En el Ejemplo 6.2.1 se muestra la salida tras la etapa análisis léxico para dos oraciones empleando el conjunto de etiquetas de grano fino.

Ejemplo 6.2.1. Etiquetación correcta[5] para las sentencias 5.1 y 5.2, una vez han sido *tokenizadas*:

$O_1 etiquetas =$	[(**Ese**	d:postype_demonstrative-gen_m-num_s)
	(**ordenador**	n:postype_common-gen_m-num_s)
	(**es**	v:postype_semiauxiliary-num_s-person_3-mood_indicative-tense_present),
	(**rápido**	a:postype_qualificative-gen_m-num_s),
	(**y**	c:postype_coordinating),
	(**fiable**	a:postype_qualificative-gen_c-num_s),
	(**,**	f:punct_comma),
	(**pero**	c:postype_coordinating),
	(**no**	r:postype_negative),
	(**recomiendo**	v:postype_main-num_s-person_1-mood_indicative-tense_present)
	(**comprarlo**	v:postype_main-mood_infinitive),
	(**.**	f:punct_period)]

[5]Hay que recordar que la precisión teórica del etiquetador utilizado es de aproximadamente un 97 % por lo que en la práctica habrá palabras que no se anoten correctamente.

$O_2 etiquetas =$ [(**Sin_duda** r)
(**es** v:postype_semiauxiliary-num_s-person_3-mood_indicative-tense_present))
(**muy** r)
(**caro** a:postype_qualificative-gen_m-num_s)
(**,** f:punct_comma)
(**2.700€** z)
(**.** f:punct_period)]

■

Capítulo 7

Análisis sintáctico de dependencias

Desarrollar un sistema de MO capaz de interpretar y comprender de forma efectiva un texto, requiere emplear técnicas de análisis sintáctico capaces de determinar las relaciones entre los componentes de las oraciones más allá del simple conocimiento léxico, técnica habitual en los entornos de MO actuales. En este capítulo se explica cómo se ha desarrollado esta tarea, describiendo el algoritmo empleado y centrándose en la búsqueda de un modelo óptimo para el castellano.

7.1. Introducción

Como ya se ha comentado, MaltParser [13] es el software en el que se apoya nuestra propuesta en lo que a extracción de árboles de dependencias en textos se refiere. Antes de comenzar a explicar el proceso aplicado es necesario introducir algunos conceptos relativos a esta herramienta:

- MaltParser es un generador automático de analizadores de dependencias que permite obtener modelos a partir de un conjunto de entrenamiento para poder analizar a continuación nuevas oraciones. El conjunto de entrenamiento debe representarse en un formato estándar, por ejemplo cualquiera de los formatos CoNLL-X o el MALT-TAB[1]. Nótese que los documentos del Ancora Corpus Dependencies [14] ya se encuentran representados siguiendo la notación CoNLL [9], lo que ha permitido que se puedan utilizar como conjunto de entrenamiento. MaltParser implementa hasta 9 algoritmos determinísticos de análisis de dependencias entre los que se encuentra el *Nivre arc-eager* [8]

[1] Representación alternativa a los formatos CoNLL-X para codificar árboles de dependencias como ficheros XML.

que ha sido el empleado en este proyecto, y que será detallado en este mismo capítulo.

- Todo modelo de análisis de dependencias generado por este software necesita de un *modelo de características*, que en esencia es un fichero XML que permite al analizador identificar durante la fase de aprendizaje y de análisis cuáles son los campos a tener en cuenta para establecer las dependencias. Configurar un modelo de características adecuado es un proceso largo y complejo que requiere probar múltiples alternativas. Aunque MaltParser proporciona un fichero XML por defecto para cada uno de los algoritmos que implementa, éste no es válido si lo que se quiere es obtener un analizador sintáctico óptimo para un lenguaje. Para comprender mejor como funciona y cómo se estructura esta tarea se detalla un caso sencillo en el ejemplo 7.1.2.

7.1.1. El algoritmo *Nivre arc-eager*

El *Nivre arc-eager* es un método de análisis sintáctico de dependencias basado en un autómata de pila que sirve de motor a una estrategia de avance/reducción[2]. Antes de describir el funcionamiento del algoritmo, es necesario introducir algunas definiciones.

Definición 7.1.1. Sea $S = [w_0, w_1, ..., w_n]$ la lista de palabras de una oración con w_0 como el nodo artificial, ROOT, y sea $R = \{r_0, r_1, ..., r_m\}$ el conjunto de todos los tipos de dependencias[3], se define un *estado* o *configuración* del autómata de pila en un momento dado como una tupla $c = (\sigma, \beta, A)$ donde:

1. σ es la pila de palabras $w_i \in S$ que han sido procesadas hasta ese momento.
2. β es un *buffer* que representa las palabras $w_j \in S$ que aún quedan por analizar.
3. A es el conjunto de las relaciones de dependencia establecidas, es decir, el fragmento del árbol de dependencias construido en ese instante.
4. $c_0 = ([w_0], [w_1, ..., w_n], \varnothing)$ es el único *estado inicial* para una oración S.
5. $(\sigma, [\,], A)$ es un *estado final* para cualquier σ y para cualquier A.

■

El objetivo del algoritmo es partir del estado inicial hasta llegar a uno de los estados finales, donde A constituirá el árbol de dependencias para la oración. Para ello, se aplican

[2]Para la definición formal de las transiciones se hace uso de la notación anglosajona *shift/reduce.*

[3]Las dependencias sintácticas utilizadas por el sistema, pueden encontrarse en el apéndice B.2.

una serie de transiciones que permiten cambiar de estado. En concreto, este método trabaja con cuatro tipos distintos de transiciones, definidas formalmente en el cuadro 7.1, y que hace uso de la siguiente notación:

- $\sigma|w_i$ indica que el *token* w_i se encuentra en la cima de pila.
- $w_j|\beta$ representa la lista de palabras que no han sido procesadas, donde w_j es la cabeza de la misma.

Transición		**Precondiciones**
Left-Arc$_r$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_j\|\beta, A \cup \{(w_j, r, w_i)\})$	$(w_k, r', w_i) \notin A \wedge i \neq 0$
Right-Arc$_r$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma\|w_i\|w_j, \beta, A \cup \{(w_i, r, w_j)\})$	
Reduce	$(\sigma\|w_i, \beta, A) \Rightarrow (\sigma, \beta, A)$	$(w_k, r', w_i) \in A$
Shift	$(\sigma, w_i\|\beta, A) \Rightarrow (\sigma\|w_i, \beta, A)$	

Cuadro 7.1: Transiciones del algoritmo de análisis sintáctico *Nivre arc-eager*

Más en detalle se comentan a continuación cada una de las transiciones incluidas en el cuadro 7.1:

1. Left-Arc$_r$: Crea una relación de dependencia entre la cima de la pila, que actúa como dependiente, y la primera palabra dentro de β, para a continuación desapilarla. Este tipo de transición requiere que el *buffer* y la pila no estén vacíos y w_i no puede ser el ROOT. No puede existir ninguna relación de dependencia previa donde la cima de la pila actúe como término subordinado.

2. Right-Arc$_r$: Construye una relación de dependencia donde la cabeza de la pila actúa como padre y la primera palabra de β como dependiente. Se apila la cabeza del *buffer* en σ, quitándola del *buffer*. Tanto la pila como el *buffer* han de contener algún elemento.

3. Shift: Apila en σ la primera palabra del *buffer*, quitándola del mismo.

4. Reduce: Desapila de σ. Debe existir una relación de dependencia donde la cima de la pila actúe como dependiente.

Para comprender como funcionan estas transiciones en el Ejemplo 7.1.1 se muestra el procesamiento para una oración sencilla.

Ejemplo 7.1.1. Secuencia de transiciones correctas para obtener el árbol de dependencias de la sentencia 5.1, siguiendo el algoritmo de análisis *Nivre arc-eager*.

	Transición	σ	β	A
0		[ROOT]	[Ese,...,comprarlo, .]	∅
1	Shift	[ROOT, Ese]	[ordenador,...,comprarlo, .]	∅
2	Left-Arc$_{spec}$	[ROOT]	[ordenador, ...,comprarlo, .]	A$_1$ = {(ordenador, **spec**, ese)}
3	Shift	[ROOT, ordenador]	[es,...,comprarlo, .]	A$_1$
4	Left-Arc$_{suj}$	[ROOT]	[es, ..., comprarlo, .]	A$_2$ = A$_1$ ∪ {(es, **suj**, ordenador)}
5	Right-Arc$_{sentence}$	[ROOT, es]	[rápido, ..., comprarlo, .]	A$_3$ = A$_2$ ∪ {(ROOT, **sentence**, es)}
6	Shift	[ROOT, es,rápido]	[y, ..., comprarlo, .]	A$_3$
7	Right-Arc$_{coord}$	[ROOT, es, rápido, y]	[fiable,..., comprarlo, .]	A$_4$ = A$_3$∪ {(rápido,**coord** ,y)}
8	Reduce	[ROOT, es, rápido]	[fiable, ..., comprarlo, .]	A$_4$
9	Right-Arc$_{grup.a}$	[ROOT, es, rápido, fiable]	[,, ...comprarlo, .]	A$_5$ = A$_4$∪ {(rápido, **grup.a**, fiable)}
10	Reduce	[ROOT, es, rápido]	[,, pero,...comprarlo, .]	A$_5$
11	Reduce	[ROOT, es]	[,...,comprarlo, .]	A$_5$
12	Right-Arc$_f$	[ROOT, es, ,]	[pero,..., comprarlo, .]	A$_6$ = A$_5$∪ {(es, **f**, ,)}
13	Reduce	[ROOT, es]	[pero,...,comprarlo, .]	A$_6$
14	Right-Arc$_{coord}$	[ROOT, es, pero]	[no, ..., comprarlo, .]	A$_7$ = A$_6$∪{(es, **coord** , pero)}
15	Reduce	[ROOT, es]	[no, ..., comprarlo, .]	A$_7$
16	Shift	[ROOT, es, no]	[recomiendo ,comprarlo, .]	A$_7$
17	Left-Arc$_{mod}$	[ROOT, es]	[recomiendo, comprarlo, .]	A$_8$ = A$_7$∪ {(recomiendo, **mod**, no }
18	Shift	[ROOT, es, recomiendo]	[comprarlo, .]	A$_8$
19	Left-Arc$_v$	[ROOT, es]	[comprarlo, .]	A$_9$ = A$_8$∪ {(comprarlo,**v**, recomiendo)}
20	Right-Arc$_S$	[ROOT, es, comprarlo]	[.]	A$_{10}$ = A_9∪ { (es, **S**, comprarlo)}
21	Reduce	[ROOT, es]	[.]	A$_{10}$
22	Right-Arc$_f$	[ROOT, es, .]	[ ]	A$_{11}$ = A_{10}∪ { (es, **f**, .) }

Cuadro 7.2: Secuencia de transiciones del *Nivre arc-eager* para la oración 5.1

■

El gran problema que hay que resolver para implementar en la práctica este tipo de métodos resulta de la posibilidad de aplicar en un estado más de una transición, sin que todas sean correctas. Es necesario por tanto saber seleccionar la adecuada en cada momento para poder obtener un árbol de dependencias correcto. Para reducir el impacto de este indeterminismo latente se hace uso de técnicas de aprendizaje automático, donde un modelo de características indica cuáles son los aspectos en los que deberá fijarse el modelo para tomar

la decisión correcta.

En el siguiente apartado se explica cómo se estructura un modelo de características en MaltParser, y se presenta un ejemplo que permite comprender con claridad cómo actúan las características en cada estado.

7.1.2. Estructura de un modelo de características

Como ya se comentó en la introducción de este capítulo, los modelos de características son un aspecto fundamental durante la fase de aprendizaje, ya que permiten al analizador saber cuál es la transición correcta en cada momento. MaltParser proporciona un modelo de características por defecto para cada algoritmo que implementa. Es aquí importante recordar de nuevo que tanto el conjunto de entrenamiento como los ficheros que se analizarán más tarde deben estar en un formato estándar. Así, los modelos de características se refieren a determinados campos presentes en este tipo de ficheros. En este sentido, antes de ver un ejemplo, se define cuál es la sintaxis para recuperar un determinado atributo presente en estos ficheros. Sean $e \in$ {POSTAG,FORM,LEMMA,CPOSTAG,FEATS} y s $\in$ {HEAD, DEPREL}, entonces:

- InputColumn(e, Stack[i]) recupera el valor del atributo e dentro del fichero CoNLL para el término de la oración en la posición i de la pila, donde 0 indica la cima de la pila.
- InputColumn(e, Input[j]) recupera el valor del atributo e para la palabra situada en la posición j del *buffer* de entrada, donde 0 indica la primera palabra disponible en dicho *buffer*.
- OutputColumn(s, Stack[i]) obtiene el valor del atributo de salida s para el término de la posición i de la pila, donde 0 indica la cima de la pila.
- OutputColumn(s, Input[j]) extrae el atributo s para la palabra situada en la posición j del *buffer* de entrada, donde 0 indica la primera palabra disponible en el *buffer*.

Existen además una serie de funciones que permiten recuperar elementos relacionados con los elementos de la pila o del *buffer*. Dentro de la guía de usuario[4] es posible encontrar la lista completa, pero a continuación se resumen las más comunes y las que han sido empleadas en la búsqueda del modelo óptimo para el sistema de MO:

- head(x) recupera el padre del elemento x.

[4] http://www.maltparser.org/userguide.html#featurespec Visitado por última vez en agosto de 2012.

- ldep(x) recupera el hijo más a la izquierda del elemento x.
- rdep(x) recupera el hijo más a la derecha del elemento x.
- pred(x) devuelve el elemento con el ID previo en la oración. Si no hay ninguno devolverá nulo.
- succ(x) devuelve el elemento con el ID siguiente al término x, o nulo si no hay ninguno.

El código XML debajo de estas lineas representan las características por defecto que MaltParser proporciona para el algoritmo *Nivre arc-eager*.

```
<featuremodels>
	<featuremodel name="nivreeager">
		<feature>InputColumn(POSTAG, Stack[0])</feature>
		<feature>InputColumn(POSTAG, Input[0])</feature>
		<feature>InputColumn(POSTAG, Input[1])</feature>
		<feature>InputColumn(POSTAG, Input[2])</feature>
		<feature>InputColumn(POSTAG, Input[3])</feature>
		<feature>InputColumn(POSTAG, Stack[1])</feature>
		<feature>OutputColumn(DEPREL, Stack[0])</feature>
		<feature>OutputColumn(DEPREL, ldep(Stack[0]))</feature>
		<feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
		<feature>OutputColumn(DEPREL, ldep(Input[0]))</feature>
		<feature>InputColumn(FORM, Stack[0])</feature>
		<feature>InputColumn(FORM, Input[0])</feature>
		<feature>InputColumn(FORM, Input[1])</feature>
		<feature>InputColumn(FORM, head(Stack[0]))</feature>
	</featuremodel>
</featuremodels>
```

Ejemplo 7.1.2. Se muestran ahora de modo orientativo, en el cuadro 7.3, las secuencias de características[5] que se tendrían en cuenta para obtener el árbol del Ejemplo 7.1.1, si para ello se consideraran los campos indicados en el siguiente código XML[6]:

```
<featuremodels>
	<featuremodel name="nivreeager">
		<feature>InputColumn(CPOSTAG, Stack[0])</feature>
		<feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
		<feature>InputColumn(FORM, Stack[0])</feature>
		<feature>InputColumn(FORM, Input[0])</feature>
		<feature>InputColumn(FORM, Input[1])</feature>
	</featuremodel>
</featuremodels>
```

Features$_i$	**Stack[0]**$_{form}$	**Stack[0]**$_{postag}$	**Input[0]**$_{form}$	**Input[1]**$_{form}$	**ldep(Stack[0])**$_{deprel}$
f_0	ROOT	null	Ese	ordenador	null
f_1	Ese	d	ordenador	es	null
f_2	ROOT	null	ordenador	es	null
f_3	ordenador	n	es	rápido	spec
f_4	ROOT	null	es	rápido	null
f_5	es	v	rápido	y	suj
f_6	rápido	a	y	fiable	null
f_7	y	c	fiable	,	null
f_8	rápido	a	fiable	y	coord
f_9	fiable	a	,	pero	null
f_{10}	rápido	a	,	pero	grup.a
f_{11}	es	v	,	pero	atr
f_{12}	,	f	pero	no	null
f_{13}	es	a	pero	no	f
f_{14}	pero	c	no	recomiendo	null
f_{15}	es	v	no	recomiendo	coord
f_{16}	no	r	recomiendo	comprarlo	null
f_{17}	es	v	recomiendo	comprarlo	coord
f_{18}	recomiendo	v	comprarlo	.	mod
f_{19}	es	v	comprarlo	.	coord
f_{20}	comprarlo	v	.	null	null
f_{21}	es	v	.	null	S
f_{22}	.	f	null	null	null

Cuadro 7.3: Secuencias de características utilizadas para la oración del cuadro 7.2

■

[5]Una celda con valor *null* indica que ese valor no está disponible.

[6]Se usa un modelo de características sencillo para comprender el ejemplo.

7.2. Generación de un modelo de análisis para el castellano

Tal y como se comentó anteriormente, el 90 % del *corpus* fue utilizado para desarrollar el modelo. No obstante, dada la variedad de modelos que fue necesario evaluar y dado el elevado tiempo de aprendizaje de MaltParser, se redujo el tamaño del *corpus* de entrenamiento original para abordar esta tarea en un tiempo razonable. Así, todos los modelos han sido entrenados con un *corpus* cuatro veces más pequeño, y de igual forma se redujo el tamaño del conjunto de evaluación. Para los modelos para los que se obtuvieron los mejores resultados sí se realizó un entrenamiento completo para obtener unos resultados de evaluación más fiables.

El modelo de características es otro factor crucial para el entrenamiento del analizador. Para el desarrollo de esta parte del sistema se siguieron tres enfoques distintos para tratar de obtener el mejor modelo posible:

1. CPOSTAG + FORM: Estrategia centrada en modelos donde sólo se tienen en cuenta las etiquetas de grano grueso, y las propias formas de los términos, para tratar de seleccionar la transición correcta.

2. POSTAG + FORM: Alternativa centrada en considerar las etiquetas de grano fino como forma de decidir una nueva configuración.

3. CPOSTAG + FEATS + FORM: Una versión ampliada del primer enfoque, donde además se tienen en cuenta las propias características.

Previamente hemos comentado que el Ancora Corpus Dependencies [14] carecía de información en su campo POSTAG y que éste había sido creado combinando la información de CPOSTAG y FEATS. Teóricamente ello supone que las alternativas 2 y 3 emplean la misma información. Sin embargo, durante el entrenamiento de los distintos modelos se observó que el comportamiento no era el mismo y por eso se consideraron las dos estrategias por separado. Los modelos entrenados están listos ahora para analizar nuevos textos, cuya salida se vuelca en un fichero de texto plano en formato CoNLL.

Ejemplo 7.2.1. Representación del texto de las sentencias 5.1 y 5.2 en formato CoNLL (cuadro 7.4) e ilustración de dichas sentencias gráficamente como árboles de dependencias (figuras 7.1 y 7.2).

ID	FORM	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL
1	Ese	d	d:postype_demonstrative-gen_m-num_s	postype=demonstrative|gen=m|num=s	2	spec
2	ordenador	n	n:postype_common-gen_m-num_s	postype=common|gen=m|num=s	3	suj
3	es	v	v:postype_semiauxiliary-num_s-person_3-mood_indicative-tense_present	postype=semiauxiliary|num=s|person=3|mood=indicative|tense=present	0	sentence
4	rápido	a	a:postype_qualificative-gen_m-num_s	postype=qualificative|gen=m|num=s	3	atr
5	y	c	c:postype_coordinating	postype=coordinating	4	coord
6	fiable	a	a:postype_qualificative-gen_c-num_s	postype=qualificative|gen=c|num=s	4	grup.a
7	,	f	f:punct_comma	punct=comma	3	f
8	pero	c	c:postype_coordinating	postype=coordinating	3	coord
9	no	r	r:postype_negative	postype=negative	11	mod
10	recomiendo	v	v:postype_main-mood_gerund	postype=main|mood=gerund	11	v
11	comprarlo	v	v:postype_main-mood_infinitive	postype=main|mood=infinitive	3	S
12	.	f	f:punct_period	punct=period	3	f
1	Sin_duda	r	r	_	2	mod
2	es	v	v:postype_semiauxiliary-num_s-person_3-mood_indicative-tense_present	postype=semiauxiliary|num=s|person=3|mood=indicative|tense=present	0	sentence
3	muy	r	r	_	4	spec
4	caro	a	a:postype_qualificative-gen_m-num_s	postype=qualificative|gen=m|num=s	2	atr
5	,	f	f:punct_comma	punct=comma	6	f
6	2700€	z	z	_	2	cc
7	.	f	f:punct_period	punct=period	2	f

Cuadro 7.4: Árbol de dependencias en formato CoNLL 2006 de la oración 5.1

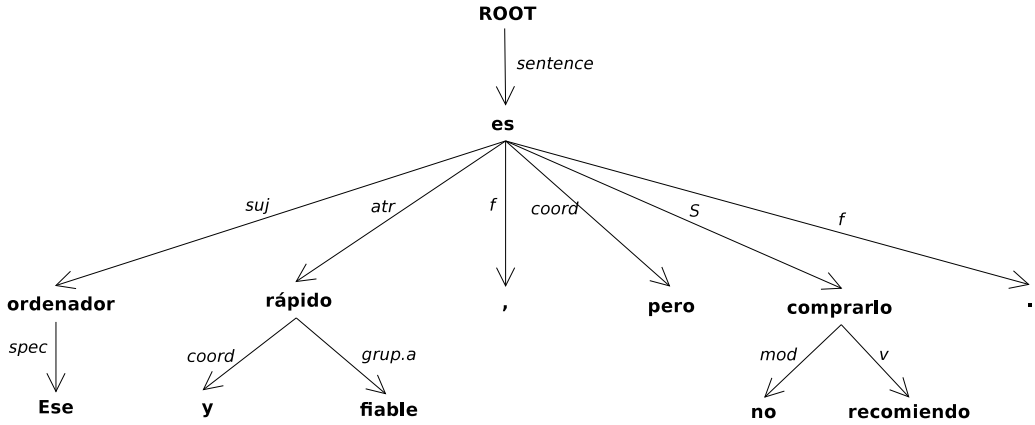


Figura 7.1: Árbol de dependencias para la oración 5.1

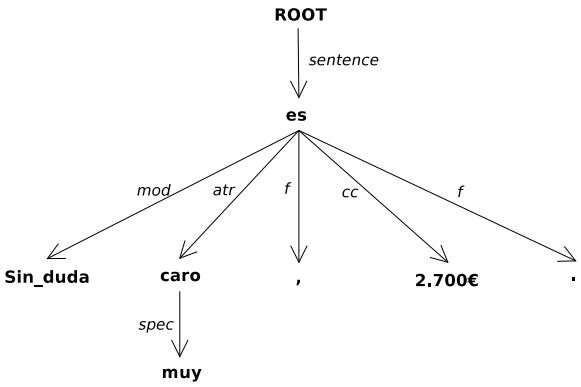


Figura 7.2: Árbol de dependencias para la oración 5.2

■

7.3. Evaluación

Existen distintas formas para medir el rendimiento de un analizador sintáctico [8], pero la medida más empleada es sin duda la *Attachment score*. Ésta se ha convertido en un estándar para la evaluación de estos entornos y su cometido es medir el porcentaje de palabras a las que se les ha asignado correctamente el padre en el árbol de dependencias. Se dispone de dos variantes de esta métrica, según se tenga en cuenta el tipo de dependencia asignada o no. Así, se distingue:

- LAS *(Labeled Attachment Score)*: Métrica que mide el porcentaje de palabras a las que tanto el padre como el tipo de dependencia fueron asignados correctamente.

- UAS *(Unlabeled Attachment Score)*: Métrica que sólo tiene en cuenta que el padre de la relación de dependencia esté bien asignado.

Existe una tercera medida también usada con frecuencia, la *Label Accuracy Score*, LAS2, que mide sólo que el tipo de dependencia haya sido correctamente anotado. En la evaluación de los modelos se tuvieron en cuenta las tres métricas explicadas, LAS, UAS y LAS2, para disponer de mayor información sobre el comportamiento de los analizadores. No obstante, para el sistema de MO es importante que tanto el padre como la relación sintáctica estén asignadas correctamente para procesar los sintagmas, por ello se ha prestado mayor atención a la métrica LAS.

Comentadas las medidas consideradas para la selección de los mejores modelos, bajo estas líneas se enumeran aquellos para los que se obtuvo una mayor precisión con la métrica LAS, para cada una de la tres estrategias seguidas[7]. Para la evaluación se empleó el *script* oficial de la CoNLL-X SHARED TASK [9], *eval.pl*[8], que detalla toda clase de información sobre el modelo generado, como la propia LAS o detalles estadísticos sobre los fallos del analizador. En la estrategia CPOSTAG+FEATS+FORM se detallan dos modelos dado que se obtuvo prácticamente el mismo rendimiento para ambos.

- Estrategia CPOSTAG + FORM: Entre un total de 26 alternativas distintas, el modelo de características con el que se obtuvo mayor precisión fue:

```
<featuremodels>
	<featuremodel name="nivreeager">
		<feature>InputColumn(CPOSTAG, Stack[0])</feature>
		<feature>InputColumn(CPOSTAG, Input[0])</feature>
		<feature>InputColumn(CPOSTAG, Input[1])</feature>
		<feature>InputColumn(CPOSTAG, Input[2])</feature>
		<feature>InputColumn(CPOSTAG, Input[3])</feature>
		<feature>InputColumn(CPOSTAG, Stack[1])</feature>
		<feature>InputColumn(CPOSTAG, Stack[2])</feature>
		<feature>InputColumn(CPOSTAG, Stack[3])</feature>
		<feature>OutputColumn(DEPREL, Stack[0])</feature>
		<feature>OutputColumn(DEPREL, ldep(Stack[0]))</feature>
		<feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
```

[7]Con el conjunto de entrenamiento reducido antes comentado.

[8]http://ilk.uvt.nl/conll/software.html Visitado por última vez en agosto de 2012.

```
            <feature>OutputColumn(DEPREL, ldep(Input[0]))</feature>
            <feature>InputColumn(FORM, Stack[0])</feature>
            <feature>InputColumn(FORM, Input[0])</feature>
            <feature>InputColumn(FORM, Input[1])</feature>
            <feature>InputColumn(FORM, Stack[1])</feature>
            <feature>InputColumn(FORM, head(Stack[0]))</feature>
        </featuremodel>
</featuremodels>
```

- Estrategia POSTAG+FORM: Se probaron 23 modelos distintos sin que ninguno de ellos consiguiera mejorar significativamente los resultados obtenidos para la alternativa CPOSTAG+ FORM. Uno de los pocos que logró mejorar ligeramente esa precisión es el que se muestra bajo estas líneas.

```
<featuremodels>
        <featuremodel name="nivreeager">
            <feature>InputColumn(POSTAG, Stack[0])</feature>
            <feature>InputColumn(POSTAG, Input[0])</feature>
            <feature>InputColumn(POSTAG, Input[1])</feature>
            <feature>InputColumn(POSTAG, Input[2])</feature>
            <feature>InputColumn(POSTAG, Input[3])</feature>
            <feature>InputColumn(POSTAG, Stack[1])</feature>
            <feature>InputColumn(POSTAG, succ(Stack[0]))</feature>
            <feature>InputColumn(POSTAG, head(succ(Stack[0])))</feature>
            <feature>OutputColumn(DEPREL, Stack[0])</feature>
            <feature>OutputColumn(DEPREL, ldep(Stack[0]))</feature>
            <feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
            <feature>OutputColumn(DEPREL, ldep(Input[0]))</feature>
            <feature>InputColumn(FORM, Stack[0])</feature>
            <feature>InputColumn(FORM, Input[0])</feature>
            <feature>InputColumn(FORM, Input[1])</feature>
            <feature>InputColumn(FORM, head(Stack[0]))</feature>
        </featuremodel>
</featuremodels>
```

- Estrategia CPOSTAG+FEATS+FORM: Se evaluaron 15 modelos distintos, muchos de ellos mejorando cualquiera de los resultados obtenidos con anterioridad. En concreto hubo dos modelos para los que se consiguió casi la misma precisión, ambos se ilustran en formato XML a continuación. Con objeto de distinguirlos al primer modelo se le denomina m_1 y al segundo m_2.

  m_1:

```
<featuremodels>
        <featuremodel name="nivreeager">
```

```
                <feature>InputColumn(CPOSTAG, Stack[0])</feature>
                <feature>InputColumn(CPOSTAG, Input[0])</feature>
                <feature>InputColumn(CPOSTAG, Input[1])</feature>
                <feature>InputColumn(CPOSTAG, Input[2])</feature>
                <feature>InputColumn(CPOSTAG, Input[3])</feature>
                <feature>InputColumn(CPOSTAG, Stack[1])</feature>
                <feature>InputColumn(CPOSTAG, Stack[2])</feature>
                <feature>InputColumn(CPOSTAG, Stack[3])</feature>
                <feature>OutputColumn(DEPREL, Stack[0])</feature>
                <feature>OutputColumn(DEPREL, ldep(Stack[0]))</feature>
                <feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
                <feature>OutputColumn(DEPREL, ldep(Input[0]))</feature>
                <feature>Split(InputColumn(FEATS,Stack[0]),\|)</feature>
                <feature>Split(InputColumn(FEATS,Input[0]),\|)</feature>
                <feature>Split(InputColumn(FEATS,Input[1]),\|)</feature>
                <feature>InputColumn(FORM, Stack[0])</feature>
                <feature>InputColumn(FORM, Input[0])</feature>
                <feature>InputColumn(FORM, Input[1])</feature>
                <feature>InputColumn(FORM, Stack[1])</feature>
                <feature>InputColumn(FORM, head(Stack[0]))</feature>
        </featuremodel>
</featuremodels>
```

m_2:

```
<featuremodels>
        <featuremodel name="nivreeager">
                <feature>InputColumn(CPOSTAG, Stack[0])</feature>
                <feature>InputColumn(CPOSTAG, Input[0])</feature>
                <feature>InputColumn(CPOSTAG, Input[1])</feature>
                <feature>InputColumn(CPOSTAG, Input[2])</feature>
                <feature>InputColumn(CPOSTAG, Input[3])</feature>
                <feature>InputColumn(CPOSTAG, Stack[1])</feature>
                <feature>InputColumn(CPOSTAG, Stack[2])</feature>
                <feature>InputColumn(CPOSTAG, Stack[3])</feature>
                <feature>OutputColumn(DEPREL, Stack[0])</feature>
                <feature>OutputColumn(DEPREL, ldep(Stack[0]))</feature>
                <feature>OutputColumn(DEPREL, rdep(Stack[0]))</feature>
                <feature>OutputColumn(DEPREL, ldep(Input[0]))</feature>
                <feature>Split(InputColumn(FEATS,Stack[0]),\|)</feature>
                <feature>Split(InputColumn(FEATS,Input[0]),\|)</feature>
                <feature>Split(InputColumn(FEATS,Input[1]),\|)</feature>
                <feature>Split(InputColumn(FEATS,Input[2]),\|)</feature>
                <feature>InputColumn(FORM, Stack[0])</feature>
                <feature>InputColumn(FORM, Input[0])</feature>
                <feature>InputColumn(FORM, Input[1])</feature>
                <feature>InputColumn(FORM, Stack[1])</feature>
                <feature>InputColumn(FORM, head(Stack[0]))</feature>
```

```
        </featuremodel>
    </featuremodels>
```

Estrategia	**LAS(%)**	**UAS(%)**	**LASC(%)**
CPOSTAG+FORM	78,38	84,90	83,92
POSTAG+FORM	78,55	84,65	83,84
CPOSTAG+FEATS+FORM$_{m_1}$	79,72	85,29	85,42
CPOSTAG+FEATS+FORM$_{m_2}$	79,71	85,32	85,32

Cuadro 7.5: Precisión de modelos sobre el *corpus* reducido

Las precisiones de estos modelos, mostradas en el cuadro 7.5, indican que la estrategia CPOSTAG+FEATS+FORM logra un rendimiento sensiblemente superior a las demás alternativas, tantos en las métricas LAS y UAS como en la LAS2. Sobre los dos mejores modelos de esta alternativa sí se realizó un entrenamiento sobre el *corpus* inicial, formado por el 90 % del *corpus* Ancora, y la evaluación, sobre el 10 % restante. La evaluación con el nuevo conjunto de entrenamiento obtuvo de nuevo resultados muy parecidos, aunque m_1 se comportó ligeramente mejor, por lo que fue el modelo de características seleccionado para nuestro sistema de MO. En el cuadro 7.6 se comparan los analizadores generados a partir de m_1 y m_2 con el conjunto de entrenamiento original y en el cuadro 7.7 se detalla más en detalle el LAS para m_1, desglosando la precisión según los distintos tipos de dependencias existentes, donde las columnas tienen el siguiente significado:

- *Total*: Nº de dependencias de ese tipo en el *corpus* de evaluación.
- *Correctas*: Dependencias de ese tipo que el etiquetador marcó correctamente.
- *Sistema*: Nº de dependencias de ese tipo anotadas, ya sea correcta o incorrectamente.
- *Recall*: Representa el porcentaje $Correctas/Total$.
- *Precisión*: Indica el porcentaje $Correctas/Sistema$.

Modelo	**LAS(%)**	**UAS(%)**	**LAS2(%)**
m_1	81,79	86,74	86,92
m_2	81,58	86,67	86,66

Cuadro 7.6: Precisión de los mejores modelos de características

Dependencia	Total	Correctas	Sistema	Recall (%)	Precision (%)
ROOT	0	0	604	NaN	0.00
S	2805	1903	2815	67.84	67.60
a	15	8	11	53.33	72.73
ao	270	81	205	30.00	39.51
atr	609	509	639	83.58	79.66
c	43	38	44	88.37	86.36
cag	163	129	152	79.14	84.87
cc	3508	2462	3712	70.18	66.33
cd	2774	2122	2829	76.50	75.01
ci	313	185	271	59.11	68.27
conj	1012	809	971	79.94	83.32
coord	1409	952	1394	67.57	68.29
cpred	206	129	203	62.62	63.55
creg	539	288	435	53.43	66.21
d	233	196	233	84.12	84.12
et	136	80	107	58.82	74.77
f	4	1	3	25.00	33.33
grup.a	73	52	65	71.23	80.00
grup.adv	3	1	2	33.33	50.00
grup.nom	412	247	369	59.95	66.94
impers	49	23	40	46.94	57.50
inc	47	7	31	14.89	22.58
infinitiu	54	34	45	62.96	75.56
interjeccio	1	0	1	0.00	0.00
mod	511	418	486	81.80	86.01
morfema.pronominal	315	235	333	74.60	70.57
morfema.verbal	6	0	0	0.00	NaN
n	10	6	15	60.00	40.00
neg	23	15	24	65.22	62.50
p	3	0	2	0.00	0.00
participi	1	0	0	0.00	NaN
pass	181	102	184	56.35	55.43
r	33	19	23	57.58	82.61
relatiu	33	2	19	6.06	10.53
s	103	78	118	75.73	66.10
s.a	2423	2190	2487	90.38	88.06
sa	11	0	2	0.00	0.00
sadv	250	150	255	60.00	58.82
sentence	1646	1314	1513	79.83	86.85
sn	8444	7679	8358	90.94	91.88
sp	4325	3455	4257	79.88	81.16
spec	7820	7613	7858	97.35	96.88
suj	3136	2423	2908	77.26	83.32
v	980	795	917	81.12	86.70
w	2	1	1	50.00	100.00
z	81	68	74	83.95	91.89

Cuadro 7.7: Precisión detallada del analizador del sistema

Capítulo 8

El analizador de la orientación semántica

Constituye la parte central del sistema de MO. En este capítulo se explica en profundidad el núcleo del mismo y se justifican las decisiones de diseño tomadas para lograr un sistema fácilmente extensible. Por último se describe un análisis de la orientación semántica para un pequeño texto de ejemplo formado por las frases 5.1 y 5.2, introducido en capítulos anteriores y que servirá también para comprender las nuevas funcionalidades explicadas en los siguientes.

8.1. El núcleo del analizador

Esta parte del sistema se preocupa de proporcionar las funcionalidades necesarias para realizar un análisis básico de la polaridad en textos. Ello requirió decidir previamente aspectos relacionados con la estructura de datos y con los elementos de apoyo léxico y semántico que el sistema de MO utiliza.

8.1.1. Estructura de datos

Con el modelo del analizador sintáctico integrado en el proyecto, la funcionalidad del sistema de MO previa al análisis de la orientación semántica ya estaba totalmente operativa. En este momento, dado un texto donde se expresara una opinión, ya era posible segmentar, *tokenizar*, etiquetar y analizar sintácticamente sus oraciones. Sin embargo, analizar la orientación semántica de un texto a partir de los árboles de dependencias representados en formato CoNLL 2006 [9] es algo complejo e ineficiente, dadas las características de los ficheros. Para resolverlo se optó por convertir el fichero de salida que proporciona el analizador sintáctico a

una instancia de la clase *nltk.parse.dependencyGraph.DependencyGraph*, que representa una estructura de datos equivalente a los árboles de las figuras 7.1 ó 7.2. Un objeto de esta clase está constituido por varios nodos, donde cada uno de ellos es un diccionario[1] que almacena varios campos que se corresponden con las columnas ID, FORM, POSTAG, HEAD Y DEPREL del fichero CONLL que representa las oraciones:

- *word*: El propio término que constituye ese nodo. Almacena el valor de la columna FORM.
- *deps*: Los ID's de los nodos de los que es el padre.
- *rel*: El tipo de relación de dependencia de ese nodo con su padre. Representa el campo DEPREL en el fichero CONLL.
- *tag*: La etiqueta con toda la información morfológica para esa palabra, es decir la columna POSTAG.
- *address*: El identificador para ese término dentro de la oración. Es la columna ID para ese elemento.

Esta estructura de datos permitió enfocar la algoritmia de esta parte del sistema desde un punto de vista recursivo, que es la forma natural en la que los grafos y los árboles se recorren y analizan, lo que supuso una ventaja añadida.

Ejemplo 8.1.1. En la oración 5.1 *"Es un ordenador rápido y fiable, pero no recomiendo comprarlo"* presentada en formato CONLL en el cuadro 7.4 y cuyo árbol de dependencias sería el expuesto en la figura 7.1, el nodo asociado a la palabra *"es"* tendría la siguiente información:

- $word_{es} : es$
- $deps_{es} : [2, 4, 7, 8, 11, 12]$
- $rel_{es} : sentence$
- $tag_{es} : v : postype\_semiauxiliary - num\_s - person\_3 - mood\_indicative - tense\_present$
- $address_{es} : 3$

■

[1] Colección de Python que permite almacenar pares clave:valor.

8.1.2. Diccionarios de orientación semántica

Hemos comentado que para resolver la polaridad de un texto se hace uso de una aproximación semántica, donde una serie de diccionarios[2] almacenan valores que reflejan el sentimiento de una colección de palabras de distintas categorías gramaticales. Estos diccionarios únicamente almacenan la orientación semántica para la forma canónica de una palabra, esto es, su lema. Ello supone un problema, como queda reflejado en el ejemplo 8.1.2, ya que sólo es posible sumar el sentimiento de las palabras que aparezcan en los mismos.

Ejemplo 8.1.2. El término *"simpático"* aparece anotado en el SODictionariesV1.11Spa con una polaridad de 3, pero el diccionario no dispone de entradas para los términos *"simpática"*, *"simpáticos"* o *"simpáticas"*, palabras que expresan exactamente el mismo sentimiento. De esta manera, al procesar la sentencia *"es una persona simpática"*, no se encuentra ninguna entrada en los diccionarios de orientación semántica para la palabra *"simpática"* y el sistema entendería que esa oración no tiene ningún término de opinión, retornando una polaridad nula. La situación se agrava con los tiempos verbales. El diccionario de verbos sólo contiene entradas para el infinitivo[3], de modo que si no se toma ninguna medida, el analizador del sentimiento no puede procesar semánticamente ninguna otra forma verbal. Así, se podría procesar el término *"ocultar"* que está anotado con un valor de -1, pero sería imposible analizar palabras como *"ocultaste"*, *"ocultó"* u *"ocultaríamos"*.

■

La solución a este problema pasó por emplear un diccionario de lemas. Para todas las palabras del Ancora Corpus Dependencias anotadas en formato CoNLL se dispone de su correspondiente forma canónica en la columna LEMMA. A partir de ahí se generó dicho diccionario en formato texto y constituido por tres columnas: la categoría gramatical de la palabra, el propio término y la forma canónica de esa palabra, tal y como se ilustra en el cuadro 8.1. Este fichero de lemas incorporó también todas las palabras presentes en los diccionarios de orientación semántica, para asegurarse de que cualquier palabra presente en el SODictionariesV1.11Spa tenga su correspondiente entrada en el diccionario. Para mejorar la eficiencia del sistema, cuando éste se ejecuta, el fichero de lemas se carga en memoria[4] y se crea un diccionario Python que indexa a partir de la categoría gramatical y la forma de la palabra. Así se evita el acceso continuo a disco para recuperar el contenido de los diccionarios.

[2] SODictionariesV1.11Spa.

[3] Los infinitivos son las formas canónicas para los verbos.

[4] Los diccionarios de OS también son cargados en memoria.

Categoría gramatical	Palabra	Lema
n	canciones	canción
s	por	por
p	sí	él
p	mismas	mismo
f	,	,
r	no	no
c	porque	porque
p	les	él
v	haya	haber
v	caído	caer
a	simpática	simpático

Cuadro 8.1: Fragmento del diccionario de lemas

En resumen, al analizar la orientación semántica de una palabra, se obtiene en primer lugar su forma canónica y a continuación se busca su polaridad en el diccionario de orientación semántica correspondiente. De esta forma la búsqueda en este último se hace siempre utilizando el lema de la palabra.

Ejemplo 8.1.3. Retomando el ejemplo 8.1.2, en *"es una persona simpática"*, para la palabra *"simpática"*, se obtendría su lema indexando a partir de la clase gramatical adjetivo y de la propia palabra, dando como resultado *"simpático"*; término que si aparece en diccionario de OS de adjetivos anotada con un valor de 3, con lo que el sentimiento de la sentencia ya se obtendría correctamente.

■

8.1.3. Tratamiento de adjetivos, sustantivos, verbos y adverbios

En los primeros intentos por determinar el sentimiento de un texto, la MO se centraba únicamente en el análisis de los adjetivos o las frases adjetivales [3], ya que son las construcciones en las que se refleja una mayor subjetividad. Esta aproximación permitía obtener una buena parte del sentimiento expresado en el enunciado, pero para conseguir una extracción completa era necesario tratar otros muchos elementos. Por ejemplo, retomando el texto formado por las sentencias 5.1 y 5.2: *"Ese ordenador es rápido y fiable, pero no recomiendo comprarlo. Sin duda es muy caro, 2.700€. "*, los adjetivos *"rápido"*, *"fiable"* y *"caro"* son una buena muestra de la polaridad del texto, pero otras palabras como *"recomiendo"* tam-

bién tienen una connotación subjetiva. Al respecto, los trabajos recientes [2, 3] ya tienen en cuenta otros términos como los sustantivos, los verbos, o los adverbios y otras palabras relevantes que influyen en la polaridad. Este texto presenta también otros elementos que pueden modificar el sentimiento de la oración como son el intensificador *"muy"*, la negación *"no"* o la conjunción adversativa *"pero"*. Estos aspectos serán tratados a fondo en los siguientes capítulos.

Así pues, el núcleo del sistema propuesto ha tratado adjetivos, sustantivos, verbos y adverbios con el fin de conseguir una versión inicial funcional que considere los aspectos fundamentales que debe tener cualquier sistema de análisis del sentimiento, pero dado que todos estos aspectos se resuelven tratando palabras individualmente, la aproximación sintáctica no proporciona ninguna ventaja respecto a las alternativas léxicas[5]. La clase *SentimentAnalyzer* es la encargada de modelar todo el proceso y para obtener la polaridad de un texto agrega el sentimiento de cada una de las n oraciones que conforman el enunciado[6]:

$$OS_{texto} = \sum_{i=1}^{n} OS_{oracion_i} \tag{8.1}$$

El método de *SentimentAnalyzer* encargado de esta tarea es *analyze(self,file)*. Esta función interactúa con el analizador sintáctico para recuperar los árboles de dependencias de un texto, evaluando la estructura sintáctica para extraer su polaridad. El siguiente pseudocódigo, cercano al lenguaje Python, describe el comportamiento de este método:

```
def analyze(self,file):
  text_orientation = 0
  dependencyGraphs = self.parser.parse(file)
  for dg in dependencyGraphs:
    phrase_orientation = self.evaluate(dg,0)
    text_orientation += phrase_orientation.get_so()
  return text_orientation
```

donde la variable *dependencyGraphs* representa la lista de los árboles de dependencias obtenidos por el analizador (*parser*) para el texto en cuestión. La variable *text_orientation* mantiene la OS para el fragmento del texto ya procesado y *phrase_orientation* almacena la polaridad del

[5]La situación será muy distinta cuando se traten construcciones lingüísticas más complejas donde sin emplear análisis sintáctico es complicado realizar un tratamiento preciso de las mismas.

[6]Respecto a la clasificación en positivos o negativos, si el documento tiene una puntuación mayor que cero, se considera como positivo mientras que en otro caso consideramos que es negativo.

elemento de la lista de árboles que es procesado en esa iteración, tarea de la que se encarga la función de evaluación *evaluate*. Respecto a este método hubo dos aspectos de diseño que hubo que tener muy en cuenta:

- *El tipo de nodo que se esté evaluando.* Recorrer la estructura sintáctica de una oración implica analizar, entre otros nodos, conjunciones, intensificaciones, verbos, negaciones y adjetivos. Cada uno tiene unas características distintas, lo que requiere un análisis particular en casa caso. Fue por tanto necesario diseñar una estrategia de evaluación que pudiese añadir nuevas funcionalidades sin que repercutiese en el resto de métodos, y que además fuese fácilmente mantenible. La solución consistió en desarrollar una serie de funciones de evaluación específicas, que se detallan más adelante para los casos de sustantivos, adjetivos, adverbios y verbos, y a las que a partir de ahora denominaremos *funciones de visita*. Estos métodos son fundamentales en el sistema propuesto y su explicación se extenderá por varios capítulos, de forma paralela a los fenómenos lingüísticos estudiados.

- *La información devuelta.* Las llamadas recursivas a los descendientes de un nodo devuelven información que el padre utiliza para tratar de resolver la polaridad correctamente. En las funcionalidades desarrolladas para el núcleo, sería suficiente con devolver la orientación semántica, pero hay otros elementos que durante su evaluación pueden necesitar información a mayores. Fue por tanto necesario definir el concepto de *información semántica*, que tiene como cometido dar a conocer todos los detalles relevantes asociados con la subjetividad y el sentimiento de un nodo del árbol. Esto obligó a emplear un mecanismo (la clase *SentimentInfo*) que pudiese ampliar la información semántica en el futuro sin interferir en las funcionalidades que ya estén implementadas, aunque para las funcionalidades del núcleo sólo se incluyó la OS, representada como un atributo de esta clase llamado *so*. Así, *SentimentAnalizer* proporciona el método *get_sentiment_info(self,node)* que devuelve un objeto *SentimentInfo* para el nodo solicitado. En este sentido, las funciones encargadas de procesar dichos nodos retornan también instancias de este tipo.

Respecto a la implementación de la función de evaluación, se proporciona bajo estas líneas su pseudocódigo. Su cometido es procesar cada uno de los árboles de dependencias para lo que sigue un enfoque recursivo. Si es un nodo hoja, ya no queda nada más que analizar en esa rama y simplemente se devuelve un objeto *SentimentInfo* con la información para ese

término. En otro caso, es necesario analizar en detalle las características del nodo y según ellas devolver una función de visita, que es la que realmente se encarga de analizar dicho nodo y que ahora esbozamos:

```
def evaluate(self,graph,address):
  node = graph.get_by_address(address)
    if self.is_leaf(node):
      return self.get_sentiment_info(node)
    else:
      semantic_category = self.get_semantic_category(graph,node)
      f = self.visit_function(semantic_category)
      return f(graph,node)
```

donde la función *evaluate* recibe dos parámetros: *graph*, que debe ser una instancia de *DependencyGraph* y *address*, que representa el identificador del nodo a partir del cual se quiere realizar el análisis. La función que se encarga de decidir el tipo de nodo que se está procesando en cada momento es *get_semantic_category(self,graph,node)*, que atiende tanto a criterios léxicos como sintácticos para tomar esa decisión. En el caso de adjetivos, nombres, verbos y adverbios únicamente se tiene en cuenta la categoría gramatical, pero en capítulos posteriores se ilustra cómo, para llamar a funciones de visita que resuelvan construcciones lingüísticas complejas, es necesario considerar también información sintáctica. La función *visit_function* es la encargada de devolver la función de visita propia de cada tipo de palabra, pero dada la importancia de este punto su resolución se trata en un apartado aparte.

8.1.4. Las funciones de visita

Como ya se comentó, el concepto de las funciones de visita surgió a raíz de la necesidad de analizar distintas clases de nodos en el árbol, cada una de ellas con sus propias características léxicas, sintácticas y semánticas.

La solución propuesta en este trabajo permite que el análisis de un nuevo elemento lingüístico se reduzca a crear una nueva función de visita para esa construcción, clasificando cada nodo del árbol de forma que permita a *visit_function* devolver la función adecuada para ese término.

En el núcleo del analizador se consideraron las funciones de visita para verbos, sustantivos, adjetivos y adverbios. En la implementación actual todas tienen un comportamiento

similar, pero se decidió separarlas en funciones especializadas previendo futuras diferenciaciones que pudieran incluirse en el proceso de evaluación del sentimiento[7]. Así pues, el esqueleto base de la función de selección *visit_function*, que veíamos en la función *evaluate*, se ilustra en el siguiente pseudocódigo:

```
def visit_function(self, category):
  def visit_noun(graph,node):
    ...
  def visit_verb(graph,node):
    ...
  def visit_adjetive(graph,node):
    ...
  def visit_adverb(graph,node):
    ...
  def visit_other(graph,node):
    ...
  functions_dict= {
                   "n": visit_noun,
                   "a": visit_adjetive,
                   "r": visit_adverb,
                   "v": visit_verb
                  }
  if functions_dict.has_key(category):
    return functions_dict[category]
  else:
    return visit_other
```

donde el parámetro *category* proporciona a la función el tipo de nodo que se está visitando. Las funciones de visita se almacenan dentro de un diccionario Python que indexa por dicho parámetro. Este diccionario se irá ampliando conforme se incorporen nuevas clases de nodos[8]. La OS en una de las ramas cuyo origen sea una de las cuatro categorías gramaticales consi-

[7]Por ejemplo, podría ser interesante dar una mayor importancia a un nombre cuando este actúe como el sujeto de la oración, o priorizar determinados descendientes de un nodo según su categoría gramatical.

[8]La intensificación, la negación o las oraciones adversativas serán construcciones que requieren nuevas funciones de visita y por tanto nuevas entradas en el diccionario.

deradas en el núcleo se obtiene como la suma de la polaridad del nodo raíz de dicha rama y la polaridad acumulada de todos sus descendientes:

$$OS_{rama} = OS_{nodo} + \sum_{i=1}^{n} OS_{descendiente_i} \tag{8.2}$$

Respecto a la función *visit_other*, tiene como cometido propagar las llamadas recursivas en nodos sin orientación semántica con el fin de evaluar el sentimiento sobre sus descendientes, de modo que su expresión para calcular la polaridad es ligeramente distinta:

$$OS_{rama} = \sum_{i=1}^{n} OS_{descendiente_i} \tag{8.3}$$

Hasta aquí, hemos mostrado el esqueleto de la función de selección, pero apenas hemos dicho nada sobre las funciones de visita. Para comprender el proceso de análisis de la polaridad para uno de los nodos con OS tratados en el núcleo, ilustramos a continuación el código correspondiente a la función de visita de los adjetivos:

```
def visit_adjetive(graph,node):
  children = self.get_deps(node)
  node_s_info = self.get_sentiment_info(node)
  so_node = node_s_info.get_so()
  so_children= 0

  for child in children:
    s_child = self.evaluate(graph,child)
    so_children = so_children + s_child.get_so()
    so_node = so_node + so_children
  return SentimentInfo(so_node)
```

donde la variable *children* es una lista que almacena los ID's de los términos dependendientes de un nodo *node* y *node_s_info* es un objeto *SentimentInfo* que guarda la información semántica para dicho nodo. En un proceso iterativo se realizan las llamadas recursivas para evaluar a todos los descendientes de *node* para recuperar la orientación semántica de cada uno de ellos y así poder devolver la polaridad total acumulada en esa rama del árbol. La implementación de la función *visit_other* es ligeramente distinta, para adaptarse a las características del resto de nodos. Concretamente se elimina el cálculo de la información semántica para estos nodos, tal y como se ilustra en el siguiente pseudocódigo:

```
def visit_other(graph,node):
  so_node = 0
  children = self.get_deps(node)
  for child in children:
    s_child = self.evaluate(graph,child)
    so_node = so_node + s_child.get_so()
  return SentimentInfo(so_node)
```

En los siguientes capítulos se comentarán las decisiones de diseño e implementación relacionadas con las funciones de visita de la negación, intensificación y oraciones subordinadas adversativas.

8.2. Otros aspectos del sistema

En el núcleo de sistema también se consideró el tratamiento de otros aspectos del discurso del texto donde la sintaxis no es de utilidad. Estas funcionalidades, aunque se integraron en esta versión inicial, no fueron habilitadas hasta el final del desarrollo, para evitar que influyeran en la evaluación de las construcciones lingüísticas que todavía no habían sido consideradas. Concretamente, los aspectos considerados fueron:

- *La mayor importancia de las frases finales*: Se había comentado que era común que las últimas oraciones de un texto crítico actuasen a modo de resumen o conclusión de la opinión expresada [1]. Para aprovechar estas características del discurso se optó por que el sistema potenciara la orientación semántica de las tres últimas frases de los enunciados. La evaluación empírica del sistema final determinó que el mejor factor de ponderación era 1.75.

- *Una mayor distribución de los valores de las orientaciones semánticas en los datos*: Las polaridades de los elementos del SODictionariesV1.11Spa utilizado varían entre -5 y 5, pero en ocasiones si se aumenta el intervalo de valores posibles se puede clasificar de forma más precisa. Sobre el sistema final se realizaron pruebas que determinaron que potenciar en un 20 % la orientación semántica de todos los términos subjetivos presentes en el diccionario semántico, mejoraba la precisión del sistema de forma sensible.

Ejemplo 8.2.1. El desarrollo del núcleo del sistema permitió obtener una primera versión funcional del sistema, capaz de analizar la orientación semántica de textos de una forma

básica. En las figuras 8.1 y 8.2 se muestra como esta primera versión funcional procesaría las dos oraciones del texto: *"Ese ordenador es rápido y fiable, pero no recomiendo comprarlo. Sin duda es muy caro, 2.700€."*. En color verde se indican aquellos nodos con una orientación semántica positiva, mientras que los términos en rojo suponen una polaridad negativa. Para una mayor claridad se obviaron las llamadas a los términos sin orientación semántica.

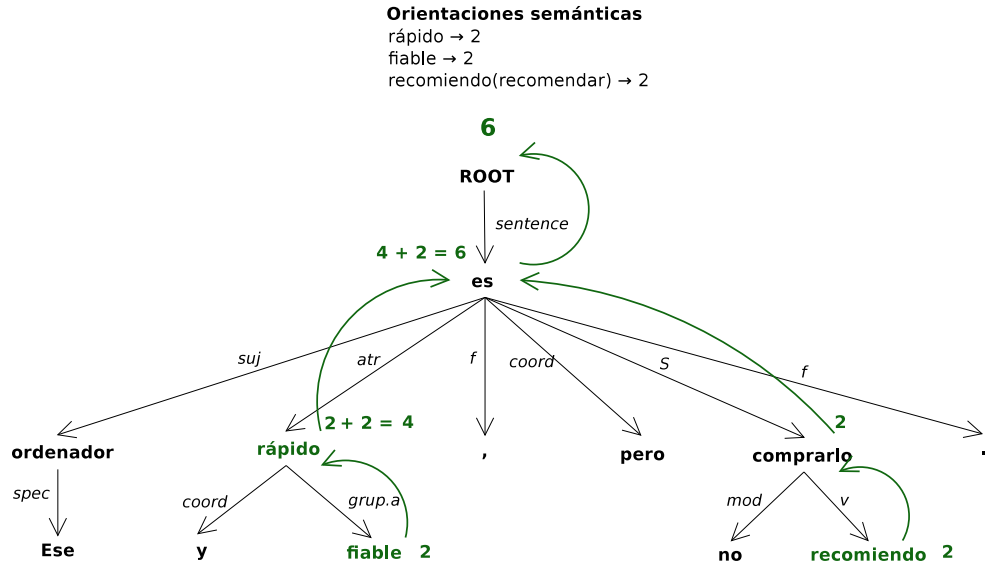


Figura 8.1: Análisis básico de la OS para la oración 5.1

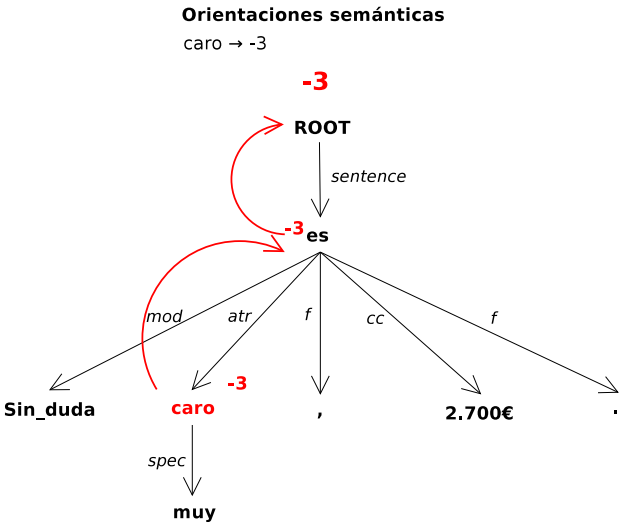


Figura 8.2: Análisis básico de la OS para la oración 5.2

El resultado final para el texto compuesto de estas dos oraciones sería:

$$OS(Or_1) = \quad 6$$
$$OS(Or_2) = \quad \text{-3}$$

$$OS_{total} = OS(Or_1) + OS(Or_2) = 6 + (-3) = 3$$

Por tanto, el sistema determinaría que este texto es positivo, algo que no parece ser correcto. El problema está en que hay varios aspectos sintácticos que no fueron tratados:

1. *La intensificación*: El término *"muy"* debería enfatizar la orientación semántica de la palabra *"caro"*, aumentando la negatividad de este elemento

2. *La negación*: Esta versión tampoco trata de ninguna forma la negación, una construcción lingüística fundamental en un entorno de MO ya que cambia la polaridad de los elementos a los que afecta. Más concretamente, en el ejemplo, la negación *"no"* debería cambiar el sentimiento del término *"recomiendo"*, que pasaría a tener una connotación negativa.

3. *La conjunción adversativa "pero"*: La aparición de este nexo también parece indicar cierta subjetividad priorizando la idea expuesta después de esta conjunción.

En definitiva, ignorar estos factores impide conseguir una extracción completa del sentimiento expresado en el texto. En los siguientes capítulos veremos como han sido tratados cada uno de estos aspectos y se mostrará cómo, conforme se va ampliando la funcionalidad del sistema, la orientación semántica del ejemplo se va ajustando más al sentimiento real expresado.

■

Capítulo 9

La intensificación

En el capítulo anterior se explicó cómo se desarrolló una versión inicial de nuestro sistema de MO, capaz de obtener el sentimiento general expresado en un texto. Sin embargo, presentaba ciertos problemas a la hora de hacer un análisis preciso, ya que no tenía en cuenta construcciones lingüísticas que influían en las orientaciones semánticas de las oraciones. En este capítulo se describe cómo se ha tratado una de esas construcciones, la intensificación, y se explican las distintas consideraciones que hubo que tener en cuenta para integrar este fenómeno en el sistema.

9.1. Construcciones lingüísticas intensificadoras

Identificar la intensificación permite conocer cuáles son los puntos de la opinión en los que el autor pone un mayor énfasis con intención de destacar una idea. Generalmente hay dos formas de expresar este fenómeno [2]:

- Utilizar términos adverbiales como *"muy", "demasiado", "poco", "mucho"o "bastante"*, para modificar la polaridad de las palabras con las que se asocian. Para realizar un tratamiento completo de este tipo de elementos, el sistema se apoyó en el SODictionariesV1.11Spa, que contiene un diccionario específico de intensificadores formado por 157 términos.

- Emplear ciertas conjunciones como *"pero"* para resaltar o contraponer la idea expuesta en la oración subordinada adversativa.

A mayores, en el lenguaje escrito y en particular en un ámbito de MO hay otros dos elementos que suelen representar una intensificación:

- *Las mayúsculas*, cuya utilización en internet es considerada como una forma de exaltación de la idea que se expresa.

- *La utilización de signos de exclamación*, que se utilizan para enfatizar y dar un mayor peso a ciertas oraciones del texto.

En este contexto, nos centramos en describir cómo se resolvió la intensificación representada mediante el empleo de adverbios y signos de exclamación, dos situaciones donde el análisis sintáctico puede resultar de gran utilidad. El tratamiento de las palabras mayúsculas se solucionó con una ponderación heurística, ya que es un fenómeno que no reviste naturaleza sintáctica, sino más bien léxica. Respecto a las cláusulas adversativas, han sido tratadas de forma separada por sus peculiaridades sintácticas y semánticas. Su resolución se describirá en el siguiente capítulo.

El uso de intensificadores introduce nuevos factores a tener en cuenta en la implementación de nuestra propuesta. Así, para tratar nodos como los adjetivos o los nombres, la información léxica que proporcionaba la categoría gramatical era suficiente para determinar el tipo de nodo, y así conocer cuál era la función de visita a la que había que llamar. Sin embargo, este enfoque no es válido cuando se habla de intensificadores. La categoría gramatical no identifica a un término como intensificador, sino que es la relación que mantiene con otro elemento lo que hace que actúe como tal. Por ello, es necesario emplear información sintáctica para saber cuando una palabra es un amplificador o un decrementador. Los casos que mostraremos a continuación, extraídos del Ancora Corpus Dependencies [14], muestran las distintas opciones contempladas cuando la categoría gramatical del intensificador es bien un adverbio o bien cuando se emplean signos de exclamación:

- *Intensificación nominal*: Trabajos previos [2, 3] consideraban que los sustantivos sólo podían modificarse mediante el uso de adjetivos y nunca mediante la utilización de intensificadores propios de adjetivos o adverbios. Sin embargo, la forma en que el *corpus* de Ancora codifica las relaciones entre elementos hace posible que existan casos en los que un sustantivo se encuentre intensificado, como el mostrado en la figura 9.1, que representa como árbol de dependencias la sentencia *"Rosa es casi una principiante"*. La relación de dependencia $sustantivo \leftrightarrow intensificador$ se anota siempre como un sintagma adverbial, que Ancora codifica con la etiqueta *sadv*.

- *Intensificación adjetival*: Ancora anota la intensificación sobre los adjetivos de dos formas, sin que quede claro cuando se debe emplear una u otra. Las sentencias *"El ministro*

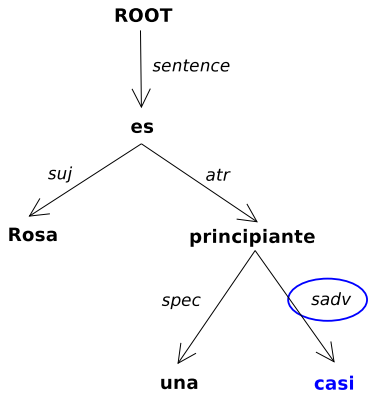


Figura 9.1: Análisis de dependencias con intensificación de sustantivos

sale muy debilitado" y *"El autor tiene muy clara su visión"* ilustradas en las figuras 9.2 y 9.3 respectivamente, muestran estos dos tipos de relaciones de dependencia que permiten identificar este fenómeno sobre los adjetivos. En el primer caso, el tipo de dependencia $adjetivo \leftrightarrow intensificador$ se anota como un complemento circunstancial, *cc* según Ancora; mientras que el segundo se clasifica como un especificador, representado por el *corpus* por *spec*.

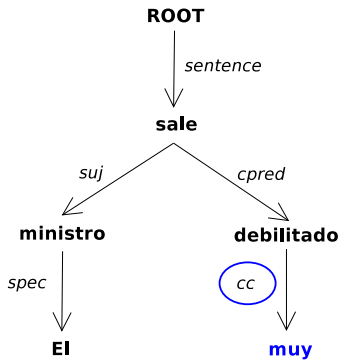


Figura 9.2: Análisis de dependencias con intensificación de adjetivos

- *Intensificación adverbial*: La intensificación de adverbios se realiza de forma análoga a la considerada en uno de los enfoques seguidos para los adjetivos, concretamente al que anota las dependencias como un especificador. En la figura 9.4 se ilustra la estructura sintáctica empleada para tratar este fenómeno en adverbios sobre la frase *"La delegación cubana está bastante bien"*-

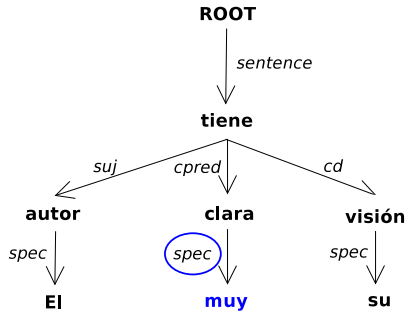


Figura 9.3: Otro análisis de dependencias con intensificación de adjetivos

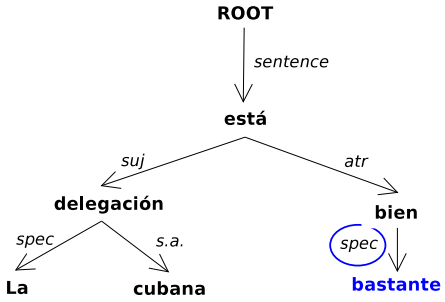


Figura 9.4: Análisis de dependencias con intensificación de adverbios

- *Intensificación verbal*: La intensificación verbal se anota también siguiendo el modelo ya comentado en uno de los enfoques empleados con los adjetivos. En este caso, cuando exista un verbo intensificado, Ancora asigna a esa relación de dependencia la etiqueta de complemento circunstancial. En la figura 9.5 se muestra el caso de un verbo intensificado en la frase *"Las negociaciones se deteriorarán seriamente"*.

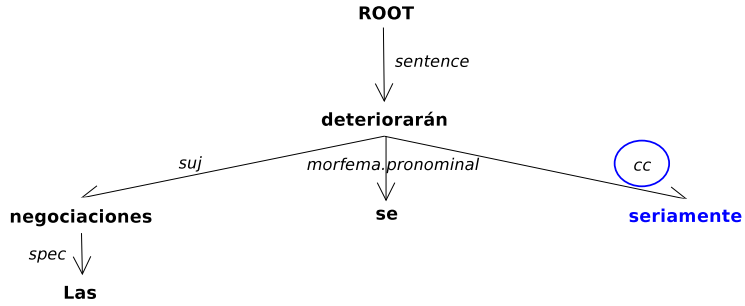


Figura 9.5: Análisis de dependencias con intensificación verbal

- *Intensificación exclamativa*: La intensificación exclamativa difiere sensiblemente de las expuestas hasta ahora, donde solamente se aumentaba o decrementaba la orientación semántica de la palabra con la que se mantenía la relación de dependencia. Sin embargo, con signos de exclamación, todos los elementos entre estos dos símbolos en una oración deberían ser potenciados. Como ilustración de ello, consideremos la potenciación del sentimiento de la rama del nodo *"es"* en la frase *"¡Qué hermosa es esta película y qué dura!"*, que en este caso sería una combinación de las palabras *"hermosa"* y *"dura"*; tal y como ilustra la figura 9.6.

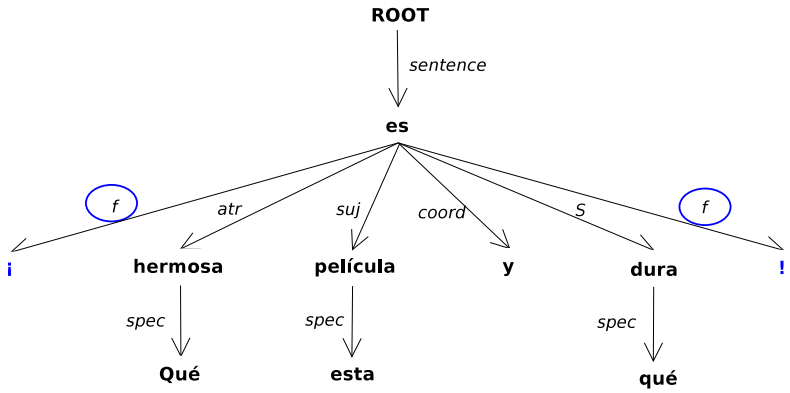


Figura 9.6: Análisis de dependencias con intensificación exclamativa

- *Intensificación múltiple*: En todos los casos adverbiales vistos hasta ahora sólo había un intensificador que modificara a la palabra en cuestión, pero es relativamente común que varios términos se empleen conjuntamente para amplificar o decrementar la orientación semántica de un *token*. Uno de los casos más típicos es el indicado en la figura 9.7, donde dos intensificadores actúan de forma anidada para modificar la polaridad de un término, en este caso en la frase *"Óscar tiene muy bien definido su futuro"*.

Los seis casos anteriormente detallados representan las situaciones de intensificación que se abordaron en el presente trabajo. El sistema analiza los tipos de dependencias entre los términos y, cuando alguno de estos casos se detecta, la función de visita propia de la intensificación resuelve la situación.

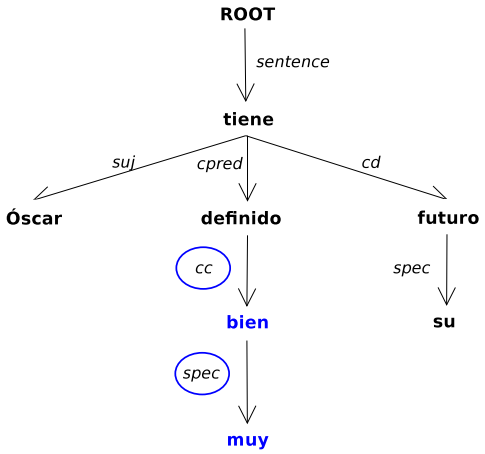


Figura 9.7: Análisis de dependencias con intensificación anidada

9.2. Tratamiento de la intensificación

Dos son las principales estrategias con las que es posible tratar la intensificación:

1. *Intensificación basada en adiciones y sustracciones*. Esta estrategia [3] propone sumar o restar una cantidad fija a la OS de una palabra, según el intensificador sea un amplificador o un decrementador. Así, si *"inteligente"* tiene una polaridad de 3, *"bastante inteligente"* podría tener un valor de 4 y *"ligeramente inteligente"* tendría un sentimiento total de 2. El gran problema en este caso es que no se tienen en cuenta las orientaciones semánticas de los términos involucrados en la intensificación, por lo que dos adverbios como *"bastante"* y *"extremadamente"* amplificarían en igual magnitud, cuando en realidad *"extremadamente"* debería potenciar más.

2. *Intensificación basada en porcentajes*: Esta aproximación propone que cada intensificador tenga un porcentaje asociado, positivo si es amplificador y negativo si es un decrementador. Esto permite que las orientaciones semánticas se modifiquen de forma que se tenga en cuenta las polaridades de los elementos involucrados. El sistema propuesto sigue este enfoque que también es empleado por otros analizadores del sentimiento como *The English SO Calculator* [3] o *The Spanish SO Calculator* [2]. En el cuadro 9.1 se muestra un fragmento del diccionario de intensificadores en el que se apoya el sistema. Este diccionario se amplió para que también contuviera un porcentaje de intensificación para el signo de exclamación *"!"*. No se incluyó la apertura exclamativa porque eso supondría intensificar dos veces la misma oración. Se descartó la idea de incluir el *"¡"*

en vez del *"!"*, ya que en un entorno web es común obviar las aperturas exclamativas. La evaluación empírica sobre el SFU Spanish Review Corpus determinó que lo óptimo era incrementar el valor de la oraciones exclamativas en un 20 %.

Intensificador	**Modificación(%)**
bastante	0,1
extremadamente	0,35
en_absoluto	-1
ligeramente	-0,5
como_mucho	-0,2

Cuadro 9.1: Fragmento del diccionario de intensificadores

La fórmula[1] que permite calcular la orientación semántica con esta aproximación en el sistema es:

$$OS_{intensificacion} = OS_{intensificado} \times (1 + \sum_{i=1}^{n} Int_{intensificador_i}) \tag{9.1}$$

Ejemplo 9.2.1. Efecto de la intensificación basada en porcentajes para el término *"inteligente"* cuando se ve afectado por algunos amplificadores y decrementadores del cuadro 9.1.

$OS_{bastante\ inteligente} = OS_{inteligente} \times (1 + Int_{bastante}) = 3 \times (1 + 0{,}1) = 3{,}3$
$OS_{ligeramente\ inteligente} = OS_{inteligente} \times (1 + Int_{ligeramente}) = 3 \times (1 - 0{,}5) = 1{,}5$
$OS_{extremadamente\ inteligente} = OS_{inteligente} \times (1 + Int_{extremadamente}) = 3 \times (1 + 0{,}35) = 4{,}05$
$OS_{como\_mucho\ bastante\ inteligente} = OS_{inteligente} \times (1 + \sum_{i=1}^{2} Int_{intensificador_i}) = OS_{inteligente} \times (1 + (Int_{bastante} + Int_{como\_mucho})) = 3 \times (1 + (0{,}1 - 0{,}2)) = 2{,}7$

■

Integrar este enfoque en el sistema requirió aumentar la información semántica de los objetos *SentimentInfo* para incluir un atributo *int* que contiene el grado de intensificación acumulado. El porcentaje de intensificación (Int) es un valor que debe propagarse a través de las funciones recursivas hasta llegar al término al que afecta, donde su efecto se disipa para combinarse con la orientación semántica del nodo en cuestión. Un ejemplo de esta propagación y combinación se muestra en la figura 9.9.

Con estos cambios, la expresión que permitía calcular la orientación semántica en un adjetivo, adverbio, verbo o nombre varió respecto a la fórmula 8.2 para incluir el porcentaje de

[1] Se denomina con Int al porcentaje de intensificación de un término.

intensificación por el que se ven afectados esta clase de nodos[2] y además ahora sus funciones de visitas deben devolver información sobre su propia intensificación, que en estos casos siempre será cero:

$$Int_{rama} = 0 \tag{9.2}$$

$$OS_{rama} = OS_{nodo} \times (1 + \sum_{i=1}^{n} Int_{descendiente_i}) + \sum_{i=1}^{n} OS_{descendiente_i} \tag{9.3}$$

El siguiente fragmento de pseudocódigo muestra como se obtenía en esta versión la información semántica para estos cuatro tipos de palabras, tomando como ejemplo la función *visit_adjetive*. El parámetro *intensification* mantiene el porcentaje de intensificación acumulado en ese nodo, para aplicarlo en el cálculo de la OS:

```
def visit_adjetive(graph,node):
  children = self.get_deps(node)
  node_s_info = self.get_sentiment_info(graph,node)
  so_node = node_s_info.get_so()
  intensification = 0
  so_children= 0
  for child in children:
    s_child = self.evaluate(graph,child)
    intensification += s_child.get_int()
    so_children += s_child.get_so()
    so_node = so_node * (1+intensification) + so_children
  return SentimentInfo(0,so_node)
```

Sin embargo las expresiones propuestas en las fórmulas 9.2 y 9.3 no son válidas si lo que se quiere es obtener la información semántica de un término adverbial intensificador. En estos nodos la intensificación tendrá un valor distinto de cero y no disponen de OS por si mismos[3]. Por ello, fue necesario establecer nuevas fórmulas para esta clase de elementos, lo que en términos de implementación derivó en una nueva función de visita que pudiese obtener la información de estos elementos. Así pues, para el cálculo del porcentaje de intensificación

[2]Nótese que un nodo puede estar afectado por intensificadores situados en distintos hijos, por ello la fórmula 9.3 agrega los porcentajes de intensificación de éstos.

[3]Por ejemplo, el término *"muy"* no expresa subjetividad por si mismo, por lo que su OS es cero, pero si tiene un porcentaje de intensificación asociado que puede modificar la OS de otros nodos del árbol.

en uno de estos nodos, fue necesario tener en cuenta el propio porcentaje de dicho nodo, así como los posibles intensificadores anidados en algunos de sus hijos para cubrir el caso de la intensificación múltiple:

$$Int_{rama} = Int_{nodo} + \sum_{i=1}^{n} Int_{descendiente_i} \tag{9.4}$$

Para la OS, la expresión es análoga a la de la fórmula 9.1, ya que estos nodos carecen de orientación semántica propia y su único objetivo es propagar el sentimiento de sus descendientes:

$$OS_{rama} = \sum_{i=1}^{n} OS_{descendiente_i} \tag{9.5}$$

A la nueva función de visita se le denominó *visit_intensifier* y su pseudocódigo se muestra bajo estas líneas. Esta función es devuelta por *visit_function(self,category)* siempre que el método *get_semantic_category(self,graph,node)*, encargado de asignar el valor del parámetro *category*, determine que el nodo que se esté procesando en ese momento es un intensificador[4]:

```
def visit_intensifier(graph,node):
  children = self.get_deps(node)
  node_s_info = self.get_sentiment_info(graph,node)
  int_node = node_s_info.get_int()
  so_node = node_s_info.get_so()

  for child in children:
    s_child = self.evaluate(graph,child)
    so_node += s_child.get_so()
    int_node += s_child.get_int()
  return SentimentInfo(int_node,so_node)
```

[4]Concretamente para estos nodos se asigna la etiqueta *int* como valor de *category*.

Ejemplo 9.2.2. En el ejemplo 8.2.1 se presentó un ejemplo que realizaba un análisis de la orientación semántica sobre un pequeño texto. Este ejemplo retoma dicho análisis tratando el fenómeno de la intensificación. La oración 5.1 no sufre ningún cambio porque no contiene ninguna construcción lingüística intensificadora. No ocurre lo mismo con la 5.2, donde se establece una relación de intensificación adjetival entre los términos *"muy"* y *"caro"*, como se observa en la figura 9.9. En color azul se representa la propagación de la intensificación a través de la rama del árbol afectada.

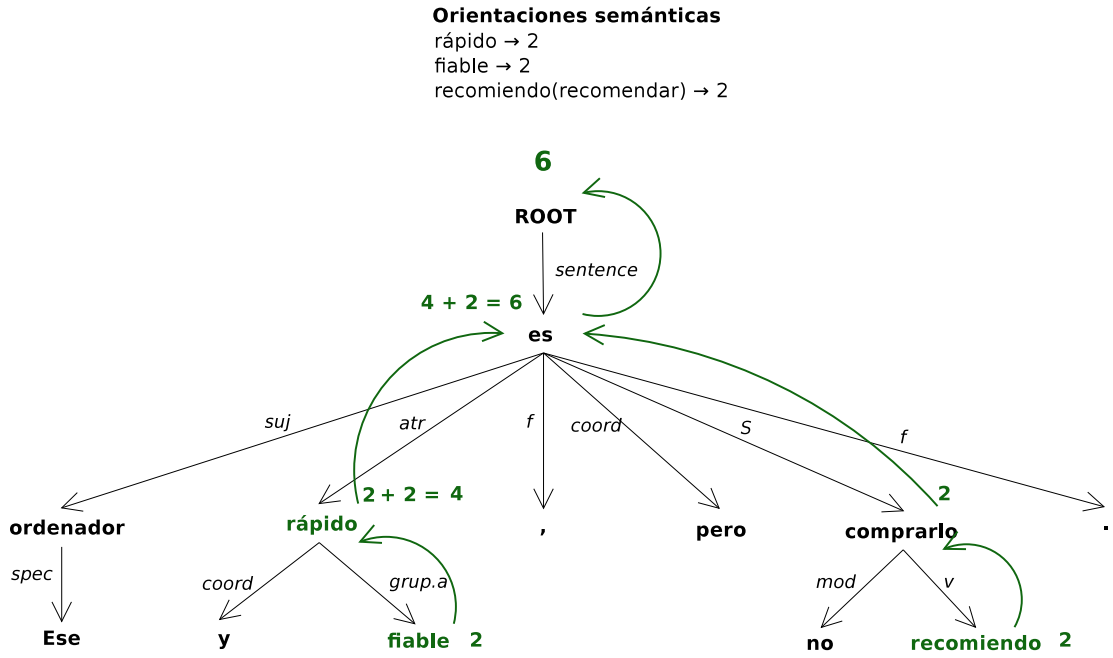


Figura 9.8: Análisis de la OS con tratamiento de la intensificación para la oración 5.1

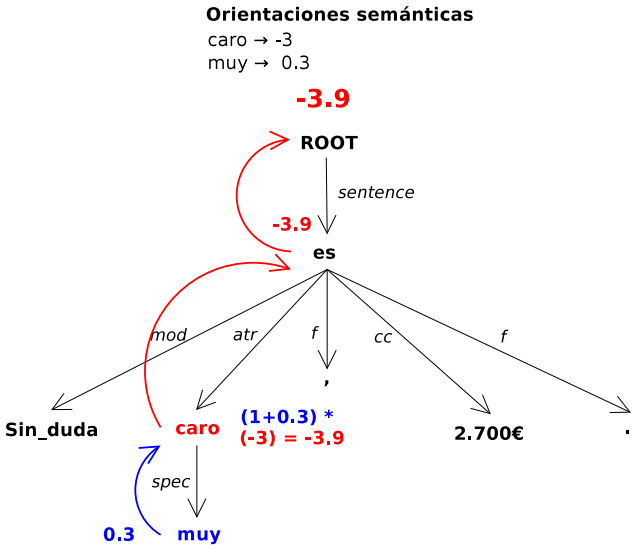


Figura 9.9: Análisis de la OS con tratamiento de la intensificación para la oración 5.2

De este modo, la polaridad total del texto sería en este momento:

$$OS(Or_1) = \quad 6$$
$$OS(Or_2) = \quad \text{-3.9}$$

$$OS_{total} = OS(Or_1) + OS(Or_2) = 6 + (-3{,}9) = 2{,}1$$

En resumen, el tratamiento de la intensificación permite obtener un sentimiento más preciso para la oración 5.2 y reducir la orientación semántica del enunciado, acercándose más a su polaridad real.

■

Capítulo 10

Las oraciones subordinadas adversativas

En el capítulo anterior se estudió el tratamiento de la intensificación con adverbios y exclamaciones, pero había otro tipo de intensificación basada en las oraciones adversativas que no fue tratada por sus peculiaridades sintácticas y semánticas[1]. En este capítulo se describe cómo se ha planteado y desarrollado este caso especial de intensificación. Se detalla la estrategia seguida para analizar este tipo de conjunciones, se justifican las decisiones de diseño tomadas y todo ello se ilustra con un ejemplo práctico.

10.1. Conjunciones adversativas

Las conjunciones adversativas permiten contraponer, excluir o resaltar las ideas expresadas entre dos oraciones, lo que comúnmente se considera como un tipo de intensificación. En este tipo de construcciones, una frase ve decrementada su orientación semántica en favor de la amplificación del sentimiento de otra. Existen muchos nexos[2] adversativos que pueden dividirse en dos grandes grupos:

1. *Los restrictivos*: Resaltan la orientación semántica de la subordinada respecto a la principal. Por ejemplo, en la frase *"La actuación del protagonista fue mala, pero la película me encanto"*, se intenta reducir la importancia de que el actor no desempeñase bien su papel, para resaltar que la película fue buena de todos modos.

[1] Este tipo de intensificación afecta a dos oraciones del texto y la estructura sintáctica que emplea Ancora para representarlas es diferente y más compleja que las tratadas en el capítulo anterior.

[2] Palabras cuya función sintáctica es separar palabras, sintagmas u oraciones.

2. *Los excluyentes*: Eliminan enteramente la orientación semántica expresada en la oración subordinante. Un buen ejemplo de este tipo de nexos es la conjunción *"sino"*. Por ejemplo, en la oración *"No se portó bien sino de una forma ruin y malvada"*, se excluye totalmente lo expresado en la primera frase ya que se indica algo que no ha sucedido.

En nuestro caso, nos hemos centrado en tratar las conjunciones restrictivas *"pero"*, *"mientras"*, *"mientras que"* y las excluyentes *"sino"* y *"sino que"*; que representan una muestra de los casos más importantes y que además comparten una estructura común, tal y como se indica en la figura 10.1. En esta línea, la siguiente sección detalla como se incorporaron estas nuevas capacidades.

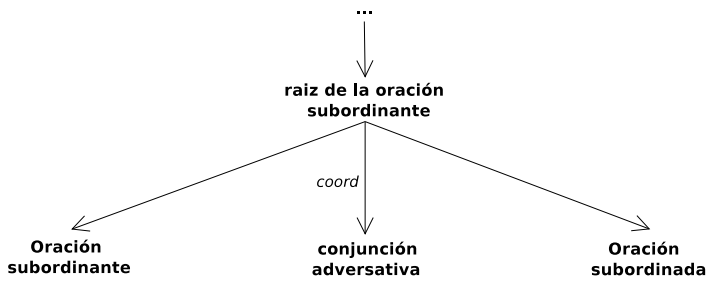


Figura 10.1: Estructura del árbol de dependencias para oraciones adversativas

Existen oraciones subordinadas adversativas conectadas por otras conjunciones, pero el *corpus* de Ancora las anota de distintas formas, algunas de ellas poco intuitivas, llegando incluso a emplear otra estructura sintáctica de dependencias, como es el caso de la conjunción *"aunque"*, de la que se puede ver un ejemplo en la figura 10.2. Dada la heterogeneidad de estas otras anotaciones su tratamiento no se ha considerado.

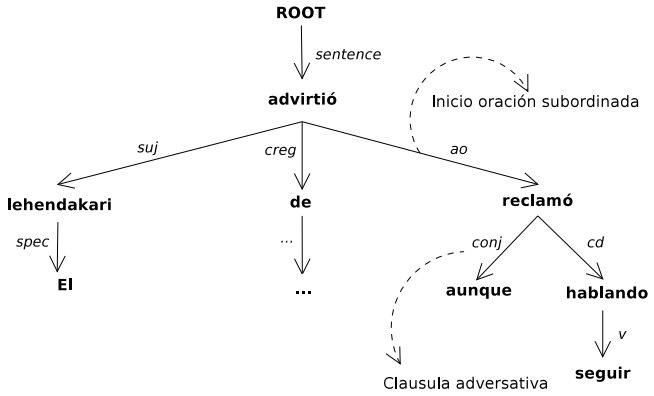


Figura 10.2: Árbol de dependencias con conjunción *"aunque"*

10.2. Tratamiento de las subordinadas adversativas

La estructura de dependencias para las oraciones adversativas ilustrada en la figura 10.1, aunque válida, presentaba algunos inconvenientes a la hora de analizar la orientación semántica, que pasamos a describir:

1. *Dificultad en la identificación del alcance de una negación en una oración subordinante*: Las conjunciones adversativas indican un límite de hasta donde es posible que la negación tenga efecto. Esta representación complicaba esa identificación, ya que sería necesario examinar todos los hijos de un nodo para saber cuales están situados a la derecha del nodo *"pero"* y así saber cuales no están afectados por la negación.

2. *El calculo de la orientación semántica entre la oración subordinante y la subordinada debía hacerse en la raíz de la oración principal*: Esto dificultaba la creación de una función de visita propia y complicaba la ponderación de las dos oraciones.

Para resolver estos problemas se optó por reorganizar la estructura del árbol de dependencias mediante la creación de un nodo artificial que delimitase lo que aparecía antes y después de la conjunción adversativa, para conseguir una estructura como la que se muestra en la figura 10.3.

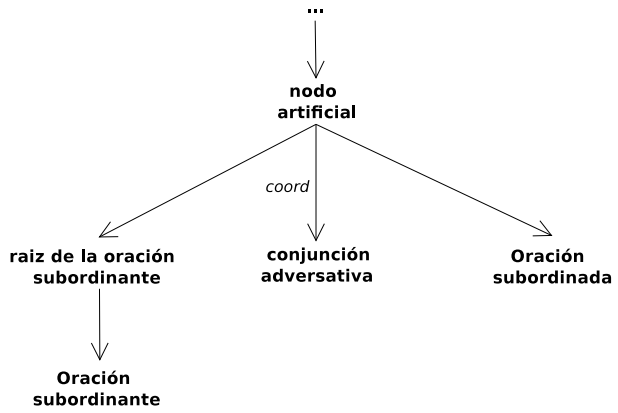


Figura 10.3: Estructura del árbol de dependencias para oraciones adversativas reorganizadas

La reorganización del árbol de dependencias se optó por hacerla sobre el fichero CoNLL proporcionado por el analizador sintáctico de dependencias, en vez de reestructurar la instancia del *DependencyGraph*. Se trata de una opción más eficiente y menos compleja que sólo requirió adaptar algunos campos de la columna HEAD y crear una nueva entrada con toda la

información para el *token* artificial adversativo. Justo antes de crear la instancia del *DependencyGraph* es cuando el sistema reorganiza el árbol. Se genera un nodo artificial, representado por [], como último término de la oración en la representación en formato CoNLL. Se crean igualmente una etiqueta especial denominada *art_adversative* y un nuevo tipo de dependencia llamada *art_rel_adversative*. La etiqueta del nodo artificial también contiene el campo ID del nodo correspondiente al nexo adversativo, así como el tipo de conjunción: *restrict* si es un nexo restrictivo y *exclude* si es excluyente. Así, en el momento de obtener la instancia del *DependencyGraph*, este nuevo nodo no requiere ningún tipo de consideración especial.

Ejemplo 10.2.1. La oración 5.1, seguida a lo largo de toda la memoria y cuyo fichero CoNLL original se indicó en el cuadro 7.4, tiene un nexo adversativo *"pero"* que es necesario reestructurar. La reorganización daría lugar a un nuevo árbol CoNLL representado en el cuadro[3] 10.1. En la figura 10.4 se muestra la instancia del *DependencyGraph* del ejemplo una vez ya reestructurado.

ID	**FORM**	**POSTAG**	**HEAD**	**DEPREL**
1	Ese	...	2	spec
2	ordenador	...	3	suj
3	es	...	*13*	sentence
4	rápido	...	3	atr
5	y	...	4	coord
6	fiable	...	4	grup.a
7	,	...	3	f
8	pero	...	*13*	coord
9	no	...	11	mod
10	recomiendo	...	11	v
11	comprarlo	...	*13*	S
12	.	...	3	f
13	*[]*	*art_adversative:restrict@8*	*0*	*art_rel_adversative*

Cuadro 10.1: Árbol reestructurado en formato CoNLL de la oracion 5.1

[3]Por claridad solo se representan las columnas que han sufrido alguna modificación respecto al cuadro 7.4.

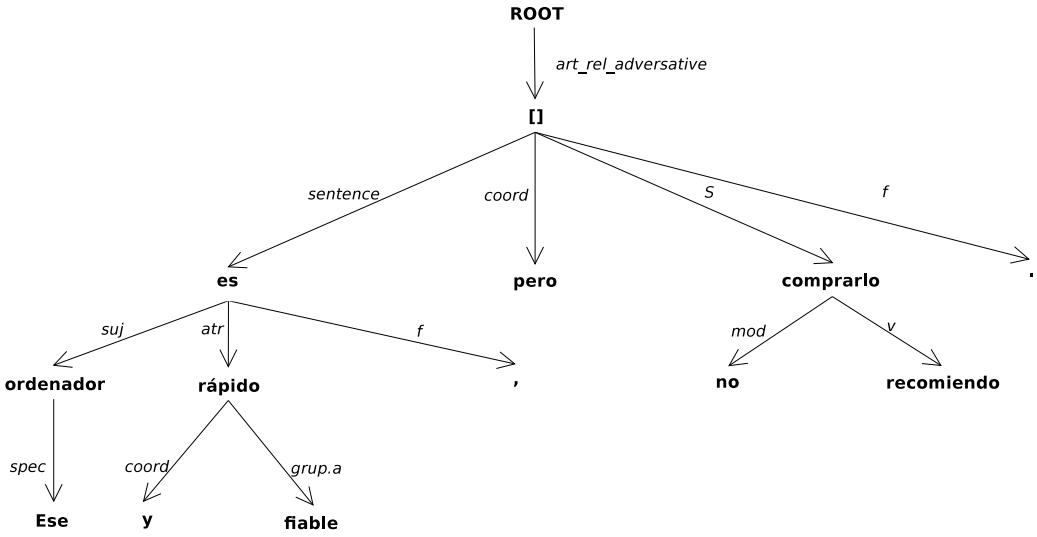


Figura 10.4: Árbol de dependencias reestructurado de la oración 5.1

■

Reestructurado el árbol, ya es posible identificar las oraciones adversativas y resolver su sentimiento de forma sencilla. La orientación semántica en uno de estos nodos artificiales se calcula ponderando las polaridades de lo que aparece tanto antes de la conjunción como después. En lo referido al porcentaje de intensificación en estos nodos, se siguió la misma estrategia que para verbos, sustantivos, adjetivos y adverbios[4]. Formalmente, las expresiones que definen el cálculo de toda esta información semántica son[5]:

$$Int_{nodo\_artificial} = 0 \tag{10.1}$$

$$OS_{nodo\_artificial} = Pond_{subordinante} \times OS_{subordinante} + Pond_{subordinada} \times OS_{subordinada} \tag{10.2}$$

En cuanto a los factores de ponderación varían según se trate de una conjunción adversativa excluyente o restrictiva, mostrándose en el cuadro 10.2 los valores con los que trabaja el sistema final. Estos factores se obtuvieron empíricamente a través de la evaluación del SFU Spanish Review Corpus.

[4]No se propaga intensificación en estos nodos.

[5]Se denomina con *Pond* a los factores de ponderación para las oraciones subordinada y subordinante.

Clausula adversativa	**Ponderación**$_{subordinante}$	**Ponderación**$_{subordinada}$
Restrictiva	0,75	1,4
Excluyente	0	1

Cuadro 10.2: Factores de ponderación para las oraciones subordinadas adversativas

Ejemplo 10.2.2. Se retoma el texto del ejemplo 9.2.2 para representar el análisis de las oraciones 5.1 y 5.2 una vez incluido considerado el tratamiento las conjunciones adversativas. Las figuras 10.5 y 10.6 muestran la propagación del análisis sobre los árboles, reflejado en lila se muestra la parte del árbol donde se resuelve esta construcción.

Nótese que en este momento el tratamiento del nexo adversativo no es correcto. El problema es que se está ponderando una oración subordinada cuyo sentimiento está mal calculado, ya que no se está analizando el término negativo *"no"*. En el capítulo siguiente mostraremos una propuesta de tratamiento del fenómeno de la negación y como su consideración permite conseguir una evaluación más adecuada del sentimiento para este caso.

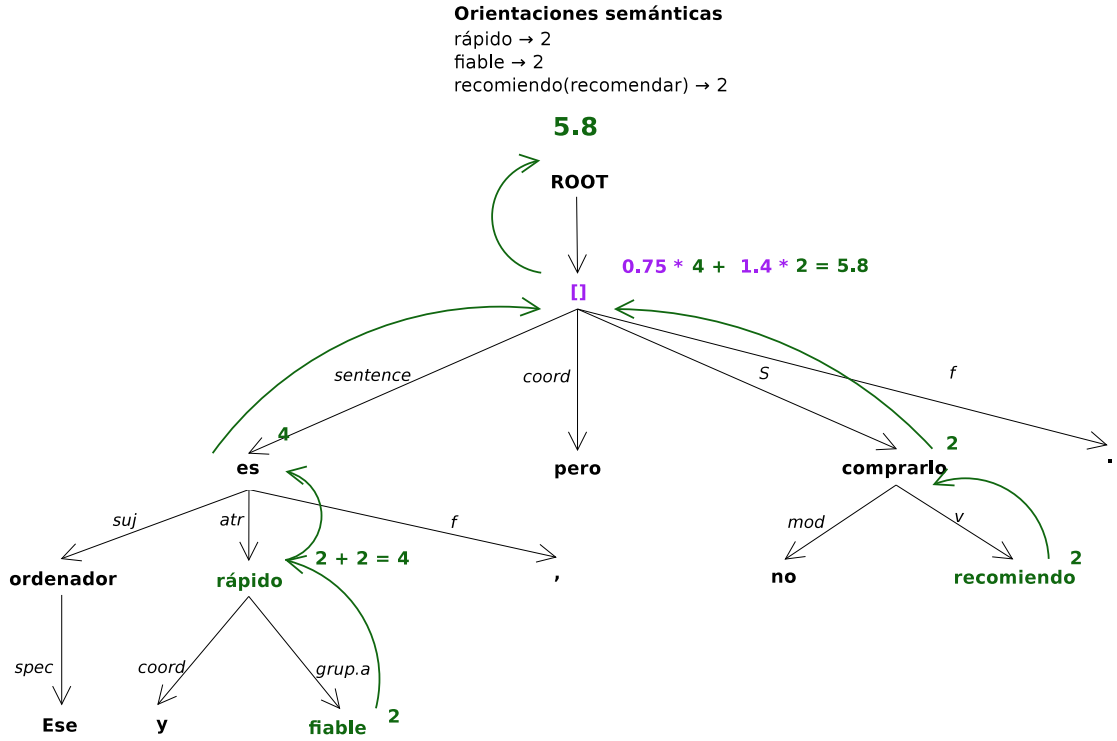


Figura 10.5: Análisis de la OS con tratamiento de las adversativas para la oración 5.1

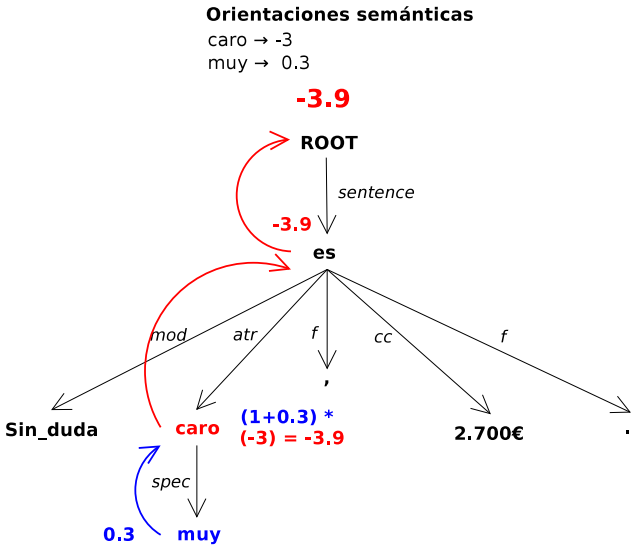


Figura 10.6: Análisis de la OS con tratamiento de las adversativas para la oración 5.2

De este modo la puntuación en este momento, para el texto completo, sería:

$$OS(Or_1) = \quad 5.8$$
$$OS(Or_2) = \quad \text{-}3.9$$

$$OS_{total} = OS(Or_1) + OS(Or_2) = 5{,}8 + (-3{,}9) = 1{,}9$$

En resumen, disminuir la importancia de lo expresado en la opinión general ayuda ligeramente a acercar el texto a su polaridad real, aunque la ausencia del tratamiento del *"no"* en la oración subordinada impide obtener todavía el sentimiento correcto para esta oración.

■

Capítulo 11

La negación

La negación es una de las construcciones lingüísticas que más influye en el análisis de la polaridad de los textos. En este capítulo se introducen algunas propuestas seguidas a lo largo de los últimos años para modelar la negación y así comprender mejor las ventajas de tratar este fenómeno desde un punto de vista sintáctico. En este trabajo en particular, se presenta un procedimiento para analizar la negación a partir de un árbol de dependencias, basándose en una serie de reglas heurísticas y se introduce el concepto de *alcance de la negación*. Al final del capítulo se incluye un ejemplo donde se trata la negación según el procedimiento propuesto.

11.1. Construcciones lingüísticas negativas

La negación es una construcción habitual a la hora de expresar una opinión. Una forma de negar una idea en castellano es mediante el término *"no"*, aunque existen otros negadores como *"sin"* o *"nunca"* que también son utilizados con frecuencia. Hay igualmente otros muchos términos que permiten invertir la polaridad de un término como es el caso de *"en absoluto"*, *"lo menos"* o *"lo mínimo"*, pero sus características sintácticas hacen que a menudo sea preferible tratarlos como intensificadores. Ese ha sido el caso de este sistema. En este proyecto sólo se han tratado como tales los negadores *"no"*, *"sin"* y *"nunca"*, que pasamos a describir:

- La estructura sintáctica para la negación *"no"* es prácticamente idéntica en todas las situaciones posibles. La única diferencia es el tipo de dependencia con el que se anota cuando el negador depende de un verbo en vez de cualquier otro tipo de construcción, tal y como se aprecia en la figura 11.1.

Figura 11.1: Estructura del árbol de dependencias para el negador *no*

- La estructura sintáctica de *"nunca"*, cuando actúa como negador, es equivalente a la del *"no"*. Los únicos cambios se producen en los tipos de dependencias con los que se anotan las relaciones, siendo algunas de sus etiquetas posibles las de complemento circunstancial, sintagma adverbial o modificador. Tanto en el caso de *"no"* como en el de *"nunca"*, el padre de la relación de dependencia no es en muchos casos, como podría pensarse en un primer momento, el término que debe ser negado. En este sentido, más adelante se detalla cómo actúa el sistema para saber cuáles son los términos que deben cambiar su polaridad.

- El negador *"sin"* sí tiene una estructura sintáctica distinta a la de los dos negadores anteriores, que pretende reflejar su carácter más local, como se observa en la figura 11.2.

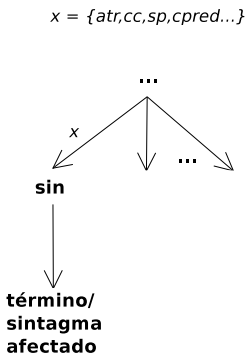


Figura 11.2: Estructura del árbol de dependencias para el negador *sin*.

El tipo de relación de dependencia x representado en la figura 11.2, varía según la función que desempeñe en la oración el sintagma afectado, de ahí que el arco pueda estar anotado como un atributo, un complemento circunstancial o un sintagma preposicional entre otros.

11.2. Aproximaciones de modelado

El interés creciente suscitado por la MO ha provocado, en particular, que hayan sido muchos los esfuerzos y las aproximaciones estudiadas para conseguir un tratamiento de la negación lo más completo posible. A continuación se introducen algunos de los enfoques más relevantes [15].

11.2.1. Inversión de la polaridad

Es uno de los enfoques léxicos más simples, donde la negación se modela invirtiendo la polaridad de los términos afectados. La expresión formal que emplea esta técnica para modificar un término con una OS de x se resume como:

$$Negación(x) = -x \tag{11.1}$$

Ejemplo 11.2.1. La siguiente tabla refleja el tratamiento de la negación para los términos subjetivos *"horrible"*, *"decepcionado"*, *"genial"* y *"magistral"* mediante inversión de la polaridad.

Término	OS		Término_{negado}	OS_{negada}
horrible	-4	→	no horrible	4
decepcionado	-3	→	no decepcionado	3
genial	4	→	no genial	-4
magistral	5	→	no magistral	-5

■

No obstante, esta solución no parece la más correcta para obtener la polaridad de los términos negados, algo comprensible si nos fijamos en el ejemplo 11.2.1, donde se observa que la negación de *"genial"* tiene una polaridad de -4, un sentimiento equivalente al de *"horrible"*, algo que no parece muy adecuado dado que *"no genial"* no expresa un sentimiento excesivamente negativo. La expresión 11.1 tampoco permite tratar correctamente la aparición conjunta de los fenómenos de negación e intensificación [3], como ilustra el siguiente ejemplo. En este caso la negación invertiría la polaridad del término una vez ya intensificado[1].

Ejemplo 11.2.2. Considerando el tratamiento conjunto de la intensificación y la negación para el término *"genial"*, de forma individual, mediante la inversión de la polaridad.

[1]Se supone la aproximación de intensificación propuesta en este trabajo, representada formalmente en la expresión 9.1.

$$\begin{aligned} &\text{no genial} && = \text{-4} \\ &\text{no muy genial} && = \text{-}(4 \times (1 + 0.3)) = \text{-5.2} \end{aligned}$$

de forma que resulta que el sentimiento de *"no muy genial"* es más negativo que el de *"no genial"*, lo que realmente no tiene sentido.

■

Otro de los inconvenientes de esta aproximación es saber cuáles son las palabras modificadas por un negador, esto es, el *alcance de la negación*. Se trata de un problema generalizado en el tratamiento de la negación, que se ve acentuado en las aproximaciones léxicas. En un primer momento podría pensarse en una estrategia que negase siempre el término a continuación de la negación, pero esto no es válido. Por ejemplo, en la frase *"No se lo había pasado bien"*, el negador *"no"* no afectaría a la polaridad de *"se"* sino a la de *"bien"*. Para resolver este problema se hace uso de los delimitadores[2]. Una opción clásica es negar toda la oración hasta encontrar un separador como un punto, o un nexo que indique el comienzo de una nueva frase. Otra opción más sofisticada, seguida por *The English SO Calculator*, se basa en emplear como delimitadores determinadas etiquetas de las palabras, como la de *pronombre* o *conjunción*. De esta forma, el alcance de la negación llegaría hasta un término etiquetado con una categoría gramatical que indique el fin del alcance de la negación. Nótese que esta opción no resolvería el caso de *"No se lo había pasado bien"*, ya que se detectaría que la etiqueta de *"se"* es un pronombre, se concluiría que finaliza ahí el efecto del negador y se realizaría un tratamiento incorrecto de esta construcción.

En resumen, aunque fácil de comprender, esta técnica presenta múltiples inconvenientes, razón por la que fue totalmente descartada. Para resolver los problemas de cambio de polaridad existe otra aproximación que detallamos a continuación, denominada *modificación de la polaridad*. Respecto al alcance de la negación introduciremos igualmente un mecanismo basado en el análisis sintáctico, que permite identificarlo de una forma mucho más fiable y precisa.

11.2.2. Modificación de la polaridad

Es la solución, también léxica, que implementan tanto *The English SO Calculator* [3] como *The Spanish SO Calculator* [2] para resolver la negación. En vez de invertir la polaridad, se modifica su valor en una cantidad fija de signo contrario al sentimiento de la palabra

[2]Elemento que representa un límite de hasta donde puede llegar el efecto de un negación.

afectada por el negador, con lo que se consigue un enfoque más realista de la negación. En los dos sistemas dicha cantidad es 4. Formalmente, para un término con polaridad con valor x, la modificación de la polaridad se obtiene como:

$$Negación(x_{positivo}) = x - 4 \tag{11.2}$$

$$Negación(x_{negativo}) = x + 4 \tag{11.3}$$

Ejemplo 11.2.3. En relación al tratamiento de la negación para los términos subjetivos *"horrible"*, *"decepcionado"*, *"genial"* y *"magistral"*, de forma individual, mediante la técnica de modificación de la polaridad, se tiene que:

Término	OS		Término_{negado}	OS_{negada}
horrible	-4	→	no horrible	(-4 + 4) = 0
decepcionado	-3	→	no decepcionado	(-3 + 4) = 1
genial	4	→	no genial	(4 - 4) = 0
magistral	5	→	no magistral	(5 - 4) = 1

■

A la vista de los resultados del ejemplo 11.2.3, esta técnica parece reflejar de forma adecuada la mezcla de sentimiento representada cuando se niega un término muy positivo o negativo, como se puede ver con el caso de *"genial"*. El efecto es más evidente aún con los términos más negativos o positivos como ocurre con *"magistral"*. Esta aproximación también resuelve correctamente la aparición conjunta de negación e intensificación sobre un mismo término. Como en la técnica de inversión de la polaridad, la negación actúa sobre la OS del término ya intensificado.

Ejemplo 11.2.4. Al considerar el tratamiento conjunto de la intensificación y la negación para el término *"genial"*, a nivel individual, mediante la modificación de la polaridad, tenemos que:

$$\begin{aligned} &\text{no genial} && = 4 - 4 = 0 \\ &\text{no muy genial} && = (4 \times (1 + 0.3)) - 4 = 1.2 \end{aligned}$$

En este caso *"no muy genial"* sí tiene una mejor valoración que *"no genial "*, incluso se advierte un sentimiento ligeramente positivo.

■

Respecto a como identificar los términos afectados por un negador, al tratarse de un método léxico es necesario emplear las mismas técnicas que en la inversión de la polaridad.

11.2.3. Control de la tendencia positiva en el lenguaje humano

No nos referimos aquí a una técnica de procesamiento de la negación, sino más bien a un mecanismo que permite contrarrestar el impreciso tratamiento que se hace de este fenómeno [2]. Las estrategias léxicas de MO presentan una predisposición a la clasificación positiva como consecuencia de la tendencia humana a emplear un lenguaje también positivo, aún cuando lo que se expresa sea una crítica negativa. El problema reside en que son muchas las personas que para expresar una opinión desfavorable no hacen uso de términos negativos, sino que emplean la negación del correspondiente antónimo: *"No ... bueno"* en vez de *"... malo"*, *"no ... bonito"* en vez de *"... feo"* o *"no ... barato"* en vez de *"... caro"* son algunos de los ejemplos de esta situación. De este modo, si la negación no se detecta correctamente, puede haber términos positivos cuya polaridad no es modificada, lo que incrementa la orientación semántica positiva del texto en cuestión.

La solución propuesta en sistemas de MO con base léxica pasa por potenciar la orientación semántica cuando se detecta una palabra negativa. *The English SO Calculator* [3] o *The Spanish SO Calculator* [2] aumenta actualmente la ponderación de este tipo de palabras en un 50 %. En nuestro proyecto, con el tratamiento de la negación realizado, no fue necesario potenciar la OS de estos términos para lograr un buen análisis de este fenómeno.

11.2.4. Identificación sintáctica de la negación

Este método [16] es el que ha demostrado obtener los mejores resultados a la hora de resolver la polaridad de una oración cuando en ella aparecen ocurrencias de términos negativos. El gran problema de las técnicas comentadas hasta ahora es la forma en la que identifican el alcance de la negación. Al ser aproximaciones léxicas necesitan emplear heurísticas y patrones para conocer dicho alcance, ya que no pueden comprender la estructura del texto. Esto lleva en ocasiones a interpretaciones incorrectas, sobre todo en idiomas como el castellano donde la colocación de palabras en la oración dispone de un grado elevado de libertad.

La solución propuesta representa un procedimiento formal y complejo basado en el análisis sintáctico para identificar el alcance real de la negación. En un primer paso se obtiene el *alcance candidato* de una sentencia, que contiene todas las palabras que aparecen después de la negación en la misma sentencia. Para obtener el *alcance real* se han de tener en cuenta la presencia de algunos factores adicionales:

- *Delimitadores estáticos*: Son elementos como signos de puntuación o conjunciones que

representan inequívocamente el comienzo de una nueva oración.

- *Delimitadores dinámicos*: Es una solución similar a la utilizada por *The English SO Calculator*, donde se buscan una serie de términos con una determinada etiqueta que permita identificar que ese elemento representa el inicio de una nueva frase, como pueden ser una conjunción o un pronombre.

- *Reglas heurísticas sobre la estructura sintáctica*: Establecen patrones que obtienen el alcance de la negación mediante el análisis de determinadas funciones sintácticas como el complemento directo o el atributo, así como la consideración de verbos, nombres o adjetivos de sentimiento.

Con estas reglas y delimitadores se define un procedimiento donde es posible descartar parte del alcance candidato hasta obtener solamente los términos que realmente están afectados por un negador.

11.3. Tratamiento de la negación

El sistema propuesto ha realizado un tratamiento de la negación basado en las técnicas de la identificación sintáctica de la negación y de la modificación de la polaridad, pero antes de explicar cómo se integró esta funcionalidad es conveniente introducir un problema asociado con el tratamiento de la negación en un entorno de MO. Para ello nos serviremos de un ejemplo sencillo. Supongamos una oración sin opinión, como el caso de la siguiente:

"No estoy en casa"

Si no se hace ninguna consideración a mayores, la negación se detectaría, y se calcularía su alcance para obtener la orientación semántica de los términos afectados, que en este caso[3] sería 0 ya que no aparece ningún término subjetivo. Luego se modificaría la polaridad lograda para dicho alcance, lo que según se tome el 0 como positivo o negativo, daría un resultado de -4 o +4. Ello indicaría la existencia de subjetividad en una oración que carecía de ella. Si las negaciones que afecten a todas las sentencias objetivas son tenidas en cuenta de esta forma, es muy probable que el sistema clasifique de forma sesgada y disminuya su precisión.

Esta situación hizo necesario considerar un mecanismo que permita determinar al sistema cuándo una rama del árbol de dependencias tiene asociada opinión o no, y así tener la

[3]Recordar que la OS de una oración se calcula, *a grosso modo* como la suma de las polaridades de sus términos.

posibilidad de controlar si es conveniente hacer el cambio de polaridad o no. La solución propuesta en el proyecto consistió en ampliar el objeto *SentimentInfo* que devolvían las funciones de visita, y que contenía la orientación semántica y la intensificación acumuladas en un nodo. Con el tratamiento de la negación se incorporó un nuevo atributo booleano, *subjectivity*, que determina si la rama que empieza en ese nodo tiene opinión asociada[4] o no.

Resuelto el tema de la existencia de opinión, el tratamiento de cualquier tipo de negación requiere llevar a cabo tres pasos, aunque el desarrollo de cada uno de ellos puede realizarse de una forma distinta según la estructura sintáctica que represente a cada negador:

1. La identificación del nodo de negación en el árbol sintáctico de dependencias.

2. La obtención del alcance de esa negación. En este punto, el sistema sigue un procedimiento puramente sintáctico para identificar el alcance de un negador, sin basarse en delimitadores léxicos.

3. La modificación de la polaridad acumulada en ese nodo.

A continuación se describe en detalle la realización de estos tres pasos para los negadores considerados en el sistema.

11.3.1. Los negadores *"no"* y *"nunca"*

En la figura 11.1 se mostraba la estructura sintáctica que el Ancora Corpus [14] utiliza para representar estas construcciones lingüísticas. En ella se aprecia que esta clase de negadores se representan siempre como nodos hojas en los árboles de dependencias, por lo que su alcance nunca puede determinarse a partir de sus descendientes. Ante esta situación, trabajos previos [16] y un estudio sobre el propio *corpus* sirvieron para determinar que los términos afectados por el negador son siempre nodos que bien actúan como el padre de la negación o bien como uno de sus hermanos[5] en el árboles de dependencias. Este es un aspecto fundamental que se tuvo en cuenta en las decisiones diseño de como identificar el nodo y el alcance de la negación.

[4]Un nodo del árbol de dependencias es subjetivo si él o alguno de sus descendientes tiene una entrada en los diccionarios de orientación semántica.

[5]Nodos al mismo nivel que el negador.

Identificación del nodo de negación

Se estudiaron dos alternativas en la identificación de estas dos construcciones, que pasamos a describir más en detalle:

1. *Identificar la negación en el propio nodo negador*: Los negadores *"no"* y *"nunca"* conocen su relación de dependencia, pero como cualquier otro nodo de la instancia del *DependencyGraph*, ignoran el ID de su padre y el de sus hermanos. Ello impide detectar el alcance de la negación en este punto porque se desconoce información fundamental sobre los posibles términos afectados.

2. *Identificar la negación en el nodo padre del negador*: Fue la opción utilizada en este trabajo. En el nodo padre es donde se puede conocer de forma inmediata[6] todos los nodos que se pueden ver afectados por la negación y por tanto es el punto en el que el alcance y la polaridad pueden ser determinados de forma eficiente. El inconveniente de esta aproximación es la necesidad de analizar los tipos de dependencia que mantiene cada nodo padre con sus hijos para ver si existe algún negador. Sin embargo, no es un problema grave ya que lo normal es que un nodo tenga pocos hijos e identificar un hijo negador consiste en examinar sus campos *word* y *rel*, lo que computacionalmente es poco costoso.

Identificada una construcción de este tipo, es necesario indicarlo de alguna manera para que la función de selección *visit_function* pueda devolver la función de visita que se encarga de resolver las negaciones. A este propósito, el método *get_semantic_category(self,graph,node)* asigna la etiqueta *neg* a los nodos que se encuentren negados por alguno de estos dos términos.

Alcance de la negación

La identificación de la negación sigue un enfoque sintáctico basado en [16], pero las reglas y el procedimiento son modificados y adaptados para aprovecharse de las características de los árboles de dependencias obtenidos por el analizador sintáctico del sistema. La estrategia seguida determina un alcance candidato y luego elimina los términos que realmente no forman parte de ese alcance, basándose en una serie de reglas heurísticas que emplean la estructura sintáctica.

[6] El nodo padre guarda los ID's de todos sus hijos, por tanto conoce el identificador del nodo negador y de los hermanos de éste.

El alcance candidato de estos negadores está formado por el nodo padre y por sus ramas hermanas en el árbol de dependencias. Para tratar de obtener el alcance real de una negación el sistema trabaja con las siguientes reglas heurísticas:

1. *Regla del padre subjetivo*: Una negación sintácticamente dependiente de un verbo, adjetivo o adverbio con orientación semántica solamente modifica la polaridad del término del que depende.

2. *Regla del complemento directo, complemento predicativo o atributo*: Si alguna de las ramas mantiene una de estas relaciones de dependencia con el nodo padre, entonces dicha rama también ve modificada su polaridad.

3. *Regla del complemento circunstancial*: La negación modifica la polaridad de la rama del árbol que actúa como complemento circunstancial. En el caso de que existan varios de éstos, sólo se modifica la orientación semántica de la rama más próxima al negador.

Estas reglas son analizadas en el orden de prioridad en el que han sido presentadas. Si un nodo cumple con alguna de las reglas, ésta se aplica y las reglas que queden por analizar no son procesadas. Si ninguna regla encaja con el nodo en cuestión, el alcance candidato (exceptuando el nodo padre[7]) es considerado como el alcance real, y la OS acumulada de todas las ramas ve modificada su polaridad. La función encargada de obtener el alcance de una negación de este tipo es *scope_detection()*, una subrutina interna de la función de visita para esta clase de negación, que es presentada en el siguiente punto. Este método tiene como objetivo devolver las ramas del árbol que están dentro del alcance real de la negación, la que están fuera de él y el tipo de de regla que se ha aplicado, para saber cómo debe calcularse la información semántica en el momento de la evaluación. Para ello, se creó una clase denominada *NegationInfo* que permite guardar esta información y que contiene tres atributos:

- *neg_branches*: Lista que almacena los ID's de los hijos del nodo que representan ramas que deben modificar su polaridad.

- *other_branches*: Lista que contiene los ID's de las raíces de las ramas que están fueran del alcance de la negación.

[7]Si no se cumple la regla del padre subjetivo se asegura que es un nodo objetivo que no aporta nada al cálculo de la polaridad.

- *delimiter*: Indica el tipo de delimitador utilizado para obtener las listas *neg_branches* y *other_branches*. Su valor viene determinado por la regla heurística utilizada para obtener el alcance de la negación.

Modificación de la polaridad

Los distintos alcances posibles para un negador de este tipo obligaron a definir expresiones asociadas que permitiesen invertir sólo la parte de la orientación semántica acumulada en el nodo padre. Todas estas expresiones tienen en común la intensificación existente en un nodo de esta clase, dado por:

$$Int_{nodo\ con\ negacion} = 0 \tag{11.4}$$

En relación al cálculo de la orientación semántica, éste varió según la regla aplicada y fue necesario estudiar cómo hacer el cambio de la polaridad (*Flip*) para las distintas reglas según la rama sea subjetiva (*Subj*)[8] o no. El valor fijo utilizado para la modificación de la polaridad fue 4, el mismo que el del sistema *The Spanish SO Calculator*, ya que fue el valor con el que se obtuvieron los mejores resultados sobre el SFU Spanish Review Corpus. La expresión que calcula la polaridad de un nodo con negación, que encaja con la regla del padre subjetivo, es la siguiente:

$$OS_{neg} = Flip((1 + \sum_{i=1}^{n} Int_{descendiente_i}) \times OS_{padre}) \times Subj_{padre} + \sum_{i=1}^{n} OS_{descendiente_i} \tag{11.5}$$

Para las reglas del complemento directo, atributo o complemento predicativo y la regla del complemento circunstancial, la orientación semántica se calcula siguiendo la expresión 11.6. En este caso modificar la polaridad aún sin que la rama sea subjetiva demostró mejorar sensiblemente el rendimiento del sistema. Una posible justificación para este fenómeno es que esta estrategia simula el control de la tendencia positiva del lenguaje humano seguida por los sistema léxicos de MO, comentado anteriormente. Así pues, la expresión que permite calcular la OS en para un nodo que cumpla una de estas reglas, es la siguiente:

$$OS_{neg\ nodo} = Flip(\sum_{i=1}^{n} OS_{neg\_branch_i}) + \sum_{i=1}^{m} OS_{other\_branch_i} \tag{11.6}$$

Cuando no se cumple ninguna de las reglas establecidas, la polaridad se modifica negando la totalidad acumulada en el nodo padre, algo que en términos formales se representa como:

[8] *Subj* vale 1 si hay subjetividad, y 0 en otro caso.

$$OS_{neg\ nodo} = Flip(\sum_{i=1}^{n} OS_{descendiente_i}) \times Subj_{nodo} \tag{11.7}$$

donde

$$Subj_{nodo} = Subj_{descendiente_1} \bigvee Subj_{descendiente_2} \bigvee ... \bigvee Subj_{descendiente_n}$$

Aunque las expresiones para calcular la polaridad de cada regla son distintas, la función de visita aplicada a estos nodos es siempre la misma, *visit_negation*, cuyo esqueleto se muestra en el siguiente pseudocódigo:

```
def visit_negation(graph,node):

  def scope_detection():
    ...

  def neg_function(delimiter):
    ...

  scope_of_negation = scope_detection()
  neg_branches = scope_of_negation.get_neg_branches()
  other_branches = scope_of_negation.get_other_branches()
  type_delimiter = scope_of_negation.get_delimiter()
  return neg_function(type_delimiter)(node,neg_branches,other_branches)
```

La función interna *scope_detection()* es la encargada de obtener el alcance de la negación en un determinado nodo, como se había explicado en el punto anterior. En cuanto al método *neg_function(delimiter)*, tiene como objetivo devolver la función que permita calcular toda la información semántica en un nodo con negación basándose en el tipo de regla que encajó con la estructura del mismo. Su diseño sigue una estrategia muy similar al empleado por el mecanismo de selección de las funciones de visita.

11.3.2. El negador *"sin"*

El carácter local de este negador, junto con las diferencias en la representación en el árbol de dependencias respecto a los demás tipos de negaciones, obligó a que esta construcción recibiera un tratamiento especial en el sistema.

Identificación del nodo de negación

Con los negadores *"no"* y *"nunca"*, se habían señalado las ventajas de identificar la negación en el nodo padre por la forma en la que se representaban las dependencias, pero con *"sin"* esta estrategia no tiene sentido. La razón es que en este caso los términos afectados por la negación son elementos dependientes de nodo negador, como quedaba reflejado en la figura 11.2. En otras palabras, este nodo siempre actúa como padre del elemento cuya OS debe ser modificada, lo que permite realizar una identificación más directa que la seguida con el resto de negadores. Así, detectar esta clase de negador se reduce a procesar el campo *word* del nodo en cuestión, para ver si se corresponde con un término *"sin"*. Identificado el nodo, la función *get_semantic_category(self,graph,node)* le asigna la etiqueta *neg_sin* para que el selector de funciones de visita sepa cuál es la función que debe devolver.

Alcance de la negación

La estructura sintáctica utilizada en este tipo de negación también evitó tener que seguir el procedimiento de identificación del alcance de la negación empleado por *"no"* y *"nunca"*. En este caso no es necesario partir de un alcance candidato, ni establecer reglas heurísticas para la selección del alcance real, siempre formado por los descendientes del nodo *"sin"*. Normalmente esta clase de nodo sólo tiene un hijo, aunque excepcionalmente puede no ser así.

Modificación de la polaridad

Las expresiones que obtienen toda la información semántica en un negador *"sin"* son las siguientes:

$$Int_{nodo\ sin} = 0 \tag{11.8}$$

y

$$OS_{neg\ nodo} = Flip(\sum_{i=1}^{n} OS_{escendiente_i}) \times Subj_{sin} \tag{11.9}$$

con

$$Subj_{sin} = Subj_{descendiente_1} \bigvee Subj_{descendiente_2} \bigvee ... \bigvee Subj_{descendiente_n}$$

y donde una de las peculiaridades es la modificación del valor del *Flip* para esta negación. Negar con *"no"* o *"nunca"* indica expresar la inexistencia del término negado, mientras que la

utilización del *"sin"* parece reflejar una negación más relajada. Los mejores resultados sobre la evaluación SFU Spanish Review Corpus determinaron 3,5 como el valor de modificación de la polaridad que debía utilizar esta construcción, un valor menos agresivo que el empleado para el resto de los negadores. La modificación de la OS sólo tiene efecto cuando alguno de sus descendientes es una rama subjetiva.

Ejemplo 11.3.1. El tratamiento de la negación permite retomar el ejemplo 10.2.2 y tratar el fragmento de la oración 5.1 *"no recomiendo comprarlo"* de forma correcta, como se ilustra en la figura 11.3 en color naranja. Así, al procesar el nodo *"comprarlo"*, se detecta que es un nodo negado y se procede a la identificación del alcance de la negación. Se establece el alcance candidato, que en este caso está formado sólo por *"comprarlo"* y *"recomiendo"*, y se inicia el procedimiento de identificación de los términos afectados. En este caso ninguna de las tres reglas heurísticas con las que el sistema trabaja coinciden, por lo que el alcance real estaría formado por el término *"recomiendo"*; y la polaridad en el nodo padre se calcularía siguiendo la expresión 11.7.

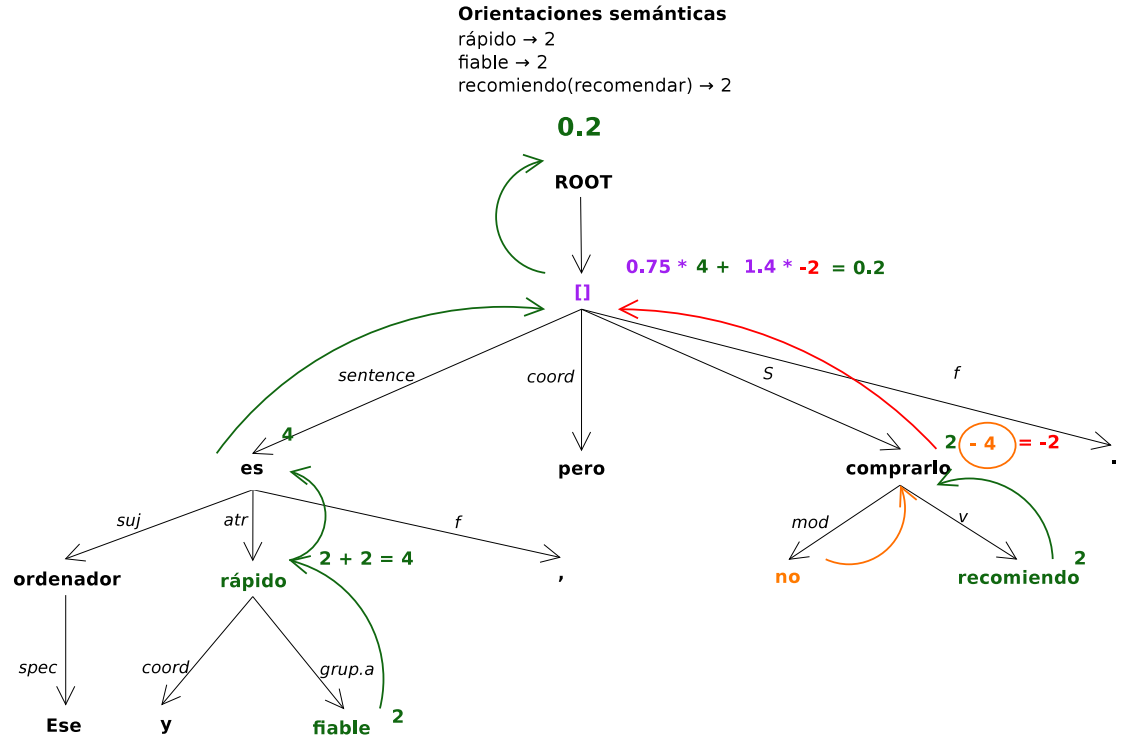


Figura 11.3: Análisis de la OS con tratamiento de la negación para la oración 5.1

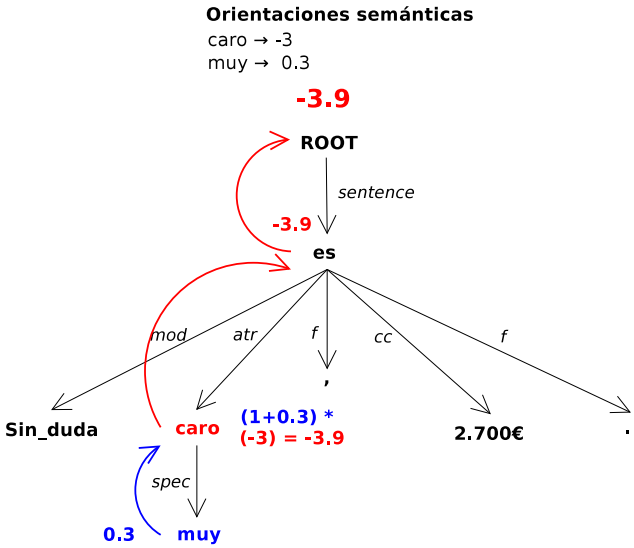


Figura 11.4: Análisis de la OS con tratamiento de la negación para la oración 5.2

La puntuación del texto completo, formado por la sentencias 5.1 y 5.2, una vez incorporados todos los aspectos sintácticos, sería la siguiente:

$$OS(Or_1) = \quad 0.2$$
$$OS(Or_2) = \quad \text{-}3.9$$

$$OS_{total} = OS(Or_1) + OS(Or_2) = 0{,}2 + (-3{,}9) = -3{,}7$$

El sentimiento general parece expresar correctamente el sentir del texto. Para la oración 5.1 se obtiene un sentimiento cercano a 0, lo que refleja que la opinión para esta sentencia contrapone tanto ideas a favor como en contra. Por contra, la oración 5.2 es una crítica claramente negativa, algo que también queda reflejado en el cálculo de la OS y que determina que al final el texto completo sea también clasificado como negativo.

■

Capítulo 12

Resultados y rendimiento

En este capítulo se presentan los resultados obtenidos para el sistema con las distintas funcionalidades desarrolladas. El *corpus* empleado para el desarrollo y la evaluación fue el SFU Spanish Review Corpus, de forma que el rendimiento final del sistema propuesto en este proyecto puede compararse fácilmente con el del sofisticado *The Spanish SO Calculator* [2]. También se muestran los resultados obtenidos con los *corpus* HOpinion y Spanish Movie Reviews, para así conocer cómo se comporta el sistema sobre *corpus* distintos a los utilizados para su desarrollo.

12.1. Pruebas realizadas

Para cada versión del sistema se realizaron pruebas de precisión que permitiesen realizar un análisis exhaustivo del funcionamiento del sistema y del origen de los resultados. Así, se realizaron mediciones para las distintas versiones, donde cada una incluye las funcionalidades de la anterior:

1. Tratamiento de adjetivos, verbos, sustantivos y adverbios individualmente.
2. Incorporación de construcciones lingüísticas intensificadoras.
3. Integración del tratamiento de las oraciones subordinadas adversativas.
4. Incorporación del fenómeno de la negación.
5. Agregación de factores léxicos.

Se había comentado que durante el desarrollo del sistema, el conjunto de documentos utilizado para la evaluación había sido el *corpus* de críticas de la SFU, formado por opiniones

de la página web de Ciao. En esta web, los autores clasifican sus documentos en una escala de valoración en estrellas de 1 a 5, donde 1 es lo más negativo y 5 lo más positivo. De las 400 opiniones del *corpus*, las 200 negativas son críticas clasificadas con 1 ó 2 estrellas y las 200 positivas fueron etiquetadas con 4 ó 5 estrellas[1], es decir, el punto medio representado por las 3 estrellas, es ignorado. Para las distintas categorías de documentos de este *corpus* se desglosa la precisión obtenida tanto en textos positivos como negativos, de forma que se pueda comprender claramente cómo se comporta el sistema en los distintos ámbitos. Dado el reducido tamaño de este *corpus* la totalidad del mismo fue empleada tanto para el desarrollo como para la evaluación, lo que conlleva un riesgo de tomar decisiones de diseño adaptadas a estos documentos, pero no extrapolables a otros. Para comprobar que el sistema se comporta correctamente en otros *corpus*, la versión final del mismo fue evaluada también con los conjuntos de documentos de HOpinion y Spanish Movie Reviews. Se había comentado que estos documentos estaban también clasificados en una escala de 1 a 5 estrellas. No obstante, en la línea del *corpus* de la SFU, el sistema sólo analizó los textos de 1 y 2 estrellas, como negativos, y los de 4 y 5, como positivos. La categoría de 3 estrellas es ignorada ya que en ella se encuentran tanto textos favorables como desfavorables.

12.2. Evaluación sobre el SFU Spanish Review Corpus

El cuadro 12.1 muestra los resultados obtenidos para los distintos dominios cuando son evaluados por el sistema base. En los cuadros 12.2 y 12.3 se indican respectivamente las precisiones obtenidas una vez el analizador de la orientación semántica es capaz de tratar los fenómenos de la intensificación y las oraciones adversativas. El cuadro 12.4 representa los resultados para el sistema una vez se incorporó el tratamiento de la negación. Por último, en el cuadro 12.5, se muestra el rendimiento cuando son tratados determinados aspectos del discurso, que no pueden resolverse a nivel sintáctico. A continuación se detallan los resultados para cada versión del sistema.

- Los resultados del sistema base, mostrados en el cuadro 12.1, indican que en términos generales el cálculo de la OS para textos positivos parece realmente prometedor, aunque no es así para los enunciados negativos. Sin embargo, estos buenos resultados para las

[1]En los documentos no se encuentra anotada la clasificación en estrellas que había sido asignada por su autor. Únicamente se distingue entre documentos negativos y positivos.

opiniones favorables están condicionados por la marcada tendencia positiva[2] de esta versión, donde para 323 de las 400 críticas que componen el *corpus* de SFU se obtuvo esa valoración. Es decir, la elevada precisión positiva no se debe al buen tratamiento que se hacía de este tipo de textos, sino al sesgo inicial del sistema que provoca que casi cualquier enunciado sea clasificado como una opinión favorable. De hecho, si realizamos un test de significatividad chi-cuadrado respecto al número de positivos y negativos esperados, 200 de cada tipo en el *corpus*, obtenemos un valor $p < 0,01$; lo que en este caso significaría que esta versión sufre una tendencia positiva. También es importante destacar que para los dominios culturales, donde los gustos personales son un factor importante, como es el caso de los libros, la música o las películas, su precisión es menor que la media general obtenida para el sistema base.

Categoría	**Precisión(%)$_{neg}$**	**Precisión(%)$_{pos}$**	**Precisión(%)$_{media}$**
Hoteles	28	100	64
Libros	8	92	50
Móviles	32	100	66
Coches	44	88	66
Musica	28	96	62
Ordenadores	40	92	66
Películas	20	88	54
Lavadoras	48	84	66
Media	31	92.5	61.75

Cuadro 12.1: Precisión del sistema base

- La precisión media del sistema una vez incorporado el tratamiento de la intensificación mejora sensiblemente (+4,25 %) respecto a la versión inicial del sistema, como queda reflejado en el cuadro 12.2. Por otra parte, un resultado difícil de explicar en esta versión del sistema, es el gran incremento del rendimiento en la extracción de la OS para las críticas negativas, concretamente un aumento de un 14 %. A este respecto, en términos teóricos no hay ninguna razón para que el tratamiento de la intensificación mejore el rendimiento de los textos negativos y lo empeore para los positivos, como ha sido el caso. Desde un punto de vista lingüístico, podría servir para detectar más fielmente críticas donde los usuarios realmente estén descontentos y para reflejarlo utilicen construcciones

[2]Probablemente debida a lo ya comentado en capítulos anteriores sobre la tendencia positiva del lenguaje humano y al no tratamiento de la negación.

intensificadoras.

Aún así, en esta versión se mantenía una predisposición positiva: 284 críticas clasificadas como positivas frente a sólo 116 negativas y el resultado del test chi-cuadrado seguía siendo estadísticamente significativo ($p < 0,01$). Respecto a los resultados para las distintas categorías, la precisión media para los ámbitos de libros, películas y música sigue sin superar la media general del sistema, mientras que en dominios donde la calidad del producto o servicio no se ve afectada tanto por los gustos personales la precisión es mejor, como por ejemplo, en los hoteles, los móviles o las lavadoras.

Categoría	**Precisión(%)**$_{neg}$	**Precisión(%)**$_{pos}$	**Precisión(%)**$_{media}$
Hoteles	52	96	74
Libros	24	88	56
Móviles	44	100	72
Coches	44	84	64
Musica	40	92	66
Ordenadores	44	84	64
Películas	40	76	58
Lavadoras	72	76	74
Media	45	87	66

Cuadro 12.2: Precisión al incorporar la intensificación

- La inclusión de las oraciones subordinadas adversativas apenas tuvo influencia en el rendimiento general del sistema (+1 %), como se ilustra en el cuadro 12.3, ni el número de textos clasificados como positivos o negativos, ni el resultado del test de chi-cuadrado sufrieron cambios significativos. La razón de que el incremento sea reducido puede estar en el tratamiento incompleto de este tipo de cláusulas. Hay que recordar que el sistema procesa los nexos *"pero"*, *"mientras"*, *"mientras que"*, *"sino"* y *"sino que"*; pero ignora otros muchos dadas las múltiple formas en que los representaba el *corpus* Ancora en el árbol de dependencias.

- La incorporación del tratamiento de la negación es claramente beneficiosa para el sistema, como se observa en el cuadro 12.4, incrementando en un 8,5 % el rendimiento general respecto a la versión anterior. Nótese que el rendimiento respecto a las críticas negativas aumenta en casi un 30 %, aunque sea a costa de perder algo de precisión en la clasificación de opiniones positivas. El tratamiento de la negación permitió también

Categoría	**Precisión(%)$_{neg}$**	**Precisión(%)$_{pos}$**	**Precisión(%)$_{media}$**
Hoteles	52	96	74
Libros	24	88	56
Móviles	40	100	70
Coches	44	88	66
Musica	40	96	68
Ordenadores	52	84	68
Películas	40	76	58
Lavadoras	72	80	76
Media	45,5	88.5	67

Cuadro 12.3: Precisión al incorporar las subordinadas adversativas

contrarrestar la tendencia positiva del sistema y del lenguaje humano presente en los mismos: 204 críticas clasificadas como positivas por 196 negativas, obteniendo para el test chi-cuadrado un valor $p = 0,6892$, lo que implica que no se trata de una diferencia estadísticamente significativa. Este resultado, unido a la precisión del sistema; indica que en estos momentos ya se está realizando un buen análisis de la polaridad. Sobre los resultados para las distintas categorías, parece confirmarse que para dominios como las películas o los libros existen más dificultades para resolver su OS correctamente, mientras que para ámbitos como los hoteles o las lavadoras los resultados ya son realmente buenos.

Categoría	**Precisión(%)$_{neg}$**	**Precisión(%)$_{pos}$**	**Precisión(%)$_{media}$**
Hoteles	92	80	86
Libros	64	76	70
Móviles	64	88	76
Coches	72	76	74
Musica	68	84	76
Ordenadores	76	80	78
Películas	64	56	60
Lavadoras	96	72	84
Media	74,5	76,5	75,5

Cuadro 12.4: Precisión al incorporar la negación

- Los resultados del cuadro 12.5 ilustran el rendimiento del sistema una vez incorporados aspectos léxicos, como la mayor importancia de las últimas frases del texto o la

ampliación del espectro de las polaridades de los diccionarios de orientación semántica. La precisión del sistema se incrementó hasta en un 3,5 %, clasificando 218 textos como positivos y 182 como negativos[3]. Sin embargo, aunque menos textos fueron clasificados como negativos en comparación con la versión anterior, la precisión en este tipo de críticas no se redujo apenas. Esto implica que estos aspectos no sintácticos permitieron corregir algunos textos positivos que habían sido mal clasificados.

Categoría	**Precisión(%)$_{neg}$**	**Precisión(%)$_{pos}$**	**Precisión(%)$_{media}$**
Hoteles	92	88	90
Libros	64	84	74
Móviles	72	88	80
Coches	68	80	74
Musica	64	88	76
Ordenadores	80	92	86
Películas	64	68	66
Lavadoras	88	76	82
Media	74	83	78,5

Cuadro 12.5: Precisión del sistema final

A la vista de las pruebas, entre los resultados obtenidos para las construcciones lingüísticas, la negación parece ser el fenómeno con el que se logra una mayor mejora en el sistema. Además permite controlar en buena medida la tendencia positiva del lenguaje humano, algo que los sistemas léxicos no lograban tratar directamente.

Entre los resultados para las distintas categorías destaca el elevado rendimiento obtenido para dominios como hoteles u ordenadores, tanto para las críticas negativas como las positivas, pero también son relevantes las precisiones para las categorías de libros, música y en especial películas, por debajo de la media del sistema en prácticamente todas las versiones. Hay varios factores que pueden influir en ello:

- *La invalidez de la polaridad genérica para determinados ámbitos.* Palabras como *"guerra"*, *"combate"* o *"huérfano"* aparecen anotadas en el SODictionariesV1.11Spa con una connotación negativa, pero es probable que en un ámbito como el de los libros o las películas no sea así, sino que sean utilizadas para ambientar la temática o describir un

[3]Para resultado para el test chi-cuadrado se obtuvo un valor $p = 0,0719$ por lo que la diferencia no sería significativa a un nivel de confianza del 99 %.

argumento. Véase con un fragmento de un texto del *corpus* SFU cuya polaridad debería ser positiva:

"Los pequeños dejan sus juegos porque se trasladan junto con sus padres a otro lugar de la finca, una especie de casita de campo que ellos llaman la cabaña, en la creencia de que estarán a salvo de los combates (estamos en el llamado frente del este durante la II Guerra Mundial) que según dicen se librarán cerca de la carretera. Hannibal contará 6 o 7 años, su hermanita Mischa 3. Desafortunadamente un tanque ruso parará junto a la cabaña para aprovisionarse de agua...y poco después...se ven envueltos entre proyectiles, como resultado los niños quedarán huerfanos. [..] Una película buena , interesante y muy bien rematada [...]"

- *Los gustos personales de cada persona.* Aun dentro de un mismo dominio, hay palabras cuya connotación puede variar según quién la emplee o la lea. Esta subjetividad no es tan frecuente en dominios como hoteles u ordenadores donde la calidad suele tener unos criterios establecidos y aceptados por la mayoría. Sin embargo, es muy distinto en ámbitos de entretenimiento, como es el caso de la música, los libros o las películas. Así, que la temática de una película o una novela sea *"romántica "*, o que la historia que se cuente sea *"lenta"* o *"sencilla"*, podría resultar muy aburrido para una persona, pero interesante para otra.

Analizados los resultados para el trabajo propuesto, queda comparar su rendimiento con otras propuestas. Ya hemos comentado que *The Spanish SO Calcultator* es un sistema de MO para el castellano que también emplea como recurso semántico el SODictionariesV1.11Spa y el *corpus* de la SFU para la evaluación. Esto nos permite comparar la aproximación léxica de ese sistema con la alternativa sintáctica propuesta en este trabajo. Los resultados parecen prometedores como se observa en la figura 12.1, que compara las versiones de nuestro sistema con *The Spanish SO Calcultator* (74,5 %), ya que la solución sintáctica ha conseguido mejorar los resultados de forma sensible, aún tratando menos construcciones lingüísticas[4]. Como en cualquier aplicación de PLN, la precisión del sistema se ve afectada por la incertidumbre a la que debe hacer frente. Los segmentadores, los etiquetadores o los analizadores sintácticos no son totalmente precisos, invonveniente que se ve agravado cuando actúan conjuntamente, como en este proyecto. A esta imprecisión hay que sumar dos factores más en el ámbito que nos ocupa: la calidad de entrada de los textos y la incompletitud y subjetividad[5] de las entradas

[4]The Spanish SO Calcultator es capaz de tratar semánticamente verbos modales y condicionales para calcular la polaridad de un texto.

[5]Dependendiente de los gustos y percepciones de quiénes los crean.

de los diccionarios. Todos estos factores acaban influyendo negativamente en el cálculo de la OS y afectan inevitablemente al rendimiento del sistema.

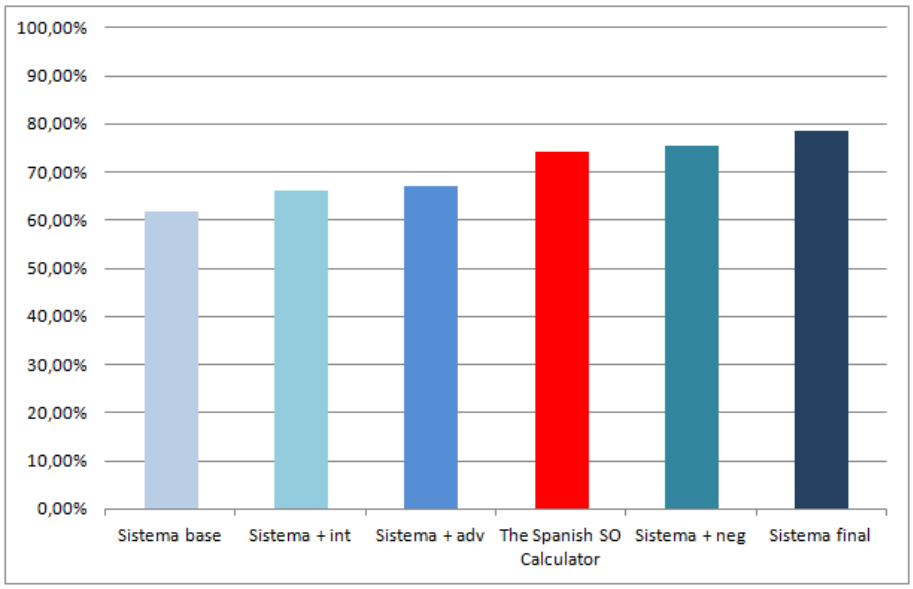


Figura 12.1: Comparación entre nuestro sistema y The Spanish SO Calculator

12.3. Evaluación sobre el HOpinion

Visto el comportamiento del sistema para este *corpus*, se comprobó el rendimiento del mismo con otros *corpus*. Se había comentado que la versión final de este trabajo también se evaluó con el HOpinion[6], formado por críticas de hoteles. Respecto a estas críticas se observa en el cuadro 12.6 que el rendimiento para los textos positivos es realmente bueno, en especial con las opiniones más favorables. La precisión para los negativos también es buena, llegando a un 83 % para los textos clasificados con una estrella, pero el rendimiento parece resentirse sensiblemente para los textos de dos estrellas, donde el sentimiento ya es más difícil de analizar. En estos documentos la mezcla de argumentos a favor y en contra es común y la clasificación por parte de su autor como bastante negativo es, en muchas ocasiones, cuestionable. Véase un ejemplo de un documento[7] de dos estrellas de este *corpus*:

[6]No nos consta que ningún otro sistema haya evaluado este *corpus*, por lo que no ha sido posible comparar resultados.

[7]Ejemplo extraído del HOpinion *corpus* sin realizar ningún tipo de corrección ortográfica ni sintáctica.

"Hotel cerca de las ruinas de Uxmal, si tienes la suerte de estar en la 2 o superior planta puedes ver el manglar y al fondo las ruinas de Uxmal. Muy parecido a un hotel que estuve en Tunez en medio del desierto, pero esta vez en medio del manglar. Acojedor a simple vista, poca gente. Piscina correcta con red para jugar al volley. Habitaciones con Aire acondicionado, son muy grandes. En cuestion de limpieza mas o menos limpias aunque al estar en pleno manglar existen gran numero de insectos y es dificil tenerlo siempre limpio, muchos mosquitos, y las ventanas tienes rendijas por las cuales se puede colar cualquier cosa, quizas deberian modificar las ventanas a la terraza. Tiene un baño completo, ducha caliente a partir de las 5, y TV. Respecto a la cena y desayuno muy malos, poca variedad y de calidad mala, precio excesivo de las bebidas. Hotel para estar 1 dia como mucho, hotel de transito."

Nº estrellas	**Acertados**	**Total**	**Precisión(%)**
1	704	841	83,71
2	835	1269	65,80
4	5592	6244	89,56
5	5798	6112	94,86

Cuadro 12.6: Precisión sobre el HOpinion

12.4. Evaluación sobre el Spanish Movie Reviews

Respecto a la evaluación del *corpus* de críticas de cine, Spanish Movie Reviews, el rendimiento en general es bueno, como se ilustra en el cuadro 12.7 e incluso mejora el obtenido para el SFU Spanish Reviews para las críticas positivas, aunque funciona bastante mal para los documentos de dos estrellas, como en el caso anterior.

Nº estrellas	**Acertados**	**Total**	**Precisión(%)**
1	243	351	69,23
2	369	923	39,48
4	706	890	79,33
5	366	461	79,39

Cuadro 12.7: Precisión sobre el Spanish Movie Reviews

Este *corpus* también ha sido evaluado por otros trabajos como [17]. En dicho trabajo, se crea un diccionario específico para este *corpus*; concretamente se obtienen bigramas con polaridad asociada partiendo de palabras semilla. A este respecto, se proponen distintos

conjuntos semilla: uno simple (v1), formado sólo por los términos *"excelente"* y *"malo"*, y otro múltiple (v2) constituido por varios términos positivos y negativos. Así, es posible lograr un conjunto especializado de términos que expresan opinión dentro de este ámbito, lo que supone una ventaja importante respecto a nuestra propuesta. Por ejemplo, *"banda sonora"* se considera como una expresión positiva, cuando en realidad es una expresión objetiva carente en sí misma de polaridad[8], por lo que nuestro sistema sería incapaz de extraer opinión de ella, dado que en los diccionarios semánticos utilizados no se advierte que estos términos indiquen sentimiento; al menos desde un punto de vista genérico. Además es importante destacar que estos resultados, ilustrados en el cuadro 12.8, no son totalmente comparables; dado que en [17] sólo se evalúan 400 críticas[9] (200 positivas, 200 negativas). Aun así, nuestra propuesta parece mejorar el rendimiento en los textos positivos, aunque no sucede lo mismo con las críticas desfavorables debido al bajo rendimiento que obtuvo nuestro sistema con los textos clasificados con dos estrellas.

Sistema	**Precisión**$_{pos}$**(%)**	**Precisión**$_{neg}$**(%)**
Nuestro sistema	79,35 %	48,04 %
Sistema bigramas$_{v1}$	35,5 %	91,5 %
Sistema bigramas$_{v2}$	70 %	69 %

Cuadro 12.8: Comparación de la precisión del Spanish Movie Reviews con otros sistemas

[8]Es probable que el hecho de considerar esta expresión como positiva se deba a que en el conjunto de documentos considerado sólo se hable de la banda sonora cuando ésta es buena; en aquellos casos en que la banda sonora no es destacable, simplemente no se menciona.

[9]Este trabajo también descarta las críticas de 3 estrellas.

Capítulo 13

Conclusiones y trabajo futuro

En este capítulo presentamos las conclusiones sobre el trabajo desarrollado y se resaltan los aspectos más importantes sobre los resultados obtenidos. También se realiza una exposición sobre las líneas futuras, enfocadas a lograr un sistema MO más eficiente y preciso.

13.1. Conclusiones

La MO es un área reciente de investigación que ha visto como su importancia se ha acrecentado notablemente en los últimos años, debido fundamentalmente al interés de las empresas en analizar las opiniones que los consumidores expresan en la web.

Hasta la fecha, muchas de las soluciones resolvían la orientación semántica de los textos mediante un enfoque casi exclusivamente léxico. Esta estrategia permite extraer correctamente el sentimiento de términos a nivel individual (adjetivos, verbos, sustantivos, adverbios), pero presenta serias limitaciones cuando lo que se quiere tratar son construcciones lingüísticas más complejas, donde es necesario interpretar el contenido de las oraciones.

En este trabajo se propone una aproximación sintáctica que pretende solventar las carencias de las soluciones actuales. Para ello, se emplean técnicas de análisis de dependencias que extraen los árboles sintácticos de las oraciones para así tratar tres construcciones lingüísticas de impacto en este ámbito: la intensificación, las oraciones subordinadas adversativas y la negación. Respecto a los dos primeras, se sigue una estrategia basada en ponderaciones para obtener el sentimiento en los fragmentos del árbol que representan estos fenómenos. Para el fenómeno de la negación se establece un procedimiento puramente sintáctico, aprovechando las características de los árboles de dependencias, para determinar cuáles son los términos de la oración afectados por un negador. Así, la orientación semántica de un texto se resuelve

mediante una evaluación de árbol de dependencias, siguiendo una aproximación semántica donde se emplean una serie de diccionarios que refieren la polaridad para términos subjetivos. El sistema considera también otros aspectos del discurso que no presentan naturaleza sintáctica, como son el empleo de las palabras mayúsculas o el mayor énfasis implícito en las últimas oraciones de un texto.

Para la evaluación del trabajo se utiliza el SFU Spanish Review Corpus, que contiene documentos de distintos dominios y que permite estudiar el comportamiento del sistema en toda clase de ámbitos. Se realizaron pruebas de precisión para evaluar el efecto de incorporar distintas construcciones lingüísticas, revelando que el tratamiento de la intensificación, y especialmente el de la negación, son fundamentales para lograr un sistema preciso y robusto. Los resultados finales muestran que el trabajo propuesto mejora sensiblemente el rendimiento respecto a otros sistemas léxicos sofisticados, como *The Spanish SO Calculator*, evaluado con el mismo *corpus* y que emplea los mismos diccionarios de OS, aún considerando menos construcciones lingüísticas. Respecto al rendimiento según el tipo de documento, el sistema se muestra más eficaz en dominios donde los gustos personales no son un factor importante, pero ve disminuida su precisión en ámbitos más subjetivos, como las películas, la música o los libros. Esto es debido en parte al discurso más literario empleado en estos ámbitos, donde la polaridad de las palabras tiene un valor más individualizado e íntimo.

El sistema final también fue evaluado sobre otros *corpus* especializados en un dominio en concreto, obteniendo para ellos resultados muy similares a los documentos de ese mismo ámbito en el SFU; lo que demuestra que el buen rendimiento del sistema no está condicionado por el conjunto de documentos empleados durante el desarrollo.

13.2. Trabajo futuro

Nuestra principal contribución se deriva de la integración sintáctica en el sistema de MO, utilizando para ello un analizador basado en dependencias, lo que en sí supone aplicar las tendencias más recientes en lo que al ámbito sintáctico se refiere [16]. En concreto, el analizador de dependencias propuesto se basa en el algoritmo *Nivre arc-eager* [8], uno de los más utilizados, pero existen otros algoritmos más recientes que con los que se podría obtener una mayor precisión en el análisis sintáctico. A este propósito se podrían utilizar algoritmos como el *2-planar* [18], que ha conseguido mejorar el estado del arte en cuatro de los *corpus* propuestos en la *CoNLL-X shared task* [9], además de reducir el tiempo de análisis respecto

a otros algoritmos, lo que supondría mitigar uno de los cuellos de botella del sistema de MO actual[1].

En lo referido a los elementos sintácticos tratados, la negación y la intensificación son los dos fenómenos lingüísticos más influyentes en un entorno de minería de opiniones, pero existen otras construcciones que también pueden variar la orientación semántica de un texto, como pueden ser los verbos condicionales o los modales. Una ampliación futura del sistema podría incorporar estas y otras construcciones lingüísticas significativas, mediante nuevas funciones de visita. La consideración de más conjunciones adversativas también podría ser un objetivo en el futuro, para tratar nexos que son representados de otras formas en el árbol de dependencias.

Extraer información sobre los sintagmas involucrados en la orientación semántica también sería una funcionalidad de gran utilidad. Así, el sistema sería capaz de extraer las críticas concretas que se hagan sobre las características de productos o servicios, lo que puede de interés para las empresas a la hora de identificar los aspectos que deben mejorar, y en qué sentido, y cuáles deben mantener.

En lo referido al conocimiento del sistema, disponer de unos recursos semánticos lo más completos posibles es fundamental para conseguir un buen análisis del sentimiento. El SODictionariesV1.11Spa podría ampliarse para que en vez de almacenar la orientación semántica genérica de un término, almacenase una lista de orientaciones semánticas para las palabras según el dominio en el que aparezcan. Uno de los problemas del análisis del sentimiento está en que la polaridad de las palabras varía según su contexto. Esta situación quedó especialmente reflejada en la evaluación del sistema, cuando en dominios de entretenimiento, como películas, libros o música el rendimiento era menor que la media del sistema, debido en parte a las polaridades inadecuadas de ciertas palabras. Así pues, tratar diferentes OS para determinados, términos según su ámbito, podría permitir al sistema tener un conocimiento más específico y extraer la polaridad de una forma más precisa. La inclusión de términos propios en los diccionarios también podría ser objeto de interés. El nombre de *Tom Hanks* en un documento acerca de una película, el de los *Beatles* en uno musical o el de *Steve Jobs* o *Bill Gates* en un ámbito de tecnología podrían tener de por sí algún tipo de connotación subjetiva [1], y por tanto su sentimiento debería ser extraído para realizar un análisis de la OS más completo.

[1]Gran parte del tiempo de análisis de la OS de un texto en el sistema actual lo consume la etapa del análisis sintáctico.

Apéndices

Apéndice A

Manual de usuario

Aunque el objeto del trabajo de este proyecto es introducir una nueva propuesta para la MO de textos en castellano, basada en en análisis de dependencias; y no generar una aplicación a nivel de usuario, en este apéndice se ilustra la instalación del sistema MO como un paquete Python, así como los requisitos necesarios para que éste pueda funcionar.

A.1. Requisitos del sistema

- Debe estar instalada una versión de Python superior a la 2.6, así como una versión del NLTK superior a la 2.01. También debe descargarse el directorio de datos de esta herramienta, *nltk_data*, para disponer de los recursos de los que hace uso nuestro sistema.
- Es necesario tener instalado un entorno de ejecución Java versión 1.6 o superior de las API's de *Java 2 Standard Edition* (J2SE), para se pueda emplear el analizador sintáctico. A este respecto, se debe tener instalada la versión de MaltParser 1.7.
- Debe disponerse de la herramienta de instalación de programas Python, *setuptools*[1]. Concretamente debe instalarse el soporte para la versión de Python 2.6.

A.2. Instalación de la aplicación

En esta sección se enumeran los pasos a seguir para instalar el paquete en una máquina Unix, de forma que pueda ser utilizado por cualquier programa Python.

1. Descomprimir el paquete *miope-1.0.0.tar.gz*.

[1] http://pypi.python.org/pypi/setuptools

2. Dentro de la carpeta *miope* se encuentra un fichero *config.conf* con la ubicación de todos los recursos que utiliza el sistema. Sólo debe modificarse el parámetro *path_maltparser* para adaptarlo a la ruta en la que cada usuario tenga instalada esta herramienta.

3. Situarse en el directorio del archivo *setup.py*, abrir un terminal y ejecutar el siguiente comando con permisos de administrador: *sudo python setup.py install*

4. Para comprobar que la instalación se ha ejecutado correctamente, es recomendable abrir un términal Python e intentar importar el módulo miope con el comando *import miope*. Si el paquete se ha instalado correctamente no debería haber ningún problema.

A.2.1. Ejecución de un texto de prueba

A continuación se ilustra brevemente como analizar un texto de opinión con el analizador. Por claridad, se ejemplifica desde un terminal Python y se parte de un texto muy simple: *"No obstante,no estoy muy contento"*. Este ejemplo asume que el enunciado de prueba se sitúa en */home/ejemplo.txt* y que está codificado en *utf-8*:

1. Abre un terminal Python.
   $ python

2. Importa el analizador semántico del sistema.
   *>>>from miope.sentimentanalyzer.SentimentAnalyzer import **

3. Crea una instancia del analizador de la orientación semántica.
   >>>s = SentimentAnalyzer()

4. ... y analiza el texto deseado.
   >>>os = s.analyze("/home/ejemplo.txt")
   Durante la ejecución se generan dos ficheros temporales dentro del directorio */tmp*:

   - *input.conll*: El contenido de */home/ejemplo.txt* en formato CoNLL 2006 antes de analizarse sintácticamente, esto es, sin completar las columnas HEAD y DEPREL.
   - *output.conll*: El fichero *input.conll* una vez analizado sintácticamente y ya con datos en las columnas HEAD y DEPREL. Este fichero es de donde parte el analizador de la orientación semántica para extraer el sentimiento del enunciado.

5. Con todos los aspectos léxicos y sintácticos habilitados el resultado sería[2]...

   >>>os

   0.5

[2]El resultado de la evaluación de un texto dará un valor positivo si el enunciado representa una opinión favorable y negativo en caso contrario. Valores próximos a cero indican textos con polaridad neutra o mixta.

Apéndice B

Conjuntos de etiquetas morfológicas y sintácticas

En este anexo se ilustra la notación morfológica y sintáctica que emplea el sistema de MO para realizar las tareas de análisis léxico y sintáctico de dependencias. Todas las etiquetas pertenecen al Ancora Corpus Dependencies [14], por lo que recuperamos contenidos extraídos de su propia documentación. Cuando esto ocurra, es explicitado claramente y se indica el recurso web desde donde es posible descargarlo.

B.1. Etiquetas morfológicas

En relación a la etiquetación se realizaron pruebas con dos conjuntos de etiquetas. Los dos subapartados bajo estas líneas detallan el contenido de estas dos colecciones.

B.1.1. Conjunto de grano grueso

Se trata de una colección de etiquetas generada a partir del campo CPOSTAG de los documentos del *corpus* de Ancora. En total son doce etiquetas que se corresponden con doce categorías gramaticales, cuya representación se indica en el cuadro B.1.

Categoría gramatical	**Codificación**
Adjetivo	a
Adverbio	r
Conjunción	c
Determinante	d
Fecha	w
Interjección	i
Nombre	n
Número	z
Preposición	s
Pronombre	p
Signo de puntuación	f
Verbo	v

Cuadro B.1: Categorías gramaticales presentes en el Ancora Corpus Dependencies

B.1.2. Conjunto de grano fino

Colección de etiquetas formadas por los CPOSTAG y FEATS. Su contenido completo se puede encontrar en:

- clic.ub.edu/corpus/webfm_send/18[1]
- http://clic.ub.edu/corpus/es/documentacion en el enlace de *Morfología*.

Se incluye a continuación el contenido del conjunto de etiquetas, extraído íntegramente de la documentación.

[1]Visitado por última vez en agosto de 2012.

NOTE ON MORPHOLOGIC ANNOTATION:

There are two common attributes to every node: **<wd>** (word) and **<lem>** (lemma), whose value depends on the lexical item they accompany, corresponding to its form (**<wd>**) and lemma (**<lem>**).
The following nodes are the only possible terminal nodes according to the AnCora notation guidelines.

Node / ***description***	**Attributes** / ***description***	**Attribute values** / ***description***	**Secondary attributes** / ***description***	**Secondary attribute values** / ***description***
<a> / *adjective*	**<gen>** / *gender*	**"c"** / *common* **"f"** / *feminine* **"m"** / *masculine*		
	<num> / *number*	**"c"** / *common* **"p"** / *plural* **"s"** / *singular*		
	<posfunction>	**"participle"**		
	<postype> / *PoS subclassification*	**"qualificative"** **"ordinal"**		
<c> / *conjunction*	**<postype>** / *PoS subclassification*	**"coordinating"** **"subordinating"**		
<d> / *determiner*	**<gen>** / *gender*	**"c"** / *common* **"f"** / *feminine* **"m"** / *masculine*		
	<num> / *number*	**"c"** / *common* **"p"** / *plural* **"s"** / *singular*		
	<postype> / *PoS subclassification*	**"article"** **"demonstrative"** **"exclamative"** **"indefinite"**	**<possessornum>** / *possessor's number (only for "possessive")*	**"c"** / *common* **"p"** / *plural* **"s"** / *singular*

		“interrogative” **“numeral”** **“ordinal”** **“possessive”**			
<f> / *punctuation*	**<punct>** / *punctuation mark type*	**“apostrophe”** / *[’]* **“bracket”** / *[(], [)]* **“sqbracket”** / *[[], []]* **“cubracket”** / *[{], [}]* **“colon”** / *[:]* **“comma”** / *[,]* **“etc”** / *[...] meaning etcetera* **“exclamationmark”** / *[¡], [!]* **“hyphen”** / *[-]* **“mathsign”** / *a sign used in mathematic formulae* **“period”** / *[.]* **“questionmark”** / *[¿], [?]* **“quotation”** / *[’], [’’], [“], [””]* **“semicolon”** / *[;]* **“slash”** / *[/]* **“revslash”** / *[\]*	**<punctenclose>** / *opens or closes the punctuation mark (only for “bracket”, “sqbracket”, “cubracket”, “exclamationmark”, “questionmark”, “quotation”)*	**“open”** **“close”**
<i> / *interjection*					
<n> / *noun*	**<gen>** / *gender*	**“c”** / *common* **“f”** / *feminine* **“m”** / *masculine*			
	<num> / *number*	**“c”** / *common* **“p”** / *plural* **“s”** / *singular*			
	<postype> / *PoS subclassification*	**“common”** **“proper”**			

<p> / *pronoun*	**<gen>** / *gender*	**"c"** / *common* **"f"** / *feminine* **"m"** / *masculine*			
	<num> / *number*	**"c"** / *common* **"p"** / *plural* **"s"** / *singular*			
	<postype> / *PoS subclassification*	**"demonstrative"** **"exclamative"** **"indefinite"** **"interrogative"** **"numeral"** **"personal"** **"possessive"** **"relative"**	**<case>** / *case (only for "personal")*	**"accusative"** **"dative"** **"nominative"** **"oblique"**	
			<person> / *person (only for "personal", "possessive")*	**"1"** / *first person* **"2"** / *second* **"3"** / *third*	
			<polite> / *polite form (only for "personal", "possessive")*	**"yes"** **"no"**	
			<possessornum> / *possessor's number (only for "possessive")*	**"c"** / *common* **"p"** / *plural* **"s"** / *singular*	
<r> / *adverb*	**<postype>** / *PoS subclassification*	**"general"** **"negative"**			
<s> / *preposition*	**<gen>** / *gender*	**"c"** / *common* **"m"** / *masculine*			
	<num> / *number*	**"c"** / *common* **"p"** / *plural* **"s"** / *singular*			
	<postype> / *PoS subclassification*	**"preposition"**			
<v> / *verb*	**<mood>** / *mode*	**"gerund"** **"imperative"**			

		“indicative” **“infinitive”** **“pastparticiple”** **“subjuntive”**				
	<num> / *number*	**“c”** / *common* **“f”** / *feminine* **“m”** / *masculine*				
	<person> / *person*	**“1”** / *first person* **“2”** / *second* **“3”** / *third*				
	<postype> / *PoS subclassification*	**“auxiliary”** **“main”** **“semiauxiliary”**				
	<tense> / *tense*	**“conditional”** **“future”** **“imperfect”** **“past”** **“present”**				
<w> / *date*						
<z> / *number*	**<postype>** / *PoS subclassification*	**“currency”** **“percentage”**				

B.2. Dependencias sintácticas

El conjunto de tipo de dependencias soportados por el *corpus* Ancora se incluye tras estas líneas. Está directamente extraído de la documentación web, a la que se puede acceder a través de:

- http://clic.ub.edu/corpus/es/documentacion en en enlace *Sintaxis-Semántica(funciones y Papeles Temáticos)- Versión 1.0*
- http://clic.ub.edu/corpus/webfm_send/22[2]

[2]Visitado por última vez en agosto de 2012.

GLOSSARY OF TERMS:
arg0 applies to the argumental complement closest to the verb, semantically (typically, the subject).
arg4 applies to the argumental complement farthest from the verb, semantically (typically, destination complements).
arg1, **arg2** and **arg3** are a gradation between those two edges: the higher the number, the further the complement, semantically.
argL applies to lexicalized complements.
argM applies to non-argumental complements.

NOTE ON ANNOTATION:
Any of the nodes detailed below may have an attribute **<repetition>**, whose values are either **"yes"** or **"no"** (default), to indicate that it appears twice in the sentence (via pronominalization, usually). If the value that applies is the default one (**"no"**), the attribute will not appear explicitly in the annotation.
void in a particular column means that attribute does not apply to the function.

Attribute <func> values *(function)*	**Attribute: <arg> values** *(argument)*	**Attribute: <tem> values** *(thematic role)*	**Secondary <attribute> and "values"**	**Possible head nodes**
"suj" / *subject*	**"arg0"**	**"agt"** / *agent* **"cau"** / *cause* **"exp"** / *experiencer* **"src"** / *source*		**<S>** / *clause* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase* **<relatiu>** / *relative*
	"arg1"	**"pat"** / *patient* **"tem"** / *theme*		
	"arg2"	**"loc"** / *locative* **"ins"** / *instrument*		
	"argL"	***void***		
"cd" / *direct object*	**"arg1"**	**"pat"** / *patient* **"tem"** / *theme*		**<S>** / *clause* **<sa>** / *adjective phrase* **<sadv>** / *adverbial phrase* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase* **<relatiu>** / *relative*
	"arg2"	**"atr"** / *attribute* **"ext"** / *extension*	**<functype>** / *function subclassification* **"quantitative"** / *only for "ext"*	

	"argL"	***void***		**<morfema.pronominal>** / *pronominal morpheme*
"ci" / *indirect object*	**"arg2"**	**"ben"** / *beneficiary* **"exp"** / *experiencer*		**<sp>** / *prepositional phrase* **<sn>** / *noun phrase* **<relatiu>** / *relative*
	"arg3"	**"ben"** / *beneficiary* **"exp"** / *experiencer*		**<morfema.pronominal>** / *pronominal morpheme*
"cag" / *agent*	**"arg0"**	**"agt"** / *agent*		**<sp>** / *prepositional phrase*
"atr" / *attribute*	**"arg2"**	**"atr"** / *attribute*		**<S>** / *clause* **<sa>** / *adjective phrase* **<sadv>** / *adverbial phrase* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase* **<relatiu>** / *relative*
"creg" / *prepositional object*	**"arg1"**	***void***		**<S>** / *clause*
	"arg2"	***void*** **"atr"** / *attribute* **"efi"** / *final state* **"ext"** / *extension* **"ins"** / *instrument* **"loc"** / *locative*		**<sa>** / *adjective phrase* **<sadv>** / *adverbial phrase* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase* **<relatiu>** / *relative*
	"arg3"	***void*** **"fin"** / *finality* **"ori"** / *origin*		
	"arg4"	**"des"** / *destination* **"efi"** / *final state*		
	"argL"	***void***		
"cpred" / *predicative*	**"arg2"**	**"atr"** / *attribute*	**<predicate>** / *modified*	**<S>** / *clause*

complement	**"arg3"**	**"atr"** / *attribute*	*complement*	**<sa>** / *adjective phrase*
	"argM"	**"atr"** / *attribute*	**"cd"** / *direct object*	**<sadv>** / *adverbial phrase*
	"argL"	***void***	**"creg"** / *prepositional object*	**<sn>** / *noun phrase*
			"suj" / *subject*	**<sp>** / *prepositional phrase*
"cc" / *adjunct*	**"arg1"**	***void***	**<functype>** / *function*	**<S>** / *clause*
		"loc" / *locative*	*subclassification*	**<sa>** / *adjective phrase*
		"tmp" / *temporal*	**"locative"**	**<sadv>** / *adverbial phrase*
	"arg2"	***void***	**"temporal"**	**<sn>** / *noun phrase*
		"atr" / *attribute*		**<sp>** / *prepositional phrase*
		"efi" / *final state*		**<relatiu>** / *relative*
		"ext" / *extension*		
		"fin" / *finality*		
		"ins" / *instrument*		
		"loc" / *locative*		
	"arg3"	**"ein"** / *initial state*		
		"fin" / *finality*		
		"ins" / *instrument*		
		"loc" / *locative*		
		"ori" / *origin*		
	"arg4"	**"des"** / *destination*		
		"efi" / *final state*		
	"argM"	**"adv"** / *non-specific adjunct*		
		"atr" / *attribute*		
		"cau" / *cause*		
		"ext" / *extension*		
		"fin" / *finality*		
		"loc" / *locative*		
		"mnr" / *manner*		

		"tmp" / *temporal*			
	"argL"	***void***			
"ao" / *sentence adjunct*	***void***	***void***			**<S>** / *clause* **<sadv>** / *adverbial phrase* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase*
"et" / *textual element*	***void***	***void***			**<S>** / *clause* **<sadv>** / *adverbial phrase* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase* **<relatiu>** / *relative* **<coord>** / *coordination*
"mod" / *verbal modifier*	***void***	***void***			**<sadv>** / *adverbial phrase* **<sn>** / *noun phrase* **<sp>** / *prepositional phrase* **<neg>** / *negation*
"impers" / *impersonality mark*	***void***	***void***			**<morfema.verbal>** / *verbal morpheme*
"pass" / *passive mark*	***void***	***void***			**<morfema.verbal>** / *verbal morpheme*

Glosario

AA Aprendizaje Automático.

LAS Labeled Attachment Score.

LAS2 Label Accuracy Score.

MO Minería de Opiniones.

NLTK Natural Language Toolkit.

OO Orientación a objetos.

OS Orientación Semántica.

PLN Procesamiento del Lenguaje Natural.

UAS Unlabeled Attachment Score.

Índice alfabético

Bibliografía

[1] Bo Pang, Lillian Lee, *Opinion Mining and Sentiment Analysis*. Hanover, MA, USA: now Publishers Inc., 2008.

[2] Brooke, J., Tofiloski, M., and Taboada, M., "Cross-linguistic sentiment analysis: From english to spanish," in *Proceedings of the International Conference RANLP-2009*, (Borovets, Bulgaria), pp. 50–54, Association for Computational Linguistics, September 2009.

[3] Taboada, M., Brooke, J., Tofiloski, M., Voll, K., and Stede, M., "Lexicon-based methods for sentiment analysis," *Computational Linguistics*, vol. 37, no. 2, pp. 267–307, 2011.

[4] Ding, X., Liu, B., and Yu, P. S., "A holistic lexicon-based approach to opinion mining," in *Proceedings of the international conference on Web search and web data mining*, WSDM'08, (New York, NY, USA), pp. 231–240, ACM, 2008.

[5] Brants, T., "TnT: a statistical part-of-speech tagger," in *Proceedings of the sixth conference on Applied natural language processing*, ANLC'00, (Stroudsburg, PA, USA), pp. 224–231, Association for Computational Linguistics, 2000.

[6] Mohseni, M., Motalebi, H., Minaei-bidgoli, B., and Shokrollahi-far, M., "A Farsi part-of-speech tagger based on Markov model," in *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, (New York, NY, USA), pp. 1588–1589, ACM, 2008.

[7] Brill, E., "A simple rule-based part of speech tagger," in *Proceedings of the workshop on Speech and Natural Language*, HLT'91, (Stroudsburg, PA, USA), pp. 112–116, Association for Computational Linguistics, 1992.

[8] Sandra Kübler, Ryan McDonald, Joakim Nivre, *Dependency Parsing*. Morgan & Clay-Pool Publishers, 2009.

[9] Buchholz, S. and Marsi, E., "CoNLL-X shared task on multilingual dependency parsing," in *Proceedings of the Tenth Conference on Computational Natural Language Learning*,

CoNLL-X'06, (Stroudsburg, PA, USA), pp. 149–164, Association for Computational Linguistics, 2006.

[10] Perkins, J., *Python Text Processing with NLTK 2.0 Cookbook*. Packt Publishing, 2010.

[11] Steven Bird, Ewan Klein, Edward Loper, *Natural Language Processing with Python*. Sebastopol, CA, USA: O'REILLY, 2009.

[12] Mark Lutz, *Learning Python*. Sebastopol, CA, USA: O'REILLY, 2009.

[13] Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E., "Maltparser: A language-independent system for data-driven dependency parsing.," *Natural Language Engineering*, vol. 13, no. 2, pp. 95–135, 2007.

[14] Taulé, M., Martí, M. A., and Recasens, M., "Ancora: Multilevel annotated corpora for catalan and spanish," in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)* (Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odjik, J., Piperidis, S., and Tapias, D., eds.), (Marrakech, Morocco), European Language Resources Association (ELRA), may 2008.

[15] Wiegand, M., Balahur, A., Roth, B., Klakow, D., and Montoyo, A., "A survey on the role of negation in sentiment analysis," in *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*, NeSp-NLP'10, (Stroudsburg, PA, USA), pp. 60–68, Association for Computational Linguistics, 2010.

[16] Jia, L., Yu, C., and Meng, W., "The effect of negation on sentiment analysis and retrieval effectiveness," in *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM'09, (New York, NY, USA), pp. 1827–1830, ACM, 2009.

[17] Cruz Mata, F. L., *Extracción de opiniones sobre características: Un enfoque práctico adaptado al dominio*. PhD thesis, Universidad de Sevilla, 2011.

[18] Gómez Rodríguez, C. and Nivre, J., "A transition-based parser for 2-planar dependency structures," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL'10, (Stroudsburg, PA, USA), pp. 1492–1501, Association for Computational Linguistics, 2010.