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

Clasificación emocional de noticias en español utilizando algoritmos de aprendizaje automático

Proyecto de Investigación

Trimestre 2018 Otoño

Ivan González Villarruel
210302618
van91.gv@gmail.com

Asesor:

José Alejandro Reyes Ortiz
Doctor en Ciencias de la Computación
Profesor Titular
Departamento de Sistemas
jaro@correo.azc.uam.mx

11 de enero de 2019

Declaratoria

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.

Dr. José Alejandro Reyes Ortiz

Yo, Ivan González Villarruel, 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.

Ivan González Villarruel

Resumen

El procesamiento de lenguaje natural es una rama de la inteligencia artificial la cual tiene como objetivo encontrar la manera más sencilla de comunicar máquinas con personas haciendo uso de lenguas naturales, como el inglés, francés, alemán, portugués o español. Las lenguas más utilizadas en el mundo digital, dentro de ellas el español, tienen una aplicación basado en el interés económico y práctico de las empresas e instituciones dedicadas al estudio del procesamiento de lenguaje natural.

Las lenguas naturales pueden expresarse de forma oral y escrita, está ultima tiene mayor avance en el tratamiento de textos ya que son más fáciles de manipular. Este tratamiento consiste en clasificar textos, agrupar contenidos, extraer información relevante o convertir textos en contenidos estructurados. En el caso de la clasificación de textos, un ejemplo es clasificar mensajes de redes sociales basado en la polaridad (positiva o negativa).

Una tarea que sea convertido en un gran reto para la comunidad científica es la clasificación de noticias, específicamente, identificar si una noticia genera un aspecto de seguridad o inseguridad en el lector. En este contexto, este proyecto se enfoca en clasificar títulos de noticias en dos categorías: segura y no segura. Para lo cual, se trabajará con textos en español, esto es con títulos de noticias con fuentes RSS de periódicos procedentes de los países Argentina, Chile, Colombia, Cuba, España, México, Perú y Venezuela aplicando técnicas de procesamiento de lenguaje natural y minería de datos para poder extraer características estadísticas, sintácticas y semánticas que permitan hacer una correcta clasificación de los títulos de noticias. Una noticia estará clasificada en segura (*safe*) o en no segura (*unsafe*), esta clasificación está basada en la emoción que puede generar un determinado tipo de noticia en el lector.

Se utilizarán tres algoritmos de aprendizaje automático para realizar la clasificación de los títulos, los cuales son *Naïve Bayes*, J48 y Máquinas de Soporte Vectorial. Este tipo de algoritmos simulan el aprendizaje humano por lo cual se dice que deben aprender con la entrada de datos de entrenamiento para que posteriormente se pueda realizar la clasificación de nuevos datos de acuerdo a lo que ya se aprendió.

Una vez hecha la clasificación y obtenidos los resultados para cada algoritmo se hará una comparativa entre ellos y se evaluaran métricas de Precisión, Cobertura y Medida F.

Tabla de contenido

Resumen ..... ¡Error! Marcador no definido.
1. Introducción 8
2. Antecedentes 8
2.1 Proyectos Terminales 8
2.2 Artículos 9
3. Justificación 9
4. Objetivos 10
4.1 Objetivo General 10
4.2 Objetivos Específicos 10
5. Marco Teórico 10
5.1 Procesamiento de lenguaje natural 10
5.2 Aprendizaje Automático 11
5.3 Algoritmo J48 12
5.4 Algoritmo Bayes Naïve 12
5.5 Algoritmo Máquinas de Soporte Vectorial 12
6. Desarrollo 12
6.1 Preprocesamiento 13
6.1.1 Limpieza de datos 14
6.1.2 Normalización de textos 15
6.2 Extracción de características 16
6.2.1 Obtención del diccionario 16
6.2.2 Pesado de características estadísticas 16
6.2.3 Pesado de características sintácticas 22
6.2.4 Pesado de características semánticas 23
6.3 Clasificación 24
6.3.1 Clasificación usando árboles de decisión 25
6.3.2 Clasificación con un algoritmo basado en reglas 29
6.3.3 Clasificación usando funciones matemáticas 30
7. Resultados 31

8. Análisis y discusión de resultados 35
9. Conclusiones 36
10. Bibliografía 36
11. Apéndices 38
Apéndice A. Código para obtener el texto de los encabezados de noticias 38
Apéndice B. Código para quitar números y signos de puntuación 39
Apéndice C. Código para eliminar palabras vacías 40
Apéndice D. Código para pasar el corpus a letras minúsculas 40
Apéndice E. Código de lematizado 41
Apéndice F. Código de pesado de características léxico-sintácticas 42
Apéndice G. Código de pesado de características semánticas 44

Índice de figuras

Figura 1. Muestra del corpus original. 13
Figura 2. Muestra del corpus con encabezados y etiquetas. 14
Figura 3. Ejemplos de listas de tokens 16
Figura 4. Ejemplo de diccionario de términos 16
Figura 5. Ventana de inicio de Weka. 17
Figura 6. Ventana de Explorer de Weka 18
Figura 7. Ventana para abrir archivo de entrada en Weka. 18
Figura 8. Selección de la clase a clasificar. 19
Figura 9. Selección del filtro StringToWordVector. 19
Figura 10. Ventana de configuración de StringToWordVector 20
Figura 11. Aplicación de filtro StringToWordVector. 20
Figura 12. Obtención del diccionario de términos. 21
Figura 13. Conteo de los títulos de noticias de acuerdo a su clasificación 21
Figura 14. Almacenamiento de pesado TF-IDF 21
Figura 15. Muestra de archivo arff con pesado TF-IDF 22
Figura 16. Ejemplos de títulos de noticias 23
Figura 17. Ejemplos de títulos de noticias para el reconocimiento de entidades nombradas 24
Figura 18. Ejemplo de salida en la consola de Eclipse para el reconocimiento de entidades nombradas 24
Figura 19. Selección del algoritmo que realiza la clasificación. 25
Figura 20. Ventana de configuración de los parámetros del algoritmo J48. 26
Figura 21. Selección del enfoque de validación, ... 27
Figura 22. Resultados de la clasificación reportados por Weka. 28
Figura 23. Proceso para guardar resultados en un archivo arff. 28
Figura 24. Ventana para guardar archivo arff con los resultados obtenidos. 29
Figura 25. Ventana de configuración de parámetros del algoritmo Naïve Bayes. . 30
Figura 26. Ventana de configuración de parámetros del algoritmo Máquinas de Soporte Vectorial. 31

Índice de tablas

Tabla 1. Muestra de Stopwords utilizadas 14
Tabla 2. Ejemplo de eliminación de Stopwords 15
Tabla 3. Ejemplos de palabras lematizadas 15
Tabla 4. Ejemplo de salida para pesado de características léxico sintácticas 23
Tabla 5. Ejemplo de salida para el reconocimiento de entidades nombradas. 24
Tabla 6. Modelo de matriz de confusión. 33
Tabla 7. Resultados de Precisión. 33
Tabla 8. Resultados de Cobertura. 34
Tabla 9. Resultados de Medida-F. 34

1. Introducción

Hoy en día se ha facilitado mucho el acceso a la información de lo que está sucediendo en todo el mundo, un ejemplo de ello son las páginas web, blogs y periódicos electrónicos, los cuales distribuyen sus últimas actualizaciones a través de los canales con el estándar RSS. Con los sistemas RSS el lector puede enterarse con mucha facilidad de las últimas noticias que son de su interés.

En este rubro, existe una gran cantidad de periódicos electrónicos que se distribuyen en internet y son escritos en las diferentes variantes del lenguaje español, por ejemplo, el utilizado en diferentes países de América que hablan este idioma. Las noticias publicadas por este tipo de periódicos suelen tener un título muy breve y sin ninguna información contextual lo que podría generar una emoción de seguridad o inseguridad (incertidumbre) en el lector. Un sistema que pueda determinar este tipo de emoción sería de gran utilidad para los lectores. En este contexto, una noticia será considerada segura cuando genera una emoción positiva o neutral en el lector o no está relacionada con temas controversiales como religión o política por mencionar algunos y una noticia no segura se define como aquella que genera emociones negativas en el lector.

Por lo tanto, en este proyecto se pretende realizar una clasificación emocional de la noticia utilizando el texto de sus títulos por medio de algoritmos de aprendizaje automático, con ello se desea saber si la noticia clasifica en segura o no segura, a partir de un corpus creado con fuentes RSS de distintos periódicos escritos en español con ocho variantes, tales como: español utilizado en Argentina, Chile, Colombia, Cuba, España, México, Perú y Venezuela. Cabe mencionar que este proyecto de integración estará ligado al proyecto de investigación aprobado en la División de CBI, denominado “SI001-18: Análisis de medios escritos en español utilizando técnicas de procesamiento de lenguaje natural”.

2. Antecedentes

2.1 Proyectos Terminales

1. Algoritmos de aprendizaje automático para el análisis de opiniones a partir de textos en español [1].

   El autor de este proyecto terminal implementa algoritmos de aprendizaje automático y hace un estudio de los resultados de estos algoritmos aplicados en el análisis de sentimientos partiendo de textos en español.

2. Análisis del comportamiento de diferentes algoritmos de aprendizaje automático [2].

En el proyecto terminal se evalúa el comportamiento de algoritmos de aprendizaje automático para obtener la clasificación de delitos ocurridos en la zona metropolitana de la ciudad de México.

3. Aplicación de Distintas Técnicas de Minería de Datos para el Tratamiento de Información [3].

   En este proyecto terminal se diseñó una aplicación de minería de datos que realiza una limpieza de datos a través de algoritmos implementados y se analiza la eficacia de estos.

2.2 Artículos

4. Visual Sentiment Analysis of RSS News Feeds Featuring the US Presidential Election in 2008 [4].

   En este artículo los autores muestran una herramienta de análisis visual para llevar el análisis de sentimientos semiautomático de grandes noticias RSS relacionadas con la elección presidencial de Estados Unidos en el año 2008.

5. LexFAR en la competencia TASS 2017: Análisis de sentimientos en Twitter basado en lexicones [5].

   En esta publicación se explica un sistema propuesto con una orientación basada en aprendizaje automático donde se utilizó el algoritmo de máquina de soporte vectorial (SVM) y dos lexicones para el análisis de sentimientos a nivel de tweets.

6. Evaluando un lexicón para la clasificación de polaridad a nivel de Tweet en español [6].

   En este artículo se hace la evaluación de un lexicón denominado MLSentiCon para llevar a cabo la clasificación de polaridad de tweets en español. Esta tarea se realiza mediante el uso de Máquinas de Soporte Vectorial.

3. Justificación

Se ha visto claramente que la inteligencia artificial en la actualidad está en crecimiento siendo así uno de los principales pilares de la cuarta revolución industrial teniendo un gran impacto en la sociedad con las diferentes innovaciones tecnológicas relacionadas con la automatización, el control y las tecnologías de la información.

Muchas empresas han adoptado desde hace tiempo la tecnología de minería de datos lo que las ayuda a procesar datos más rápido o realizar modelos estadísticos y proyecciones futuras de datos útiles para sus productos o servicios.

El desarrollo de este proyecto puede ser la base para futuras aplicaciones en el análisis de noticias que sirvan como filtros dirigidos a los lectores e igualmente para agilizar búsquedas de noticias.

4. Objetivos

4.1 Objetivo General

Implementar y comparar tres algoritmos de aprendizaje automático para la clasificación emocional de noticias seguras e inseguras a partir de títulos de noticias en español.

4.2 Objetivos Específicos

1. Diseñar e implementar un método para el preprocesamiento de títulos de noticias en ocho variantes del español, tales como: Argentina, Chile, Colombia, Cuba, España, México, Perú y Venezuela.

2. Diseñar e implementar un método para extraer características sintácticas y semánticas de los títulos de las noticias.

3. Implementar tres algoritmos de aprendizaje automático para la clasificación de noticias en seguras y no seguras: basado en reglas (Naïve Bayes), usando funciones matemáticas (máquinas de soporte vectorial) y árboles de decisión (J48).

4. Evaluar los algoritmos de aprendizaje automático implementados para la clasificación de noticias basada en su precisión.

5. Marco Teórico

5.1 Procesamiento de lenguaje natural

De acuerdo a [7] la inteligencia artificial tiene múltiples áreas de estudio, en las cuales existe una rama que se dedica esencialmente a comunicar computadoras con seres humanos mediante el uso de lenguas naturales, así como el inglés, francés, alemán, portugués, español, entre otros.

De manera oral y escrita son las dos formas en que se pueden expresar las lenguas naturales, siendo la manera escrita la que se puede tratar y obtener de una forma más fácil además de tener mucha más documentación que la manera oral.

Para que una computadora logre comprender una lengua natural se requiere de un proceso de modelización probabilística en el cual a partir de colecciones de datos de un determinado

contexto se calcula la frecuencia en que aparecen las unidades lingüísticas. A partir de este cálculo los algoritmos de aprendizaje automático pueden deducir las posibles respuestas basándose en los datos iniciales con los que aprendió.

5.2 Aprendizaje Automático

Según [8], el Aprendizaje Automático (AA, o Machine Learning, por su nombre en inglés) es la rama de la Inteligencia Artificial que tiene como objetivo desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear algoritmos capaces de generalizar comportamientos y reconocer patrones a partir de una información suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado general a partir de enunciados que describen casos particulares.

A un nivel muy básico, podríamos decir que una de las tareas del AA es intentar extraer conocimiento sobre algunas propiedades no observadas de un objeto basándose en las propiedades que sí han sido observadas de ese mismo objeto (o incluso de propiedades observadas en otros objetos similares) o, en palabras más llanas, predecir comportamiento futuro a partir de lo que ha ocurrido en el pasado. Un ejemplo de mucha actualidad sería, por ejemplo, el de predecir si un determinado producto le va a gustar a un cliente basándonos en las valoraciones que ese mismo cliente ha hecho de otros productos que sí ha probado.

Hay un gran número de problemas que caen dentro de lo que llamamos aprendizaje inductivo. La principal diferencia entre ellos estriba en el tipo de objetos que intentan predecir. En [8] se incluyen algunas clases habituales, las cuales son:

- Regresión: Intentan predecir un valor real. Por ejemplo, predecir el valor de la bolsa mañana a partir del comportamiento de la bolsa que está almacenado (pasado). O predecir la nota de un alumno en el examen final basándose en las notas obtenidas en las diversas tareas realizadas durante el curso.
- Clasificación (binaria o multiclase): Intentan predecir la clasificación de objetos sobre un conjunto de clases prefijadas. Por ejemplo, clasificar si una determinada noticia es de deportes, entretenimiento, política, etc. Si solo se permiten 2 posibles clases, entonces se llama clasificación binaria; si se permiten más de 2 clases, estamos hablando de clasificación multiclase.
- Ranking: Intentar predecir el orden óptimo de un conjunto de objetos según un orden de relevancia predefinido. Por ejemplo, el orden en que un buscador devuelve recursos de internet como respuesta a una búsqueda de un usuario.

Dentro del aprendizaje automático se cuenta con una amplia gama de algoritmos, de los cuales, a continuación, se describen tres de ellos dada su naturaleza, a saber: árboles de decisión, algoritmos basados en reglas y algoritmos centrados en funciones matemáticas.

5.3 Algoritmo J48

Este algoritmo descrito en [9], construye un árbol a partir de datos. Se construye iterativamente al ir agregando nodos o ramas que minimicen la diferencia entre los datos. Este algoritmo es un descendiente del ID3 y se extiende en el sentido de su capacidad de utilizar atributos numéricos y vacíos para generar reglas del árbol. Con el propósito de clasificación de una nueva instancia, J48 prueba cada uno de los valores del atributo de acuerdo con su estructura hasta que encuentra una hoja, la cual contiene los valores de la clase para cada instancia.

5.4 Algoritmo Bayes Naïve

Algoritmo que genera un árbol de decisión a partir del clasificador bayesiano Naïve Bayes, que es el modelo más simple de clasificación ya que asume independencia entre todos los atributos dada una clase. Por lo tanto, corresponde a un modelo de atributos independientes. En este caso, la estructura de la red es fija y sólo es necesario aprender los parámetros. El fundamento principal de este clasificador es la suposición de que todos los atributos son independientes del valor de la variable clase [9].

5.5 Algoritmo Máquinas de Soporte Vectorial

De acuerdo a [10] una máquina de soporte vectorial (*SVM*) es una técnica de clasificación la cual aprende de la superficie de decisión de dos clases distintas de los puntos de entrada. Como un clasificador de una sola clase, la descripción dada por los datos de los vectores de soporte es capaz de formar una frontera de decisión alrededor del dominio de los datos de aprendizaje con muy poco o ningún conocimiento de los datos fuera de esta frontera. Los datos son mapeados por medio de un *kernel* Gaussiano u otro tipo de *kernel* a un espacio de características en un espacio dimensional más alto, donde se busca la máxima separación entre clases. Esta función de frontera, cuando es traída de regreso al espacio de entrada, puede separar los datos en todas las clases distintas, cada una formando un agrupamiento.

6. Desarrollo

En esta sección se describe como se llevaron a cabo todas las etapas de este proyecto de integración. Las etapas que a continuación se presentan son: El pre-procesamiento del texto, extracción de características estadísticas, léxicas y semánticas. Para finalizar se realizará la clasificación con los algoritmos de aprendizaje automático antes mencionados.

6.1 Preprocesamiento

El pre-procesamiento de las noticias en español consiste en depurar la información que no es de utilidad para poder extraer características que ayuden a obtener una mejor clasificación de los títulos de noticias.

El corpus inicial con el que se trabajó cuenta con mil quinientos títulos de noticias en español, adicionalmente cada registro trae consigo un id, el país de origen de la noticia, una URL, la fecha y una etiqueta de su clasificación como se muestra en la Figura 1.

```
1  9   VEN http://www.2001.com.ve/en-la-agenda/179911/eeuu-
   -maduro-debera-explicar-sus-elecciones--ilegitimas--en-l
   a-cumbre-de-lima.html   EEUU: Maduro deberá explicar
   sus elecciones "ilegítimas" en la Cumbre de Lima
   2018-01-29 20:48:14 CET UNSAFE
2  12  PER http://www.iprofesional.com/notas/262584-justici
   a-peru-indemnizacion-corrupcion-reparacion-odebrecht-Per
   u-exige-a-Odebrecht-us1000-millones-por-danos-y-perjuici
   os  Perú exige a Odebrecht u$s1.000 millones por daños
   y perjuicios 2018-01-29 23:40:45 CET UNSAFE
3  15  PER http://www.americatv.com.pe/noticias/actualidad/
   comision-lava-jato-app-no-designara-reemplazo-su-agrupac
   ion-n308703 Comisión Lava Jato: APP no designará
   reemplazo de Marisol Espinoza  2018-01-30 01:22:59 CET
   UNSAFE
4  20  PER http://rpp.pe/mundo/estados-unidos/video-peruana
   -respondera-en-espanol-a-trump-en-el-discurso-del-estado
   -de-la-union-noticia-1102204    Peruana responderá en
   español a Trump en el discurso del Estado de la Unión
   2018-01-30 03:34:56 CET UNSAFE
5  25  ESP http://www.elmundo.es/
   tecnologia/2018/01/30/5a6f5457268e3e89418b45c2.html
   ¿Funcionará Netflix mejor con el Internet de Movistar
   ahora?    2018-01-30 03:39:27 CET SAFE
```

Figura 1. Muestra del corpus original.

Utilizando el entorno de desarrollo Eclipse y el lenguaje de programación Java se generó un código con el cual se leyó el archivo inicial línea por línea quitando los campos que no son relevantes para el objetivo del proyecto para posteriormente crear un nuevo corpus con los campos que son útiles para esta etapa, siendo únicamente el título de la noticia y su etiqueta de clasificación los campos que vamos a recuperar en el nuevo corpus como se observa en la Figura 2. El código fuente para obtener el texto de los encabezados de noticias se puede consultar en el Apéndice A.

```
1  EEUU Maduro deberá explicar sus elecciones ilegítimas
   en la Cumbre de Lima,unsafe
2  Perú exige a Odebrecht us  millones por daños y
   perjuicios,unsafe
3  Comisión Lava Jato APP no designará reemplazo de
   Marisol Espinoza,unsafe
4  Peruana responderá en español a Trump en el discurso
   del Estado de la Unión,unsafe
5  Funcionará Netflix mejor con el Internet de Movistar
   ahora,safe
```

Figura 2. Muestra del corpus con encabezados y etiquetas.

6.1.1 Limpieza de datos

Una vez obtenidos los títulos de noticias y sus respectivas etiquetas se procede a procesar el corpus en formato txt para la limpieza de los datos, es decir que eliminaremos fragmentos de texto que no están relacionados con el contexto de los datos, estos fragmentos se conocen como ruido. En este caso se eliminarán signos de puntuación, valores numéricos, caracteres especiales y palabras vacías (mejor conocidas por su nombre en inglés *Stopwords*) ya que no aportan algo realmente significativo para el procesamiento de lenguaje natural, esta limpieza de texto ayudará a que se pueda identificar mejor la frecuencia de aparición de las palabras y realizar una mejor extracción de características logrando un pesado eficiente de las mismas.

La eliminación de los signos de puntuación y los valores numéricos se realiza mediante expresiones regulares utilizando el método para cadenas de caracteres *replaceAll* para reemplazar esos caracteres por cadenas vacías. El código fuente de este proceso se puede consultar en el Apéndice B.

En lo que respecta a las palabras vacías se busca una lista de *Stopwords* para el español en internet y se guarda en un archivo de texto el cual se lee línea por línea y se hace un cruce de información con el corpus para buscar las palabras vacías en él y poder reemplazarlas por cadenas vacías. El código del método que elimina *Stopwords* se puede consultar en el Apéndice C.

En la Tabla 1 a continuación se muestran como ejemplo algunas de las *Stopwords* que se utilizaron.

Tabla 1. Muestra de Stopwords utilizadas

ser	estoy	porque	un
es	estan	por qué	una
soy	como	estado	unas
eres	en	estaba	unos
somos	para	ante	uno

En la Tabla 2 se muestra como ejemplo el título original en letras minúsculas de una noticia contenida en el corpus y ese mismo título después de que se eliminaron las *Stopwords*.

Tabla 2. Ejemplo de eliminación de Stopwords

un	informe	de	la	bcc	niega	que	la	brecha	salarial	sea	por	género
	informe	de		bcc	niega	que		brecha	salarial	sea		género

6.1.2 Normalización de textos

Esta etapa consiste en llevar los textos a una representación estándar por lo que el corpus se pasa a letras minúsculas y posteriormente se realiza la lematización del texto.

En está ocasión la razón de que el corpus se procese y el texto se lleve a letras minúsculas, es que Weka no admite letras mayúsculas. El código fuente para pasar el corpus a letras minúsculas se puede consultar en el Apéndice D.

La lematización de textos es un proceso lingüístico en el cual se obtiene la raíz de cada palabra, el lema de una palabra es la palabra que normalmente encontraríamos en un diccionario tradicional de cualquier idioma, en este caso del idioma español. Una vez realizado el lematizado encontraremos los sustantivos en su forma singular y en masculino y para los verbos en su forma en infinitivo como los ejemplos contenidos en la Tabla 3.

El proceso de lematización se lleva a cabo con la ayuda de *TreeTagger* [8] la cual es una herramienta para obtener información del lema apoyándose en el uso de un gran diccionario para el idioma que se va a analizar. El código para lematizar se puede consultar en el Apéndice E.

Tabla 3. Ejemplos de palabras lematizadas

pide	pedir
prohíba	prohibir
varones	varón
circulen	circular
motos	moto

6.2 Extracción de características

La extracción de características para este proyecto de integración consta de obtener características estadísticas, léxicas y semánticas a partir del corpus obtenido en la etapa de preprocesado, esto se logra llevando a una representación numérica almacenada en vectores del texto analizado para que Weka logre interpretarlo y se pueda hacer la correcta clasificación de los títulos de noticias.

6.2.1 Obtención del diccionario

Se define como diccionario aquel conjunto que contiene todas las palabras diferentes que se encuentran en el corpus. Para poder generar el diccionario es necesario seguir ciertos pasos, los cuales se describen y ejemplifican a continuación.

Inicialmente se debe tener la lista de tokens segmentados y limpios como se muestra en la Figura 3, para el ejemplo se toman tres títulos de noticias, en este caso cada token es una palabra.

```
1  {"supervivencia", "al", "cáncer", "aumentar", "paciente", "rico"}
2  {"ahora", "buscador", "google", "se", "encontrar"}
3  {"que", "deber", "eclipse", "de", "superluna", "azul", "de", "sangre"}
```

Figura 3. Ejemplos de listas de tokens

El siguiente paso es construir la lista de palabras del vocabulario, esta lista será el diccionario de términos, así como se aprecia en la Figura 4.

```
diccionario = {"supervivencia", "al", "cáncer", "aumentar", "paciente",
               "rico", "ahora", "buscador", "google", "se", "encontrar",
               "que", "deber", "eclipse", "de", "superluna", "azul", "sangre"}
```

Figura 4. Ejemplo de diccionario de términos

6.2.2 Pesado de características estadísticas

Para poder extraer las características estadísticas es necesario llevar el texto a una representación cuantitativa, es decir en su forma numérica. Esto se logra con lo que se conoce como hacer un pesado, es decir que se obtendrá un diccionario con todas las palabras diferentes que aparecen en todo el corpus y también se logrará la frecuencia inversa de los términos (TF-IDF), la cual será almacenada en un archivo en forma de vectores manteniendo su etiqueta de clasificación para cada título de noticia.

Para generar el archivo que contiene el pesado nos apoyaremos en la herramienta Weka, a continuación, se muestran los pasos detalladamente para poder realizarlo.

6.2.2.1 Proceso para el pesado de características estadísticas

Como primer paso entramos a la aplicación Weka [10] versión 3.8.2 y se mostrará como primera pantalla la que podemos apreciar en la Figura 5 y se da clic en la opción del botón *Explorer*.

Figura 5. Ventana de inicio de Weka.

Enseguida se despliega otra ventana en la que se da clic en *Open File* como se ilustra en la Figura 6.

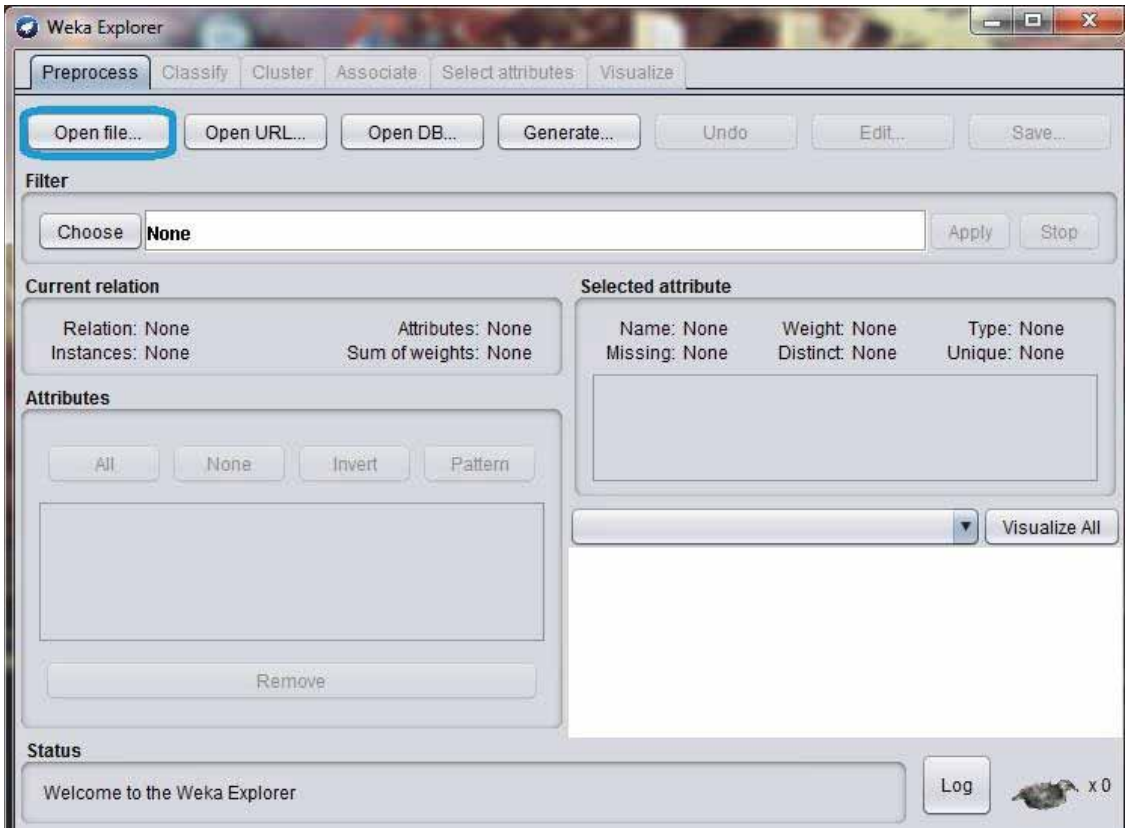

Figura 6. Ventana de Explorer de Weka.

Después se busca el corpus lematizado en letras minúsculas y se selecciona el botón Abrir como se aprecia en la Figura 7.

Figura 7. Ventana para abrir archivo de entrada en Weka.

En la ventana siguiente en la sección de atributos se seleccionará *tag* que es como está identificada nuestra clase para clasificar y en la sección para elegir el filtro que se usará se da clic en el botón *Choose* como se ejemplifica en la Figura 8.

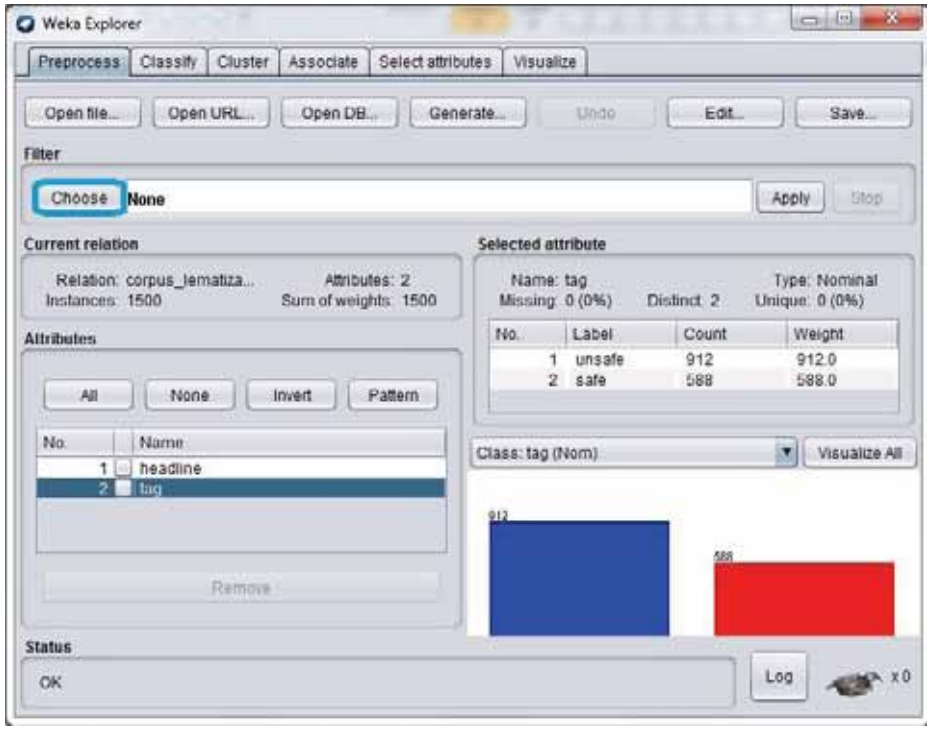

Figura 8. Selección de la clase a clasificar.

La selección anterior desplegara un menú con distintas carpetas, mediante la ruta weka/filters/unsupervised/attribute se selecciona el filtro *StringToWordVector*. Una vez elegido el filtro se da clic sobre el texto de *StringToWordVector* para abrir las configuraciones del filtro como se muestra en la Figura 9.

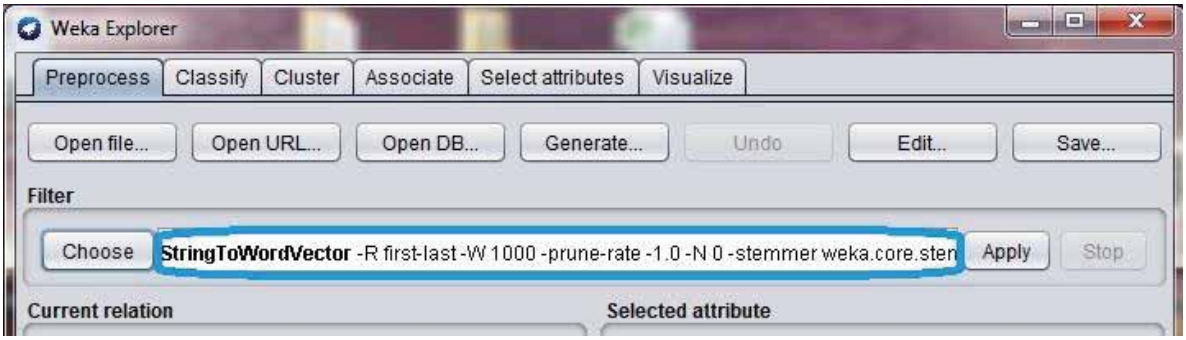

Figura 9. Selección del filtro StringToWordVector.

En la siguiente ventana se ponen los valores de *IDFTransform* y *TFTransform* en *True* y se da clic en el botón *OK* para guardar la configuración como se señala en la Figura 10.

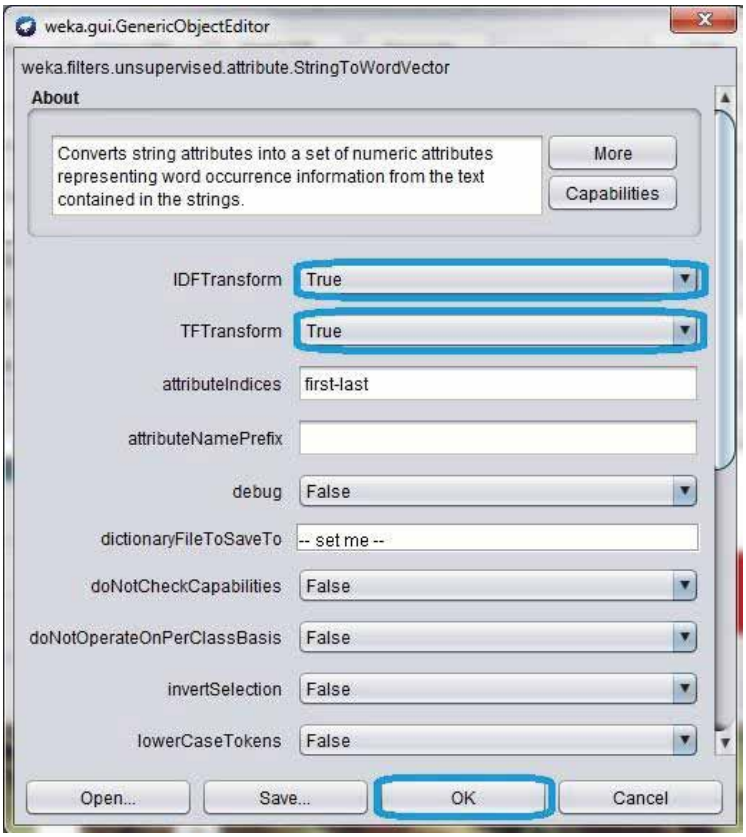

Figura 10. Ventana de configuración de StringToWordVector.

Se regresa a la ventana anterior donde primero se selecciona el botón *Apply* para aplicar el filtro que hemos elegido con sus configuraciones como se muestra en la Figura 11, esto genera un listado de palabras las cuales no se repiten como se aprecia en la Figura 12. En la sección *Selected attribute* se observa el conteo de los títulos de noticias de acuerdo a su clasificación como se observa en la Figura 13

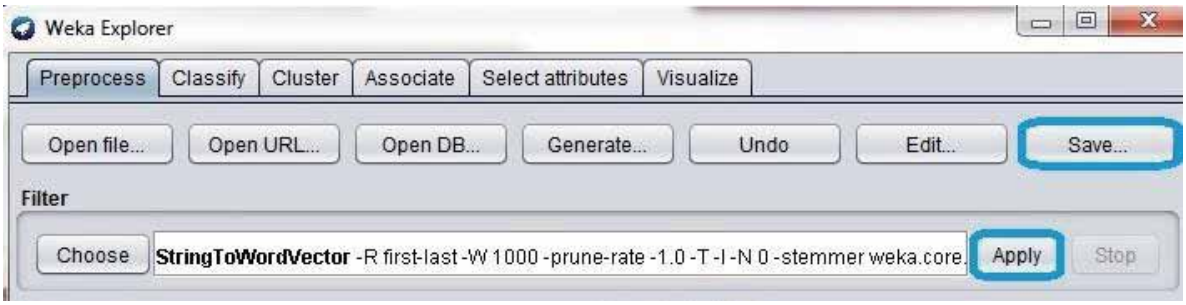

Figura 11. Aplicación de filtro StringToWordVector.

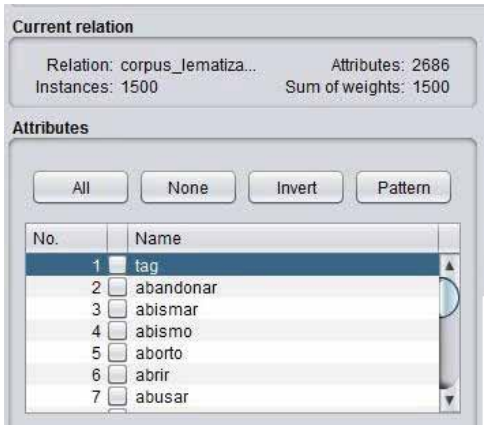

Figura 12. Obtención del diccionario de términos.

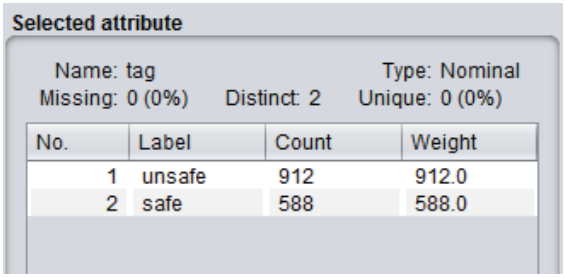

Figura 13. Conteo de los títulos de noticias de acuerdo a su clasificación

Como último paso se guarda el archivo en formato arff con el pesado dando clic en la opción *Save* como aparece en la Figura 11 y se selecciona la ruta de almacenamiento y el nombre del archivo de salida como se muestra en la Figura 14 y finalmente damos clic en el botón Guardar.

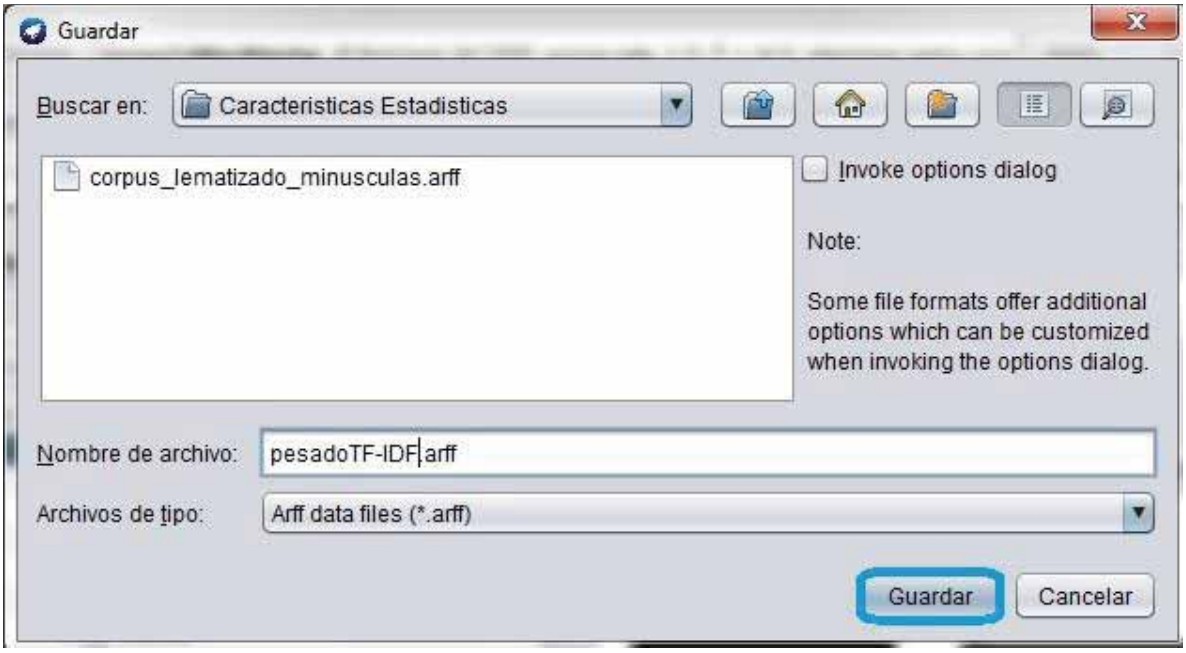

Figura 14. Almacenamiento de pesado TF-IDF

El resultado de este proceso da como salida un archivo de tipo arff que contiene valores numéricos almacenados en vectores para cada título de noticia, en la Figura 15 se muestra un fragmento de este archivo.

```
2690  @data
2691  {271 3.407044,282 0.347943,284 3.346732,342 3.473108,346
      2.837983,597 3.346732,624 2.784638,1650 5.069138}
2692  {281 4.588685,400 3.627779,665 3.065685,722 4.108232,776
      2.992655,1022 3.953561,1063 1.180949}
2693  {67 4.588685,227 3.953561,282 0.347943,561 4.307638,588
      4.307638,703 1.942449,1507 5.069138}
2694  {282 0.347943,292 1.362806,377 3.407044,775 3.720336,886
      3.953561,1003 2.529751,1019 4.307638,1535 5.069138}
2695  {0 safe,35 3.627779,146 3.065685,282 0.347943,1719
      5.069138,1863 4.588685,2045 5.069138,2078 5.069138}
2696  {37 1.663945,205 4.108232,282 0.347943,426 3.953561,481
      3.546138,662 3.291251,687 2.400419,889 3.953561,1063
      1.180949,2685 5.069138}
2697  {0 safe,711 3.953561,1144 5.069138,1460 4.108232,1680
      4.588685,1743 5.069138,1894 5.069138,2497 4.307638}
2698  {142 4.307638,282 0.347943,472 4.108232,529 3.827185,694
      3.953561,855 1.4071,919 4.307638,941 2.585232,1242 5.069138}
2699  {282 0.347943,598 4.588685,943 4.588685,2217 4.108232}
2700  {120 3.827185,243 2.624851,299 3.627779,546 3.028209,713
      4.307638,1031 4.307638}
```

Figura 15. Muestra de archivo arff con pesado TF-IDF

6.2.3 Pesado de características sintácticas

En la literatura se pueden encontrar varias características léxicas-sintácticas, pero para el contexto del corpus solo se eligieron las más relevantes. Estas características son diversidad léxica (número de palabras sin repetir en el documento), longitud del vocabulario (diversidad léxica entre el total de palabras del documento), número de verbos, número de sustantivos, número de adjetivos, número de determinantes y el número de palabras capitalizadas.

Para poder reconocer cada una de las características léxicas-sintácticas es necesario realizar un etiquetado gramatical PoS (*Part of Speech*), este nos permite asignar a cada una de las palabras del texto su categoría gramatical. El etiquetado *PoS* se realizó con la ayuda de las librerías de *StanfordCoreNLP* [10] para Java con las cuales se identificó el tipo de palabra y se contabilizo cada característica léxico-sintáctica para cada título de noticia. La lista de las etiquetas PoS se puede consultar en el sitio web de *The Stanford Natural Language Processing Group* [11]. Finalmente para obtener el pesado, los valores numéricos encontrados fueron almacenados en vectores en un archivo CSV.

Se consideran tres títulos de noticias contenidos en el corpus para ejemplificar el pesado de las características léxicas-sintácticas como se muestra en la Figura 16.

```
1  el supervivencia al cáncer aumentar paciente rico ,safe
2  ahora buscador Google se encontrar ,safe
3  todo que deber eclipse de superluna azul de sangre ,safe
```

Figura 16. Ejemplos de títulos de noticias

Para identificar fácilmente las características se tomará en cuenta una abreviación para cada una como se indica en la siguiente lista:

- Longitud del vocabulario (RV)
- Diversidad léxica (DL)
- Número de verbos (NV)
- Número de sustantivos (NS)
- Número de adjetivos (NA)
- Número de determinantes (ND)
- Número de palabras capitalizadas (NPC)

Después de realizar el pesado de las características léxico-sintácticas se obtendrá una salida como la que se observa en la Tabla 4 y se almacena en un archivo CSV. El código fuente del pesado se puede consultar en el Apéndice F.

Tabla 4. Ejemplo de salida para pesado de características léxico sintácticas

DL	RV	NV	NS	NA	ND	NCP	Tag
1.000	7.000	2	2	1	2	1	safe
1.000	5.000	1	0	1	0	1	safe
0.888	8.000	1	3	1	1	0	safe

6.2.4 Pesado de características semánticas

A partir del contexto del corpus se pueden detectar algunas características semánticas las cuales serán identificadas por el reconocimiento de entidades nombradas (*NER* por sus siglas en inglés) con el apoyo de de las librerías de *StanfordCoreNLP* [10] para Java. Las características son las siguientes:

- Número de organizaciones
- Número de personas
- Número de ubicaciones
- Número de porcentajes
- Número de tiempos
- Número de fechas
- Número de monedas

En la Figura 17 se muestran tres títulos de noticias que se tomarán como ejemplo para el reconocimiento de entidades nombradas.

```
1  Alista Hacienda nuevo esquema de estimulo ante reforma fiscal de EU,safe
2  Facebook gano 12.800 millones de euros en 2017 un 56% mas,safe
3  Anuncio de TV muestra a ilegal disparandole a una mujer en Estados Unidos,unsafe
```

Figura 17. Ejemplos de títulos de noticias para el reconocimiento de entidades nombradas.

Para cada título de noticia se aplicará el reconocimiento de entidades nombradas y se hará el conteo de frecuencia de cada una de las características semánticas antes mencionadas, ese pesado se almacenará en vectores para cada noticia, manteniendo su etiqueta de clasificación y guardando el archivo en formato CSV como se observa en la Tabla 5. El código fuente para extraer las características semánticas se puede consultar en el Apéndice G.

Tabla 5. Ejemplo de salida para el reconocimiento de entidades nombradas.

org	pers	loc	percent	time	date	money	tag
3	0	0	0	0	0	0	safe
1	0	0	2	0	1	0	safe
1	0	2	0	0	0	0	unsafe

Para la segunda noticia se proporciona el ejemplo en la Figura 18 de cómo se realizó la prueba en Eclipse de la salida que hace el *NER* al identificar las entidades nombradas.

```
Facebook ORG
gano O
12.800 NUMBER
millones O
de O
euros O
en O
2017 DATE
un O
56 PERCENT
% PERCENT
mas O
```

Figura 18. Ejemplo de salida en la consola de Eclipse para el reconocimiento de entidades nombradas.

6.3 Clasificación

En esta sección se presentan los enfoques de clasificación utilizados en este trabajo para determinar la clase de una noticia de entrada. Las clases consideradas son segura o no segura.

6.3.1 Clasificación usando árboles de decisión

Los árboles de decisiones son algoritmos que permiten realizar una clasificación supervisada buscando una variable dependiente concreta. Al ser obtenido el árbol de decisión del algoritmo J48 este establece una regla de decisión, esto permite segmentar en grupos de acuerdo a las características extraídas de los títulos de noticias y decidir cuáles son relevantes para posteriormente llevar a cabo la clasificación. Con base en las reglas de decisión se puede hacer la clasificación de títulos nuevos de noticas de acuerdo a sus atributos.

Para poder llevar a cabo la clasificación en la herramienta Weka es necesario abrir el archivo arff donde se almaceno el pesado de las características, en este caso se tomará como ejemplo la clasificación de características estadísticas más características léxicas. Después de abrir el archivo arff se selecciona el algoritmo con el cual se hará la clasificación como se ilustra en la Figura 19.

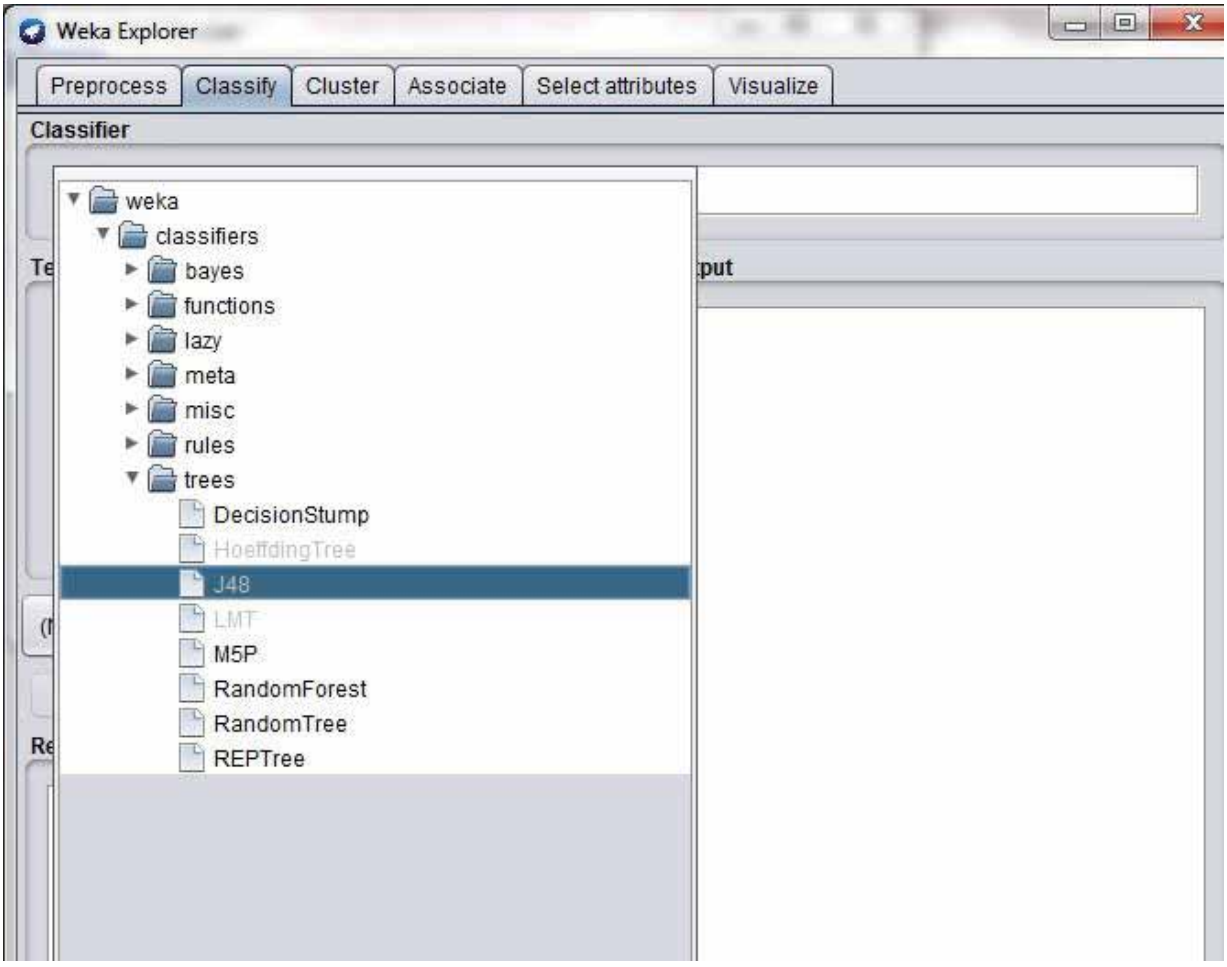

Figura 19. Selección del algoritmo que realiza la clasificación.

Una vez que se seleccionó el algoritmo J48 se da clic en el nombre del algoritmo para que se despliegue la ventana que muestra los parámetros y sus valores por default como se observa en la Figura 20.

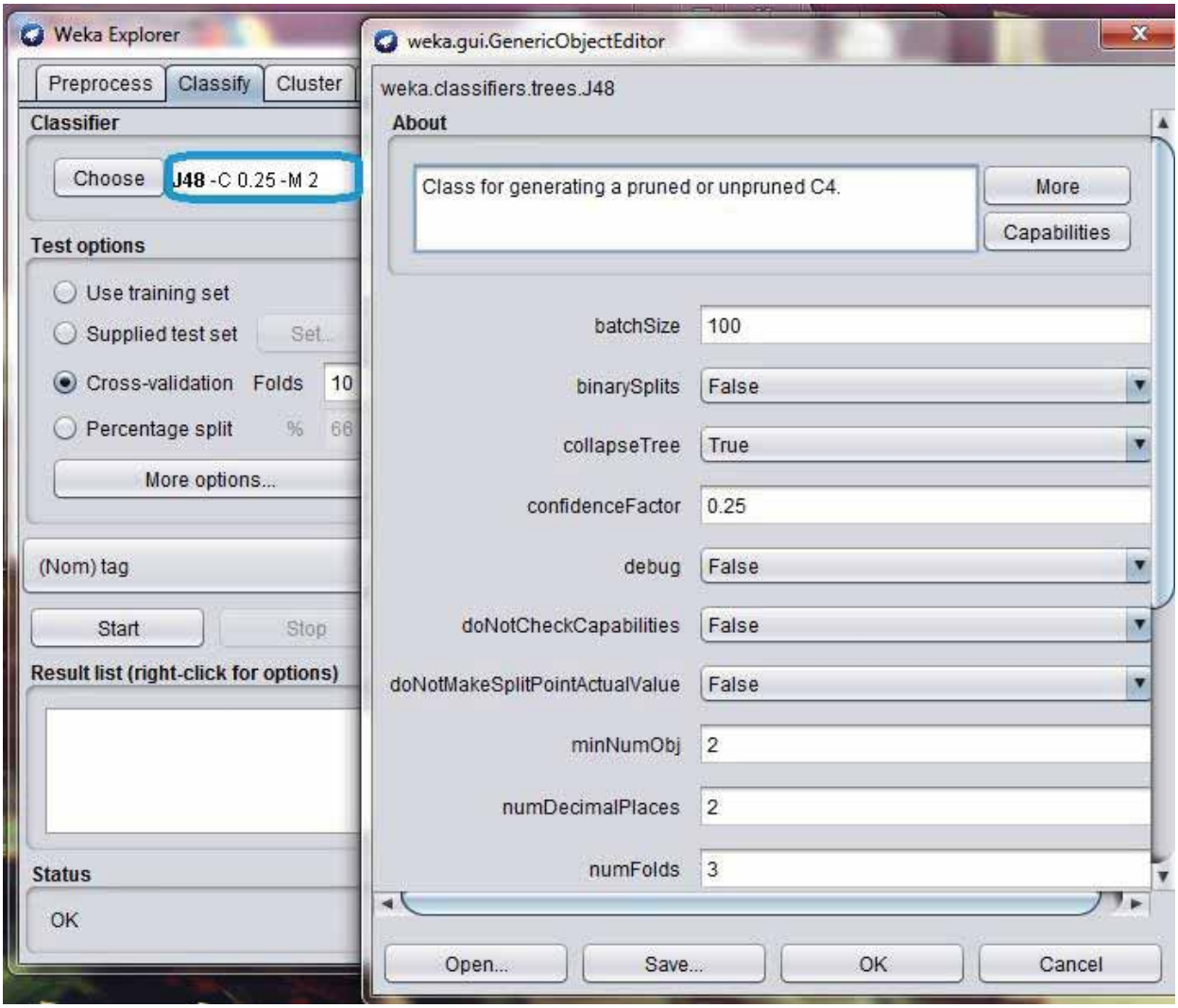

Figura 20. Ventana de configuración de los parámetros del algoritmo J48.

Posteriormente se elige en la sección *Test Options* la opción de validación cruzada (*Cross Validation*) con el valor de *Folds* por default en diez. También se seleccionara la clase por la cual se hará la clasificación y finalmente para ejecutar el algoritmo se da clic en el botón *Start* como se aprecia en la Figura 21.

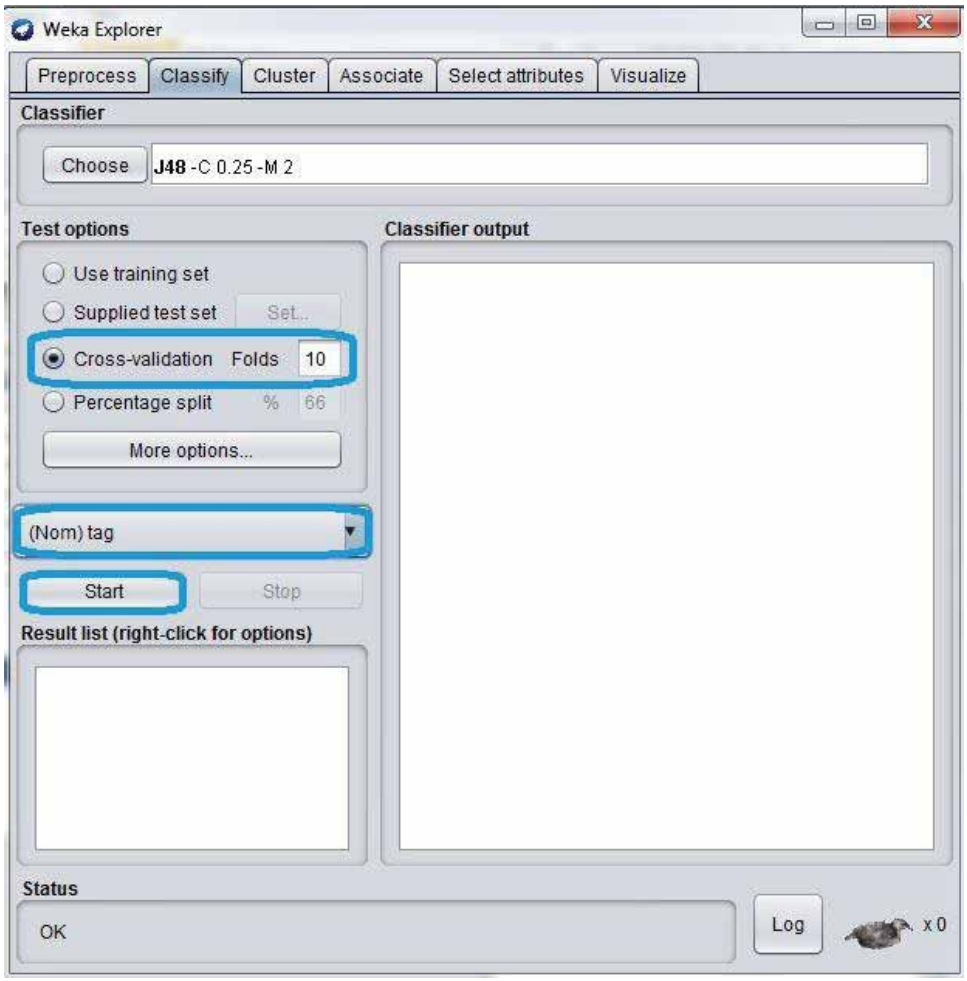

Figura 21. Selección del enfoque de validación,

Al término de la ejecución del algoritmo se obtendrá una salida como la que se muestra en la Figura 22 donde se pueden apreciar los valores reportados por las métricas y los porcentajes de las instancias correctamente clasificadas y las incorrectamente clasificadas, así como también se puede observar la matriz de confusión.

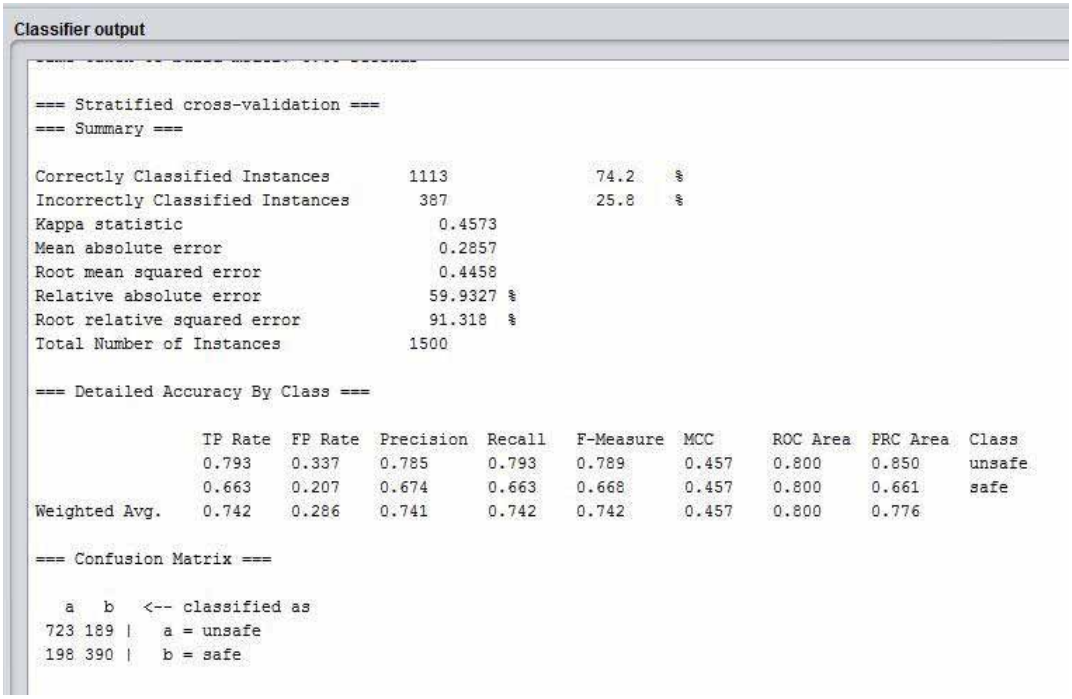

Figura 22. Resultados de la clasificación reportados por Weka.

Para guardar los resultados en un archivo se procede a dar clic derecho en la sección *Results list* en el registro de la salida que generó Weka y se selecciona la opción *Save Result Buffer* como se muestra en la Figura 23.

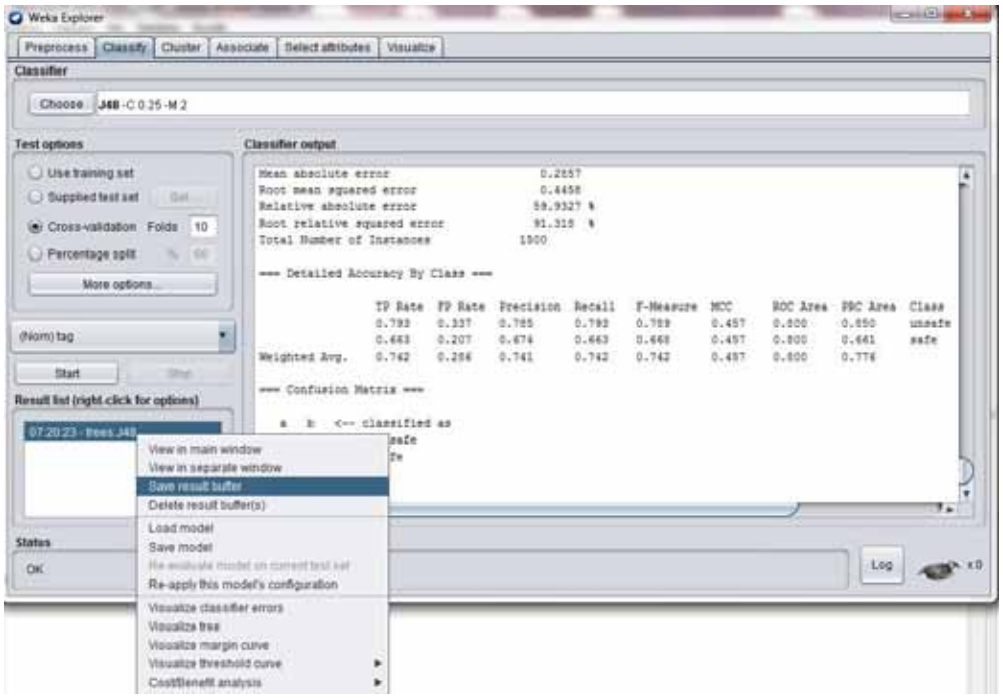

Figura 23. Proceso para guardar resultados en un archivo arff.

Finalmente solo se seleccionará la ruta donde se almacenará el archivo de salida como se observa en la Figura 24.

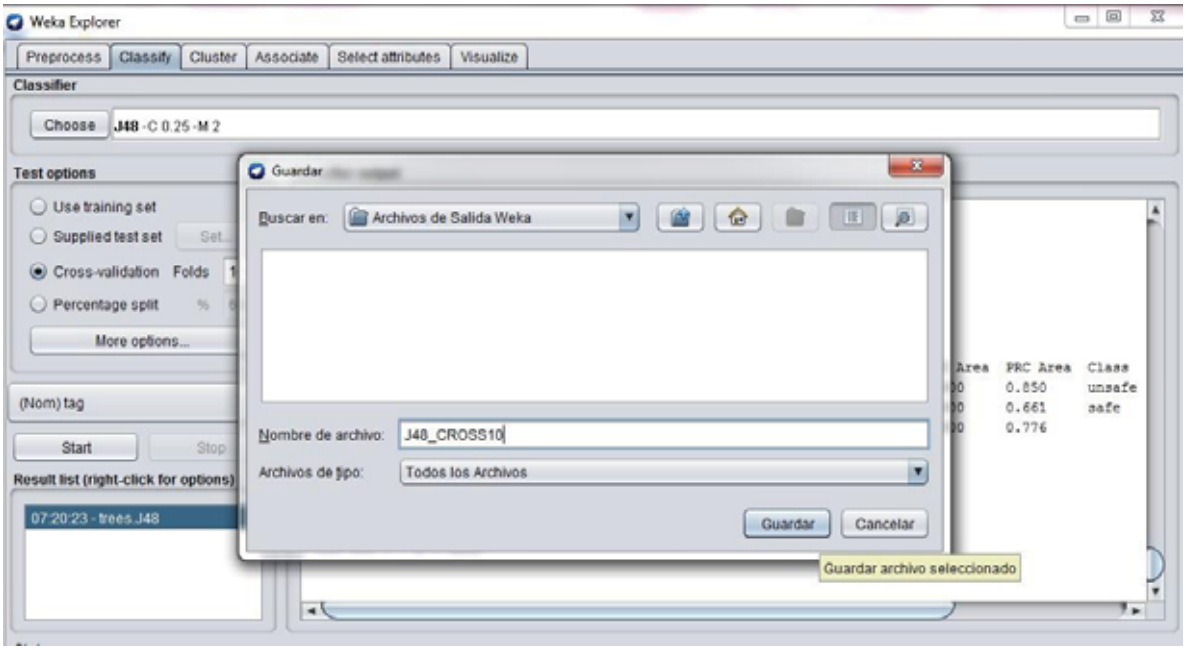

Figura 24. Ventana para guardar archivo *arff* con los resultados obtenidos.

6.3.2 Clasificación con un algoritmo basado en reglas

El algoritmo de *Naïve Bayes* genera un árbol de decisión a partir del clasificador bayesiano *Naïve Bayes,* este modelo de clasificación asumirá independencia entre todos los atributos de la clase, en este caso segura (*safe*) y no segura (*unsafe*). Este clasificador tiene como cimiento la suposición de que todos los atributos son independientes del valor de la clase. Esta clasificación lleva exactamente el mismo proceso que se mostró en el punto anterior con la herramienta Weka, lo único en que difiere es en los parámetros del clasificador y en su configuración como se muestra en la Figura 25.

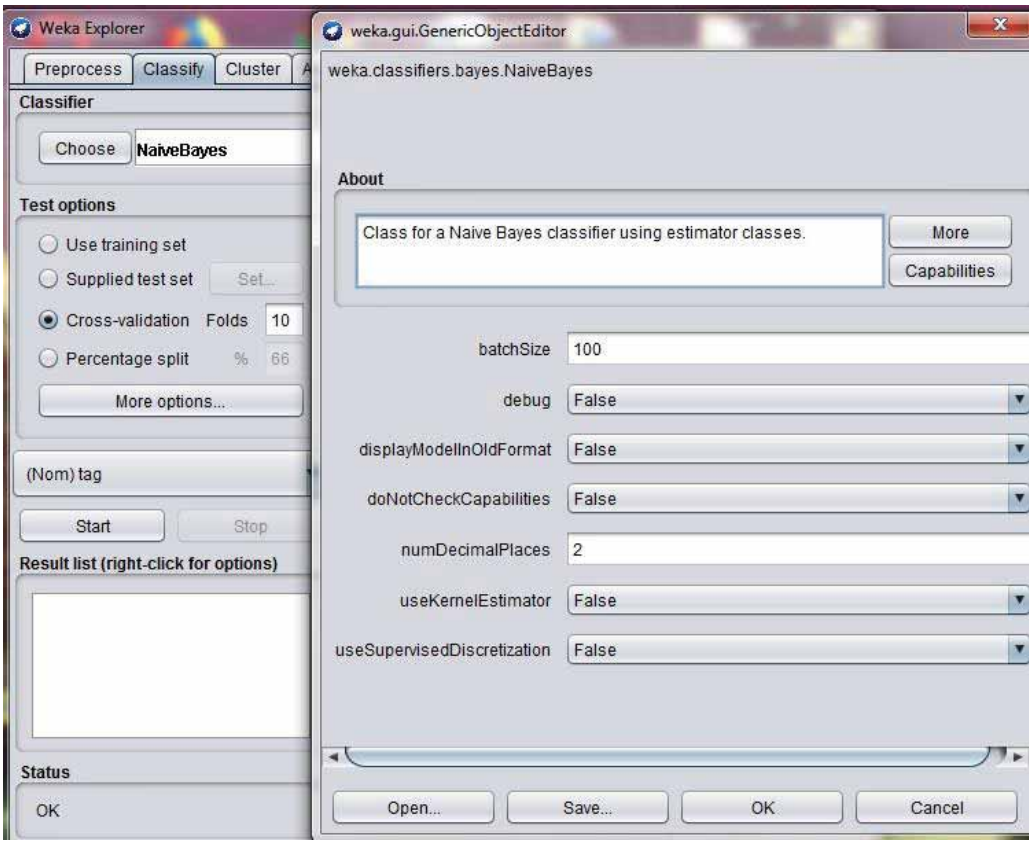

Figura 25. Ventana de configuración de parámetros del algoritmo Naïve Bayes.

6.3.3 Clasificación usando funciones matemáticas

El algoritmo de Máquinas de Soporte Vectorial (*SVM*) logra hacer la clasificación de los títulos de noticias gracias a que aprende de los elementos de las clases segura (*safe*) y no segura (*unsafe*) para establecer una frontera entre estas dos clases que maximiza la distancia entre los ejemplos de cada clase que no tengan elementos entre ellos. Con la ayuda de un *kernel* los datos pueden ser mapeados a una dimensión más alta donde se buscar la máxima separación entre clases. Cuando la función frontera es traída de regreso al espacio dimensional inicial es posible separar y agrupar los datos en todas las clases.

El proceso para llevar a cabo la clasificación en la herramienta Weka fue idéntico como en los dos puntos anteriores a diferencia de los parámetros propios del algoritmo Máquinas de Soporte Vectorial (*SVM*) como se muestra en la Figura 26.

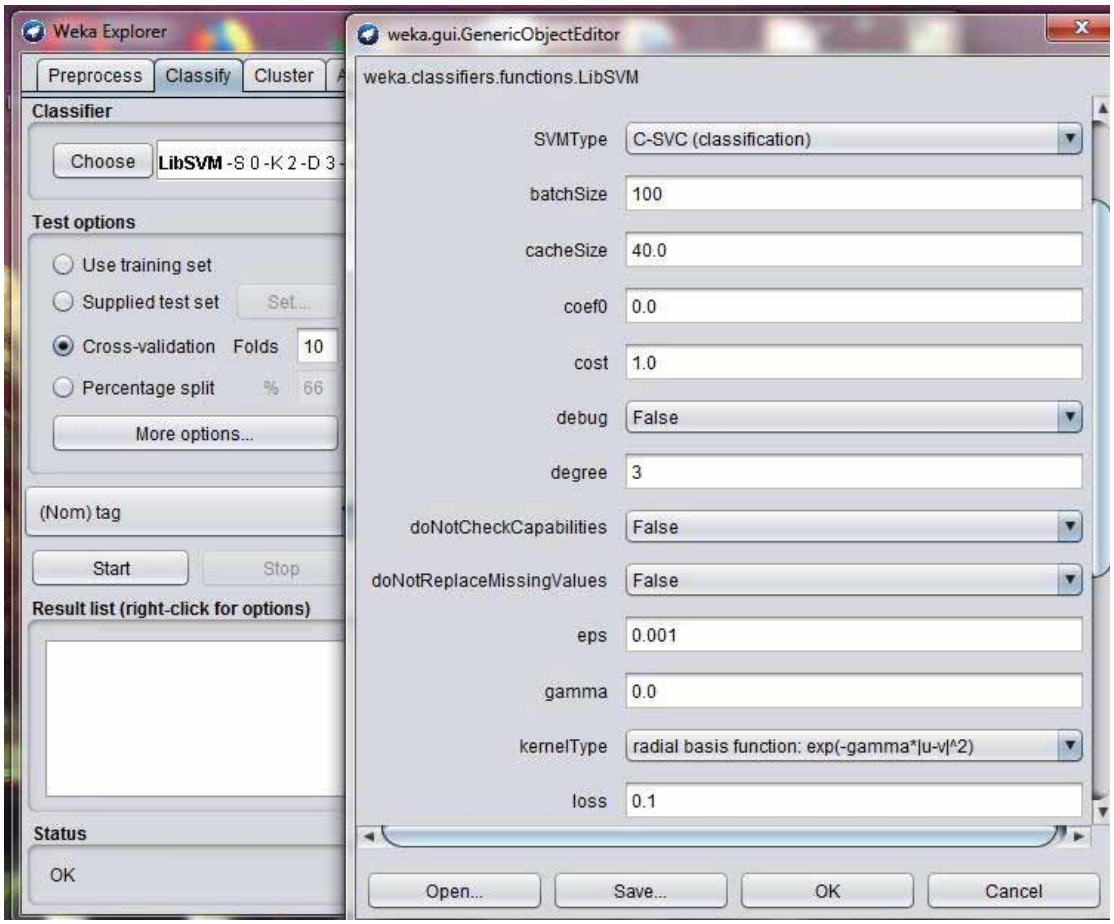

Figura 26. Ventana de configuración de parámetros del algoritmo Máquinas de Soporte Vectorial.

7. Resultados

Para la generación de resultados se ejecutaron los algoritmos J48, *Naïve Bayes* y máquinas de soporte vectorial con los enfoques de validación *Cross Validation* y *Percentage Split* para cada tipo de características extraídas y sus múltiples combinaciones las cuales dieron un total de cuarenta y dos archivos de salida.

Con la finalidad de obtener los mejores resultados en la evaluación de los algoritmos de aprendizaje automático se realizaron múltiples combinaciones de las características extraídas, las cuales se muestran en el siguiente listado:

- Características estadísticas
- Características léxicas
- Características semánticas
- Características léxicas + Características semánticas
- Características estadísticas + Características léxicas
- Características estadísticas + Características semánticas
- Características estadísticas + Características léxicas + Características semánticas

En la etapa de evaluación se utilizaron dos enfoques de validación, *Cross Validation* y *Percentage Split*. En [16] se dice que la validación cruzada (*Cross Validation*), una técnica de evaluación estándar, es una forma sistemática de ejecutar divisiones porcentuales repetidas. Divide un conjunto de datos en 10 piezas ("pliegues"), luego extiende cada pieza para probarlas y entrenar con las nueve restantes. Esto da 10 resultados de evaluación, que se promedian. En la validación cruzada "estratificada", al hacer la división inicial, nos aseguramos de que cada pliegue contenga aproximadamente la proporción correcta de los valores de clase. Después de realizar 10 veces la validación cruzada y calcular los resultados de la evaluación, Weka invoca el algoritmo de aprendizaje una última vez (11) en todo el conjunto de datos para obtener el modelo que se imprime.

De acuerdo a [17] el enfoque de validación *Percentage Split* realiza una división porcentual, ésta división del conjunto de datos se hace de forma aleatoria. Weka por default destina un 66 por ciento de los datos del corpus para entrenamiento y un 34 por ciento para prueba.

Para poder evaluar la ejecución de los algoritmos Weka maneja ciertas métricas que comúnmente se han utilizado en proyectos de clasificación de texto las cuales se describen a continuación.

Precisión (*Precision*): Métrica que está dedicada a medir la cantidad de términos correctamente reconocidos respecto al total de términos pronosticados, ya sean estos términos verdaderos o falsos.

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Cobertura (*Recall*): Es la métrica que calcula la proporción de términos correctamente identificados respecto al total de términos reales.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Medida-F (*F-Measure*): Es la media armónica entre cobertura y precisión. Esta métrica tiene un parámetro que establece la compensación entre cobertura y precisión. El estándar de la medida-F es F1, que otorga igual importancia a la recuperación y precisión: se basa en conseguir un valor único ponderado entre ellos

$$F1 = 2.\frac{PPV.TPR}{PPV + TPR}$$

Verdaderos positivos (TP): Son aquellas noticias que fueron clasificadas como seguras (*safe*) o no seguras (*unsafe*).

$$TPR = \frac{TPR}{TP + FN}$$

Falsos positivos (FP): Es la proporción de ejemplos que fueron clasificados como pertenecientes a una clase, pero en realidad corresponden a otra.

$$FPR = \frac{FN}{FN + TP} = 1 - TPR$$

Para poder entender mejor la clasificación que se llevó a cabo Weka también nos despliega lo que se conoce como matriz de confusión, que es una forma sencilla de evaluar los resultados obtenidos. Esta matriz nos permite visualizar por medio de una tabla de contingencia la distribución de errores cometidos por un clasificador. En la Tabla 6 se observa un ejemplo de matriz de confusión.

Tabla 6. Modelo de matriz de confusión.

	Clase Predicha	
Clase Real	Safe	Unsafe
Safe	Verdaderos Positivos (TP)	Falsos Negativos (FN)
Unsafe	Falsos Positivos (FP)	Verdaderos Negativos (TN)

A continuación, en las Tablas 7, 8 y 9 se muestran los resultados obtenidos con las métricas de Precisión (*Precision*), Cobertura (*Recall*) y Medida-F (*F-Measure*) de la evaluación de los tres algoritmos utilizados en los enfoques de validación utilizados (*Cross Validation* y *Percentage Split*).

Tabla 7. Resultados de Precisión.

Precisión	**Algoritmos**					
	J48		Naïve Bayes		SVM	
Características	Cross Validation	Percentage split	Cross Validation	Percentage split	Cross Validation	Percentage split
estadísticas TF-IDF	0.74	0.711	0.629	0.646	0.729	0.717
léxicas	0.562	0.582	0.588	0.556	0.543	0.549
semánticas	0.581	0.564	0.59	0.578	**0.524**	0.535
léxicas + semánticas	0.575	0.565	0.609	0.588	0.579	0.585
estadísticas TF-IDF + léxicas	0.741	**0.742**	0.629	0.646	0.732	0.714
estadísticas TF-IDF + semánticas	0.718	0.716	0.629	0.646	0.727	0.713
estadísticas TF-IDF + léxicas + semánticas	0.741	0.738	0.629	0.646	0.728	0.715

En relación a la métrica de Precisión (*Precision*) en la Tabla 7 se observan mejores resultados con el algoritmo J48 con características estadísticas TF-IDF + léxicas.

Tabla 8. Resultados de Cobertura.

Cobertura	**Algoritmos**					
	J48		Naïve Bayes		SVM	
Características	Cross Validation	Percentage split	Cross Validation	Percentage split	Cross Validation	Percentage split
estadísticas TF-IDF	0.727	0.7	0.59	0.62	0.733	0.722
léxicas	0.595	0.608	0.611	0.578	0.557	0.561
semánticas	0.611	0.598	0.616	0.606	**0.493**	0.537
léxicas + semánticas	0.595	0.573	0.627	0.604	0.593	0.596
estadísticas TF-IDF + léxicas	0.742	**0.743**	0.59	0.62	0.736	0.718
estadísticas TF-IDF + semánticas	0.723	0.72	0.59	0.62	0.731	0.718
estadísticas TF-IDF + léxicas + semánticas	0.741	0.741	0.59	0.62	0.732	0.72

Como se puede observar en la Tabla 8 la métrica de Cobertura (*Recall*) tuvo resultados satisfactorios con características estadísticas TF-IDF + léxicas utilizando el algoritmo J48.

Tabla 9. Resultados de Medida-F.

Medida-F	**Algoritmos**					
	J48		Naïve Bayes		SVM	
Características	Cross Validation	Percentage split	Cross Validation	Percentage split	Cross Validation	Percentage split
estadísticas TF-IDF	0.73	0.703	0.594	0.623	0.729	0.716
léxicas	0.552	0.553	0.579	0.557	0.548	0.552
semánticas	0.546	0.536	0.553	0.541	**0.498**	0.536
léxicas + semánticas	0.575	0.568	0.603	0.587	0.582	0.588
estadísticas TF-IDF + léxicas	**0.742**	0.735	0.594	0.623	0.731	0.714
estadísticas TF-IDF + semánticas	0.717	0.711	0.594	0.623	0.726	0.712
estadísticas TF-IDF + léxicas + semánticas	0.741	0.737	0.594	0.623	0.726	0.714

En la tabla 9 se puede observar que se obtienen buenos resultados con respecto a la Medida-F (*F-Measure*), para la combinación de características estadísticas TF-IDF + léxicas con el algoritmo J48 utilizando una evaluación cruzada.

8. Análisis y discusión de resultados

De acuerdo a los resultados obtenidos se puede observar claramente que el algoritmo que tuvo la mejor ejecución con el valor de 0.742, siendo el más alto en Medida-F (*F-Measure*) fue el basado en arboles de decisión J48 utilizando el enfoque de validación cruzada (*Cross Validation*) con la combinación de características estadísticas más características léxicas.

El algoritmo de máquinas de soporte vectorial (*SVM*) fue el algoritmo que dio los resultados menos satisfactorios reportando un valor de 0.498 para Medida-F (*F-Measure*) igualmente en el enfoque de validación cruzada con las características semánticas. Entre los resultados obtenidos con los dos algoritmos antes mencionados se puede notar que existe una gran diferencia, esto se debe en parte a que las características semánticas extraídas no son muy representativas para el contexto del corpus con el que se trabajó. De las siete características semánticas que se pudieron reconocer, las organizaciones y las personas fueron las más representativas en el corpus, siendo las características restantes casi nulas, por esta razón los resultados no fueron favorables.

En el escenario de las características estadísticas más características léxicas fueron mucho mejores los resultados que se obtuvieron dado que es más precisa la frecuencia inversa de los términos (TF-IDF) que se adquiere con la extracción de características estadísticas y por otra parte en los títulos de noticias contenidos en el corpus por lo regular siempre se tendrán características léxicas muy representativas como los verbos, sustantivos y adjetivos.

9. Conclusiones

Al termino de este proyecto se logró hacer una clasificación emocional de títulos de noticias en español almacenados en un corpus con mil quinientos registros apoyándose en la etiqueta contenida en cada título de noticia, la cual puede ser segura (*safe*) o no segura (*unsafe*). Esta clasificación se pudo llevar a cabo con los algoritmos de aprendizaje automático J48, *Naïve Bayes* y Máquinas de Soporte Vectorial haciendo uso de los enfoques de validación *Cross Validation* y *Percentage Split.*

En la etapa de evaluación de resultados se detectó que el algoritmo basado en arboles de decisión J48 reporto el valor de 0.742 que es el mejor resultado en Medida-F (*F-Measure*), un valor de 0.742 para Cobertura *(Recall) y* 0.741 *para Precisión (Precision)*. El valor obtenido para Medida-F (*F-Measure*) representa el 74.2 % de las instancias correctamente clasificadas, este porcentaje alcanzado se logró utilizando el enfoque de validación cruzada (*Cross Validation*) y la combinación de características estadísticas + características léxicas. Es importante mencionar que la combinación de características extraídas fue de gran peso para obtener un resultado favorable, esto se debe al dominio del texto con el que se trabajó.

La comparativa entre los resultados obtenidos, la identificación del algoritmo que fue más efectivo en el contexto de noticias en español y la metodología utilizada en este proyecto de integración se considera una aportación para futuros proyectos en el área de procesamiento de lenguaje natural y en minería de textos en español. Estos proyectos pueden ser aplicaciones en el análisis de noticias que se utilicen como filtros dirigidos a los lectores y de igual manera para agilizar búsquedas de noticias.

10. Bibliografía

[1] J. Paniagua Reyes “Algoritmos de aprendizaje automático para el análisis de opiniones a partir de textos en español” Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2017.

[2] S. Anaya García “Análisis del comportamiento de diferentes algoritmos de aprendizaje automático” Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2018.

[3] N. Guzmán González, "Aplicación de Distintas Técnicas de Minería de Datos para el Tratamiento de Información", Licenciatura, Universidad Autónoma Metropolitana, 2011.

[4] F. Wanner, C. Rohrdantz, F. Mansmann, D. Oelke and D. Keim, "Visual Sentiment Analysis of RSS News Feeds Featuring the US Presidential Election in 2008", *Visual Interfaces to the Social and the Semantic Web*, 2009.

[5] J. Reyes Ortiz, F. Paniagua Reyes, B. Priego and M. Tovar, "LexFAR en la competencia TASS 2017: Análisis de sentimientos en Twitter basado en lexicones", *TASS 2017: Workshop on Semantic Analysis at SEPLN, septiembre 2017*, pp. 51-57, 2017.

[6] F. Paniagua Reyes, J. Reyes Ortiz and B. Priego Sánchez, "Evaluando un lexicón para la clasificación de polaridad a nivel de Tweet en español", Research in Computing Science, vol. 144, pp. 179–189, 2017.

[7] "¿Qué es el PLN o Procesamiento de Lenguaje Natural?", *El Huffington Post*, 2018. [Online]. Available: https://www.huffingtonpost.es/instituto-de-ingenieria-del-conocimiento/que-es-el-pln-o-procesamiento-de-lenguaje-natural_a_23253781/. [Accessed: 26- Dec- 2018].

[8] F. Caparrini and W. Work, "Introducción al Aprendizaje Automático - Fernando Sancho Caparrini", *Cs.us.es*, 2018. [Online]. Available: http://www.cs.us.es/~fsancho/?e=75. [Accessed: 24- Dec- 2018].

[9] R. Barrientos Martínez1 et al., "Árboles de decisión como herramienta en el diagnóstico médico", *Revista médica de la Universidad Veracruzana*, vol. 9, pp. 19-24, 2009.

[10] G. Betancourt, "Las máquinas de soporte vectorial (SVMs)", *Scientia et technica*, no.27, vol. 1, pp. 67-72, 2005.

[11] "TreeTagger-a part of speech tagger for many languages", *Cis.uni-muenchen.de*, 2018. [Online]. Available: http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/. [Accessed: 13- Jun- 2018].

[12] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java", *Cs.waikato.ac.nz*, 2018. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/. [Accessed: 03- Dec- 2018].

[13] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60. [pdf] [bib]

[14] "The Stanford Natural Language Processing Group", *Nlp.stanford.edu*, 2018. [Online]. Available: https://nlp.stanford.edu/software/spanish-faq.shtml. [Accessed: 03- Dec- 2018].

[15] "The Stanford Natural Language Processing Group", *Nlp.stanford.edu*, 2018. [Online]. Available: https://nlp.stanford.edu/software/CRF-NER.shtml. [Accessed: 03- Dec- 2018].

[16] "Cross-validation - Data Mining with Weka", *FutureLearn*, 2018. [Online]. Available: https://www.futurelearn.com/courses/data-mining-with-weka/0/steps/25384. [Accessed: 26- Dec- 2018].

[17] c*s.waikato.ac.nz*, 2019. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/mooc/dataminingwithweka/transcripts/Transcript2-2.txt. [Accessed: 06- Jan- 2019].

11. Apéndices

Apéndice A. Código para obtener el texto de los encabezados de noticias

```
public class Principal {

private static final String FILENAME = "corpus_head_tag.txt";
//Nombre del archivo de salida
public static void main (String[]args) {

//Se crea una lista para almacenar los nuevos valores
LinkedList <Conversor>lista = new LinkedList<Conversor>();
BufferedReader br = null;
/*Se lee el archivo linea por linea y solo se guardan en
  la lista los valores del encabezado y la etiqueta*/
try {
        br  =  new  BufferedReader(new  InputStreamReader(new  FileInputStream("corpus_inicial.txt"),
        "utf-8"));
        String line;
        int i=1;
while ((line = br.readLine()) != null) {
StringTokenizer stringTokenizer = new StringTokenizer(line, "\t");

        while (stringTokenizer.hasMoreElements()) {
        Conversor conv1 = new Conversor();
        String id = stringTokenizer.nextElement().toString();
        String pais = stringTokenizer.nextElement().toString();
        String url = stringTokenizer.nextElement().toString();
        String encabezado = stringTokenizer.nextElement().toString();
        String fecha = stringTokenizer.nextElement().toString();
        String etiqueta= stringTokenizer.nextElement().toString();

        conv1.setEncabezado(encabezado);
        conv1.setEtiqueta(etiqueta);

        lista.add(conv1);
        }
  }

        System.out.println("Done");

} catch (IOException e) {
    e.printStackTrace();
} finally {
try {
        if (br != null)
            br.close();

  } catch (IOException ex) {
     ex.printStackTrace();
  }
}
/*Se itera sobre la lista y se recupera cada elemento
  para poder guardarlo en un archivo nuevo*/
 try {

 BufferedWriter out = new BufferedWriter(new
 OutputStreamWriter(new FileOutputStream(FILENAME), "utf-8"));

 for(Conversor conv: lista )
    {
        out.write(conv.toString() + "\n");
    }

    System.out.println("Done");
    out.close();

   } catch (IOException e) {
```

```
    e.printStackTrace();
   }
 }
}
```

Apéndice B. Código para quitar números y signos de puntuación

```
public class Principal {

private static final String FILENAME = "corpus_sin_signos.txt";

public static void main(String[] args) {


        LinkedList <Registro>lista = new LinkedList<Registro>();
        BufferedReader br = null;
        try {
        br = new BufferedReader(new InputStreamReader(new FileInputStream("corpus_head_tag.txt"),
        "utf-8"));
        String line;

        while ((line = br.readLine()) != null) {
           StringTokenizer stringTokenizer = new StringTokenizer(line, "\t");

        while (stringTokenizer.hasMoreElements()) {
        Registro reg = new Registro();
        String encabezado1 =null;
        String encabezado = stringTokenizer.nextElement().toString();
        String etiqueta= stringTokenizer.nextElement().toString();

        encabezado1                    =                    encabezado.replaceAll("[0-9|:|;|$|¿|?|¡|!|+|-
        +"|*|/|#|(|)|=|%|,|_|\"|'|“|”|‘|’|[|]]","");
        encabezado1 = encabezado1.replaceAll("[^a-Za-Z|^á|^é"
                        +"|^í|^ó|^ú|^ü|^Á|^É|^Í|^Ó|^Ú|Ü|^ñ|^Ñ]", " ");
        reg.setEncabezado(encabezado1);
        reg.setEtiqueta(etiqueta);

        lista.add(reg);
                   }
                }
                System.out.println("Done");

        } catch (IOException e) {
                e.printStackTrace();
        } finally {
        try {
                if (br != null)
                        br.close();
        } catch (IOException ex) {
                ex.printStackTrace();
        }
}

        try {
        BufferedWriter      out      =      new      BufferedWriter(new      OutputStreamWriter(new
        FileOutputStream(FILENAME), "utf-8"));

                for(Registro conv: lista )
                {
                out.write(conv.toString()+"\n");
                }

                System.out.println("Done");
                out.close();

        } catch (IOException e) {
                e.printStackTrace();
        }

   }
```

```
}
```

Apéndice C. Código para eliminar palabras vacías

```
public String palabrasVacias(String entrada) {
final String FILENAME = "stopwords.txt";
String salida = "";
BufferedReader br = null;
//Descomposicion de la cadena entrada y almacenada en un arreglo
String[] result = entrada.split("\\s");

        try {
        br = new BufferedReader(new InputStreamReader(new           FileInputStream(FILENAME), "utf-
        8"));
        String line;
        String token="";

        while ((line = br.readLine()) != null)
        {
                StringTokenizer stringTokenizer = new StringTokenizer(line, "\n");

                while (stringTokenizer.hasMoreElements()) {
                token = stringTokenizer.nextElement().toString();

/* ciclo for que  compara la similitud de las cadenas almacenadas en el array con las stopwords
desde un archivo de texto */

        for (int x=0; x<result.length; x++)
        {
                if(result[x].compareTo(token)==0) {
                result[x]= "";
                }
        }
    }
  }
/*Ciclo for para concatenar las cadenas almacenadas en el array result*/
        for (int x=0; x<result.length; x++)
        {
                salida=salida.concat(result[x] + " ");
        }

} catch (IOException e) {
    e.printStackTrace();
} finally {
try {
        if (br != null)
        br.close();
  }catch (IOException ex) {
    ex.printStackTrace();
  }
}
return salida;
}
```

Apéndice D. Código para pasar el corpus a letras minúsculas

```
public class Principal {

private static final String FILENAME = "corpus_lematizado_minusculas.csv";

public static void main(String[] args) {

LinkedList <Registro>lista = new LinkedList<Registro>();
BufferedReader br = null;
try {
        br = new BufferedReader(new InputStreamReader(new
                        ("corpus_sin_signos.txt"), "utf-8"));
        String line;
```

```
while ((line = br.readLine()) != null) {

StringTokenizer stringTokenizer = new StringTokenizer(line, ",");

        while (stringTokenizer.hasMoreElements()) {

        Registro reg = new Registro();
        String encabezado1 =null;
        String etiqueta1 = null;
        String encabezado = stringTokenizer.nextElement().toString();
        String etiqueta= stringTokenizer.nextElement().toString();

        encabezado1 = encabezado.toLowerCase();
        etiqueta1 = etiqueta.toLowerCase();

        reg.setEncabezado(encabezado1);
        reg.setEtiqueta(etiqueta1);

        lista.add(reg);

   }
}

        System.out.println("Done");

} catch (IOException e) {
        e.printStackTrace();
} finally {
        try {
                if (br != null)
                 br.close();

        } catch (IOException ex) {
                ex.printStackTrace();
        }
}

try {
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(FILENAME), "utf-
8"));

for(Registro conv: lista )
        {
        out.write(conv.toString()+"\n");
        }
System.out.println("Done");
out.close();

} catch (IOException e) {
        e.printStackTrace();
  }
 }
}
```

Apéndice E. Código de lematizado

```
public class TreeTagger {
private static final String FILENAME = "corpus_lematizado.txt";
public static void main(String[] args) throws IOException, TreeTaggerException{
LinkedList <Registro>lista = new LinkedList<Registro>();
BufferedReader br = null;
try {
br = new BufferedReader(new InputStreamReader(new FileInputStream("corpus_stopwords.txt"), "utf-
8"));
String line;
while ((line = br.readLine()) != null) {
StringTokenizer stringTokenizer = new StringTokenizer(line, ",");
        while (stringTokenizer.hasMoreElements()) {
```

```
        Registro reg = new Registro();
        String encabezado = stringTokenizer.nextElement().toString();
        String[] cadena = encabezado.split("\\s");
        String etiqueta= stringTokenizer.nextElement().toString();
        String encabezado1 = Etiquetar(cadena);
        reg.setEncabezado(encabezado1);
        reg.setEtiqueta(etiqueta);
        lista.add(reg);
        }
}

System.out.println("Done");
} catch (IOException e) {
        e.printStackTrace();
} finally {
        try {
                if (br != null)
                    br.close();
        } catch (IOException ex) {
                ex.printStackTrace();
        }
}
try {
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(FILENAME), "utf-
8"));
        for(Registro conv: lista )
        {
        out.write(conv.toString()+"\n");
        }
        System.out.println("Done");
        out.close();
} catch (IOException e) {
        e.printStackTrace();
        }
}

public static String Etiquetar(String [] cadenaEntrada) throws IOException, TreeTaggerException{

final ArrayList<Token> tokens = new ArrayList<Token>();
String salida="";
// Point TT4J to the TreeTagger installation directory. The executable is expected
// in the "bin" subdirectory - in this example at "/opt/treetagger/bin/tree-tagger"
System.setProperty("treetagger.home", "recursos/TreeTagger");
TreeTaggerWrapper<String> tt = new TreeTaggerWrapper<String>();
try {                tt.setModel("recursos/TreeTagger/modelos/sp.par:iso8859-1");
tt.setHandler(new TokenHandler<String>() {
        public void token(String token, String pos, String lemma) {
     tokens.add(new Token(lemma));
     }
});
        tt.process(asList(cadenaEntrada));
}
finally {
        tt.destroy();
}
for (int x=0; x<tokens.size(); x++)
{
salida=salida.concat(tokens.get(x) + " ");
}

      return salida;
    }
}
```

Apéndice F. Código de pesado de características léxico-sintácticas

```
public class NLPTest {
public Registro pesado(String text) {
Registro conteo = new Registro();
```

```
Properties props=new Properties();
props.setProperty("annotators","tokenize, ssplit, pos,lemma");
props.setProperty("ssplit.isOneSentence", "true");
props.setProperty("pos.model", "edu/stanford/nlp/models/pos-tagger/spanish/spanish-distsim.tagger");
props.setProperty("parse.model", "edu/stanford/nlp/models/srparser/spanishSR.ser.gz");
props.setProperty("tokenize.language", "es");
StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
float divLexica = 0;
float longVocabulario=0;
Integer numVerbos = 0;
Integer numSustantivos = 0;
Integer numAdjetivos = 0;
Integer numDeterminantes = 0;
Integer numPalabrasCapitalizadas=0;
String[] longVoc = text.split("\\s");
LinkedList <String> sinRepetir = new LinkedList();
sinRepetir.add(0, longVoc[0]);

for(int j=1; j<longVoc.length; j++)
{
        if(sinRepetir.contains(longVoc[j])==false)
        {
        sinRepetir.add(longVoc[j]);
        }
}
longVocabulario = sinRepetir.size();
divLexica = (longVocabulario/longVoc.length);

conteo.setDivLexica(divLexica);
conteo.setLongVocabulario(longVocabulario);
Annotation document = new Annotation(text);
// run all Annotators on this text
pipeline.annotate(document);
// these are all the sentences in this document
// a CoreMap is essentially a Map that uses class objects as keys and has values with custom types
List<CoreMap> sentences = document.get(SentencesAnnotation.class);
        for(CoreMap sentence: sentences)
        {
        // traversing the words in the current sentence
        // a CoreLabel is a CoreMap with additional token-specific methods
                   for (CoreLabel token: sentence.get(TokensAnnotation.class)) {
                  // this is the text of the token
                   String word = token.get(TextAnnotation.class);
                   // this is the POS tag of the token
                   String pos = token.get(PartOfSpeechAnnotation.class);
                        if(pos.startsWith("v"))
                                numVerbos++;
                        if(pos.startsWith("nc"))
                                numSustantivos++;
                        if(pos.startsWith("a"))
                                numAdjetivos++;
                        if(pos.startsWith("d"))
                                numDeterminantes++;
                        if(pos.startsWith("np"))
                                numPalabrasCapitalizadas++;
                  }
                  conteo.setNumVerbos(numVerbos);
                  conteo.setNumSustantivos(numSustantivos);
                  conteo.setNumAdjectivos(numAdjetivos);
                  conteo.setNumDeterminantes(numDeterminantes);
                  conteo.setNumCapitalizadas(numPalabrasCapitalizadas);
                }
                return conteo;
        }
}
```

Apéndice G. Código de pesado de características semánticas

```
public class NLPTest {
public Registro pesado(String text) {
Registro conteo = new Registro();
Properties props=new Properties();
props.setProperty("annotators","tokenize, ssplit, pos, lemma, ner");
props.setProperty("ssplit.isOneSentence", "true");
props.setProperty("pos.model", "edu/stanford/nlp/models/pos-tagger/spanish/spanish-distsim.tagger");
props.setProperty("ner.model",
"edu/stanford/nlp/models/ner/spanish.ancora.distsim.s512.crf.ser.gz");
props.setProperty("parse.model", "edu/stanford/nlp/models/srparser/spanishSR.ser.gz");
props.setProperty("tokenize.language", "es");
// build pipeline
StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

Integer numOrganizaciones = 0;
Integer numPersonas = 0;
Integer numCiudades = 0;
Integer numPorcentajes = 0;
Integer numTiempos = 0;
Integer numFechas = 0;
Integer numMonedas = 0;
// read some text in the text variable
Annotation document = new Annotation(text);
// run all Annotators on this text
pipeline.annotate(document);
// these are all the sentences in this document
// a CoreMap is essentially a Map that uses class objects as keys and has values with custom types
List<CoreMap> sentences = document.get(SentencesAnnotation.class);
for(CoreMap sentence: sentences) {
// traversing the words in the current sentence
// a CoreLabel is a CoreMap with additional token-specific methods
for (CoreLabel token: sentence.get(TokensAnnotation.class)) {
// this is the text of the token
String word = token.get(TextAnnotation.class);
// this is the POS tag of the token
//String pos = token.get(PartOfSpeechAnnotation.class);
// this is the NER label of the token
String ne = token.get(NamedEntityTagAnnotation.class);
        if(ne.equals("ORG"))
        numOrganizaciones++;
        if(ne.equals("PERS"))
        numPersonas++;
        if(ne.equals("LUG"))
        numCiudades++;
        if(ne.equals("PERCENT"))
        numPorcentajes++;
        if(ne.equals("TIME"))
        numTiempos++;
        if(ne.equals("DATE"))
        numFechas++;
        if(ne.equals("MONEY"))
        numMonedas++;
}

        conteo.setNumOrganizaciones(numOrganizaciones);
        conteo.setNumPersonas(numPersonas);
        conteo.setNumCiudades(numCiudades);
        conteo.setNumPorcentajes(numPorcentajes);
        conteo.setNumTiempos(numTiempos);
        conteo.setNumFechas(numFechas);
        conteo.setNumMonedas(numMonedas);
        }
        return conteo;
  }
}
```