POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

Máster Universitario en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

TRABAJO FIN DE MÁSTER

Anotadores sobre UIMA para procesado de historia clínica digital

Autor: Ángel Soler Ruiz

Directora: Ernestina Menasalvas

MADRID, JUNIO 2018

i. Resumen

El presente Trabajo Fin de Máster versará sobre el problema de la detección de términos negados en documentos.

Viendo el panorama global, en el que cada vez más se exigen resultados más precisos y en el menor tiempo posible, el uso de sistemas de información es actualmente imprescindible en prácticamente cualquier ámbito o disciplina, tanto para acceder de forma rápida a información actualizada, como para compartir información entre profesionales y obtener resultados de calidad.

La detección de la negación en oraciones es importante a la hora de desarrollar un software de extracción de información sobre documentos. Es cierto que para detectar la negación en un único documento solo haría falta una persona que lo leyera y anotase las palabras que se encuentran bajo el ámbito de una negación. Pero cuando el número de documentos a analizar crece considerablemente es preferible trasladar el problema a una solución software.

En este trabajo se utilizará el framework UIMA (Unstructured Information Management Application), para implementar una aplicación que anote en los textos las negaciones, basadas en el algoritmo NegEx y así identificar términos específicos negados. Se pretende con ello facilitar la labor de detección de la negación en el contexto de aplicaciones basadas en minería de textos.

Conceptos clave: anotador, negación, información no estructurada, UIMA, Analysis Engine, NegEx.

ii. Summary

The present Master Thesis will deal with the detection of denial of terms in general documents.

Seeing the global panorama in which more and more precise results are required and in the shortest period of time, the use of information systems is currently essential in any field or discipline, both to quickly access updated information and to share information among professionals to obtain quality results.

The detection of denial in sentences is a problem to take into account when developing a knowledge extraction software over documents. It is true that for the detection of the negation in a single document we would only take a single person to read it and he could write down those words under a negation scope. But when the number of documents to analyse increases considerably it is preferable to transfer the problem into a software solution.

In this thesis, the UIMA (Unstructured Information Management Application) framework will be used to implement an application that records the negations based on the NegEx algorithm in the medical texts and thus detect the denied terms. It is intended to facilitate the work of detection in the context of medical applications based on text mining.

Key words: annotator, family health history, negation, unstructured information, UIMA, Analysis Engine, NegEx.

Tabla de contenidos

i. Resumen III

ii. Summary IV

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS 1

1.1. Planteamiento del problema 1

1.2. Objetivos del proyecto 1

CAPÍTULO 2. ESTADO DEL ARTE 2

2.1. Algoritmo NegEx 2

2.2. Algoritmo NegEx para textos en español 6

2.2.1. Método de NegEx en español 6

2.3. UIMA 9

2.3.1. Conceptos clave de UIMA 10

2.3.2. Funcionamiento básico de UIMA 13

2.4. Anotadores UIMA Remotos 14

2.4.1. Despliegue de un componente UIMA 16

2.4.2. Llamada a un servicio UIMA 17

CAPÍTULO 3. ARQUITECTURA DE LA APLICACIÓN 18

3. 18

3.1. Visión general de la aplicación 18

3.2. Especificación de Requisitos Software (ERS) 20

3.2.1. Introducción 20

3.2.1.1. Propósito 20

3.2.1.2. Ámbito del sistema 20

3.2.1.3. Definiciones, acrónimos y abreviaturas 21

3.2.1.4. Referencias 21

3.2.1.5. Visión general del documento 21

3.2.2. Descripción General 22

3.2.2.1. Perspectiva del Producto ... 22
3.2.2.2. Funciones del Producto ... 22
3.2.2.3. Características de los Usuarios ... 23
3.2.2.4. Restricciones ... 23
3.2.2.5. Suposiciones y Dependencias ... 23
3.2.2.6. Requisitos futuros ... 23
3.2.3. Requisitos Específicos ... 24
3.2.3.1. Interfaces Externas ... 24
3.2.3.1.1. Interfaces de Usuario ... 24
3.2.3.1.2. Interfaces Hardware ... 24
3.2.3.1.3. Interfaces Software ... 24
3.2.3.1.4. Interfaces de Comunicación ... 25
3.2.3.2. Requisitos Funcionales ... 25
3.2.3.3. Requisitos de Rendimiento ... 25
3.2.3.4. Restricciones de Diseño ... 26
3.3. Solución ... 26
3.4. Proceso de construcción ... 28
3.4.1. Definición de los tipos (TypeSystem) ... 28
3.4.2. Generar las clases Java correspondientes con esos tipos ... 29
3.4.3. Preparación del diccionario de palabras ... 29
3.4.3.1. Escribir el diccionario de palabras ... 29
3.4.4. Crear el descriptor del Analysis Engine ... 31
3.4.4.1. Añadir diccionario de predicates negatives ... 32
3.4.5. Incluir TypeSystem remoto de Tokenization ... 34
3.4.6. Escribir el código del anotador ... 34
3.4.6.1. Paso 1: recuperación de las anotaciones previas ... 34
3.4.6.2. Paso 2: obtención del texto ... 35
3.4.6.3. Paso 3: averiguar si existe una “persistent negative” ... 35
3.4.6.4. Paso 4: crear anotaciones parciales ... 37

3.4.6.5. Paso 5: identificar al término negado 38
3.4.6.6. Paso 6: crear anotaciones finales 40
CAPÍTULO 4. EJECUCIÓN DE PRUEBAS Y RESULTADOS 41
4.1. Ejecución de pruebas 41
4.1.1. Prueba 1. Prueba básica correcta 41
4.1.2. Prueba 2. Prueba básica parcialmente correcta 42
4.1.3. Prueba 3. Prueba básica parcialmente correcta 43
4.1.4. Prueba 4. Prueba avanzada 44
4.1.5. Prueba 5. Prueba básica 46
4.1.6. Prueba 6. Prueba básica 47
4.1.7. Prueba 7. Prueba compleja 48
4.1.8. Prueba 8. Prueba básica 49
4.1.9. Prueba 9. Prueba básica 50
CAPÍTULO 5. CONCLUSIONES GENERALES Y LÍNEAS FUTURAS . 52
5.1. **Conclusiones generales** 52
5.2. Líneas futuras 53
Bibliografía 54

Tabla de Ilustraciones

Ilustración 1. Rendimiento del algoritmo básico 4
Ilustración 2. Rendimiento de NegEx 4
Ilustración 3. UIMA... 9
Ilustración 4.Ejemplo de red con servidor Vinci de anotadores remotos ... 15
Ilustración 5. Ejemplo de descriptor de despliegue de un anotador 16
Ilustración 6.Ejemplo de descriptor de llamada de un anotador remoto.... 17
Ilustración 7 Secuencia de ejecución de los anotadores 18
Ilustración 8. Visión completa de la aplicación... 19
Ilustración 9. Definición del TypeSystem de la negación............................. 28
Ilustración 10. Clases Java de negación generadas 29
Ilustración 11. Método "initialize" del anotador de la negación................. 30
Ilustración 12. Pestaña TypeSystem del descriptor de la negación............. 31
Ilustración 13. Información de la interfaz SharedResourceObject 32
Ilustración 14. Pestaña "Resources" mostrando la inclusión del diccionario 33
Ilustración 15. Recuperación de las anotaciones de tipo Sentence.............. 35
Ilustración 16. Recuperación del texto cubierto por la anotación Sentence 35
Ilustración 17. Método para obtener la lista de índices de la palabra en la oración 36
Ilustración 18. Recuperación de las anotaciones entre dos índices 38
Ilustración 19. Búsqueda de los tokens negados 39
Ilustración 20. Creación anotaciones finales 40
Ilustración 21 Prueba básica correcta 41
Ilustración 22. Prueba básica parcialmente correcta 42
Ilustración 23. Prueba básica parcialmente correcta 43
Ilustración 24. Prueba avanzada 44
Ilustración 25. Prueba básica.. 46
Ilustración 26. Prueba básica.. 47
Ilustración 27. Prueba compleja... 48
Ilustración 28. Prueba básica.. 49
Ilustración 29. Prueba básica.. 50

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS

1.1. Planteamiento del problema

El problema que se plantea consiste en averiguar si un término específico de un documento se encuentra bajo la influencia de una negación o no.

Los textos que analizar son información no estructurada, esto es, no tienen una estructura interna identificable como podría ser un modelo relacional de tipo Entidad/ Relación en el que la información se almacena en tablas estructuradas con un tipo primitivo identificable. Es un conglomerado masivo y desorganizado de varios objetos que no tienen valor hasta que se identifican y se almacenan de manera organizada. Una vez que se organizan, los elementos que conforman su contenido pueden ser buscados y categorizados para obtener información. Un ejemplo podría ser el análisis de correos electrónicos que contienen datos que pueden ser importantes para una organización.

1.2. Objetivos del proyecto

A continuación, se listan los objetivos del proyecto:

- **Identificación de oraciones negadas y el ámbito de negación a partir de los términos negados**. Una vez identificado el término que indica negación, se tratará de averiguar el ámbito de la negación. Para ello se tendrán que identificar los signos de puntuación, principalmente tres: el punto, el punto y coma y la coma.

- **Desarrollo de anotadores sobre términos y oraciones negadas**. Cuando se haya identificado una oración y ésta sea una oración negada, entonces podremos añadir un anotador de negación sobre esa oración. A lo largo del documento se aclarará el término anotador dentro del framework UIMA.

CAPÍTULO 2. ESTADO DEL ARTE

2.1. Algoritmo NegEx

En el ámbito clínico, la mayoría de la información contenida en los registros médicos de los pacientes está en forma de narración y por tanto no está directamente disponible (en forma de tabla).

Desde hace años, investigadores del área que se centra en la extracción de información, llevan creando métodos cada vez más eficientes para automáticamente indexar documentos clínicos narrativos y facilitar así la búsqueda de términos relevantes. Sin embargo, las técnicas utilizadas para la recuperación de información generalmente no discriminan entre términos que están presentes, es decir, que son un hecho, y términos que están negados.

En los informes médicos la presencia de un término no indica necesariamente la presencia de la condición clínica representada por ese término. De hecho, la mayoría de los términos y enfermedades más frecuentes presentes en documentos referentes a informes radiológicos, históricos o exámenes físicos están negados. Normalmente los facultativos, respecto a la negación, anotan que una enfermedad en particular puede ser descartada o que un hallazgo que está relacionado con una enfermedad sospechosa se encuentra ausente.

Mientras que la negación en predicados lógicos está bien definida y sintácticamente es simple, la negación del lenguaje natural es compleja y lleva siendo estudiada desde hace siglos. Identificar una oración negada asociada a una condición clínica, supone identificar una proposición asignada a esa condición clínica de una persona y determinar si está negada en el texto.

NegEx es un algoritmo desarrollado por los investigadores Wendy W. Chapman, Will Bridewell, Paul Hanbury, Gregory F. Cooper y Bruce G. Buchanan [1] del *Center for Biomedical Informatics* de la universidad de Pittsburg en Pennsylvania (EE UU).

El algoritmo recibe como entrada una oración con un conjunto de elementos hallados en ella y un conjunto de enfermedades. La salida es un booleano indicando si la porción de oración (o palabra) que se le pasa como argumento está negada o no.

Previamente el algoritmo debe identificar términos de un diccionario médico (en este caso el diccionario médico UMLS (Unified Medical Language System)). Si el algoritmo no encuentra un término UMLS entonces no se puede encontrar una oración negada.

UMLS es un conjunto de software que ofrece varios vocabularios médicos y biomédicos bajo un grupo de estándares para hacer posible la interoperabilidad entre sistemas computacionales.

Uno de los usos de UMLS es la asociación de información médica, términos médicos, nombres de fármacos, y una serie de códigos entre diferentes sistemas. Un posible ejemplo sería la asociación de términos y códigos entre un doctor, un farmacéutico y la compañía aseguradora del paciente.

Los investigadores se basaron en un algoritmo previo para poder construir NegEx. En su primera versión (2001) identificaron treinta y cinco oraciones que indicaban cierto tipo de negación y fueron divididas en dos grupos: pseudo-negativas y frases que indican cierta negación de enfermedades cuando se utilizan siguiendo uno de los dos patrones o expresiones regulares siguientes:

Patrón II-A: (oración negada) * (término UMLS)

Patrón II-B: (término UMLS) * (oración negada)

En ambos casos el * indica hasta un máximo de cinco elementos de separación que pueden encontrarse entre la oración negada y el término UMLS.

Las siguientes tablas son los resultados obtenidos después de ejecutar el algoritmo base y NegEx en los grupos uno y dos. Se tratan de imágenes extraídas del artículo [1] de Wendy Chapman concretamente de la sección tres "Results".

TABLE 2
Performance of the Baseline Algorithm

	Group 1 sentences (i.e., containing NegEx negation phrases) (n = 500) (%)	Group 2 sentences (i.e., not containing NegEx negation phrases) (n = 500) (%)	All sentences (n = 1000) (%)
Sensitivity	88.27	0.00	88.27
Specificity	52.69	100.00	85.27
PPV	68.42	—	68.42
NPV	79.46	96.99	93.01

Note. Sensitivity = Number of terms NegEx correctly negated/number of terms negated by rater; Specificity = Number of terms NegEx correctly did not negate/number of terms not negated by rater; PPV = Number of terms NegEx correctly negated/number of terms NegEx negated; NPV = Number of terms NegEx correctly did not negate/number of terms NegEx did not negate. Sensitivity is 0% in group 2 because group 2 did not contain sentences with negation phrases used by NegEx.

Ilustración 1. Rendimiento del algoritmo básico

TABLE 3
Performance of NegEx

	Group 1 sentences (i.e., containing NegEx negation phrases) (n = 500) (%)	Group 2 sentences (i.e., not containing NegEx negation phrases) (n = 500) (%)	All sentences (n = 1000) (%)
Sensitivity	82.41	0.00	77.84
Specificity	82.50	100.00	94.51
PPV	84.49	—	84.49
NPV	80.21	96.99	91.73

Ilustración 2. Rendimiento de NegEx

Se pueden observar dos características clave que diferencian a ambos algoritmos. El algoritmo base proporciona un poco más de sensibilidad respecto a NegEx. Como se explica en la **Ilustración 1. Rendimiento del algoritmo básico**, la sensibilidad es el cociente entre el número de términos que NegEx niega correctamente y el número de términos que se negaron manualmente, es decir la persona que en una primera fase se dedicó a encontrar en los mismos textos pasados a NegEx términos negados.

Sin embargo, NegEx proporciona mucha más especificidad, esto es el número de términos que NegEx no negó correctamente (lo que se denomina como “true negative”) entre el número de términos que la persona encargada de realizar manualmente la negación, no negó.

Los resultados que se obtuvieron una vez realizadas las pruebas sobre NegEx concluyeron que un algoritmo simple basado en expresiones regulares puede detectar con bastante precisión una gran cantidad de lo que los autores han denominado como *“pertinent negatives”* en los textos clínicos. Identificando un conjunto de oraciones pseudo negadas, un conjunto de oraciones negadas, y dos expresiones regulares simples es todo lo que se necesita para identificar a la mayoría de las *“pertinent negatives”* en registros médicos. El conjunto de las oraciones negadas y de las expresiones regulares presentadas en el estudio realizado por los investigadores está abierto a cambios y nos da la libertad de incluir nuevas oraciones.

Algunas de las oraciones que se emplearon fueron las siguientes:

Término en inglés
can be ruled out can rule him out can rule out
No, not
No evidence
No new
No support for
No suspicion of

Tabla 1. Ejemplo de "pertinent negatives" utilizadas

2.2. Algoritmo NegEx para textos en español

El desarrollo de una aplicación orientada a la minería de textos tiene que considerar el idioma/lengua de los textos que va a analizar, pues las reglas gramaticales difieren entre lenguas.

Este apartado resume el método empleado en el estudio realizado [4] en el que se tomaron como conjunto de datos, datos no estructurados (textos) de registros médicos (EHR), los cuales no están estandarizados, para extraer información relevante con el objetivo de realizar nuevas investigaciones.

2.2.1. Método de NegEx en español

En este apartado resumo el método que emplearon los investigadores para adaptar el algoritmo NegEx a los textos en español.

El método llevado a cabo, consta de los siguientes cuatro puntos

- Anotación manual de textos clínicos.
- Etiquetado de los términos.
- Cálculo de la frecuencia
- Evaluación

En los siguientes cuatro apartados detallo lo esencial de cada punto.

Para poder detectar la negación, uno de los requisitos del algoritmo es tener un corpus de expresiones en español. Puesto que no existía en el momento de desarrollar el proyecto un "Gold Standard" lo suficientemente bueno, los investigadores tuvieron que crear uno sobre el cual ejecutar el algoritmo de detección de la negación.

El siguiente paso fue la traducción de los términos del Gold Standard escrito en inglés al castellano, a la par que se fue enriqueciendo con términos propios de la lengua castellana. Un ejemplo de estos términos es el siguiente:

Término en inglés	Término en castellano
can be ruled out can rule him out can rule out	Se puede descartar
No, not	No
No evidence	Sin evidencia No evidencia
No new	Sin novedad
No support for	No hay soporte para
No suspicion of	Ninguna sospecha de

Tabla 2. Tabla de traducción de términos

Una vez realizada la traducción de cada término, se le asignó una categoría indicando la certidumbre de negación que representaba. Las categorías son tres: i) Término Definitivamente Negado; ii) Término Probablemente Negado; iii) Término Pseudo Negado.

Una vez que todas las oraciones del corpus fueron categorizadas, se calculó la frecuencia de cada término del corpus. La media de la frecuencia de cada término también se calculó y los términos se pudieron clasificar utilizando tres categorías de frecuencia: a) sin aparición: aquellos términos con frecuencia cero; b) infrecuente: aquellos términos con frecuencia mayor que cero, pero menor que la media; c) frecuente: aquellos términos con frecuencia mayor que la media.

La solución en español se evaluó sobre quinientos informes. El corpus (dataset) estaba compuesto de aproximadamente un millón ciento sesenta y cuatro mil setecientas cuarenta y dos palabras de las cuales sesenta y cinco mil seiscientas cinco eran diferentes.

Los experimentos realizados revelaron los siguientes resultados.

	Precisión	Recall	F-Measure	Accuracy
Definite Negated Terms	49,47%	55,70%	52,38%	83,37%
Definite Existence	86,86%	95,2%	90,84%	84,78%

Una de las conclusiones que se observaron fueron que para aproximar más los resultados a los obtenidos por la versión en inglés de NegEx, era necesario, en el futuro, aumentar el número de términos que disparaban al algoritmo (triggering terms) pero añadiendo reglas gramaticales adaptadas al español.

2.3. UIMA

UIMA es el acrónimo para Unstructured Information Management. Es un conjunto de aplicaciones que son capaces de analizar un gran volumen de información no estructurada con el objetivo de extraer conocimiento que sea relevante para un usuario final. Un ejemplo sería una aplicación UIMA que consumiera textos en formato plano e identificase entidades como personas, lugares, organizaciones o relaciones como "localizado en".

UIMA es un framework que se divide en varios componentes como se muestra en la **Ilustración 3. UIMA** .

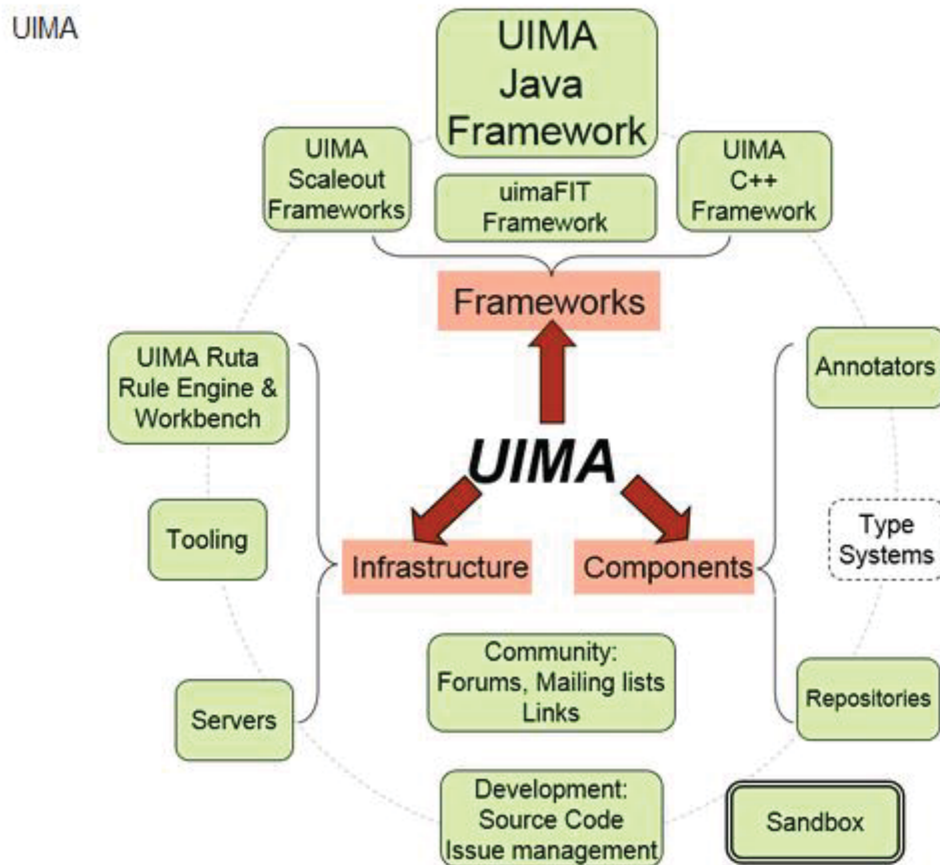


Ilustración 3. UIMA

La documentación del framework de UIMA es extensa a la vez que completa. Para este trabajo se ha revisado la documentación referente al tutorial básico que ofrece UIMA en su página web y algunas secciones de la documentación del framework. En esta memoria de TFM no se va a explicar al detalle el funcionamiento de UIMA, pero sí que se va a resumir en el siguiente apartado 2.3.1, los conceptos que el lector ha de entender previamente para poder localizarse en el global del proyecto.

2.3.1. Conceptos clave de UIMA

Como se ha explicado en el apartado anterior, este subapartado tiene la intención de ofrecer una lista de conceptos clave que son necesarios para comprender el framework UIMA y parte de este TFM. Todos ellos son importantes, pues todos ellos han intervenido en el desarrollo de este proyecto.

- **Analysis Engine (AE).** Es un programa que analiza elementos (ej: documentos, imágenes, sonidos) e infiere información sobre ellos.

- **Aggregate Analysis Engine.** Es un AE construido a partir de varios AEs.

- **Annotation.** Es la asociación de metadatos, como una etiqueta, a una región del texto (o de otro elemento).

- **Annotator.** Es un componente software que implementa la interfaz Annotator de UIMA. Los anotadores se implementan para producir y almacenar las anotaciones sobre regiones del elemento que está siendo analizado (ej: texto, audio o vídeo).

- **CAS.** Son las siglas de *Common Analysis Structure*. Es la estructura de datos principal que utilizan los componentes UIMA para representar y compartir los resultados de los análisis. Un CAS contiene:
    - Un elemento que analizar (en la nomenclatura UIMA se le conoce como “artifact”). Este es el objeto que está siendo analizado como un texto, audio o vídeo.

- Un descriptor de Type System. Este descriptor indica los tipos, subtipos y propiedades de un CAS.
- Metadatos del análisis. Son "pre anotaciones" que describen al "artifact" o a regiones específicas del "artifact".
- Un repositorio indexado para proporcionar un acceso eficiente y la posibilidad de iterar sobre los resultados del análisis.

- **JCas.** Es un objeto que hace de interfaz hacia los contenidos del CAS. Esta interfaz utiliza clases Java adicionales en las cuales cada tipo del CAS está representado como una clase Java con el mismo nombre, cada propiedad ("feature") está representada con métodos *getters y setters*, y cada instancia del tipo está representado por un objeto Java de la correspondiente clase Java.

- **Primitive Analysis Engine.** Es un *Analysis Engine* que está compuesto de un único *Annotator*, y que no tiene más que un solo AE.

- **Type.** Es una especificación de un objeto del CAS utilizado para almacenar los resultados de los análisis. Los tipos normalmente tienen asociadas "features", las cuales son atributos, o propiedades del tipo. Un tipo se puede equiparar a una clase en lenguajes de programación orientados a objetos o a una tabla en bases de datos.

- **Type System.** Es una colección de tipos relacionados. Todos los componentes que pueden acceder a un CAS, incluyendo *Aplicaciones, Analysis Engines,* Collection Readers, o CAS Consumers declaran el *Type System* que utilizan.

- **Unstructured Information.** El ejemplo por excelencia de información no estructurada es un documento de texto escrito en lenguaje natural. El significado del contenido del documento es implícito y la interpretación correcta por un software requiere cierto grado de análisis para explicar las semánticas del documento.

- **XML Metadata Interchange (XMI).** Es un estándar OMG para la representación de grafos en formato XML, los cuales utiliza UIMA para serializar los resultados de los análisis contenidos en el CAS en su representación XML.

2.3.2. Funcionamiento básico de UIMA

UIMA es una arquitectura en la cual los bloques básicos, llamados Analysis Engines (AEs), se construyen para analizar un documento e inferir y registrar atributos descriptivos sobre el documento en su totalidad, o en regiones de éste. Esta información descriptiva, producida por un AE, es conocida como **resultado del análisis.** Los resultados de los análisis típicamente representan metadatos sobre el contenido del documento. Una manera de describir mentalmente a un AE es equipararlo con un software que automáticamente descubre y registra metadatos sobre el contenido original.

Los **resultados de los análisis** pueden incluir sentencias que describan regiones del texto en lugar de referirse al documento en su totalidad. Se utiliza el término **span** para referirse a una secuencia de caracteres en un texto.

Un pequeño ejemplo podría ser un documento con el identificador D102 que contiene un *span*, "Fred Centers" y que empieza en la posición 101. Un AE que se haya programado para detectar nombres de personas en el texto podría representar la siguiente afirmación como un **resultado del análisis:**

The span from position 101 to 112 in document D102 denotes a Person

En la afirmación anterior hay un término predefinido o lo que se denomina en UIMA como *Type*. En la oración anterior el *Type* es *Person*. Los tipos de UIMA van a determinar las clases de resultados que un AE puede crear.

El framework UIMA trata a los AEs como objetos "*pluggables",* compuestos, descubiertos o administrados. En el corazón de los AEs se encuentran los **algoritmos de análisis** que realizan todo el trabajo de analizar los documentos y registrar los resultados.

UIMA proporciona un componente básico diseñado para albergar a los algoritmos de análisis que se ejecutan dentro de un AE. Las instancias de este componente se denominan **Annotator**. Una de las primeras tareas de un desarrollador va a ser implementar un anotador. UIMA proporciona los métodos adecuados para crear estos anotadores y los *Analysis Engines.*

Para resumir debe quedar claro que un AE encapsula a un anotador añadiéndole las APIs y la infraestructura necesaria para la composición y el despliegue de los anotadores dentro de framework UIMA. El AE más sencillo contiene exactamente un único anotador. Los AEs más complejos pueden estar compuestos por otros AEs.

2.4. Anotadores UIMA Remotos

Los anotadores remotos UIMA surgen para evitar tener que implementar localmente funcionalidad común. Cierto número de los anotadores, son anotadores que están siendo utilizados por otros anotadores y además de igual manera en todos ellos.

Con un anotador como “servicio” no solo evitamos tener que ejecutar el mismo anotador en todos y cada uno de los sistemas individuales, además favorecemos la “mantenibilidad” y actualización del mismo.

UIMA permite fácilmente tomar un Analysis Engine o un CAS Consumer y desplegarlo como servicio utilizando varios protocolos de red. Así ambos pueden ser llamados desde una máquina remota sin necesidad de implementar su funcionalidad de manera local.

El protocolo de red utilizado en este trabajo se denomina “**Vinci**”. Es una versión ligera del protocolo SOAP, y forma parte de Apache UIMA.

El principal uso de este servicio es la creación de un proxy por parte de un Analysis Engine local hacia el remoto y trabajar con el anotador remoto como si fuera local.

Para el correcto funcionamiento se necesita, por un lado, un **servidor** en el que desplegar el anotador remoto, y un **cliente** al cual indicarle de “manera especial” en dónde se está ejecutando ese anotador y cómo puede utilizarlo.

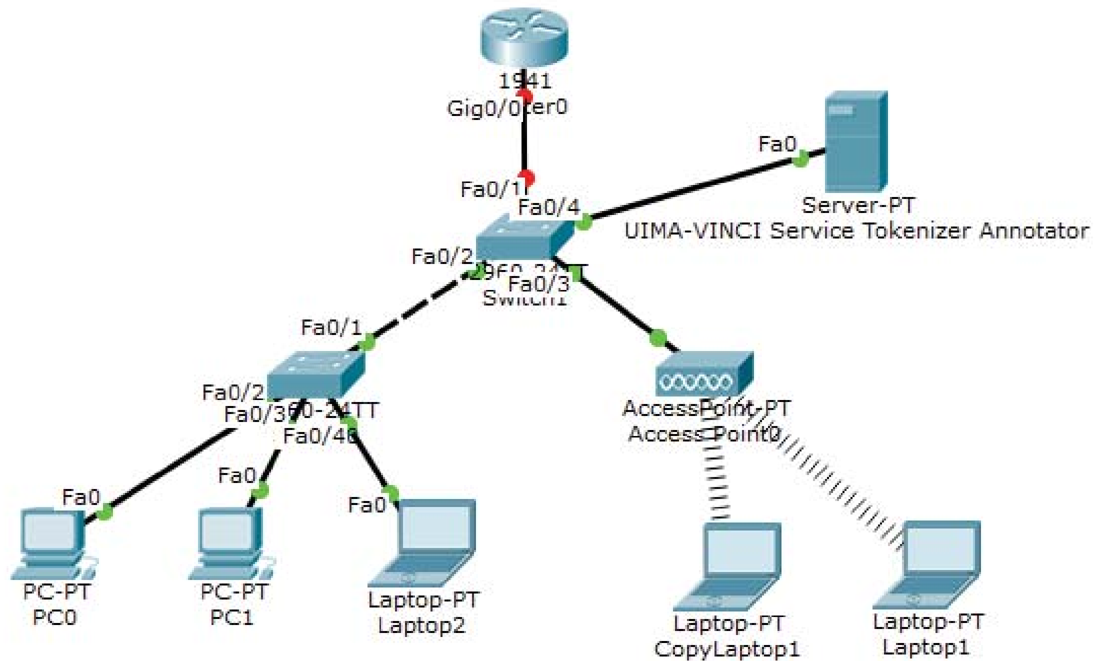


Ilustración 4.Ejemplo de red con servidor Vinci de anotadores remotos

En el ejemplo de la Ilustración 4.Ejemplo de red con servidor Vinci de anotadores remotos los dispositivos de tipo PC y Laptop acceden al servidor Server-PT en donde se encuentra el anotador remoto. Ahora bien, para hacer disponible este anotador hay que hacer "el despliegue" del servicio y una vez desplegado podremos invocarlo.

Como todo componente en UIMA, un anotador remoto se especifica con su correspondiente descriptor XML tanto para el despliegue del servicio como para la invocación desde el cliente, pero estos dos descriptores son diferentes como se explica en los dos siguientes apartados.

2.4.1. Despliegue de un componente UIMA

Para el despliegue de un anotador UIMA se tiene que crear un archivo XML con la siguiente estructura:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <deployment name="Negation Annotator Service">
3     <service name=                    .Negation0.0.1" provider="vinci">
4         <parameter name="resourceSpecifierPath" value="NoDetectorAnnotator.xml" />
5         <parameter name="numInstances" value="1" />
6     </service>
7 </deployment>
```

Ilustración 5. Ejemplo de descriptor de despliegue de un anotador

Los elementos importantes son:

- **Deployment name**: este es el nombre con el cual se identifica al despligue. No es significativo para Vinci.
- **Service name**: En Vinci los servicios son identificados por este nombre. Si hay más de un servicio con el mismo nombre, Vinci asume que son equivalentes y enrutará las consultas recibidas de manera aleatoria hacia ambos, incluso cuando se estén ejecutando en máquinas que físicamente están separadas. Por tanto, es muy importante que los servicios tengan nombres únicos.

El servicio de Vinci es accedido por su nombre, pero en una red, los dispositivos únicamente entienden de direcciones IP. De igual manera que el servicio DNS resuelve los nombres de las direcciones de máquinas a direcciones IP, y de manera transparente al usuario, Vinci tiene su propio servidor que resuelve los nombres de los anotadores a direcciones IP y puertos. Este servicio se denomina Vinci Naming Services ó VNS.

En este trabajo no se ha realizado la configuración y puesta en marcha de este servidor de nombres ni del servidor de despliegue, se ha utilizado un servidor ya en funcionamiento, por tanto, no se va a detallar cómo hay que configurar y desplegar un servidor Vinci. Se puede consultar el apartado 3.6 Working with Remote Services de la documentación de UIMA para ver los detalles.

2.4.2. Llamada a un servicio UIMA

Una vez que se ha desplegado un Analysis Engine o un CAS Consumer como servicio, puede ser utilizado por cualquier aplicación UIMA de la misma manera que se utiliza un Analysis Engine o CAS Consumer local.

En esta ocasión, es necesario crear un descriptor XML de tipo cliente. Su estructura es la siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<uriSpecifier xmlns="http://uima.apache.org/resourceSpecifier">
    <resourceType>AnalysisEngine</resourceType>
    <uri>                          Tokenization0.0.1</uri>
    <protocol>Vinci</protocol>
    <parameters>
        <parameter name="VNS_HOST" value=".            "/>
        <parameter name="VNS_PORT" value="    "/>
  </parameters>
</uriSpecifier>
```

Ilustración 6.Ejemplo de descriptor de llamada de un anotador remoto

En este caso se especifica:

- <**uri**>: el nombre del servicio que queremos utilizar, (debe de haber un servicio con el mismo nombre desplegado en el servidor Vinci)
- <**protocol**>: siempre Vinci
- <**parameters**>: normalmente son “VNS_HOST” y “VNS_PORT”, dirección IP y puerto del servicio

Ya tenemos acceso al anotador de manera remota. No es necesario especificar ningún Type System ni realizar ningún otro procesamiento.

Nota: Los descriptores remotos se encargan de proporcionar la especificación del Type System de los anotadores remotos, pues sin ellos no sabríamos que forma tiene el anotador remoto y no podríamos tratarlo. Por cuestiones de seguridad nunca se manda la información de nuestro anotador local, ni su type system ni su descriptor, al anotador remoto, no es necesario.

CAPÍTULO 3. ARQUITECTURA DE LA APLICACIÓN

En los capítulos anteriores se ha descrito el problema que queremos abarcar, una introducción al framework UIMA y pequeños avances sobre los elementos utilizados en la aplicación como el uso de un diccionario o los anotadores remotos.

En este capítulo se van a describir los componentes que forman la aplicación, así como los elementos externos necesarios para su correcto funcionamiento.

3.1. Visión general de la aplicación

El anotador desarrollado en este TFM forma parte de una cadena de anotadores que conjuntamente conforman una aplicación final y que a su vez hacen uso también de anotadores remotos que son comunes para todos los demás anotadores.

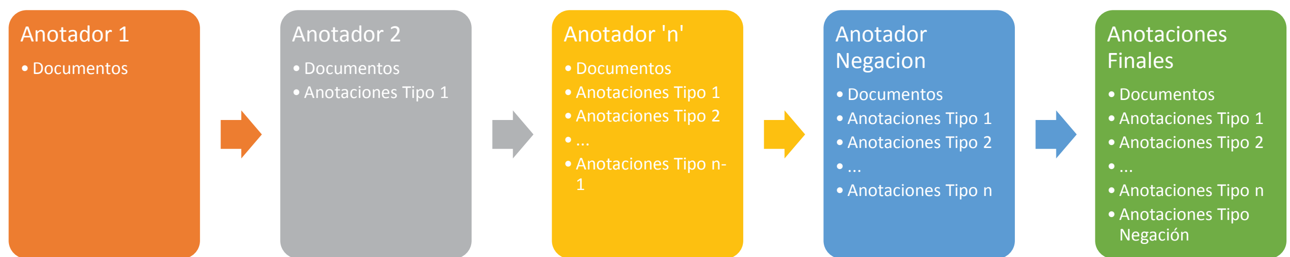


Ilustración 7 Secuencia de ejecución de los anotadores

En el momento de escribir esta memoria, el anotador de la negación es el último de la cadena, lo que quiere decir que es el último en ejecutarse. Como recordatorio, puesto que algunos de los anotadores dependen de los resultados producidos por anotadores previos, la ejecución es puramente secuencial, independientemente de que utilicen anotadores remotos. En la siguiente ilustración se muestra un esquema de cómo encaja el anotador de la negación en la cadena de anotadores.

Como se muestra en la ilustración anterior, cada anotador recibe el/los documentos que están siendo procesados y además, a excepción del primer anotador, recibe las anotaciones generadas por los anotadores anteriores de cada uno de los documentos.

Notar que la ilustración anterior sólo muestra en dónde encaja el anotador de la negación en la cadena final. En la Ilustración 8. **Visión completa de la aplicación** se muestra la visión completa de la aplicación incluyendo los anotadores remotos.

Cada anotador puede hacer uso de las anotaciones previas, de algunas o de ninguna de ellas.

En el caso del anotador de negación sólo se utilizan las anotaciones de tipo **"Token"** y de tipo **"Sentence".** El primero de ellos por cada palabra o símbolo de puntuación, interrogación, exclamación … genera una anotación de tipo Token. El segundo de ellos, probablemente basándose en el primer anotador, genera anotaciones de oraciones completas. La implementación de estos dos anotadores no forma parte de este TFM, simplemente se utilizan los resultados generados por ellos como base para construir el anotador de negación.

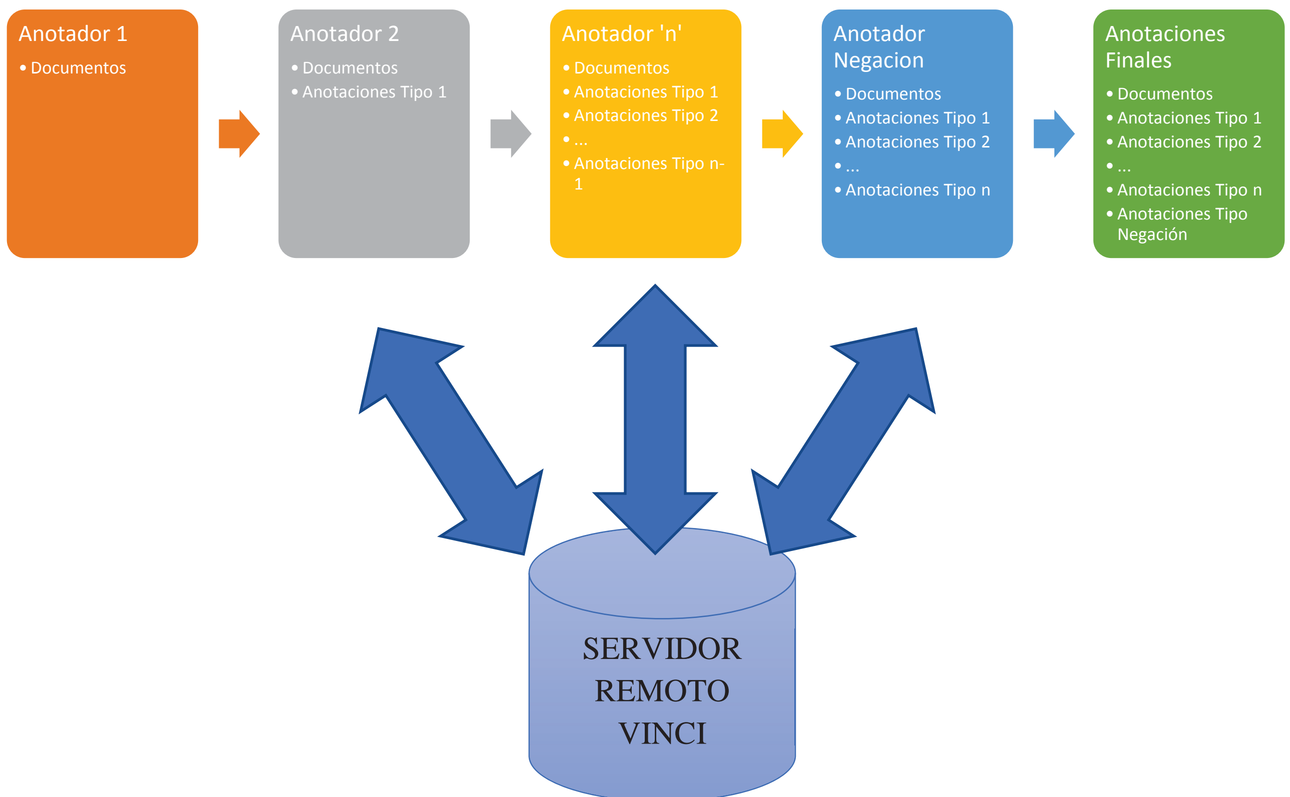


Ilustración 8. Visión completa de la aplicación

3.2. Especificación de Requisitos Software (ERS)

3.2.1. Introducción

En este apartado se recoge la Especificación de Requisitos Software (ERS) del módulo que se está implementando, un componente Java para la anotación de metadatos sobre documentos que será integrado dentro de una aplicación mayor. Esta especificación se ha estructurado inspirándose en las directrices dadas en el documento "Especificaciones de requisitos software según el estándar IEEE 8302 (IEEE Std. 830-1998).

Cada requisito se identificará de forma única dentro de la ERS siguiendo el siguiente patrón de nombrado:

Requisito $_XX: *Requisito en lenguaje natural*

3.2.1.1. Propósito

El propósito del presente documento es el de definir de manera clara y precisa las funcionalidades y restricciones del componente que se quiere construir. El documento va dirigido, al grupo de calidad que puedan necesitar en algún momento consultar los requisitos para una posible ampliación u optimización de este, y en cierta medida a los usuarios finales para que conozcan los objetivos del módulo, conocido como anotador UIMA.

3.2.1.2. Ámbito del sistema

El motor que impulsa el desarrollo de este módulo es la necesidad de identificar en los textos recibidos palabras específicas que se encuentren negadas.

El módulo, recibirá información de entrada, que previamente ya ha sido procesada por otros anotadores, podrá utilizar esa información que se ha ido añadiendo y aportará nuevo conocimiento extraído del análisis de los resultados de los anotadores previos, así como del texto a analizar.

El módulo no modificará el texto recibido, ni realizará cambios sobre los anotadores previos.

3.2.1.3. Definiciones, acrónimos y abreviaturas

Acrónimo	Descripción
UIMA	Unstructured Information Management Arquitecture

Tabla 3. Acrónimos

Término	Definición
Persistent Negative	Palabra o conjunto de palabras que indican o sugieren negación

Tabla 4. Definiciones

3.2.1.4. Referencias

Todo el documento se ha redactado siguiendo las especificaciones detalladas por el estándar IEEE *Recommended Practice for Software Requirements Specifications (IEEE Std. 830-1998).*

3.2.1.5. Visión general del documento

El apartado de especificación de requisitos software se encuentra en el primer apartado del capítulo 5, *Arquitectura de la Aplicación,* el cual se divide en tres secciones:

- Introducción**:** sección en donde se describe el formato de la Especificación de Requisitos Software (ERS).
- Descripción General**:** sección en donde se describen todos aquellos factores que afectan al producto y a sus requisitos.
- Requisitos Específicos**:** esta sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los diseñadores diseñar un sistema que satisfaga estos requisitos.

3.2.2. Descripción General

En esta sección se describirán todos aquellos factores que afectan al producto y a sus requisitos. Se describe únicamente el contexto en el que se enmarcan los requisitos, no los requisitos en sí. Esto permitirá definir con detalle los requisitos en la sección 3 (Requisitos Específicos) haciendo que sean más fáciles de entender.

3.2.2.1. Perspectiva del Producto

Este módulo forma parte de una aplicación mayor formada por otros anotadores que realizan otro tipo de tareas. Los requisitos generales del sistema se enfocan a la extracción de conocimiento a partir de documentos, y crear metadatos asociados a los textos.

3.2.2.2. Funciones del Producto

En términos generales el módulo deberá proporcionar soporte a las siguientes tareas:

- **Identificación de términos negados en secciones específicas de un historial clínico**. Será necesario identificar previamente la sección del historial en la que se esté interesado extraer información.

- **Identificación de oraciones negadas y el ámbito de negación a partir de los términos negados**. Una vez identificado el término que indica negación, se tratará de averiguar el ámbito de la negación. Para ello se tendrán que identificar los signos de puntuación, principalmente tres: el punto, el punto y coma y la coma.

- **Desarrollo de anotadores sobre términos y oraciones negadas**. Cuando se haya identificado una oración y ésta sea una oración negada, entonces podremos añadir un anotador de negación sobre esa oración. A lo largo del documento se aclarará el término anotador dentro del framework UIMA.

3.2.2.3. Características de los Usuarios

No son necesarios conocimientos específicos de ninguna rama para poder ejecutar el anotador. El usuario debe de conocer cómo ejecutar los anotadores previos de los cuales el anotador de la negación obtiene la información necesaria para poder llevar a cabo su función.

3.2.2.4. Restricciones

Debido a que este módulo depende de dos módulos anteriores para su correcto funcionamiento, la única restricción es que se ejecuten previamente estos dos módulos.

Es un módulo desarrollado en Java en su versión 8, así que deberá ser necesario instalarlo en un servidor que tenga instalado una JVM para esta versión.

3.2.2.5. Suposiciones y Dependencias

En el momento en el que por las razones que fueran, se decidiera cambiar el nombre de los anotadores previos, será necesario actualizar el nombre que se haya impuesto al anotador remoto.

Puesto que el anotador depende también de un diccionario de palabras y oraciones en castellano, sin las cuales no va a ser posible detectar la negación, si se pretenden analizar textos en otras lenguas, o textos en los que aparezcan nuevas "persistent negatives".

3.2.2.6. Requisitos futuros

El módulo debería de poder implementar la capacidad para interpretar un Part of Speech. Con un Part of Speech, se obtendrá el significado morfológico de cada término

del texto que se analiza. Con ello se pretende incrementar el rendimiento del anotador de la negación, pues se conocerá a qué tipo de término está negando.

3.2.3. Requisitos Específicos

En esta sección se van a detallar los requisitos funcionales que deberán ser satisfechos por el módulo de detección de la negación para permitir a los diseñadores construir un sistema que satisfaga estos requisitos, y que permita al equipo de pruebas planificar y realizar las pruebas que demuestren si el sistema satisface o no, los requisitos. Todo requisito aquí especificado describirá comportamientos externos al sistema, perceptibles en parte por los usuarios, operadores y otros componentes.

3.2.3.1. Interfaces Externas

En este apartado se detallan los requisitos que afectan a la interfaz de usuario, interfaz con otros sistemas (hardware y software) e interfaces de comunicaciones.

3.2.3.1.1. Interfaces de Usuario

En este documento no se van a especificar requisitos de interfaz de usuario.

3.2.3.1.2. Interfaces Hardware

- **Requisito $_01:** el módulo funcionará sobre diferentes plataformas hardware (AMD, Intel, Sparc…).
- **Requisito $_02:** el módulo funcionará sobre diferentes arquitecturas hardware X86 o X64.

3.2.3.1.3. Interfaces Software

- **Requisito $_03:** el módulo se desarrollará con la versión 8 de Java.

3.2.3.1.4. Interfaces de Comunicación

- **Requisito $_04:** el módulo se comunicará con el servidor remoto utilizando el protocolo ligero Vinci que forma parte del entorno UIMA.
- **Requisito $_05:** el módulo se comunicará con el servidor remoto intercambiando la información en formato XMI.

3.2.3.2. Requisitos Funcionales

En este apartado se detallan las funciones que debe realizar el módulo para su correcto funcionamiento. Los requisitos funcionales se han listado por objetos, es decir detallando los atributos y funciones que deberá de ofrecer cada objeto.

- **Requsito $_06:** El módulo consultará un diccionario de términos de "persistent negatives" y cargará cada una de ellas en una lista temporal.
- **Requisito $_07:** El módulo tendrá que recorrer las oraciones del texto recibido y averiguar si existe una "persistent negative" en cada una de ellas.
- **Requisito $_08:** El módulo anotará las "persistent negative" encontradas en cada oración.
- **Requisito $_09:** El módulo anotará, partiendo de las anotaciones encontradas, el término al cual está negando.
- **Requisito $_10:** El módulo añadirá la anotación creada, al conjunto de anotaciones finales.

3.2.3.3. Requisitos de Rendimiento

- **Requisito $_11:** El módulo inicializará una sola vez el diccionario de las "persistent negatives" la primera vez que entre en ejecución.

3.2.3.4. Restricciones de Diseño

No hay por el momento restricciones en el diseño de este módulo.

3.3. Solución

Como se ha comentado, la solución propuesta pasa por utilizar el framework UIMA en su versión para Java.

El primer paso para la construcción del anotador es definir un “diccionario” previo, que se ha tomado prestado, de palabras que indican negación en español, y se configurará la herramienta para que analice si en cada oración que se le pase existe o no la presencia de una o varias palabras de ese diccionario y comenzar así la búsqueda del supuesto término negado. Se tendrá que recorrer cada palabra de ese diccionario.

Va a tener cierta relevancia el formato de construcción del diccionario. Uno de los problemas que pueden aparecer está relacionado con el conjunto de caracteres con el que se construye el diccionario, y el conjunto de caracteres con el que está codificado el texto a analizar. Estos aspectos se irán explicando en los diferentes apartados de esta memoria.

Por otra parte, será importante fragmentar correctamente las oraciones. Para ello utilizaremos otro anotador (implementado previamente), el cual particiona los documentos en párrafos, oraciones y palabras o signos de puntuación (tokens).

Tiene que quedar claro que la detección de la negación siempre ha sido una tarea compleja y que aún queda mucho por hacer, ni mucho menos está resuelto. No se pretende en este trabajo realizar un algoritmo de negación como NegEx. Se tomará NegEx como referencia para hacer notar que la negación de oraciones en el área de NLP no está al cien por cien resuelto, aunque se han realizado numerosos avances y mejoras en los algoritmos.

El anotador que se desarrolla en este trabajo forma parte de una aplicación mayor compuesta de varios anotadores que realizan otras funciones y anotan otro tipo de resultados sobre los textos que son analizados. Como se explicará en los siguientes apartados, UIMA permite “concatenar” anotadores de distinto propósito en una misma

aplicación, pudiendo un anotador de esa cadena utilizar los resultados (anotaciones) generados por anotadores previos y añadir sus resultados al conjunto final.

3.4. Proceso de construcción

En este apartado se describe cómo se compone el anotador y los elementos que utiliza. En los siguientes subapartados se describen los pasos seguidos y se comentan los problemas surgidos.

3.4.1. Definición de los tipos (TypeSystem)

El primer paso es el más sencillo y consiste en definir el tipo del anotador de negación. El TypeSystem de la negación es simple y no contiene ningún atributo adicional. Como aclaración cualquier anotador define obligatoriamente y por defecto tres atributos: start, begin y sofa, indicando el carácter de inicio, carácter de fin, y el “subject of analysis), elemento de tipo texto, audio o vídeo que se está analizando.

```
NegationTypeSystem.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <typeSystemDescription xmlns="http://uima.apache.org/resourceSpecifier">
3     <name>NegationTypeSystem</name>
4     <description/>
5     <version>1.0</version>
6     <vendor/>
7   <types>
8     <typeDescription>
9       <name>                        NoDetector</name>
10      <description/>
11      <supertypeName>uima.tcas.Annotation</supertypeName>
12    </typeDescription>
13  </types>
14 </typeSystemDescription>
15
```

Ilustración 9. Definición del TypeSystem de la negación

3.4.2. Generar las clases Java correspondientes con esos tipos

Cuando se guarda un descriptor que hemos modificado con el Component Descriptor Editor (plug-in de Eclipse) por defecto se generan automáticamente las clases Java correspondientes a los tipos que están descritos en ese descriptor.

Estas clases Java tienen el mismo nombre (incluido el paquete) que los tipos CAS y tendrán implementados los métodos getters y setters para cada propiedad que se haya definido.

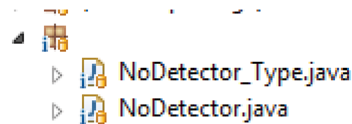


Ilustración 10. Clases Java de negación generadas

En el caso de la negación no se han añadido atributos adicionales al anotador así que no habrán ni getters ni setters.

3.4.3. Preparación del diccionario de palabras

El tercer paso y uno de los más importantes en este trabajo, es cargar el diccionario de palabras antes de realizar el procesamiento del texto.

3.4.3.1. Escribir el diccionario de palabras

El anotador está basado en una solución parcial del algoritmo NegEx explicado en el apartado **Algoritmo NegEx** de "persistencias negativas". Las persistencias negativas de este trabajo se han obtenido a partir del listado ofrecido por la implementación del algoritmo NegEx en su versión castellana [4], y adaptándola al área de estudio que representan los textos clínicos que se están analizando.

El diccionario de palabras es un archivo txt en el cual cada “persistencia negativa” conforma una línea. En la siguiente tabla aparecen algunas de las “persistencias negativas” utilizadas:

Ausente	Ausencia de	Descartando	Dudar
Excluído	Falta de	Faltos de	Fueron negativos
Improbable	Libre de	Nada	Negativo
Negó	Ninguna indicación de	Ningún	No
Nunca	Rechazado	Se descartan	Se descarta
Sin aumento	Sin cambios	Sin causa de	Sin evidencia de
Sin novedad	Sin queja de	Sin resultados de	Sin signos de
Sin signos	Sin	Suficiente para descartar	Tampoco

Tabla 5. Muestra de las "persistencias negativas"

En la solución propuesta es necesario inicializar el diccionario de palabras, y una sola vez, cuando entra a ejecutar el anotador de la negación.

Esta inicialización se realiza en un método especial de UIMA llamado *initialize*. Es en este método en donde generalmente se inicializan atributos y estructuras de datos que van a permanecer estáticos durante toda la ejecución del algoritmo.

```
/**
 * Método en el que se inicializan:<br>
 * - La lista de palabras local a partir de la lectura del archivo dictionary.txt<br>
 * - El archivo dictionary.txt se encuentra en el directorio "resources" del proyecto Java<br>
 * - El HashMap que utiliza para evitar que se anote más de una anotacion que comience en la misma posicion<br>
 * - El ArrayList de objetos de tipo Token que finalmente seran añadidos como anotacion final<br>
 * @param aContext Contexto UIMA<br>
 * @throws ResourceInitializationException si no se puede inicializar el "Dictionary"
 * @see {@link UimaContext}
 */
public void initialize(UimaContext aContext) throws ResourceInitializationException {
    super.initialize(aContext);
    try {
        StringMapResource_impl mMap = (StringMapResource_impl)getContext().getResourceObject("Dictionary");
        listaPalabras = mMap.getLista();
    } catch (ResourceAccessException e) {
        e.printStackTrace();
    }
}
```

Ilustración 11. Método "initialize" del anotador de la negación

En este caso estamos obteniendo el recurso que tiene nombre “Dictionary”. En la configuración del Analysis Engine, hay un apartado llamado “Resources” en donde se especifica el path del archivo .txt del diccionario y se le asigna un nombre para localizarlo en la aplicación. En este caso lo he denominado “Dictionary”.

3.4.4. Crear el descriptor del Analysis Engine

En el punto 3.3.3.1 Escribir el diccionario de palabras, se ha incluido un pequeño avance del contenido del descriptor del Analysis Engine, nombrando el recurso "Dictionary" el cual se especifica en este descriptor.

El Analysis Engine del anotador de negación es de tipo primitivo y tiene que ser capaz de reconocer dos tipos de TypeSystems: el TypeSystem de "Tokenización", el cual está formado por la agrupación de tres tipos de anotaciones: Paragraph, Sentence y Token, y el TypeSystem de la Negación.

Seremos capaces de construir objetos de tipo "Paragraph", "Sentence" y "Token" resultantes de los análisis del proceso de Tokenización, puesto que nuestro proyecto Maven incluye como dependencia al proyecto de Tokenización.

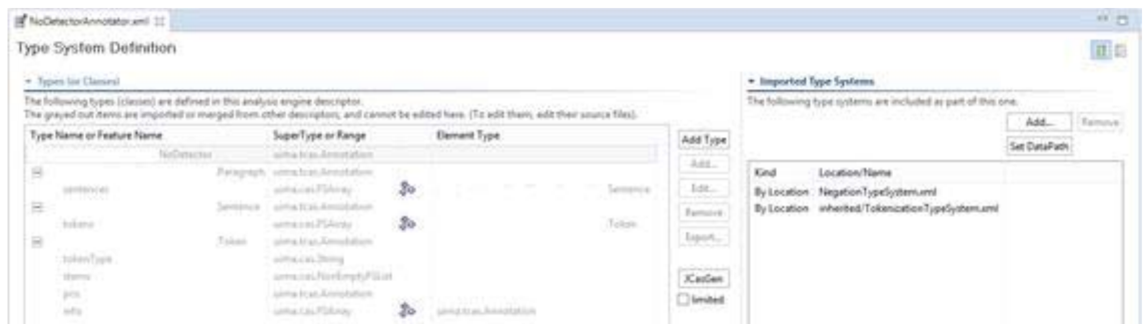


Ilustración 12. Pestaña TypeSystem del descriptor de la negación

En la ilustración anterior se muestra la pestaña "TypeSystem" del Analysis Engine de la negación, formado por el TypeSystem "TokenizationTypeSystem.xml" y el "NegationTypeSystem.xml".

No hay más información que añadir en esta pestaña. Sin embargo, nos queda irnos a la pestaña "Resources" y añadir nuestro diccionario pues es prácticamente la parte fundamental de este anotador de la negación.

3.4.4.1. Añadir diccionario de predicates negatives

A la hora de añadir un recurso externo, UIMA nos exige que la clase que utilicemos para acceder a este recurso, sea una clase que implemente el método "*load*" de la clase ***org.apache.uima.resource.SharedResourceObject***. La información de la interfaz proporcionada por el JavaDoc es la siguiente:

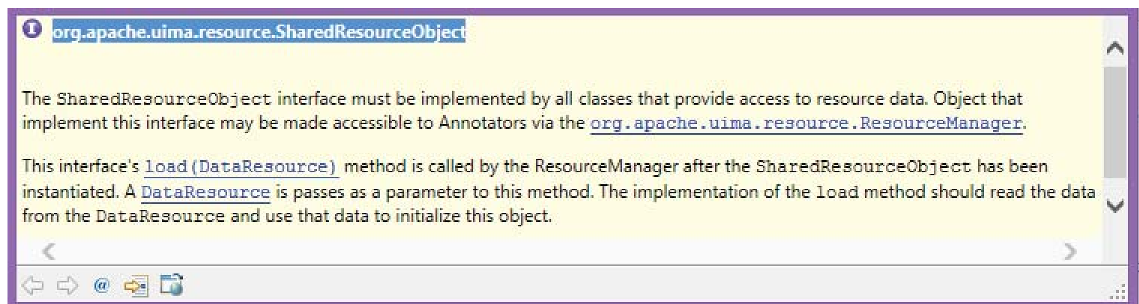


Ilustración 13. Información de la interfaz SharedResourceObject

Sólo hay un método que tenemos que implementar, el método "load". Para este proyecto estamos cargando un archivo .txt. Por tanto, utilizaremos las clases Java InputStream y BufferedReader para leer cada línea del archivo y almacenarlas en una lista de Strings.

Esta lista de Strings es la que contiene cada una de las "persistent negative" y la haremos accesible desde esta clase con su correspondiente método "get".

En la Ilustración 14. Pestaña "Resources" mostrando la inclusión del diccionario se muestra cómo hay que indicarle a UIMA en la pestaña "Resources" en la izquierda, dónde se encuentra el archivo .txt, cuál es la clase Java que implementa el método "load" que en este caso la he llamado ***…StringMapResource_Impl*** y cuál es el recurso al cual está "ligado", en este caso al recurso lo he llamado "Dictionary" y está definido en el panel de la derecha, y es el nombre por el cual vamos a recuperar en el código el diccionario de palabras.

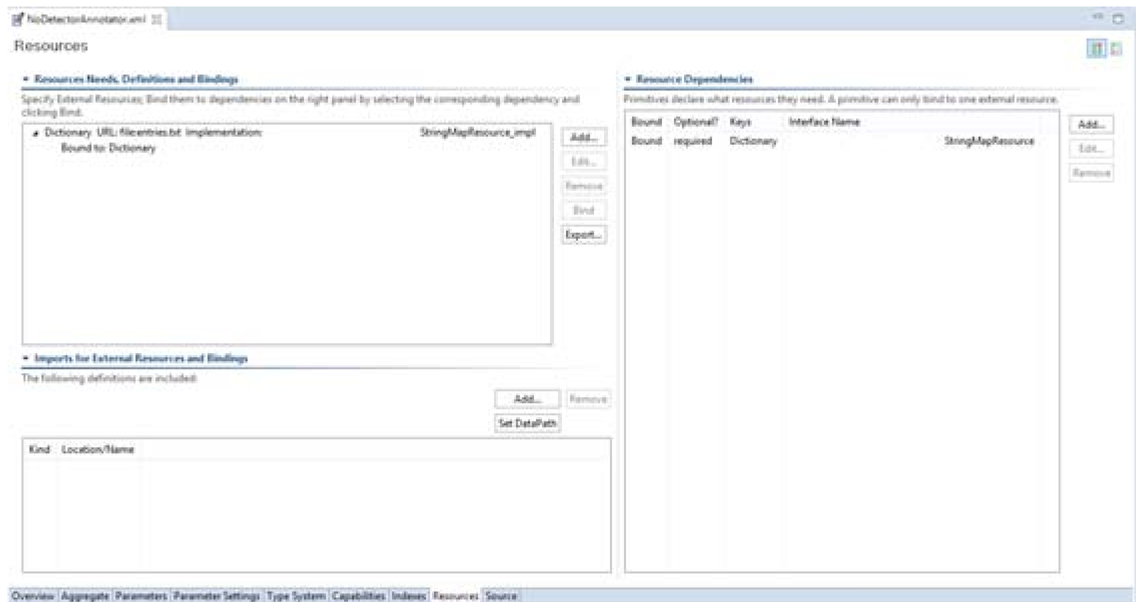


Ilustración 14. Pestaña "Resources" mostrando la inclusión del diccionario

El recurso de nombre “Dictionary” tiene que saber cuál va a ser la interfaz que va a implementar su funcionalidad (nota: no nos estamos refiriendo al método “load”, el cual está implementado en ***…StringMapResource_Impl,*** nos referimos al/los método/s propios de nuestro diccionario). En este caso sólo he implementado un método “get” para obtener la cadena.

Como se muestra en la parte izquierda de la imagen 24, el archivo “entries.txt” que es el nombre del diccionario, ha quedado ligado al recurso “Dictionary” que se ha definido en la parte derecha.

En este punto ya queda definido el diccionario de palabras y ya puede utilizarse en el anotador. Sólo es necesario, como se explicó al comienzo de este mismo punto, obtener una lista de las palabras en el método *initialize*.

3.4.5. Incluir TypeSystem remoto de Tokenization

Para asegurarnos de poder obtener correctamente las anotaciones de tipo "Parrafo", "Sentence" y "Token" ejecutados por el anotador remoto, instalamos en nuestro anotador el TypeSystem del anotador remoto, **pero este paso sólo lo realizamos a efectos de "testing" de manera local**, puesto que no podemos obtener mientras desarrollamos todas las anotaciones de la cadena.

3.4.6. Escribir el código del anotador

El último paso es escribir el código del método "process". Este método ha sufrido varios cambios, podríamos decir que "naturales", propios del desarrollo de una aplicación a lo largo del tiempo para ir adaptándolo a las necesidades de la aplicación general.

El método process se ejecuta una vez por cada documento que se recibe en el JCas pasado como argumento, por tanto, lo primero que tenemos que hacer es recuperar aquellas anotaciones que nos interesen.

3.4.6.1. Paso 1: recuperación de las anotaciones previas

Como recordatorio, todos los anotadores desarrollados en el grupo utilizan las anotaciones de tipo Paragraph, Sentence y Token generadas por el primer anotador de la cadena. En mi caso sólo utilizo las anotaciones de tipo Sentence y de tipo Token. Así que para poder recuperar los textos en los que añadir mis anotaciones de negación, tengo que **recuperar lo primero** las anotaciones de tipo Sentence como se muestra en la siguiente imagen:

```
FSIterator<Annotation> iter = jCas.getAnnotationIndex(Sentence.type).iterator();

List<Sentence> listaSentences  = new ArrayList<Sentence>();

while(iter.isValid()) {
    Sentence sentence = (Sentence) iter.get();
    listaSentences.add(sentence);
    iter.moveToNext();
}
```

Ilustración 15. Recuperación de las anotaciones de tipo Sentence

3.4.6.2. Paso 2: obtención del texto

El **segundo paso** es recorrer cada una de las anotaciones de tipo Sentence obteniendo el texto cubierto con el método *getCoveredText*.

```
while(i < listaSentences.size()) {

    Sentence s = listaSentences.get(i);

    String textoDeSentence = s.getCoveredText();
```

Ilustración 16. Recuperación del texto cubierto por la anotación Sentence

3.4.6.3. Paso 3: averiguar si existe una "persistent negative"

El **tercer paso** se realiza dentro del bucle anterior, y consiste en averiguar si existe o no una "persistent negative" en la oración. Una posible aproximación podría ser utilizar simplemente el método *substring* o el método *contains* del objeto String de Java.

Problemas que surgen en este punto

- ***Substring(int beginIndex) ó substring(int beginIndex, int endIndex):*** devuelve la subcadena que comienza en un índice determinado, ó la subcadena que está comprendida entre un índice de inicio y un índice de fin. Por tanto, no nos resuelve el problema de saber si una palabra está contenida dentro de otra.

- ***Contains(CharSecuence s):*** nos devuelve “true” si la cadena contiene en alguna parte la CharSecuence recibida como parámetro.

Solución parcial (Matcher)

Para intentar averiguar si la cadena está contenida o no en otra mayor, he utilizado la clase “Matcher” de Java, la cual encuentra, basándose en un Pattern (que puede ser una expresión regular, **una palabra o conjunto de palabras**) el índice del String en el cual aparece la primera ocurrencia de la palabra en la cadena mayor.

Este apartado se dice que es una “solución parcial” pues al encontrar sólo la primera ocurrencia de la palabra, en una oración en la que apareciera más de una vez la misma palabra, a excepción de la primera ocurrencia, no se identificarían las demás.

Por ejemplo, en la oración: “No HTA, no DM, no DL” se anotaría el primer “No” por dos razones, es la primera ocurrencia de la palabra y empieza por mayúscula, y se anotaría el segundo “no” al ser la primera ocurrencia de la palabra “no”. El tercer “no” no se anotaría y a la hora de preguntar por la enfermedad DL, se diría que esta afirmada.

Solución final (Matcher)

La solución final pasa por ir almacenando en una lista de enteros los índices en los cuales comienza la palabra que se está buscando, y puesto que sabemos la palabra que se está buscando, sabemos también su longitud en caracteres, la cual utilizaremos para crear las anotaciones.

```
private static List<Integer> isContained(String source, String subItem) {
    String pattern = "\\b" + subItem + "\\b";
    Pattern p = Pattern.compile(pattern);
    Matcher m = p.matcher(source);
    List<Integer> result = new ArrayList<Integer>();
    while(m.find()) {
        result.add(m.start());
    }
    return result;
}
```

Ilustración 17. Método para obtener la lista de índices de la palabra en la oración

Si el método "*matcher.find()*" devuelve true significa que ha encontrado una ocurrencia de la palabra en la oración. Con el método "*matcher.start()*" se obtiene en un entero el índice en el cual comienza la palabra dentro de la oración.

3.4.6.4. Paso 4: crear anotaciones parciales

El **cuarto paso** es crear las anotaciones a medida que vamos encontrando ocurrencias de las diferentes "pertinent negatives" en las oraciones.

Se viene diciendo, que el algoritmo recorre toda la lista de "pertinent negatives", y que por cada ocurrencia de una de ellas en una oración se crea una anotación. El siguiente problema al que nos enfrentamos es lo que denomino "sobreanotación". Con "sobreanotación" me refiero a la generación de anotaciones innecesarias desde el punto de vista clínico pero que debido al uso que hago de la función "contains" y del "matcher" no puedo evitar directamente que se creen anotaciones.

Problemas que surgen en este punto

En la oración "Sin síntomas de HTA", tal y como tengo construido el diccionario de "persistent negatives", el algoritmo encontraría en primer lugar "Sin síntomas de", luego encontraría "Sin síntomas" y por último "Sin". Por tanto, se me generan tres anotaciones. Es correcto, pues efectivamente en el texto aparecen las tres ocurrencias, pero no es efectivo.

La anotación final debería ser "Sin síntomas de", así que mi solución a este problema pasa por crear un HashMap de tipo HashMap<Integer,NoDetector>, en el cual la clave es un entero que indica el carácter en el cual comienza la anotación, y como valor se almacena la anotación generada.

¿Para qué estoy utilizando el HashMap? A medida que vaya encontrando las anotaciones las voy guardando en la posición en la cual comienza, pues **sólo puede haber una anotación, de tipo NoDetector, que comience en una determinada posición, no tiene sentido tener más de una anotación del mismo tipo que comience en la misma posición, aunque no digo que no se pueda hacer.**

Puesto que el HashMap sólo permite un valor por cada clave, en el caso de que me encontrara una anotación, siempre del mismo tipo (NoDetector), que comenzase en la misma posición, ya sabría si hay o no otra anotación añadida en esa misma posición.

Si no hay ninguna anotación, simplemente la añado al HashMap. Si ya hay una anotación, tengo que comparar las longitudes de la que quiero añadir y la que ya está añadida y quedarme con la de mayor longitud.

3.4.6.5. Paso 5: identificar al término negado

El **quinto paso** es identificar el **término al que niega** cada una de las anotaciones creadas en el tercer paso. Ésta va a ser la anotación final que hemos estado buscando durante todo el trabajo.

Para ello, lo primero es recuperar las anotaciones de tipo NoDetector que se encuentran dentro de una oración. En la siguiente ilustración se especifica cómo recuperar en una lista de tipo NoDetector las anotaciones de una oración.

```
//Devuelve los anotadores contenidos en la oracion comprendida entre beginSentence y endSentence
/**
 * Método para obtener en una lista los anotadores de tipo {@link NoDetector}<br>
 * contenidos entre los caracteres de inicio y fin de una oracion.
 * @param beginSentence Entero que indica la posicion del primer caracter de la oracion.
 * @param endSentence Entero que representa la posicion del ultimo caracter de la oracion.
 * @return {@link List} de tipo {@link NoDetector} de las anotaciones comprendidas entre dos posiciones en una oracion específica.
 */
private List<NoDetector> negacionesSentence(int beginSentence,int endSentence,HashMap<Integer,NoDetector> mapAux){
    List<NoDetector> result = new ArrayList<NoDetector>();
    while(beginSentence<endSentence) {
        NoDetector nAux = mapAux.get(beginSentence);
        if(nAux!=null) {
            result.add(nAux);
        }
        beginSentence++;
    }
    return result;
}
```

Ilustración 18. Recuperación de las anotaciones entre dos índices

Si el método devuelve una lista no vacía significa que en la oración existen anotaciones de tipo NoDetector.

Para cada una de las anotaciones que haya encontrado, utilizando el índice de su último carácter puedo encontrar el primer Token cuyo carácter de inicio sea mayor que el final de la anotación, es decir el primer Token que le sucede.

```
//Token supuestoNegado = null;
if(!anotacionesSentence.isEmpty()) {
    for(NoDetector nD : anotacionesSentence) {
        int j = 0;
        boolean addedToken = false;
        while(j < tokens.size() && !addedToken) {
            Token tAux = (Token) tokens.get(j);
            if(tAux.getBegin()>nD.getEnd()) {
                //Cojo el primer token cuyo begin sea mayor que el final de mi anotacion
                if (!Pattern.matches("\\p{Punct}", tAux.getCoveredText())) {
                    //Caso en el que no sea un signo de puntuacion
                    tokensANegar.add(tAux);
                    addedToken = true;
                }
            }
            j++;
        }
    }
}
```

Ilustración 19. Búsqueda de los tokens negados

En la lista "tokensANegar" almaceno los Tokens finales que "aparentemente" son los términos negados.

Problemas que surgen en este punto

En este punto estamos creando la anotación del **supuesto** término negado. Quiero resaltar que es un **supuesto** porque, como veremos en el capítulo cuatro, la solución que proporciona este anotador es una solución "naïve". Aunque ha proporcionado unos resultados decentes, hay oraciones en las que no va a ser posible, de momento, anotar correctamente el término al cual verdaderamente está negando.

Actualmente se está anotando el término que sucede a una "persistent negative", y como se ha comentado, aplicado a los textos para los cuales está diseñado este anotador, los resultados han sido buenos en general.

Las excepciones se producen cuando, entre la "persistent negative" y la enfermedad que se quiere negar están separadas. Este problema es el que intenta solucionar el algoritmo NegEx explicado en el capítulo 2.

3.4.6.6. Paso 6: crear anotaciones finales

El **sexto paso** es crear las anotaciones finales de tipo NoDetector de los Tokens contenidos en la lista “tokensANegar” y añadirlas a los índices.

```
for(Token t: tokensANegar) {
    NoDetector palabraNegada = new NoDetector(jCas);
    palabraNegada.setBegin(t.getBegin());
    palabraNegada.setEnd(t.getEnd());
    palabraNegada.addToIndexes();
}
```

Ilustración 20. Creación anotaciones finales

Habiendo ejecutado el método *addToIndexes*, los siguientes anotadores podrán utilizar las anotaciones creadas por el anotador de la negación.

En el siguiente capítulo mostraré algunas pruebas ejecutadas y los resultados obtenidos.

CAPÍTULO 4. EJECUCIÓN DE PRUEBAS Y RESULTADOS

Las pruebas se han realizado sobre la sección "Antecedentes Personales" de un conjunto de historias clínicas de pacientes anónimos.

Se ha utilizado la herramienta "UIMA CAS Visual Debugger" de UIMA desde Eclipse. Esta herramienta se instala, junto con otras herramientas de funcionalidad similar, para poder visualizar las anotaciones creadas en un texto de entrada, especificándole un Analysis Engine.

En este apartado se muestran nueve pruebas ejecutadas sobre textos con pocas líneas y simples y textos más elaborados.

4.1. Ejecución de pruebas

4.1.1. Prueba 1. Prueba básica correcta

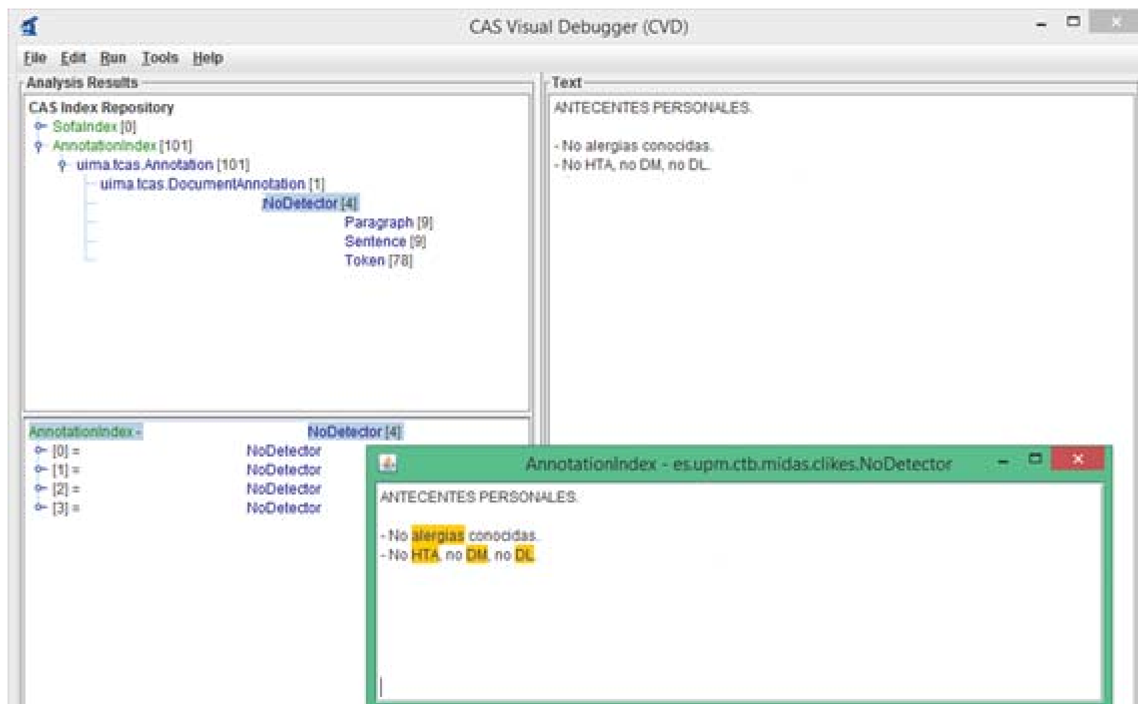


Ilustración 21 Prueba básica correcta

El resultado de la primera prueba es el correcto. Se han negado "alergias", "HTA", "DM"," DL" y efectivamente son esos los términos que se pretendía negar.

4.1.2. Prueba 2. Prueba básica parcialmente correcta

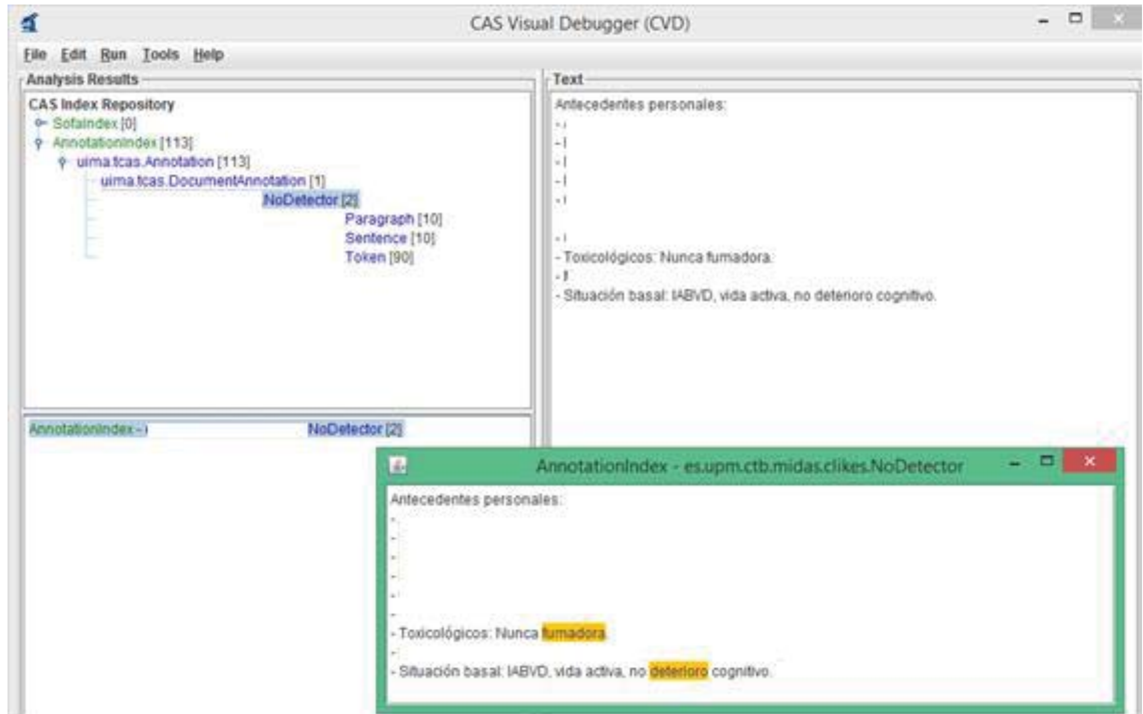


Ilustración 22. Prueba básica parcialmente correcta

El resultado desde el punto de vista del algoritmo es correcto, pues ha negado la siguiente palabra a la “persistent negative” que ha encontrado.

Ahora bien, lo deseable era que se hubiera anotado en la última oración “deterioro cognitivo” pues las dos palabras juntas representan a un solo concepto. Efectivamente está marcando que “no” hay un deterioro. Pero si el algoritmo fuera más preciso marcaría ambas palabras y nos facilitaría el tipo de deterioro (“cognitivo”).

Es esto último lo que se pretende mejorar en las siguientes versiones del anotador, pero para ello es necesario disponer de más información sobre el tipo de palabras que suceden a una anotación de negación.

4.1.3. Prueba 3. Prueba básica parcialmente correcta

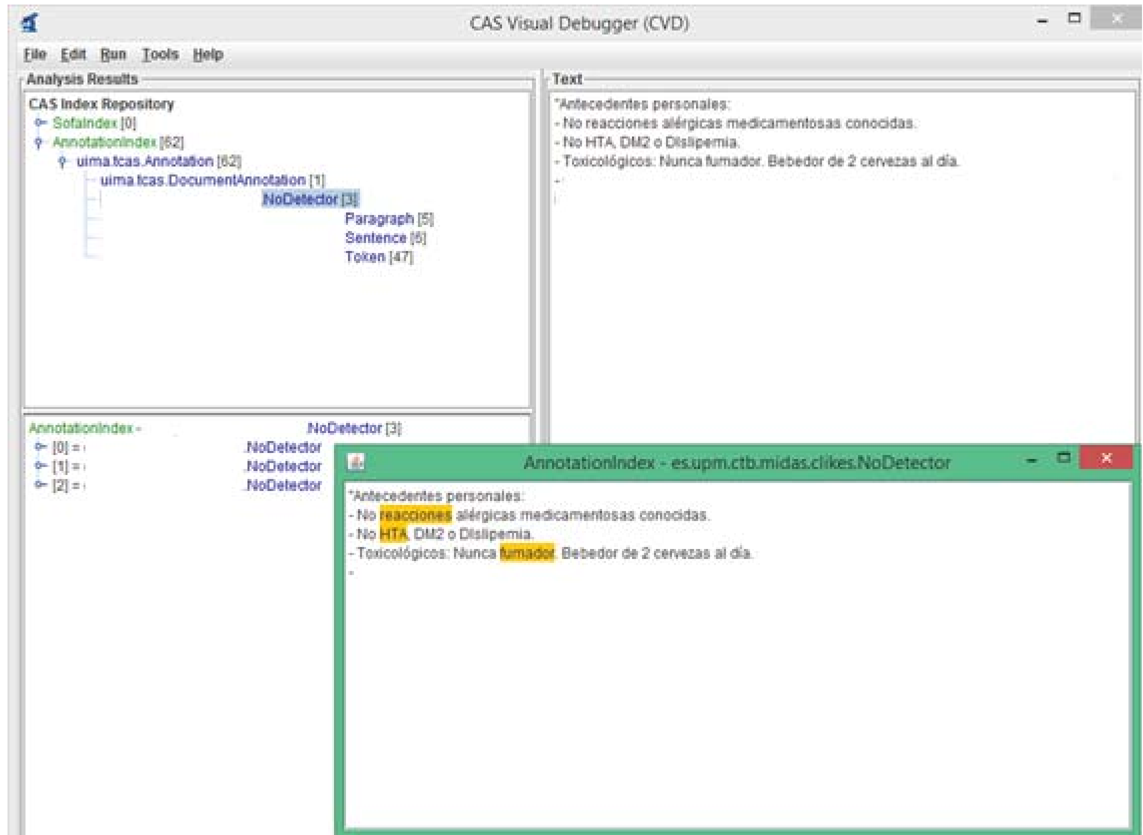


Ilustración 23. Prueba básica parcialmente correcta

En esta prueba ocurre algo parecido a la prueba anterior. Se han anotado correctamente los términos "HTA", "fumador" pero por las limitaciones de la implementación actual, el algoritmo sólo es capaz de anotar el término que sucede a la negación. En la primera anotación, se podría haber extendido a "reacciones alérgicas medicamentosas" pues es el concepto completo.

4.1.4. Prueba 4. Prueba avanzada

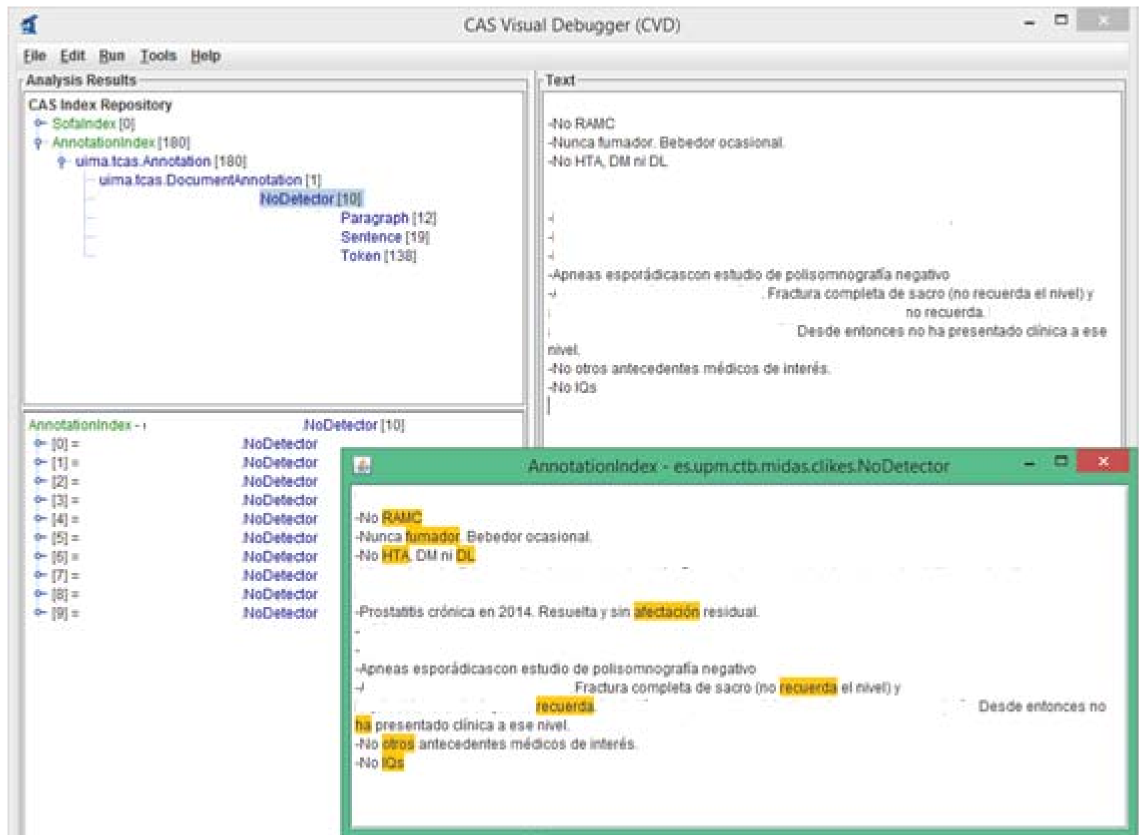


Ilustración 24. Prueba avanzada

En esta prueba se han identificado correctamente los términos “RAMC”, “fumador”, “HTA”, “DL” e “IQs”.

Como en pruebas anteriores, en la sexta oración se ha anotado “afectación” y hubiera sido más adecuado anotar “afectación residual”. El problema es el mismo que el descrito en los anteriores apartados.

El anotador ha anotado “recuerda” como palabra negada. Efectivamente el termino recuerda está negado, pero entramos en el terreno de si está proporcionando información suficiente al usuario o no.

En la novena oración, debería de haberse anotado “polisomnografía”, pues está seguida de la palabra “negativo” la cual está incluida en el diccionario de “persistent

negatives”. Debido al funcionamiento explicado del anotador, por debajo sí que se ha creado una “anotación temporal” de la palabra “negativo”, pero al no haber ninguna palabra que la suceda, no se ha creado ninguna anotación.

Esta es una de las funcionalidades a mejorar para la siguiente versión, comprobar si existe un término previo al cual puede estar negando.

La penúltima oración en la que el término “otros” está negado, es otro ejemplo de la precisión del algoritmo. Interesaría que se negasen los términos “antecedentes médicos”, aunque en este caso la oración tampoco revela información útil para el usuario, el cual está más interesado en las enfermedades, síntomas o patologías médicas.

4.1.5. Prueba 5. Prueba básica

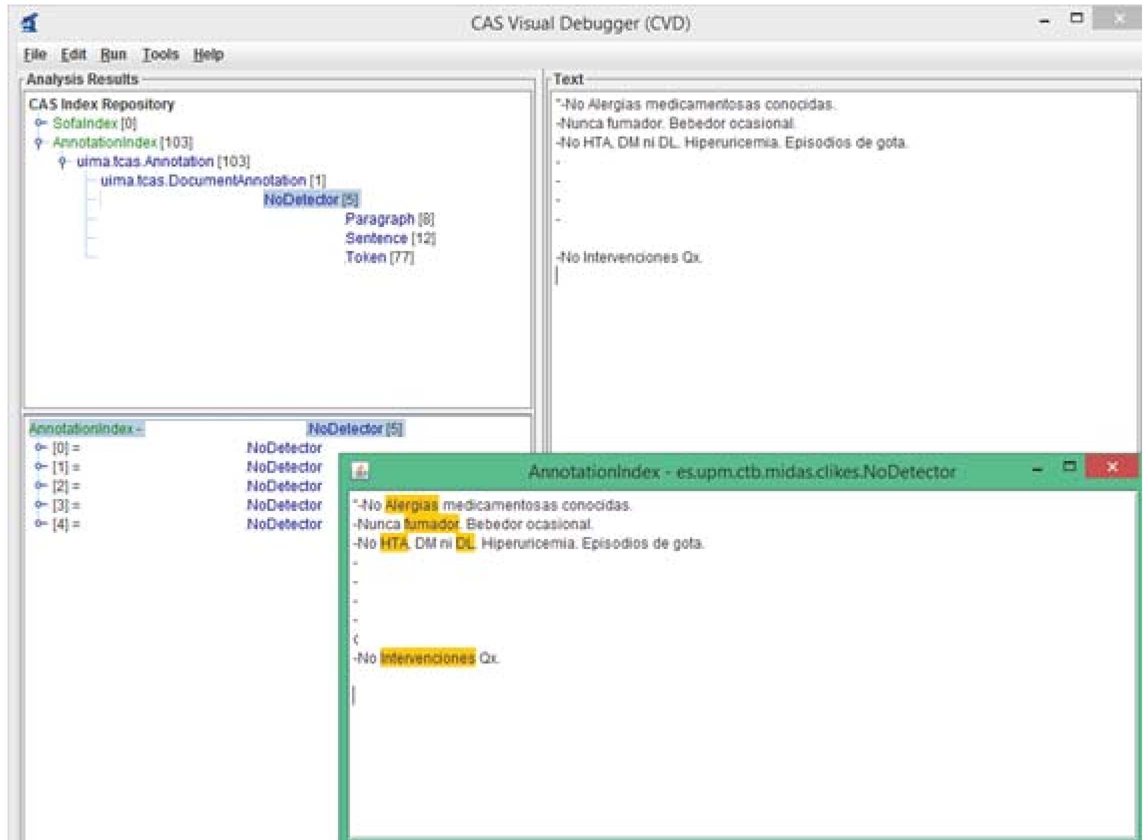


Ilustración 25. Prueba básica

En esta prueba se han identificado correctamente los términos “alergias”, “fumador”, “HTA”, “DL” e “intervenciones”.

Al igual que en pruebas anteriores, en la sexta oración, el término “negativo” se ha identificado, pero por lo explicado en la prueba anterior no se ha relacionado con ningún término, y por tanto no se ha creado ninguna anotación final.

4.1.6. Prueba 6. Prueba básica

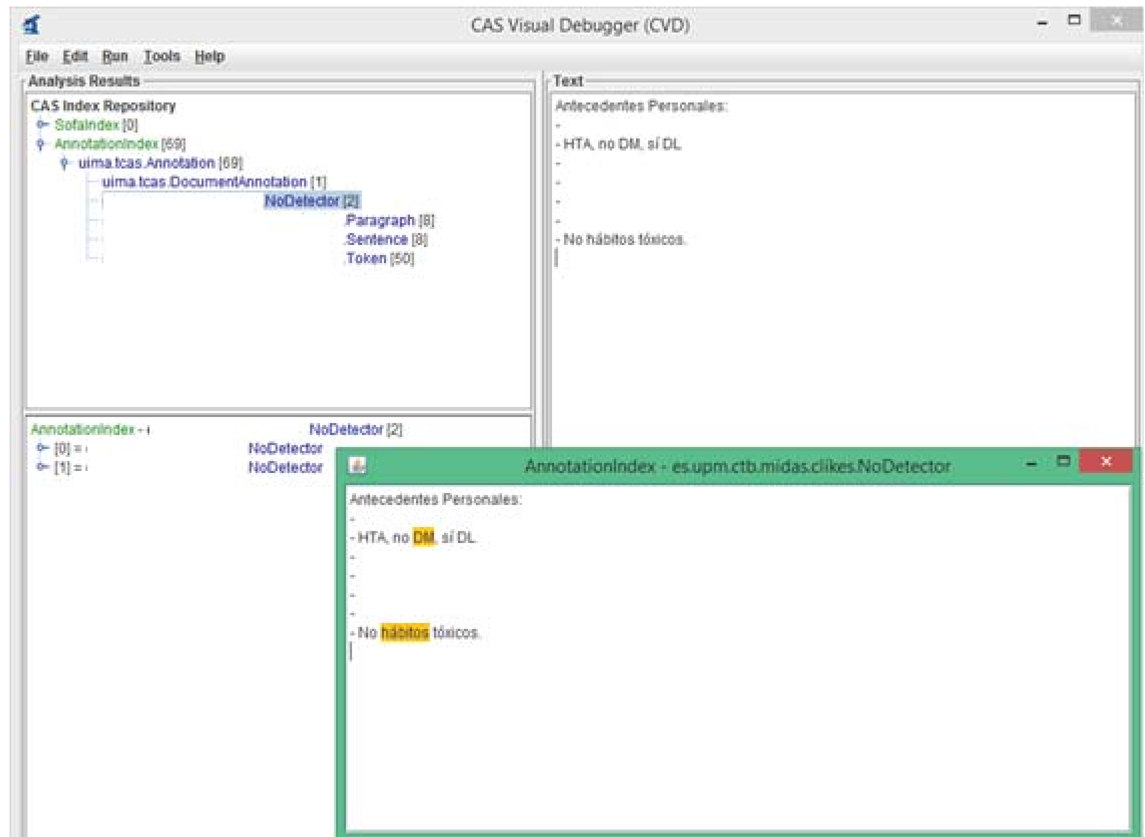


Ilustración 26. Prueba básica

El problema en esta prueba ha sido el identificar únicamente "hábitos" como término negado. Efectivamente se indica que "no hay hábitos" pero si se hubiera marcado como negado "hábitos tóxicos" sabríamos identificar los tipos de hábitos.

4.1.7. Prueba 7. Prueba compleja

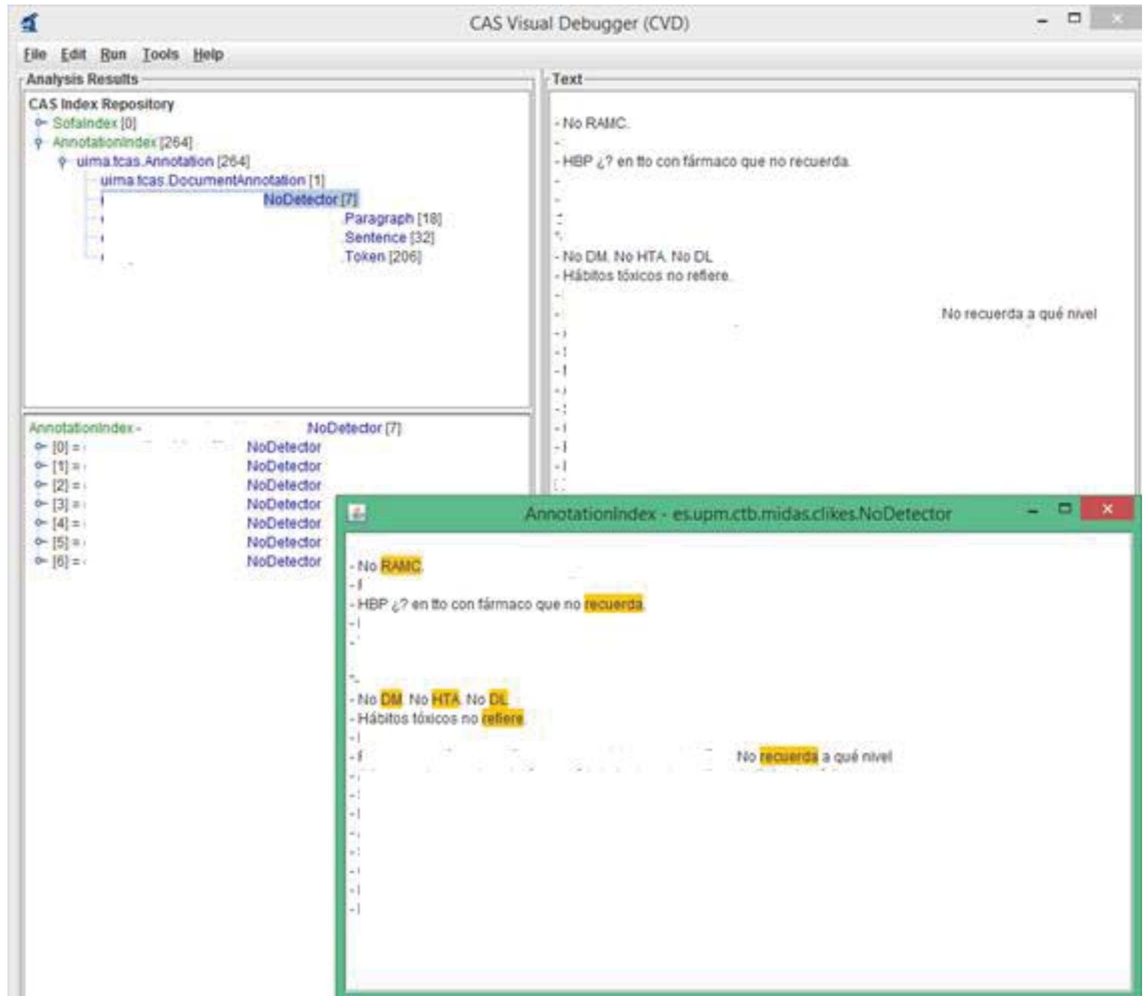


Ilustración 27. Prueba compleja

Esta prueba tiene más complejidad por la longitud del texto que por los términos a negar. Se vuelve a tratar el mismo problema que en pruebas anteriores con las palabras “refiere” y “recuerda”.

Para la palabra “refiere” la solución deseada hubiera sido anotar “hábitos tóxicos” antes que “refiere”.

Para la palabra “recuerda”, se hubiera anotado “nivel”.

4.1.8. Prueba 8. Prueba básica

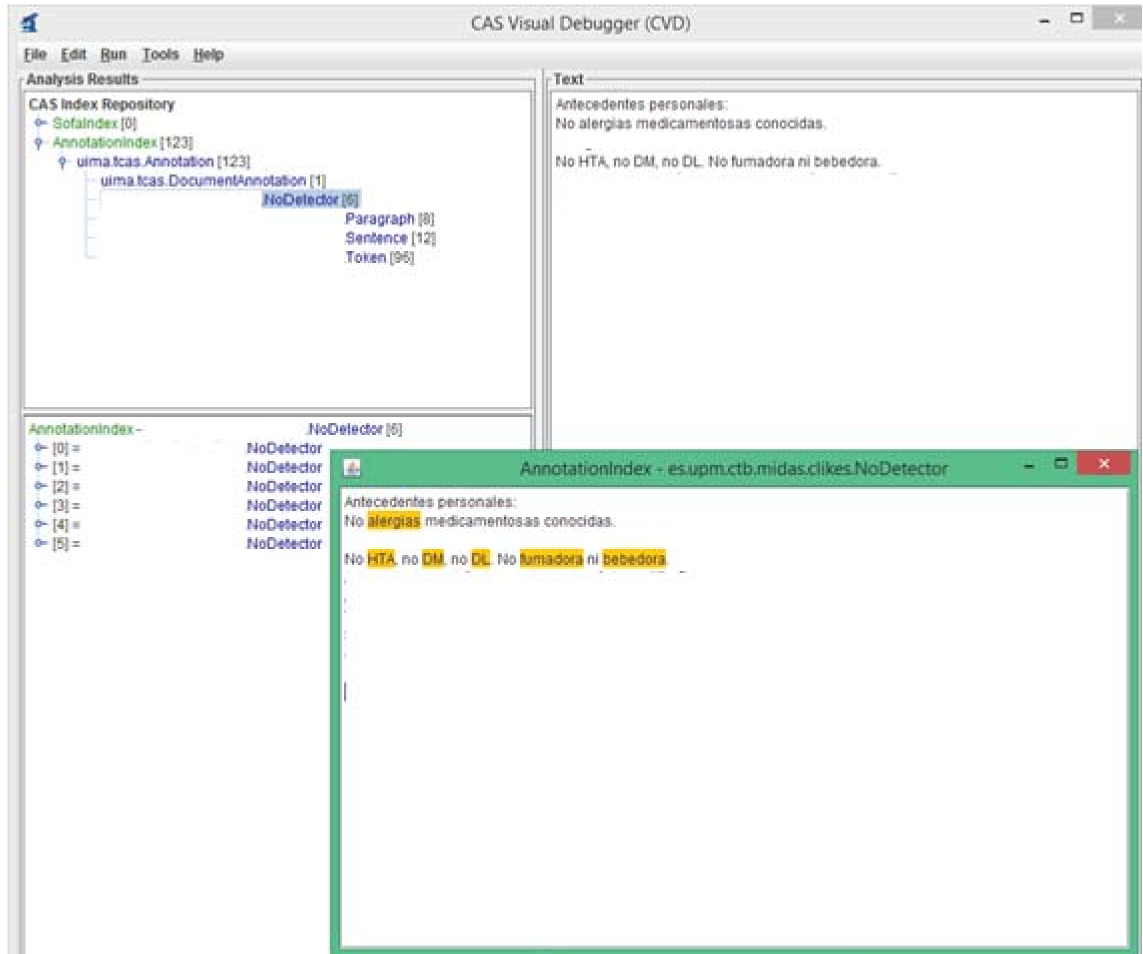


Ilustración 28. Prueba básica

Los resultados de esta prueba son bastante correctos. Se han identificado correctamente los términos “HTA”, “DM”, “DL”, “fumadora” y “bebedora”.

Sin embargo, en la primera oración el término “alergias” no refleja verdaderamente el concepto negado. La solución deseada sería negar “alergias medicamentosas” pues las dos palabras forman un concepto más preciso.

4.1.9. Prueba 9. Prueba básica

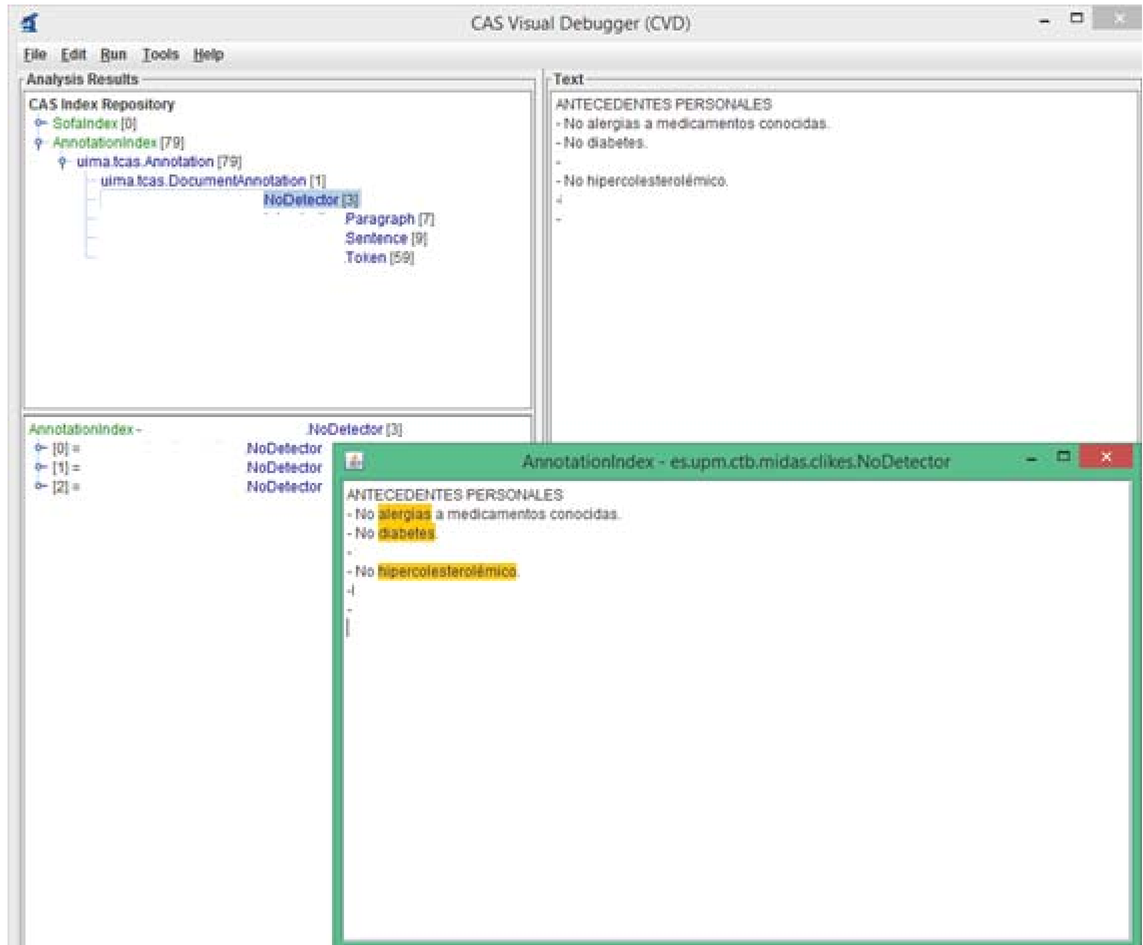


Ilustración 29. Prueba básica

En esta última prueba, al igual que en la prueba 8, el término “alergias” de la primera oración no refleja verdaderamente el concepto negado. La solución deseada sería negar “alergias medicamentosas” pues las dos palabras forman un concepto más preciso.

Sin embargo, se han anotado correctamente los términos “diabetes” y “hipercolesterolémico”.

CAPÍTULO 5. CONCLUSIONES GENERALES Y LÍNEAS FUTURAS

5.1. Conclusiones generales

En este trabajo se ha tratado de proporcionar una solución que satisfaga de una forma relativamente correcta, el problema de identificar términos negados en oraciones.

Una de las ventajas de este trabajo es que el tratamiento de la negación sólo se realiza sobre oraciones muy determinadas, en un ámbito restringido, como es el de los antecedentes personales de un historial clínico.

Lo primero que se realizó fue investigar sobre el funcionamiento de los anotadores UIMA. A partir de un ejemplo, se fue construyendo un primer anotador al que se le fueron añadiendo los anotadores remotos.

Ha sido el manejo de los anotadores remotos una parte fundamental para la ejecución del proyecto. El hecho de que en una cadena de anotadores se obtenga la misma información básica (Párrafos, Sentencias y Tokens) sobre un texto es vital para poder añadir las anotaciones de los diferentes tipos.

Después de investigar sobre los anotadores, se dieron unas directrices básicas sobre el funcionamiento del algoritmo de negación NegEx, y NegEx aplicado a textos en castellano y de cómo se ha tomado como referencia para construir una solución al problema planteado.

Se ha explicado que la solución pasa por construir un diccionario de "persistent negatives" el cual se tiene que cargar previamente en el anotador (método *initialize*) para empezar a detectar las oraciones negadas.

Han surgido problemas respecto a la cantidad de anotaciones que se generaban, lo cual se ha solucionado con un HashMap tomando como clave el índice del carácter en donde comenzaba la "persistent negative" detectada. Ha habido anotaciones (sólo he detectado un caso) que, habiéndose detectado una "persistent negative" no se ha podido identificar el término por encontrarse éste en alguna posición previa a la anotación.

A pesar de los problemas surgidos, se han logrado los objetivos marcados al comienzo del documento.

5.2. Líneas futuras

A pesar de haber logrado los objetivos propuestos, se ha mencionado que se necesitaría más información para poder mejorar las anotaciones generadas. Una posible solución sería ejecutar un anotador previo que anotase el tipo de palabra. Nos referimos a realizar un Part of Speech del documento, junto con los anotadores de Párrafos, Sentencias y Tokens.

Utilizando un Part of Speech, en términos de la negación, la precisión del anotador se vería considerablemente incrementada, pues se descartarían muchos términos que no tienen interés como pueden ser, artículos, determinantes, posesivos, adverbios incluso adjetivos, y negar únicamente las enfermedades, y así satisfacer uno de los puntos que se indicaron como objetivos que no ha sido posible alcanzar sin ayuda del Part of Speech, que es detectar el ámbito de la negación.

Como sugerencia, se puede desarrollar un anotador posterior al de la negación, que busque únicamente anotaciones negadas que sean una enfermedad. Para ello se necesitaría una base de datos, ontología o diccionario de enfermedades, por ejemplo, UMLS, SnomedCT o ICD-10 para descartar términos y sólo registrar aquellos que representen un término médico.

Bibliografía

[1] Wendy W. Chapman, Will Bridewell, Paul Hanbury, Gregory F. Cooper, Bruce G. Buchanan. "*A Simple Algorithm for Identifying Negated Findings and Diseases in Discharge Summaries*". Journal of Biomedical Informatics 34, 301-310 (2001).

[2] Wendy W. Chapman, David Chu, John N. Dowling. "*ConText: An Algorithm for Identifying Contextual Features from Clinical Text*". Department of Biomedical Informatics. University of Pittsburgh.

[3] Viviana Cotik, Vanesa Stricker, Jorge Vivaldi, Horacio Rodiguez. "*Syntactic methods for negation detection in radiology reports in Spanish*". Research Gate. https://www.researchgate.net/publication/306099791.

[4] Roberto Costumero, Federico Lopez, Consuelo Gonzalo-Martín, Marta Millan, Ernestina Menasalvas. "*An Approach to Detect Negation on Medical Documents in Spanish*". Universidad Politécnica de Madrid – Centro de Tecnología Biomédica, Madrid.

[5] Apache UIMA Development Community. "*Apache UIMA Dictionary Annotator Documentation*". Apache Software Foundation.

[6] Apache UIMA. https://uima.apache.org/

[7] The Standford Natural Language Processing Group. "*Standford Log-linear Part-Of-Speech Tagger*". https://nlp.stanford.edu/software/tagger.html

[8] Galal Aly. "*Tagging text with Standford POS Tagger in Java Applications*". http://new.galalaly.me/index.php/2011/05/tagging-text-with-stanford-pos-tagger-in-java-applications/

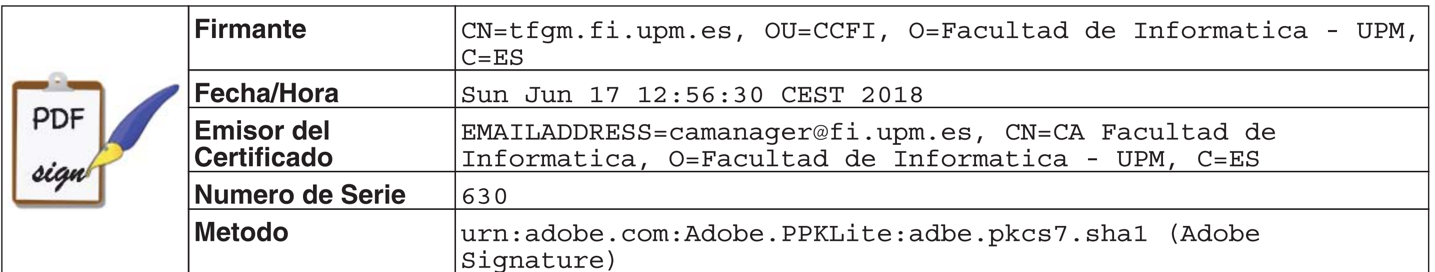
[9] Healthinformatics. *"NegEx Algorithm"*. https://healthinformatics.wikispaces.com/NegEx+Algorithm

[10] Yifan Peng, Xiaosong Wang, Le Lu, Mohammadhadi Bagheri, Ronald Summers, Zhiyong Lu. *“NegBio: a high-performance tool for negation and uncertainty detection in radiology reports”*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5961822/

[11] Saeed Mehrabi, Anand Krishnan, Sunghwan Sohn, Alexandra M Roch, Heidi Schmidt, Joe Kesterson, Chris Beesley, Paul Dexter, C. Max Schmidt, Hongfang Liu and Mathew Palakal. *“DEEPEN: A negation detection system for clinical text incorporating dependency relation into NegEx”*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5863758/

[12] ConText/NegEx. https://blulab.chpc.utah.edu/content/contextnegex

Este documento esta firmado por



	Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Sun Jun 17 12:56:30 CEST 2018
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)