Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Proyecto tecnológico

Mediciones de similitud semántica aplicados a resúmenes de artículos científicos

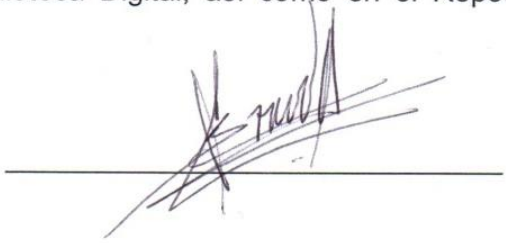
Omar Eduardo Padilla Segura
209202106
al209202106@alumnos.azc.uam.mx

Trimestre 2014 Otoño
7 de diciembre de 2014

Dra. Maricela Claudia Bravo Contreras
Profesor Asociado "D"
Departamento de Sistemas
mcbc@correo.azc.uam.mx

Dr. José Alejandro Reyes Ortiz
Profesor Titular “A”
Departamento de sistemas
jaro@correo.azc.uam.mx

Yo, Maricela Claudia Bravo Contreras, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Yo, José Alejandro Reyes Ortiz, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yo, Omar Eduardo Padilla Segura, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Tabla de contenido

Índice de figuras

Índice de tablas

Resumen

Los investigadores tienen la necesidad de localizar cierta información en artículos relacionados a su área de investigación, como puede ser: la aportación, el método utilizado y los resultados. Realizar esta búsqueda de artículos relacionados es tedioso y consume mucho tiempo.

En este proyecto se diseñó e implementó un sistema para la comparación semántica de los resúmenes de artículos científicos en inglés utilizando medidas de similitud basadas en el significado de los textos.

Las técnicas propuestas se basan en el significado del texto, esto es indispensable para otorgar una solución realmente efectiva que sirve de apoyo en la selección de artículos relacionados a un tema o trabajo específico.

1. Introducción

En las universidades existen investigadores que tienen el deber de hacer aportaciones para el desarrollo de la ciencia en cada una de sus distintas áreas de especialización. Los investigadores hacen difusión de su investigación a la sociedad a través de publicaciones científicas, ya que de nada sirve tener esos nuevos resultados si nadie más puede consultarlos.

Entre los distintos tipos de publicaciones de texto científico existen los artículos de investigación, estos artículos contienen la descripción del trabajo que se ha llevado a cabo, los objetivos, métodos, resultados obtenidos, frecuentemente son utilizados como material de referencia para realizar otras investigaciones.

Para avanzar en la elaboración de estos artículos científicos, los investigadores tienen la necesidad de buscar trabajos relacionados como artículos de investigación, en ocasiones, al realizar la búsqueda de estos textos se tiene el problema de no saber realmente si todos los artículos que se encontraron les serán de ayuda para tomarlos como referencia a su trabajo.

La tarea de búsqueda y selección de artículos de investigación relacionados con el tema a investigar es ardua, tradicionalmente el investigador revisará un gran número de artículos y después de analizarlos determinará cuales están relacionados con su trabajo.

Para facilitar esta tarea de búsqueda y selección de artículos, en este proyecto se propone diseñar e implementar una herramienta de software que compare los resúmenes de varios artículos de investigación por medio de medición de similitud semántica con el objetivo de obtener una tabla indicando el grado de similitud que se ha descubierto entre los resúmenes de los artículos científicos, con esto el investigador podrá seleccionar aquellos artículos que más se parezcan para tomarlos como referencia a su trabajo.

2. Antecedentes

La similitud semántica involucra un conjunto de documentos o términos que se encuentran en listas de términos y se les asigna una métrica basada en la semejanza de su significado o contenido semántico.

Existen varios proyectos terminales de la UAM Azcapotzalco que abordan técnicas de procesamiento de lenguaje natural y similitud semántica, estos temas se utilizan en éste proyecto de integración, a continuación se mencionan algunos proyectos terminales que sirvieron de referencia.

Sistema de recuperación de información semántico [1]

En este proyecto se crearon distintos módulos (léxico, sintáctico y semántico) de procesamiento de lenguaje natural para la extracción de información. Básicamente la idea general es la misma que nuestro proyecto de integración: tomar varios artículos científicos, extraer cierta información y procesarla para obtener a la salida cierto tipo de resultados. La diferencia se encuentra en que en nuestro proyecto de integración es que la salida indica una medida de similitud existente entre artículos científicos, ésta medición puede ser indicador de que tan relacionados se encuentran estos artículos.

Sistema de detección de plagio en archivos de texto [2]

En el proyecto de detección de plagio se emplearon técnicas para medir similitudes entre dos archivos. Nuestro proyecto de integración utiliza medidas de similitud entre archivos de texto pero utilizando diferentes técnicas para encontrar las distancia métrica.

El Sistema de detección de plagio en archivos de texto utiliza las técnicas de sub cadenas comunes más largas, trigramas y huella digital. Nuestro proyecto de integración utiliza un algoritmo de similitud textual semántica con seis distancias: Coseno, Jaro-Winkler, Levenshtein, Lin, Monge-Elkan y Wu-Palmer.

Sistema clasificador de documentos de proyectos terminales usando el concepto de memoria asociativa [3]

Este sistema tiene un módulo de procesamiento de archivos de texto en el que se le da formato adecuado al archivo (quitar acentos, convertir palabras a minúsculas, etc.), para la manipulación de la información requerida. Este proceso es similar al que se utiliza nuestro proyecto de integración, adicionalmente se tiene un módulo para extraer el resumen del texto científico y después se hace un proceso de anotación que consiste en un etiquetado.

3. Justificación

Se implementará una medición de similitud semántica entre resúmenes de artículos de investigación porque en los resúmenes se pueden identificar los objetivos principales, alcance de la investigación, metodología que se empleó, resultados y conclusiones principales. Esta herramienta facilitará el desarrollo de software que le permita al investigador de una universidad o centro de investigación seleccionar de forma rápida y eficiente sólo los artículos que en verdad sean relevantes y que estén muy relacionados en el trabajo que se esté realizando.

La similitud semántica es una buena técnica para desarrollar esta herramienta porque va más allá de la coincidencia sintáctica entre textos. Con esto se aumentan las probabilidades de obtener mejores resultados al realizar una consulta.

4. Objetivo General

Diseñar e implementar un sistema para descubrir relaciones de similitud semántica en resúmenes de artículos científicos escritos en idioma inglés.

5. Objetivos Específicos

- Diseñar e implementar modulo para la extracción de resúmenes a partir del corpus.

- Diseñar e implementar modulo para el procesamiento de texto contenido en los resúmenes.

- Diseñar e implementar una medición para obtener la similitud semántica de los resúmenes.

- Diseñar e implementar modulo para descubrir relaciones de similitud entre resúmenes.

- Diseñar e implementar un módulo para visualizar las relaciones de similitud entre resúmenes encontradas.

6. Marco Teórico

Las distintas distancias utilizadas para realizar este proyecto son: Coseno, Jaro-Winkler, Levenshtein, Lin, Monge-Elkan y Wu-Palmer. A continuación se describirá en que consiste cada una de las distancias.

6.1. Similaridad Coseno

La distancia del coseno no es propiamente una distancia sino una medida de similaridad entre dos vectores en un espacio que tiene definido un producto interior [4]. En el espacio euclidiano este producto interior es el producto escalar, ecuación 1.

$$\vec{X}_1 \cdot \vec{X}_2 = \|X_1\| \, \|X_2\| \, \cos\theta \qquad (1)$$

$$similaridad = \cos\theta = \frac{\vec{X}_1 \cdot \vec{X}_2}{\|X_1\| \, \|X_2\|} \qquad (2)$$

Para que la medida de similaridad esté en el rango (0,1), se calcula a través de la ecuación 3.

$$1 - \frac{\cos^{-1}(similaridad)}{\pi} \qquad (3)$$

6.2. Distancia Jaro-Winkler

Este algoritmo devuelve una distancia comprendida entre 0 y 1 entre dos cadenas de texto, donde 1 es un *match* entre las dos cadenas y 0 implica la mayor diferencia entre ambas [5]. La fórmula para calcular esta distancia es la ecuación 4.

$$\frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \qquad (4)$$

Dónde:

m es el número de caracteres comunes

t es el número de transposiciones

s_1 y s_2 son la cantidad de caracteres de cada palabra

6.3. Levenshtein

La distancia de Levenshtein entre dos cadenas A y B se basa en el conjunto mínimo de operaciones de edición necesarias para transformar A en B (o B en A) [6]. Las operaciones permitidas son:

1. Reemplazar un carácter de A por otro de B (o de B en A)
2. Eliminar un carácter de A o B
3. Insertar un carácter de B en A (o A en B)

Para obtener esta distancia se suele usar un algoritmo que se apoya en una matriz M de (n + 1) x (m + 1), donde n y m son las longitudes de las cadenas A y B respectivamente. El algoritmo se aprecia en la figura 1:

```
Desde i = 0 hasta n
M[i][0] := i
Desde j = 0 hasta m
M[0][j] := j
Desde i = 1 hasta n
Desde j = 1 hasta m
        Si A[i] = B[j] → costo := 0
        Sino → costo := 1
        M[i][j] := mínimo ( M[i-1][j] + 1, // borrado
                            M[i][j-1] + 1, //inserción
                            M[i-1][j-1] + costo)//sustitución
distancia := M[n][m]
```

Figura 1. Algoritmo de Levenshtein

6.4. Lin

Lin establece los siguientes principios básicos siguiendo un punto de vista basado en la teoría de la información [7]:

1. La similitud entre dos elementos se relaciona con su entorno. Si tienen características en común entonces hay una similitud.
2. La similitud entre dos elementos está relacionada con las diferencias que existe entre ellos. Entre más diferentes son, existen menos similitud.
3. La similitud máxima entre dos elementos se alcanza cuando los elementos son idénticos.

La ecuación para hacer el cálculo de similitud entre dos cadenas de texto c1 y c2 que pertenecen a una taxonomía es la ecuación 5.

$$similitud(c_1, c_2) = \frac{2 * \log p(c_3)}{\log p(c_1) + \log p(c_2)} \qquad (5)$$

Donde c3 es el padre común de c1 y c2.

6.5. Monge-Elkan

Monge y Elkan planearon una medida de distancia que combina esquemas basados en cadenas y en *tokens.* El cálculo de la función de similitud se hace a partir del cálculo de las distancias mínimas de edición entre *tokens,* y normalizando con respecto al tamaño en *tokens* de la cadena de origen [8], como se muestra en la ecuación 6:

$$simil(a, b) = \frac{1}{k} \sum_{i=1}^{K} max_{j=1}^{L} \left(sim'(a_i, b_j) \right) \qquad (6)$$

Donde sim' es una función secundaria basada en distancias, K es la cantidad de *tokens* de a y L es la cantidad de *tokens* de b.

6.6. Wu & Palmer

Wu y Palmer proponen un método para calcular la similitud que se basa en los conceptos comunes utilizando el camino dentro de la taxonomía de *WordNet.* En este método para calcular la similitud entre los conceptos *a* y *b*, se introduce el concepto *c*, que es el primer superconcepto común entre *a* y *b* [8]. La similitud entre *a* y *b* se calcula con la ecuación 7.

$$sim(a,b) = \frac{2N_{cr}}{N_{ac} + N_{bc} + 2N_{cr}} \qquad (7)$$

Dónde:

N_{ac} es el número de nodos en el camino de a a c.

N_{bc} es el número de nodos en el camino de b a c.

N_{cr} es el número de nodos en el camino de c a la raíz.

7. Desarrollo del Proyecto

Para la implementación del proyecto se utilizó el lenguaje de programación Java por su portabilidad y sencillez, NetBeans como ambiente de desarrollo, Apache PDFBox [9] que es una biblioteca Java de código abierto que permite trabajar con documentos en formato PDF, nos permite creación, manipulación y extracción del

contenido de documentos en formato PDF. Se utilizaron herramientas que se encuentran en GATE [10] para procesar el texto: ANNIE English Tokeniser para separar el texto en tokens, ANNIE Sentence Splitter para agrupar tokens en oraciones, ANNIE Gazetteer para buscar tokens que se encuentran en listas predefinidas, ANNIE Document Reset para limpiar los documentos a analizar y ANNIE NE Transducer para clasificar las palabras de texto en categorías predefinidas. Para la implementación de las medidas de similitud de Lin y Wu-Palmer se utilizó WS4J (WordNet Similarity for Java) [11] que es un API que contiene varios algoritmos de similitud semántica.

Se utilizan artículos científicos escritos en inglés, los cuales tienen un formato digital PDF del dominio de computación.

El proyecto consta de cuatro módulos que se describen a continuación.

7.1. Módulo extractor

El módulo recibe a la entrada un corpus constituido por artículos científicos escritos en inglés. En este módulo se realizaron las siguientes tareas:

- Buscar artículos científicos en la web para formar el corpus. Estos documentos son la entrada del sistema. Ejemplo en figura 2.
- Los artículos en pdf son transformados a texto plano utilizando la herramienta Apache PDFBox descrita anteriormente. El código fuente que realiza estas tareas se encuentran en el anexo A y anexo B.
- Se programó el algoritmo que se encarga de extraer los resúmenes de cada uno de los artículos por medio de expresiones regulares. El código fuente que realiza estas tareas se encuentran en el anexo C y anexo D.
- Se obtuvo un nuevo corpus que contiene sólo resúmenes de los artículos científicos como en la figura 3.

Figura 3. Documentos de entrada del módulo extractor

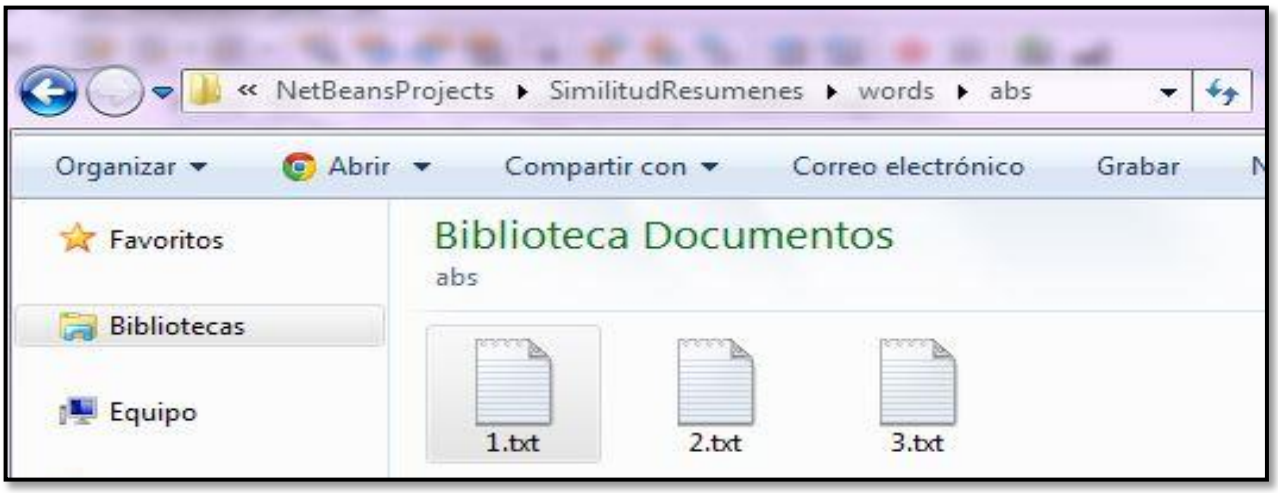


Figura 2. Documentos de salida del módulo extractor

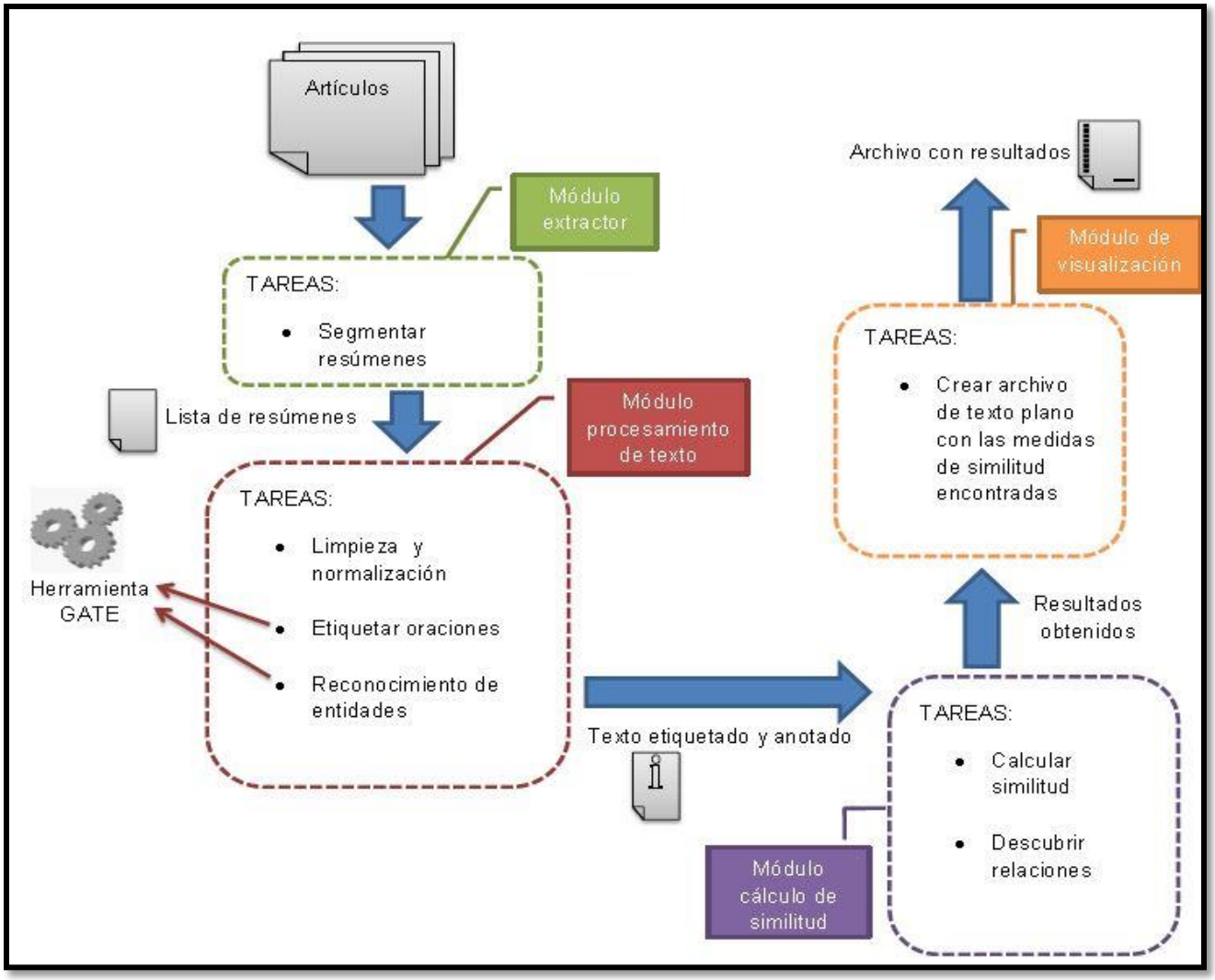


Figura 4. Arquitectura del sistema de mediciones de similitud semántica aplicados a resúmenes de artículos científicos

7.2. Módulo de procesamiento de texto

El módulo recibe de entrada los resúmenes de artículos científicos obtenidos por el módulo anterior para ser procesadas de la siguiente manera:

- Se aplica un algoritmo con expresiones regulares para remover palabras vacías y normalizar el corpus de resúmenes. Tenemos un ejemplo en la figura 5, en el recuadro superior se muestra el resumen extraído del artículo y en el recuadro inferior se encuentra el resumen limpio. El código fuente que realiza estas tareas se encuentran en el anexo E.

- Se utilizan las herramientas de GATE para hacer el etiquetado de oraciones y el reconocimiento de nombres de entidades. El código fuente que realiza estas tareas se encuentran en el anexo F.

- Las categorías de la clasificación de las palabras son: adjetivos, adverbios, números cardinales, conjunciones, determinantes, preposiciones, pronombres, sustantivos y verbos. El código fuente que realiza estas tareas se encuentran en el anexo G.
- Al final obtenemos de cada resumen una lista con todas las palabras clasificadas en cada una de las categorías mencionadas anteriormente.

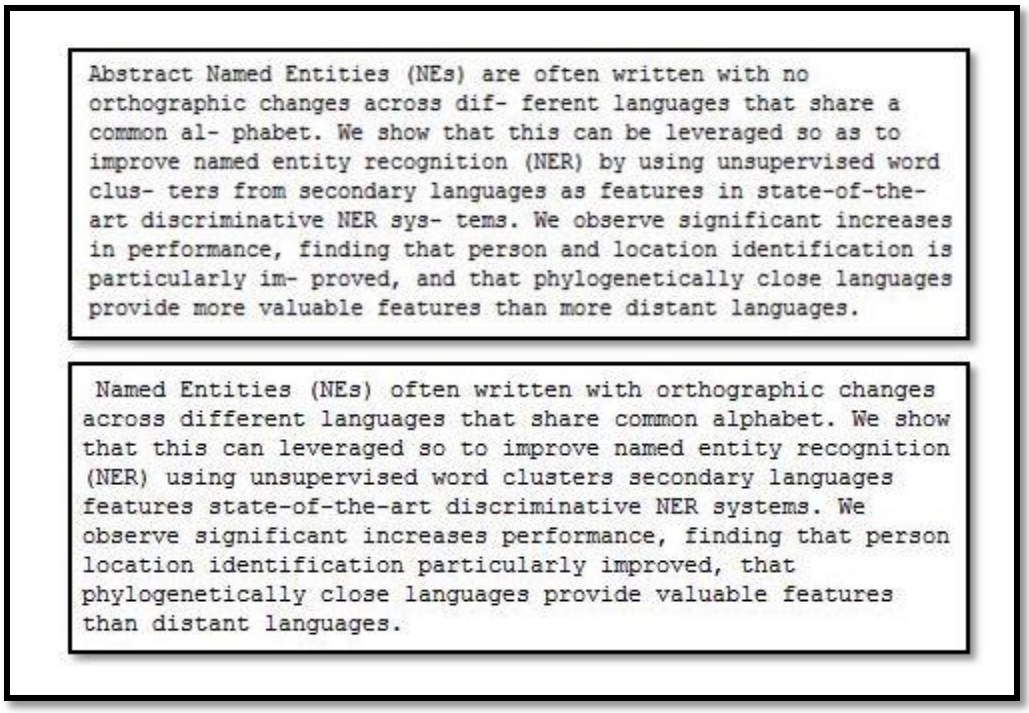


Figura 5 Comparación de archivos aplicando normalización

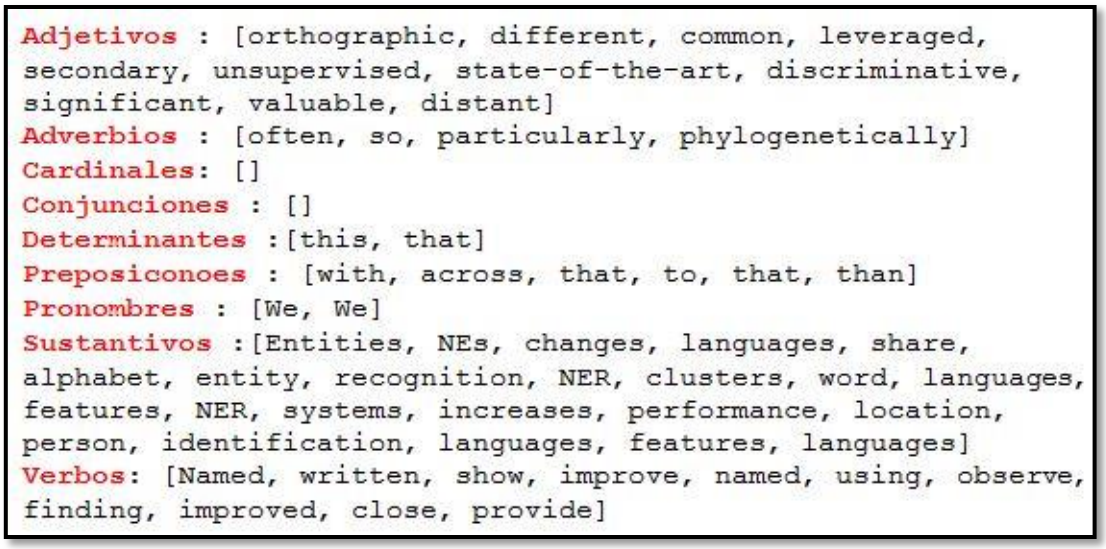


Figura 6. Salida del módulo de procesamiento de texto

En la figura 6 se muestra un ejemplo de la salida de éste módulo el cual consiste en poner cada palabra en la categoría que le corresponde.

7.3. Módulo de cálculo de similitud

En éste módulo se recibe la lista de resúmenes etiquetados por categorías y se aplica el algoritmo de “Bolsa de Palabras” para hacer el cálculo de similitud, se utilizaron cuatro de la API Java SimMetrics distancias (Coseno, Jaro-Winkler, Levenshtein y Monge-Elkan) y otras dos distancias de la API WS4J (Lin y Wu-Palmer).

En la figura 7 podemos ver que se compara la categoría X del resumen 1 contra la categoría X del resumen 2, ambas categorías deben ser la misma (adjetivos con adjetivos, verbos con verbos, etc.), donde:

- X es igual a alguna de las nueve categorías indicadas anteriormente: adjetivos, adverbios, números cardinales, conjunciones, determinantes, preposiciones, pronombres, sustantivos y verbos.
- $\text{Elemento}_{i,j}$ es igual a String i del resumen j.

Se comparan todos los Stings de la categoría del resumen 1 contra todos los Strings de la categoría del resumen 2.

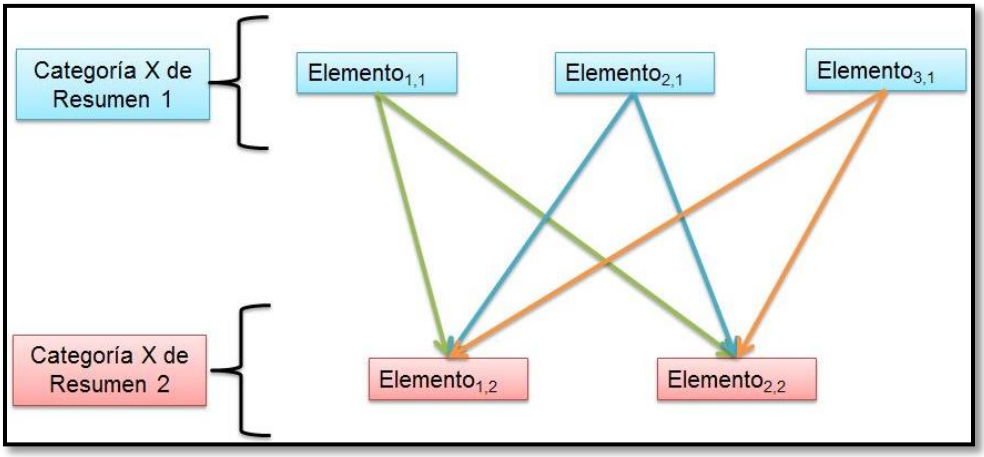


Figura 7. Comparación entre categorías

La comparación entre los elementos se hace por medio de las distancias implementadas en API Java SimMetrics y API WS4J indicadas anteriormente. El resultado de la comparación es un número entre 0 y 1, donde 0 indica los strings son diferentes y 1 indica que son equivalentes.

En el ejemplo de la figura 8 observamos que el resumen 1 tiene cuatro adverbios que se comparan contra cinco adverbios que contiene el resumen 2 obteniendo las distancias entre los strings.

Se obtiene el valor máximo de comparación por cada elemento del resumen 1. Como en la ecuación 8.

$$MaxElemento = maximo\left(distancia\left(Elemento_{i,j}, Elemento_{i,j+1}\right), distancia\left(Elemento_{i,j}, Elemento_{i+1,j+1}\right)\right) \quad (8)$$

Para obtener la similitud por categoría se promedian los máximos de cada string del resumen 1. Este proceso se hace para cada categoría y se observa en la ecuación 9.

$$SimilitudCategoria = \frac{(MaxElemento1 + MaxElemento2 + \cdots + MaxElementoN)}{N} \quad (9)$$

Figura 8. Ejemplo de comparación de adverbios con algoritmo Lin

Cuando se tienen las similitudes de cada categoría (adjetivos, adverbios, números cardinales, conjunciones, determinantes, preposiciones, pronombres, sustantivos y verbos) se procede a calcular la similitud final.

Para obtener la similitud final simplemente se saca el promedio de las similitudes por categoría como se ve en la ecuación 10.

$$Similitud = \frac{SimilitudCategoria1 + SimilitudCategoria2 + \cdots + SimilitudCateoriaN}{N} \quad (10)$$

El código fuente que realiza lo descrito anteriormente se encuentra en el anexo H.

7.4. Módulo de visualización

Éste módulo consiste en escribir un archivo de texto plano con los resultados obtenidos de cada distancia.

El resultado gold standard [12] consiste en una puntuación entre 0 y 5 por cada par de resúmenes que se interpretan de la siguiente forma:

- (5) Los dos resúmenes son completamente equivalentes y significan lo mismo.
- (4) Los dos resúmenes son en su mayoría equivalentes, solo difieren en algunos detalles sin relevancia.
- (3) Los dos resúmenes son aproximadamente equivalentes, falta o difiere alguna información importante.
- (2) Los dos resúmenes no son equivalentes, pero comparten algunos detalles.
- (1) Los dos resúmenes no son equivalentes, pero hablan del mismo tema.
- (0) Los dos resúmenes son de diferentes temas.

	Coseno	JaroW	Levenshtein	MongeElkan	WuPalmer	Lin
1 VS 2	0.8	2.9	2.0	2.7	1.4	1.1
1 VS 3	1.3	3.1	2.4	2.9	1.9	1.7
2 VS 3	2.2	3.9	3.2	3.6	3.1	2.8

Figura 9. Resultados de comparación con 6 distancias distintas

En la figura 9 observamos que con la distancia Lin interpretamos los resultados de la siguiente forma:

- Documento 1 vs Documento 2 tienen una medida de similitud de 1.1, esto quiere decir que los dos resúmenes no son equivalentes, pero hablan del mismo tema.
- Documento 1 vs Documento 3 tienen una medida de similitud de 1.7, esto quiere decir que los dos resúmenes no son equivalentes, pero comparten algunos detalles
- Documento 2 vs Documento 3 tienen una medida de similitud de 2.8, esto quiere decir que los dos resúmenes son aproximadamente equivalentes, falta o difiere alguna información importante.

Los resúmenes de los documentos 1, 2 y 3 se encuentran en las figuras 10, 11 y 12 respectivamente.

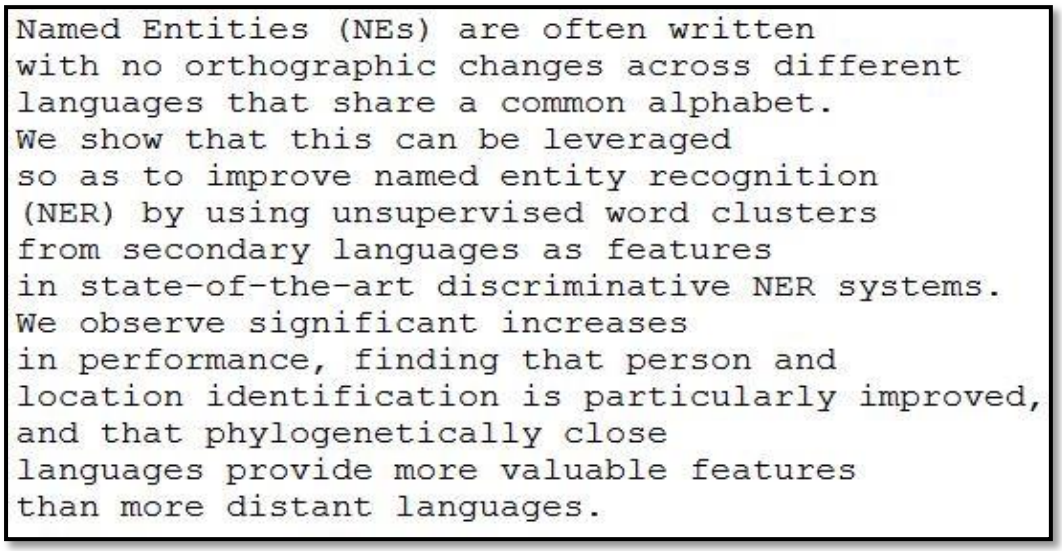


Figura 10. Resumen del documento 1

This paper describes an architecture that combines the complementary strengths of declarative programming and probabilistic graphical models to enable robots to represent, reason with, and learn from, qualitative and quantitative descriptions of uncertainty and knowledge. An action language is used for the low-level (LL) and high-level (HL) system descriptions in the architecture, and the definition of recorded histories in the HL is expanded to allow prioritized defaults. For any given goal, tentative plans created in the HL using default knowledge and commonsense reasoning are implemented in the LL using probabilistic algorithms, with the corresponding observations used to update the HL history. Tight coupling between the two levels enables automatic selection of relevant variables and generation of suitable action policies in the LL for each HL action, and supports reasoning with violation of defaults, noisy observations and unreliable actions in large and complex domains. The architecture is evaluated in simulation and on physical robots transporting objects in indoor domains; the benefit on robots is a reduction in task execution time of 39% compared with a purely probabilistic, but still hierarchical, approach.

Figura 11. Resumen del documento 2

Heart rate monitoring using wrist-type photoplethysmographic (PPG) signals during subjects' intensive exercise is a difficult problem, since the signals are contaminated by extremely strong motion artifacts caused by subjects' hand movements. So far few works have studied this problem. In this work, a general framework, termed TROIKA, is proposed, which consists of signal decomposiTion for denoising, sparse signal RecOnstructIon for high-resolution spectrum estimation, and spectral peaK trAcking with verification. The TROIKA framework has high estimation accuracy and is robust to strong motion artifacts. Many variants can be straightforwardly derived from this framework. Experimental results on datasets recorded from 12 subjects during fast running at the peak speed of 15 km/hour showed that the average absolute error of heart rate estimation was 2.34 beat per minute (BPM), and the Pearson correlation between the estimates and the ground-truth of heart rate was 0.992. This framework is of great values to wearable devices such as smart-watches which use PPG signals to monitor heart rate for fitness.

Figura 12. Resumen del documento 3

8. Evaluación

La forma en que se evaluó el proyecto fue mediante los resultados obtenidos por el International Workshop on Semantic Evaluation (SemEval-2014), ya que la temática es la similitud entre oraciones y es muy parecido a objetivo del proyecto.

Se modificó el módulo extractor y el módulo de visualización para que el sistema programado pudiera funcionar como se requería en el SemEval-2014. Las especificaciones de la competencia son las siguientes:

Dadas dos frases del texto, S1 y S2, el sistema debe calcular cuán similares son S1 y S2, devolviendo una puntuación de similitud. Se usará una escala de 0 (ninguna relación) a 5 (equivalentes).

El archivo de entrada consiste en varios pares de oraciones separadas por tabuladores como se ve en la figura 13.

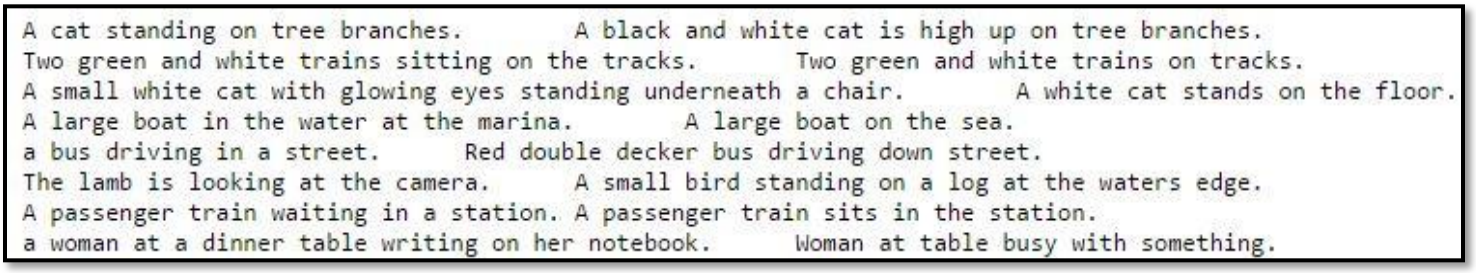


Figura 13. Fragmento de un archivo de entrada para SemEval-2014

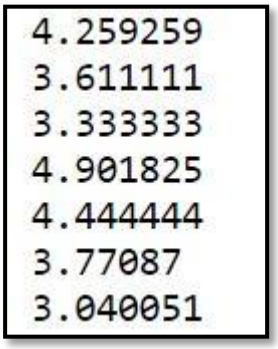


Figura 14. Ejemplo de un archivo de salida gold standard

El formato de salida es un archivo txt con un número entre 0 y 5 por cada par de frases de texto.

Para la evaluación se proporcionan seis archivos de entrada con los datos a procesar, los archivos se clasifican por:

- image description (image)
- OntoNotes and WordNet sense definition mappings (OnWN)
- news title and tweet comments (tweet-news)
- deft discussion forum and news (deft-forum)
- deft discussion forum and news (deft-news)
- news headlines (headlines)

Estos seis archivos se consideran como nuestro gold standard, lo cual significa que son los resultados ideales para cada par de textos. Estos archivos contienen los siguientes pares de textos:

- image contiene 750 pares
- OnWN contiene 750 pares
- tweet-news contiene 750 pares
- deft-forum contiene 450 pares
- deft-news contiene 300 pares
- headlines contiene 750 pares

Se aplicaron las distancias de similitud descritas anteriormente a cada archivo de texto para determinar cuál distancia es la mejor.

La evaluación de los archivos de salida se hizo mediante un script escrito en lenguaje perl que se encuentra para descargar en la página de la competencia [16].

9. Resultados

En la tabla 1 se muestran los resultados obtenidos con cada distancia de similitud aplicada, las partes marcadas con amarillo corresponden a los mejores resultados y los resultados marcados con rojo corresponden a la mejor distancia de similitud que en este caso fue Lin. La segunda mejor distancia fue Wu-Palmer y la tercera fue Coseno.

	deft-forum	deft-news	headlines	images	OnWN	tweet-news	Promedio
Coseno	0.4696	0.6521	0.5341	0.4371	0.3735	0.7493	0.53595
Jaro	0.4677	0.5924	0.4907	0.3821	0.3315	0.7361	0.500083
Leven	0.4451	0.5719	0.5217	0.4256	0.3528	0.746	0.510517
Lin	**0.4768**	**0.7636**	**0.6528**	**0.5382**	**0.5801**	**0.7801**	**0.631933**
Monge	0.4678	0.5454	0.5107	0.3992	0.3368	0.7192	0.496517
Wu	0.4691	0.637	0.5358	0.4745	0.546	0.7729	0.57255

Tabla 1. Resultados obtenidos para SemEval-2014

Adicionalmente se hizo una comparación contra todos los equipos que compitieron en SemEval-2014 quedando en el lugar 24 de 38 equipos. En la tabla 2 se muestran los resultados del lugar 18 al 27.

Se eligió la distancia de similitud Lin para comprarla contra todos los equipos porque fue la distancia que tuvo mejor promedio de las seis distancias utilizadas por el sistema.

Analizando los resultados por archivos individuales se obtuvieron los siguientes lugares:

- deft-forum: Lugar 7 con 0.4768 de puntuación.
- deft-news: Lugar 7 con 0.7636 de puntuación.

- headlines: Lugar 20 con 0.6528 de puntuación.
- images: Lugar 31 con 0.5382 de puntuación.
- OnWN: Lugar 31 con 0.5801 de puntuación.
- tweet-news: Lugar 2 con 0.7801 de puntuación.

En la tabla 3 podemos ver que aplicando el algoritmo desarrollado con el archivo tweet-news se obtuvo el segundo lugar como se mencionó anteriormente.

Team Name	deft-forum	deft-news	headlines	images	OnWN	tweet-news	Weighted mean	Rank
UMCC_DLSI_SemSim-run2	0.4689	0.6622	0.6255	0.739	0.814	0.6536	0.6756	18
BUAP-EN-run1	0.4557	0.6855	0.6888	0.6966	0.6539	0.7706	0.6715	19
RTM-DCU-run1*	0.4341	0.6974	0.6199	0.6995	0.8058	0.6882	0.6706	20
NTNU-run1	0.4369	0.7138	0.7219	0.8	0.8348	0.4109	0.6631	21
UNAL-NLP-run2	0.3826	0.7305	0.7645	0.7706	0.8268	0.4028	0.6573	22
RTM-DCU-run2*	0.3965	0.6811	0.6125	0.6656	0.7992	0.6691	0.6513	23
Lin-Omar	**0.4768**	**0.7636**	**0.6528**	**0.5382**	**0.5801**	**0.7801**	**0.6319**	**24**
StanfordNLP-run1	0.3186	0.6347	0.6361	0.7583	0.6269	0.6685	0.627	25
StanfordNLP-run3	0.3423	0.6503	0.6021	0.754	0.6087	0.638	0.6137	26
StanfordNLP-run2	0.3035	0.6791	0.6208	0.7149	0.625	0.6362	0.6101	27

Tabla 2. Resultados de SemEval-2014

Team Name	tweet-news	Rank
NTNU-run2	0.7921	1
Lin-Omar	**0.7801**	**2**
Meerkat_Mafia-pairingWords	0.7793	3
BUAP-EN-run1	0.7706	4
Meerkat_Mafia-SuperSaiyan	0.7651	5

Tabla 3. Segundo lugar con archivo tweet-news.

10. Conclusiones

Los beneficios encontrados aplicando procesamiento de lenguaje natural al sistema desarrollado son los siguientes:

- Se identifican de forma rápida los artículos científicos que tiene mayor similitud y que pueden servirle al investigador como referencia bibliográfica en el trabajo que esté realizando.
- Se puede utilizar para detectar plagios en caso de que las similitudes encontradas sean demasiado alta.

Existen muchas distancias de similitud como se ha mostrado, particularmente en este proyecto se detectó que las distancias de similitud con mejores resultados fueron las que utilizan como base una taxonomía como los son Lin y Wu-Palmer sobre las distancias que se basan en métodos probabilísticos. En nuestro caso se utilizó una taxonomía basada en la base de datos de WordNet.

Como trabajo futuro se sugiere utilizar diferentes distancias de similitud a las utilizadas en este trabajo para encontrar la más precisa y mejorar los resultados.

Los resultados encontrados se pueden agregar a un modelo de datos como puede ser una ontología o una base de datos para hacer clasificación o alguna otra resolución de problemas. Esto se puede lograr adaptando éste sistema como un módulo de otro sistema más grande.

Apéndice A: Clase Principal

Desde aquí se llaman a las clases que procesan los textos, extraen anotaciones y presentan los resultados.

```
public class Principal {
  public static void main(String[] args) throws GateException, GateRuntimeException,
  ExceptionInInitializerError {
  try {
        ArrayList<String> resumenes = new ArrayList<>();
        PDFaTXT p = new PDFaTXT();
        String sDirectorioPDF = "words\\articulos\\";
        String sDirectorioABS = "words\\abs\\";
        String sDirectorioABSLimpio = "words\\limpio\\";
        File pdf = new File(sDirectorioPDF);
        extractor res = new extractor();
        stopWords sw = new stopWords();
        if (pdf.exists()) {
           File[] ficheros = pdf.listFiles();
           for (int x = 0; x < ficheros.length; x++) {
              p.main(sDirectorioPDF + ficheros[x].getName(), sDirectorioABS +
ficheros[x].getName().replace(".pdf", ".txt"));
           }
        }
        File f = new File(sDirectorioABS);
        File[] ficheros = f.listFiles();
        for (int x = 0; x < ficheros.length; x++) {
           res.extrae(sDirectorioABS + ficheros[x].getName(), ficheros[x].getName());
        }
        File f2 = new File(sDirectorioABS);
        File[] ficheros2 = f2.listFiles();
        for (int x = 0; x < ficheros2.length; x++) {
           sw.quitaStopWords(sDirectorioABS + ficheros2[x].getName(),
ficheros2[x].getName());
        }
        File f3 = new File(sDirectorioABSLimpio);
        File[] ficheros3 = f3.listFiles();
        int cont = 0;
        for (int x = 0; x < ficheros3.length; x++) {
           resumenes.add(sDirectorioABSLimpio + ficheros3[x].getName());
           cont++;
        }
        ArrayList<ArrayList<AnotacionBean>> Lista = new ArrayList<>();
```

```
            Lista = Corpus(resumenes);
            ArrayList<AnotacionBean> subLista = new ArrayList
            CategoriaBean clasificada = null;
            ArrayList<CategoriaBean> ListaClasificada = new ArrayList<>();
            for (int x = 0; x < cont; x++) {
                subLista = Lista.get(x);
                clasificada = clasificador.clasifica(subLista);
                ListaClasificada.add(clasificada);
            }
            compara(ListaClasificada);
            BorrarTxt(sDirectorioABS);
            BorrarTxt(sDirectorioABSLimpio);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Apéndice B: Clase PDFaTXT

Esta clase hace se encarga de convertir los artículos científicos en formato pdf a formato txt.

```
public class PDFaTXT {

    PDFParser parser;
    String parsedText;
    PDFTextStripper pdfStripper;
    PDDocument pdDoc;
    COSDocument cosDoc;
    PDDocumentInformation pdDocInfo;

    public PDFaTXT() {
    }
    String pdftoText(String nomArch) {
        System.out.println("Analizando archivo pdf " + nomArch + "....");
        File f = new File(nomArch);
        if (!f.isFile()) {
            System.out.println("Archivo " + nomArch + " no existe.");
            return null;
        }
```

```
        try {
            parser = new PDFParser(new FileInputStream(f));
        } catch (Exception e) {
            System.out.println("No se puede abrir PDF");
            return null;
        }
        try {
            parser.parse();
            cosDoc = parser.getDocument();
            pdfStripper = new PDFTextStripper();
            pdDoc = new PDDocument(cosDoc);
            parsedText = pdfStripper.getText(pdDoc);
        } catch (Exception e) {
            System.out.println("Una excepción ocurrió al analizar el documento PDF.");
            e.printStackTrace();
            try {
                if (cosDoc != null) {
                    cosDoc.close();
                }
                if (pdDoc != null) {
                    pdDoc.close();
                }
            } catch (Exception e1) {
                e.printStackTrace();
            }
            return null;
        }
        System.out.println("Hecho");
        return parsedText;
    }

        void writeTexttoFile(String pdfText, String nomArch) {
        System.out.println("\nEscribiendo de PDF a archivo txt" + nomArch + "....");
        try {
            PrintWriter pw = new PrintWriter(nomArch);
            pw.print(pdfText);
            pw.close();
        } catch (Exception e) {
            System.out.println("Una excepción se produjo en la escritura pdf.");
            e.printStackTrace();
        }
        System.out.println("Hecho");
    }
```

```
    public static String main(String pdffile, String textfile) {
        PDFaTXT pdfTextParserObj = new PDFaTXT();
        String pdfToText = pdfTextParserObj.pdftoText(pdffile);
        if (pdfToText == null) {
            System.out.println("Conversión de PDF a txt fallida.");
        } else {
            System.out.println("\n Analizando archivo pdf...\n" + pdfToText);
            pdfTextParserObj.writeTexttoFile(pdfToText, textfile);
        }
        return pdfToText;
    }
}
```

Apéndice C: Clase extractor

Esta clase busca y extrae el resumen de cada artículo científico.

```
public class extractor {

    void extrae(String ruta, String nombre) {

        BufferedReader entrada = null;
        String texto = "";
        String textoAux = "";
        String rutaNueva = "words\\abs\\";
        Matcher m = null;
        int bandera = 0;
        marcador v = new marcador();
        Pattern patronBusqueda1 = Pattern.compile("^Abstract");
        try {
            File archivo = new File(ruta);
            FileReader fr = new FileReader(archivo);
            entrada = new BufferedReader(fr);
            while ((textoAux = entrada.readLine()) != null) {
                if (bandera == 0) {
                    m = patronBusqueda1.matcher(textoAux);
                    if (m.find()) {
                        bandera = 1;
                        //texto = texto + textoAux + "\r\n" ;
                        texto = texto + textoAux + " ";
                    }
```

```
            } else {
                if (!v.validaMarca(textoAux)) //  texto = texto + textoAux + "\r\n";
                {
                    texto = texto + textoAux + " ";
                } else {
                    entrada.close();
                }
            }
        }
        entrada.close();
        bandera = 0;
    } catch (Exception e) {
    }
    File archivo2 = new File(rutaNueva + nombre);
    try {
        FileWriter w = new FileWriter(archivo2);
        BufferedWriter bw = new BufferedWriter(w);
        PrintWriter wr = new PrintWriter(bw);
        wr.write(texto);
        wr.close();
        bw.close();
    } catch (Exception e) {

    }
  }
}
```

Apéndice D: Clase marcador

Esta clase encuentra el fin de un resumen para extraerlo.

```
public class marcador {

   boolean validaMarca(String linea) {
      Matcher m = null;
      Pattern patronBusqueda1 = Pattern.compile("Introduction\\s*$");
      Pattern patronBusqueda2 = Pattern.compile("INTRODUCTION\\s*$");
      Pattern patronBusqueda3 = Pattern.compile("^Index Terms");
      Pattern patronBusqueda4 = Pattern.compile("^Keywords:");
      m = patronBusqueda1.matcher(linea);
```

```
        if (m.find()) {
            return true;
        } else {
            m = patronBusqueda2.matcher(linea);
            if (m.find()) {
                return true;
            } else {
                m = patronBusqueda3.matcher(linea);
                if (m.find()) {
                    return true;
                } else {
                    m = patronBusqueda4.matcher(linea);
                    if (m.find()) {
                        return true;
                    } else {
                        return false;
                    }
                }
            }
        }
    }
}
```

Apéndice E: Clase stopWords

Clase que remueve palabras vacías y da formato al texto (quitar dobles espacios).

```
public class stopWords {

    void quitaStopWords(String ruta, String nombre) {
        BufferedReader entradaListaStopWords = null;
        BufferedReader entradaAbstract = null;
        String linea = "";
        String lineaAux = "";
        String rutaListaStopWords = "words\\stopWords\\stop.txt";
        // String rutaAbstract ="C:\\Users\\Eduardo\\Desktop\\words\\";
        String rutaNueva = "words\\limpio\\";
        ArrayList<String> stopWords = new ArrayList<String>();
        try {
            int i = 0;
```

```
            File archivo = new File(rutaListaStopWords);
            FileReader fr = new FileReader(archivo);
            entradaListaStopWords = new BufferedReader(fr);
            File archivo2 = new File(ruta);
            FileReader fr2 = new FileReader(archivo2);
            entradaAbstract = new BufferedReader(fr2);
            while ((lineaAux = entradaListaStopWords.readLine()) != null)
            {
               stopWords.add(i, lineaAux);
               i++;
            }
            entradaListaStopWords.close();
            lineaAux = "";
            while ((lineaAux = entradaAbstract.readLine()) != null)
            {
               linea = linea + lineaAux + "\r\n";
            }
            entradaAbstract.close();
            i = 0;
            while (stopWords.get(i) != null) {
               linea = linea.replaceAll("\\s" + stopWords.get(i) + "\\s", " ");
               linea = linea.replaceAll("\\. " + stopWords.get(i) + "\\s", " ");
               linea = linea.replaceAll("\\s" + stopWords.get(i) + "\\.", ".");
               linea = linea.replaceAll("\\s" + stopWords.get(i) + ",", " ");
               i++;
            }
         } catch (Exception e) {
         }
         linea = linea.replaceFirst("Abstract –", " ");
         linea = linea.replaceFirst("Abstract", " ");
         linea = linea.replaceAll("-\\s", "");
         linea = linea.replaceAll(" {2,}", " ");
         File archivo2 = new File(rutaNueva + nombre);
         try {
            FileWriter w = new FileWriter(archivo2);
            BufferedWriter bw = new BufferedWriter(w);
            PrintWriter wr = new PrintWriter(bw);
            wr.write(linea);
            wr.close();
            bw.close();
         } catch (Exception e) {
         }
      }
```

```
    static void BorrarTxt(String direccion) {
        ArrayList<String> txt = new ArrayList<>();
        File f = new File(direccion);
        File[] nomFicheros = f.listFiles();
        for (int x = 0; x < nomFicheros.length; x++) {
            txt.add(direccion + nomFicheros[x].getName());
            File f2 = new File(txt.get(x));
            f2.delete();
        }
    }
}
```

Apéndice F: Clase PluginGate

Esta clase inicializa GATE, y recupera anotaciones de los textos del corpus.

```
public class PluginGate {

    public static ArrayList<ArrayList<AnotacionBean>>
Corpus(ArrayList<String> url) throws GateException, GateRuntimeException,
ExceptionInInitializerError {
        ArrayList<ArrayList<AnotacionBean>> ListaDeListas = new ArrayList<>();
        try {
            PluginGate prin = new PluginGate();
            //Inicialización
            File gateHome = new File("gate-7.1");
            Gate.setGateHome(gateHome);
            Gate.init();
            Gate.getCreoleRegister().registerDirectories(new File("gate-
7.1/plugins/Tools").toURI().toURL());
            Corpus myCorpus = Factory.newCorpus("Abstract Corpus");
            int x = 0;
            ArrayList<Document> doc = new ArrayList();
            for (String documento : url) {
                doc.add(Factory.newDocument(new File(documento).toURI().toURL(), null));
                myCorpus.add(doc.get(x));
                System.out.println(myCorpus.getDocumentName(x) + " AGREGADO AL
CORPUS \n");
                x++;
            }
```

```
        String[] processingResources = {"gate.creole.annotdelete.AnnotationDeletePR",
            "gate.creole.tokeniser.SimpleTokeniser",
            "gate.creole.splitter.SentenceSplitter",
            "gate.creole.POSTagger",
            "gate.creole.gazetteer.DefaultGazetteer",
            "gate.creole.ANNIETransducer"};
          prin.runProcessingResources(processingResources, myCorpus);
          for (int y = 0; y < url.size(); y++) {
             AnnotationSetImpl ann = (AnnotationSetImpl) doc.get(y).getAnnotations();
             Iterator<Annotation> i = ann.iterator();
             ArrayList<AnotacionBean> DocumentoAnotado = new ArrayList<>();
             while (i.hasNext()) {
                Annotation annotation = i.next();
                String annotType = annotation.getType();
                if (annotType.contentEquals("Token")) {
                   AnotacionBean bean = new AnotacionBean();
                   bean.setCategoria(annotation.getFeatures().get("category").toString());
                   bean.setCadena(annotation.getFeatures().get("string").toString());
                   DocumentoAnotado.add(bean);
                }
             }
             ListaDeListas.add(y, DocumentoAnotado);
          }
          System.out.println("PROCESADO CORRECTAMENTE \n");
       } catch (Exception e) {
          e.printStackTrace();
       }
       return ListaDeListas;
    }
    private void runProcessingResources(String[] processingResource, Corpus corpus)
throws GateException {
       SerialAnalyserController pipeline = (SerialAnalyserController)
Factory.createResource("gate.creole.SerialAnalyserController");
       for (int pr = 0; pr < processingResource.length; pr++) {
          System.out.print("\t* Cargando " + processingResource[pr] + " ... ");
          pipeline.add((gate.LanguageAnalyser)
Factory.createResource(processingResource[pr]));
          System.out.println("Listo");
       }
       System.out.print("Creando corpus de documentos...");
       pipeline.setCorpus(corpus);
       System.out.println("Listo");
       System.out.print("Ejecutando recursos sobre el corpus...");
       pipeline.execute();
```

```
        System.out.println("Listo");
    }
}
```

Apéndice G: Clase clasificador

En esta clase se clasifican las palabras por categorías gramaticales.

```
public class clasificador {

    public static CategoriaBean clasifica(ArrayList<AnotacionBean> listaDePalabras) {
        CategoriaBean listaClasificada = new CategoriaBean();
        ArrayList<String> verbos = new ArrayList();
        ArrayList<String> adjetivos = new ArrayList();
        ArrayList<String> sustantivos = new ArrayList();
        ArrayList<String> conjunciones = new ArrayList();
        ArrayList<String> cardinales = new ArrayList();
        ArrayList<String> determinantes = new ArrayList();
        ArrayList<String> preposiciones = new ArrayList();
        ArrayList<String> pronombres = new ArrayList();
        ArrayList<String> adverbios = new ArrayList();

        for (AnotacionBean palabra : listaDePalabras) {
            if (palabra.getCategoria().startsWith("V")) {
                verbos.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("J")) {
                adjetivos.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("N")) {
                sustantivos.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("CC")) {
                conjunciones.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("CD")) {
                cardinales.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("DT")) {
                determinantes.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("IN") ||
palabra.getCategoria().startsWith("TO")) {
                preposiciones.add(palabra.getCadena());
            } else if (palabra.getCategoria().startsWith("P")) {
                pronombres.add(palabra.getCadena());
```

```
            } else if (palabra.getCategoria().startsWith("R")) {
                adverbios.add(palabra.getCadena());
            }
        }

        listaClasificada.setAdjetivos(adjetivos);
        listaClasificada.setAdverbios(adverbios);
        listaClasificada.setCardinales(cardinales);
        listaClasificada.setConjunciones(conjunciones);
        listaClasificada.setDeterminantes(determinantes);
        listaClasificada.setPreposiciones(preposiciones);
        listaClasificada.setPronombres(pronombres);
        listaClasificada.setSustantivos(sustantivos);
        listaClasificada.setVerbos(verbos);

        return listaClasificada;
    }

}
```

Apéndice H: Clase comparador

Esta clase realiza las comparaciones de resúmenes utilizando las seis distancias: Coseno, Jaro-Winkler, Levenshtein, Lin, Monge-Elkan y Wu-Palmer.

```
public class comparador {

    static void compara(ArrayList<CategoriaBean> Lista) {
        int contador = 1, j;
        ArrayList<Double> resultado = new ArrayList();
        ResultadoBean salida;
        ArrayList<ResultadoBean> ArraySalida = new ArrayList<>();
        for (int i = 0; i < Lista.size(); i++) {
            for (j = contador; j < Lista.size(); j++) {
                resultado = calcula(Lista.get(i), Lista.get(j));
                salida = new ResultadoBean();
                salida.setNombreDoc1("" + (i + 1));
                salida.setNombreDoc2("" + (j + 1));
                salida.setMetrica1(resultado.get(0));
                salida.setMetrica2(resultado.get(1));
```

```
                salida.setMetrica3(resultado.get(2));
                salida.setMetrica4(resultado.get(3));
                salida.setMetrica5(resultado.get(4));
                salida.setMetrica6(resultado.get(5));
                ArraySalida.add(salida);
            }
            contador++;
            if (i == Lista.size() - 1) {
                break;
            }
        }
       File archivoNuevo = new File("words\\resultado\\RESULTADO.txt");
        try {
            FileWriter w = new FileWriter(archivoNuevo);
            BufferedWriter bw = new BufferedWriter(w);
            PrintWriter wr = new PrintWriter(bw);
            wr.write("\t\tCoseno\t\tJaroW\t   Levenshtein\t   MongeElkan\t   WuPalmer\t
Lin\n");
            for (ResultadoBean item : ArraySalida) {
                wr.write(item.getNombreDoc1() + " VS " + item.getNombreDoc2()
                    + "\t\t " + item.getMetrica1()
                    + "\t\t " + item.getMetrica2()
                    + "\t\t" +  item.getMetrica3()
                    + "\t\t" + item.getMetrica4()
                    + "\t\t" + item.getMetrica5()
                    + "\t\t" + item.getMetrica6()+"\n");
              similitud de: "+item.getSimilitud());
            }
            wr.close();
            bw.close();
        } catch (Exception e) {
        }
    }

    static ArrayList<Double> calcula(CategoriaBean bean1, CategoriaBean bean2) {
        //System.out.println("Bean1:  " + bean1.getConjunciones());
        //System.out.println("Bean2:  " + bean2.getConjunciones());
        ArrayList<Double> res = new ArrayList();
        ArrayList<Double> Adjetivos = resultado(bean1.getAdjetivos(),
bean2.getAdjetivos());
        ArrayList<Double> Adverbios = resultado(bean1.getAdverbios(),
bean2.getAdverbios());
        ArrayList<Double> Cardinales = resultado(bean1.getCardinales(),
bean2.getCardinales());
```

```
        ArrayList<Double> Conjunciones = resultado(bean1.getConjunciones(),
bean2.getConjunciones());
        ArrayList<Double> Determinantes = resultado(bean1.getDeterminantes(),
bean2.getDeterminantes());
        ArrayList<Double> Preposiciones = resultado(bean1.getPreposiciones(),
bean2.getPreposiciones());
        ArrayList<Double> Pronombres = resultado(bean1.getPronombres(),
bean2.getPronombres());
        ArrayList<Double> Sustantivos = resultado(bean1.getSustantivos(),
bean2.getSustantivos());
        ArrayList<Double> Verbos = resultado(bean1.getVerbos(), bean2.getVerbos());
        res.add(Redondea(((Adjetivos.get(0) + Adverbios.get(0) + Cardinales.get(0) +
Conjunciones.get(0) + Determinantes.get(0) + Preposiciones.get(0) + Pronombres.get(0) +
Sustantivos.get(0) + Verbos.get(0)) / 9) * 5));
        res.add(Redondea(((Adjetivos.get(1) + Adverbios.get(1) + Cardinales.get(1) +
Conjunciones.get(1) + Determinantes.get(1) + Preposiciones.get(1) + Pronombres.get(1) +
Sustantivos.get(1) + Verbos.get(1)) / 9) * 5));
        res.add(Redondea(((Adjetivos.get(2) + Adverbios.get(2) + Cardinales.get(2) +
Conjunciones.get(2) + Determinantes.get(2) + Preposiciones.get(2) + Pronombres.get(2) +
Sustantivos.get(2) + Verbos.get(2)) / 9) * 5));
        res.add(Redondea(((Adjetivos.get(3) + Adverbios.get(3) + Cardinales.get(3) +
Conjunciones.get(3) + Determinantes.get(3) + Preposiciones.get(3) + Pronombres.get(3) +
Sustantivos.get(3) + Verbos.get(3)) / 9) * 5));
        res.add(Redondea(((Adjetivos.get(4) + Adverbios.get(4) + Cardinales.get(4) +
Conjunciones.get(4) + Determinantes.get(4) + Preposiciones.get(4) + Pronombres.get(4) +
Sustantivos.get(4) + Verbos.get(4)) / 9) * 5));
        res.add(Redondea(((Adjetivos.get(5) + Adverbios.get(5) + Cardinales.get(5) +
Conjunciones.get(5) + Determinantes.get(5) + Preposiciones.get(5) + Pronombres.get(5) +
Sustantivos.get(5) + Verbos.get(5)) / 9) * 5));
        return res;
    }

    static ArrayList<Double> resultado(ArrayList<String> array1, ArrayList<String> array2) {
        double suma1 = 0;
        double suma2 = 0;
        double suma3 = 0;
        double suma4 = 0;
        double suma5 = 0;
        double suma6 = 0;
        double resultado1 = 0;
        double resultado2 = 0;
        double resultado3 = 0;
        double resultado4 = 0;
        double resultado5 = 0;
```

```
double resultado6 = 0;
ArrayList<Double> valorSimilitud1 = new ArrayList();
ArrayList<Double> valorSimilitud2 = new ArrayList();
ArrayList<Double> valorSimilitud3 = new ArrayList();
ArrayList<Double> valorSimilitud4 = new ArrayList();
ArrayList<Double> valorSimilitud5 = new ArrayList();
ArrayList<Double> valorSimilitud6 = new ArrayList();
ArrayList<Double> valorSimilitudFinal1 = new ArrayList();
ArrayList<Double> valorSimilitudFinal2 = new ArrayList();
ArrayList<Double> valorSimilitudFinal3 = new ArrayList();
ArrayList<Double> valorSimilitudFinal4 = new ArrayList();
ArrayList<Double> valorSimilitudFinal5 = new ArrayList();
ArrayList<Double> valorSimilitudFinal6 = new ArrayList();
ArrayList<Double> ResultadosFinales = new ArrayList();
if (array1.isEmpty() && array2.isEmpty()) {
    resultado1 = 1;
    resultado2 = 1;
    resultado3 = 1;
    resultado4 = 1;
    resultado5 = 1;
    resultado6 = 1;
    ResultadosFinales.add(resultado1);
    ResultadosFinales.add(resultado2);
    ResultadosFinales.add(resultado3);
    ResultadosFinales.add(resultado4);
    ResultadosFinales.add(resultado5);
    ResultadosFinales.add(resultado6);
} else if (array1.isEmpty() && !array2.isEmpty()) {
    resultado1 = 0;
    resultado2 = 0;
    resultado3 = 0;
    resultado4 = 0;
    resultado5 = 0;
    resultado6 = 0;
    ResultadosFinales.add(resultado1);
    ResultadosFinales.add(resultado2);
    ResultadosFinales.add(resultado3);
    ResultadosFinales.add(resultado4);
    ResultadosFinales.add(resultado5);
    ResultadosFinales.add(resultado6);
} else {
    for (int i = 0; i < array1.size(); i++) {
        for (int j = 0; j < array2.size(); j++) {
            double result1 = metricaCoseno(array1.get(i), array2.get(j));
```

```
                double result2 = metricaJaroW(array1.get(i), array2.get(j));
                double result3 = metricaLevenshtein(array1.get(i), array2.get(j));
                double result4 = metricaMongeElkan(array1.get(i), array2.get(j));
                double result5 = metricaWuPalmer(array1.get(i), array2.get(j));
                double result6 = metricaLin(array1.get(i), array2.get(j));
                //System.out.println("Resultado de metrica:"+array1.get(i)+" con
"+array2.get(j)+ " = "+ result);
                valorSimilitud1.add(result1);
                valorSimilitud2.add(result2);
                valorSimilitud3.add(result3);
                valorSimilitud4.add(result4);
                valorSimilitud5.add(result5);
                valorSimilitud6.add(result6);
            }
            valorSimilitudFinal1.add(maximo(valorSimilitud1));
            valorSimilitudFinal2.add(maximo(valorSimilitud2));
            valorSimilitudFinal3.add(maximo(valorSimilitud3));
            valorSimilitudFinal4.add(maximo(valorSimilitud4));
            valorSimilitudFinal5.add(maximo(valorSimilitud5));
            valorSimilitudFinal6.add(maximo(valorSimilitud6));
            valorSimilitud1.clear();
            valorSimilitud2.clear();
            valorSimilitud3.clear();
            valorSimilitud4.clear();
            valorSimilitud5.clear();
            valorSimilitud6.clear();
        }
        for (int i = 0; i < valorSimilitudFinal1.size(); i++) {
            suma1 = suma1 + valorSimilitudFinal1.get(i);
            suma2 = suma2 + valorSimilitudFinal2.get(i);
            suma3 = suma3 + valorSimilitudFinal3.get(i);
            suma4 = suma4 + valorSimilitudFinal4.get(i);
            suma5 = suma5 + valorSimilitudFinal5.get(i);
            suma6 = suma6 + valorSimilitudFinal6.get(i);
        }
        resultado1 = suma1 / valorSimilitudFinal1.size();
        resultado2 = suma2 / valorSimilitudFinal2.size();
        resultado3 = suma3 / valorSimilitudFinal3.size();
        resultado4 = suma4 / valorSimilitudFinal4.size();
        resultado5 = suma5 / valorSimilitudFinal5.size();
        resultado6 = suma6 / valorSimilitudFinal6.size();
        ResultadosFinales.add(resultado1);
        ResultadosFinales.add(resultado2);
        ResultadosFinales.add(resultado3);
```

```
            ResultadosFinales.add(resultado4);
            ResultadosFinales.add(resultado5);
            ResultadosFinales.add(resultado6);
        }
        return ResultadosFinales;
    }

    static double maximo(ArrayList<Double> array) {
        double maximo = 0;
        if (!array.isEmpty()) {
            maximo = array.get(0);
            for (int i = 0; i < array.size(); i++) {
                if (array.get(i) > maximo) {
                    maximo = array.get(i);
                }
            }
            // System.out.println("El máximo es : "+ maximo);
        }
        return maximo;
    }

    static double metricaCoseno(String str1, String str2) {
        AbstractStringMetric metrica = new CosineSimilarity();
        double result = metrica.getSimilarity(str1, str2);
        return result;
    }

    static double metricaJaroW(String str1, String str2) {
        AbstractStringMetric metrica = new JaroWinkler();
        double result = metrica.getSimilarity(str1, str2);
        return result;
    }

    static double metricaLevenshtein(String str1, String str2) {
        AbstractStringMetric metrica = new Levenshtein();
        double result = metrica.getSimilarity(str1, str2);
        return result;
    }

    static double metricaMongeElkan(String str1, String str2) {
        AbstractStringMetric metrica = new MongeElkan();
        double result = metrica.getSimilarity(str1, str2);
        return result;
    }
```

```
    static double metricaLin(String str1, String str2) {
        double result;
        if (str1.equals(str2)) {
            result = 1.0;

        } else {
            ILexicalDatabase db = new NictWordNet();
            WS4JConfiguration.getInstance().setMFS(true);
            RelatednessCalculator rc = new Lin(db);
            List<POS[]> Lista = rc.getPOSPairs();
            result = -1.0;
            for (POS[] posPair : Lista) {
                List<Concept> synsets1 = (List<Concept>) db.getAllConcepts(str1,
posPair[0].toString());
                List<Concept> synsets2 = (List<Concept>) db.getAllConcepts(str2,
posPair[1].toString());
                for (Concept synset1 : synsets1) {
                    for (Concept synset2 : synsets2) {
                        Relatedness relatedness = rc.calcRelatednessOfSynset(synset1, synset2);
                        double score = relatedness.getScore();
                        if (score > result) {
                            result = score;
                        }
                    }
                }
            }
            if (result == -1.0) {
                result = 0.0;
            } else if (result == -0.0) {
                result = 0.0;
            } else if (result < 0.0) {
                result = 0.0;
            } else if (result > 1.0) {
                result = 1.0;
            }
        }
        return result;
    }

    static double metricaWuPalmer(String str1, String str2) {
        double result;
        if (str1.equals(str2)) {
            result = 1.0;
```

```
        } else {
            ILexicalDatabase db = new NictWordNet();
            WS4JConfiguration.getInstance().setMFS(true);
            RelatednessCalculator rc = new WuPalmer(db);
            List<POS[]> Lista = rc.getPOSPairs();
            result = -1.0;
            for (POS[] posPair : Lista) {
                List<Concept> synsets1 = (List<Concept>) db.getAllConcepts(str1,
posPair[0].toString());
                List<Concept> synsets2 = (List<Concept>) db.getAllConcepts(str2,
posPair[1].toString());
                for (Concept synset1 : synsets1) {
                    for (Concept synset2 : synsets2) {
                        Relatedness relatedness = rc.calcRelatednessOfSynset(synset1, synset2);
                        double score = relatedness.getScore();
                        if (score > result) {
                            result = score;
                        }
                    }
                }
            }
            if (result == -1.0) {
                result = 0.0;
            } else if (result == -0.0) {
                result = 0.0;
            } else if (result < 0.0) {
                result = 0.0;
            } else if (result > 1.0) {
                result = 1.0;
            }
        }
        return result;
    }

    static double Redondea(double valor) {
        DecimalFormat df = new DecimalFormat("#.#");
        return Double.valueOf(df.format(valor));
    }

}
```

11. Referencias Bibliográficas

[1] S.M. Ugalde Chávez, “Sistema de recuperación de información semántico”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2012.

[2] R.I. Morán Torres, “Sistema de detección de plagio en archivos de texto”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2013.

[3] J.L. Ugalde Anaya, “Sistema clasificador de documentos de proyectos terminales usando el concepto de memoria asociativa”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2011.

[4] V.J. Sosa Sosa. (2014, Abril 3) Representación de Documentos y Cálculo de Similaridad. [En línea] Disponible en: http://www.tamps.cinvestav.mx/~vjsosa/clases/sd/RepresentacionDocumentos_CalculoSimilaridad.pdf

[5] A. Lloret Carbonell (2014, Junio) Herramienta basada en vistas para la generación automática de anotaciones de fuentes RDF. [En línea] Disponible en: http://oa.upm.es/30992/1/PFC_ANDONI_LLORET_CARBONELL.pdf

[6] A. Lucas Villaverde (2012, Junio 5) Evaluación Automática e Interactiva de Destrezas Mediante Distancias entre Cadenas. [En línea] Disponible en: https://riunet.upv.es/bitstream/handle/10251/16576/Memoria.pdf?sequence=1

[7] M. Martí, A. Fernández, G. Vázquez (2001) Lexicografía computacional y semántica. [En línea] Disponible en: http://books.google.com.mx/books?id=LISO5_o6RpgC&printsec=frontcover#v=onepage&q&f=false

[8] S. Fernández Melián (2013, Octubre) Contribución a la alineación de ontologías utilizando lógica difusa. [En línea] Disponible en: http://dspace.uah.es/dspace/bitstream/handle/10017/20149/Tesis%20Susel%20Fdez.pdf?sequence=1&isAllowed=y

[9] Apache PDFBox (2014, Noviembre) Apache PDFBox - A Java PDF Library. [En línea] Disponible en: https://pdfbox.apache.org/index.html

[10] GATE (2014, Noviembre) GATE Developer [En línea] Disponible en: https://gate.ac.uk/family/developer.html

[11] WS4J (2014, Noviembre) WS4J [En línea] Disponible en: https://code.google.com/p/ws4j/

[12] SemEval-2014 (2014, Noviembre) Semantic Textual Similarity for English [En línea] Disponible en: http://alt.qcri.org/semeval2014/task10/index.php?id=sts-en