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

Reporte de Proyecto de Investigación

Minería de opiniones sobre textos en Twitter

Presenta:
Luis Eduardo Poot Terán
2123029543

Asesores

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

Angeles Belém Priego Sánchez
Doctora en Ciencias del Lenguaje
Profesora Asociada
Departamento de Sistemas
abps@azc.uam.mx

Trimestre 2018 – Otoño

14 de Diciembre del 2018

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, Angeles Belém Priego Sánchez, 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.

Dra. Angeles Belém Priego Sánchez

Yo, Luis Eduardo Poot Terán, 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.

Luis Eduardo Poot Terán

Resumen

Actualmente, la información se encuentra en constante expansión gracias a la posibilidad que ha proporcionado el Internet y las herramientas de comunicación que éste tiene. Con lo cual, es posible obtener desde textos científicos hasta opiniones que una persona tiene sobre cierto producto. Sin embargo, conocer e interpretar toda esta información, principalmente los comentarios, no es tarea sencilla por lo que se hace necesaria la construcción de herramientas que permitan beneficiar y ayudar con dicha tarea. Una de las áreas de la computación que permite enlazar textos escritos con su posible análisis es el Procesamiento de Lenguaje Natural, el cual a través de diversos métodos permite obtener el comportamiento de un individuo, influir en su toma de decisiones de forma imperceptible, por citar algunos ejemplos. En este trabajo se pretende ayudar con la creación de una herramienta, denominada "Analizador de Opiniones", que permite procesar comentarios de Twitter con el fin de analizar y clasificar los textos escritos emitidos por los usuarios de un producto.

Analizador de opiniones es una aplicación de Internet enriquecida[1] que trabaja en tiempo real y permite clasificar opiniones a partir de una frase de búsqueda y mediante la obtención de la acción de ésta, es decir, mediante la identificación del verbo. En el presente trabajo se exhibe el proceso de desarrollo, implementación y pruebas que permitió el origen de *Analizador de opiniones.*

[1] Se denomina aplicación de Internet enriquecida a la aplicación hibrida web-escritorio.

Contenido

Índice de Figuras

Índice de Tablas

CAPÍTULO 1. Acercamiento teórica

1.1 Introducción

Día a día la cantidad de información generada por la humanidad va en aumento, desde textos científicos hasta la opinión de una persona sobre un producto. Por lo que su análisis puede darnos resultados útiles para afrontar problemas actuales. Sin embargo, obtener estos resultados no es tarea sencilla; en consecuencia, se han desarrollado nuevas tecnologías y herramientas.

El análisis de texto surge del Procesamiento del Lenguaje Natural. El cual "pretendía que una computadora interpretara cualquier texto con el fin de encontrar la semántica asociada al conjunto de palabras que lo conforman" [1]. A lo largo de los años se ha utilizado el análisis de texto para extraer patrones basados en palabras y temas significativos, obteniendo datos cuantitativos. Logrando por ejemplo predicciones sobre el comportamiento de un individuo, así también influyendo en su toma de decisiones de forma imperceptible.

Las redes sociales pueden brindar gran cantidad de información sobre gustos, opiniones, noticias, eventos, entre otros. Las personas comparten gran parte de sus actividades diarias en las redes sociales, esta información puede ser usada por dependencias, como empresas del sector privado para saber si un producto es del agrado de sus consumidores o un partido político para conocer si sus propuestas son aceptas por la mayoría de la población.

En el presente proyecto se desarrollará un sistema web que analizará opiniones en la red social Twitter. Para llevar acabo dicho análisis se deben tener en cuenta dos aspectos fundamentales: el tema a buscar y su valoración. En el primer aspecto se descargarán tweets con el objeto de búsqueda. El segundo aspecto examinará la información de cada tweet transformándolo por medio de lexicones (listas de palabras obtenidas del mismo tweet). Se analizarán los lexicones obtenidos para llegar a una respuesta porcentual sobre el objeto de búsqueda y su valoración.

1.2 Antecedentes

Se presentan algunos proyectos y artículos relacionados particularmente con la extracción y análisis de opiniones.

1.2.1 Proyectos terminales

- *Algoritmos de aprendizaje automático para el análisis de opiniones a partir de textos en español [2].*

El proyecto terminal consta de la aplicación de 4 diferentes procesos de minería de textos: K-vecinos más cercanos (KNN), Bayes Naïve, J48, Maquinas de Soporte Vectorial (SVM). Para el análisis de textos en español extraídos de Twitter. Los procesos de minería J48 y Bayes Naïve también serán usados en este proyecto pero serán aplicados a tweets y tendrá un objeto de búsqueda especificado.

- *Extracción y Representación de servicios web escritos en SAWSDL[1] mediante una ontología [3].*

  El proyecto terminal desarrolla una aplicación que permite encontrar elementos relevantes dentro de distintos lenguajes de descripción de servicios, para transformarlos a un modelo ontológico general. Del cual se genera el objetivo para invocar el servicio de forma remota. La similitud del proyecto es la búsqueda específica de modelos y su transformación. Difiere en que el proyecto busca eventos de lenguaje de descripción de servicios, mientras que la propuesta busca en opiniones sobre la plataforma de Twitter.

1.2.2 Artículos

- *Minería de Opiniones basado en la adaptación al español de ANEW sobre opiniones acerca de hoteles [4]*

El articulo habla sobre la minería de opiniones y su alto grado de investigación en los últimos años. Plantea un sistema que extraerá, procesará e identificará sentimientos para clasificar las opciones. La similitud del proyecto es la extracción y la identificación de la opinión, pero difiere en que se usará para el caso específico de opiniones sobre hoteles.

- *Machine Learning Algorithms for Opinion Mining and Sentiment Classification. [5]*

Este artículo aborda el tema de minería de opiniones y análisis de sentimientos, lo cual permite identificar las opiniones de los usuarios expresando comentarios positivos, negativos o neutrales y citas subyacentes al texto. La similitud se encuentra en la búsqueda de comentarios con opiniones propias de los usuarios, para llegar a un resultado favorable o desfavorable del tema a buscar, el proyecto que se presenta se centrará en la extracción de comentarios de la red social Twitter y no considerará una clasificación de los sentimientos de los usuarios.

- *Algorithms for Opinion Mining and Sentiment Analysis: An Overview. [6]*

El artículo proporciona información sobre la importancia y la usanza de la información publicada en redes sociales, así como una forma de hacerlo. La similitud radica en el interés por el análisis de las opiniones dadas por individuos que utilizan plataformas sociales para expresarlas, sin embargo, difiere en que se usaran los algoritmos J48 y Bayes Naïve para la clasificación de tweets y que se enfoca en un análisis más detallado de los sentimientos y de las opiniones.

La Tabla 1 muestra un cuadro comparativo sobre las similitudes y diferencias de cada uno de los proyectos y artículos consultados como apoyo para este proyecto.

Tabla 1. Tabla comparativa de proyectos y artículos consultados

Referencia	Similitud	Diferencias
[2]	• Análisis de texto a través de tweets. • Uso de algoritmos J48 y Bayes Naïve	• Aplicación de un objetivo de búsqueda.
[3]	• Búsqueda especifica de modelos. • Transformación de modelos.	• Búsqueda de eventos de lenguaje de descripción de servicios
[4]	• Extracción de opiniones. • Identificación de opiniones.	• Uso para el caso específico de opiniones sobre hoteles.
[5]	• Búsqueda de comentarios con opiniones usuarios.	• Clasificación específica de los sentimientos de los usuarios.
[6]	• Análisis de opiniones dadas por individuos en plataformas sociales.	• Tipo de algoritmos usados para el análisis. • Análisis y clasificación de sentimientos y opiniones.

1.3 Justificación

El desarrollo de este proyecto surge debido a la popularidad y el uso de las redes sociales por personas de diversas edades y clases sociales. En donde las personas plasman su juicio sobre un producto o servicio de una manera más sencilla y recurrente. Es más fácil escribir un tweet de lo que se opina que mandar un email o hacer una llamada. Por lo que recabar y analizar esta información es más sencillo para las empresas.

Esta plataforma puede tener alcances incluso de estudiar y predecir el comportamiento humano basado en lo que escribe, por ejemplo, conocer el estado anímico de un individuo para evitar atentados en escuelas como lo es en Estados Unidos de Norte América, evitar el suicidio de un adolescente, por citar algunos ejemplos.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar un sistema para el análisis de opiniones en la red social Twitter.

1.4.2 Objetivos específicos

- Diseñar e implementar un módulo para extracción de tweets.
- Diseñar e implementar un módulo para la lematización del tweet.
- Diseñar e implementar un módulo para la clasificación del tweet basado en la valoración.
- Diseñar e implementar un módulo para el análisis y visualización del resultado.

CAPÍTULO 2. Marco teórico

En este capítulo se abordan las herramientas y el conocimiento con los cuales está desarrollado el proyecto.

2.1 Java

Java es un lenguaje de programación orientado a objetos que fue diseñado para que no se recompile el código en cada plataforma. Java también es una plataforma informática que fue comercializada en 1995 por Sun MicroSystems.

2.1.1 ¿Porque utilizar Java?

- Java es de código abierto

El código que hace funcionar a Java es de libre acceso, es decir cualquier usuario puede estudiarlo y modificarlo. Esto ha llevado a que Java este en un ciclo de mejora continua.

- Java es multiplataforma

Java no necesita compilar el código en cada plataforma donde se ejecute, es decir se compila solo una vez y puede utilizarse en plataforma distintas como Linux, Windows y Mac.

- Java cuenta con muchas librerías

La comunidad de Java está constantemente creando librerías para todo tipo se usos.

2.1.2 JavaFX

JavaFx permite crear aplicaciones de internet enriquecidas (RIA) que tienen el mismo comportamiento en distintas plataformas. JavaFX está basado en Java por lo que se puede usar cualquier librería de Java en JavaFX.

2.1.3 FXML

FXML es un lenguaje de marcado similar a HTML y XML en el cual está basado. Fue creado para definir la interfaz de usuario cuando se crea una aplicación JavaFX

2.2 Entorno de Desarrollo

2.2.1 NetBeans IDE 8.1

NetBeans es un entorno de desarrollo integrado (IDE) creado principalmente para Java. NetBeans es un IDE de código abierto que incorpora la gestión de interfaz de usuario, la gestión de configuración de usuario, la gestión del almacenamiento, la gestión de ventana, soporta diálogos paso a paso, contiene herramientas de desarrollo integrado y además soporta JavaFX [7].

2.3 TreeTagger

TreeTagger es una herramienta que permite anotar texto con descripción acerca de las etiquetas POS y lema. Desarrollado por Helmut Schmid en el proyecto TC en el Instituto de Lingüística Computacional de la Universidad de Stuttgart [9]. El paquete de software se compone de dos aplicaciones. La primera es de entrenamiento y sirve para crear archivos de parámetros a partir de un corpus y un lexicón. La segunda aplicación llamada *tagger,* etiqueta textos con etiquetas POS y lema, de acuerdo al parámetro de entrada.

2.4 Redes Sociales

Una red social es un sitio de Internet formado por comunidades de individuos, que tiene como fin el contacto entre ellos en un espacio virtual. De manera que puedan comunicarse o compartir información, como puede ser imágenes, videos, noticias entre otros.

2.4.1 Twitter

Twitter es una red social que permite enviar mensajes de máximo 280 caracteres llamados tweets, estos son mostrados en la página principal del usuario. Cada usuario puede suscribirse a otro, a esto se le llama “seguir”, este usuario “seguidor” se le denomina

follower. En la página de inicio se muestran los tweets de los usuarios a los cuales se está subscrito.

2.4.2 Tweets

Por defecto los tweets son públicos, pueden ser difundidos de manera privada para un grupo de *followers* determinados. Los usuarios pueden marcar como favorito los tweets ya sean suyos o de otro usuario, también tienen la facultad de comentar cualquier tweet que no sea privado, a su vez pueden mostrar los tweets de otro usuario en su página principal. A este último se le denomina "retweet" y es representado por las letras "RT". Se pueden escribir tweets desde la página de internet de Twitter o desde la aplicación para teléfonos y tabletas inteligentes.

Cada tweet está representado por: la imagen de perfil del usuario, el nombre del usuario, el identificar único de usuario, la fecha de creación del tweet, el mensaje del tweet, el icono para comentar, el icono para hacer un "retweet y el icono para marcar como favorito. En la Figura 1 se muestra un ejemplo de la estructura de un tweet.

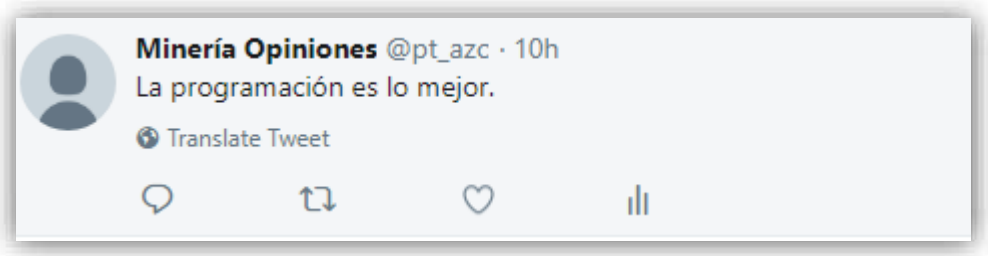

Figura 1. Estructura de un tweet

CAPÍTULO 3. Desarrollo del proyecto

En este capítulo se presenta la metodología usada en el desarrollo del proyecto. El capítulo se divide en cuatro partes, cada parte corresponde a un módulo del sistema el cual a su vez está dividido en varias secciones.

La construcción del sistema realizado se basó en el modelo presentado en la Figura 2. Este modelo está compuesto por 4 módulos: extracción, lematización, clasificación y análisis y resultado.

Figura 2.Modelo del sistema

3.1 Módulo de extracción de tweets

Este módulo se compone de dos etapas; la primera se encarga de la conexión a los servidores de Twitter y la segunda de la extracción de los tweets.

3.1.1 Conexión a Twitter

Inicialmente se creó una cuenta de desarrollador de Twitter. Posteriormente se hizo el registro del proyecto en la página de Twitter como se muestra en la Figura 3.

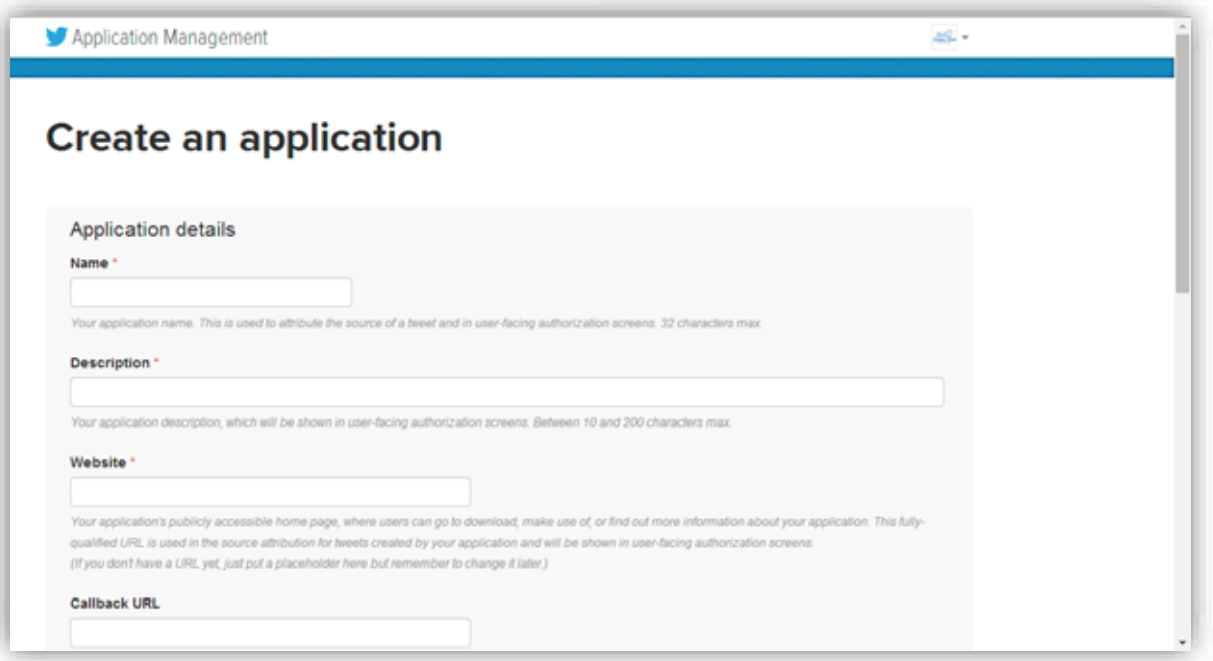

Figura 3. Formulario de registro de aplicación o proyecto en Twitter

Cuando el proyecto es aceptado se reciben cuatro llaves las cuales son: *Consumer Key, Consumer secret, Acces token* y *Acces token secret*. Todas estas llaves son proporcionadas por Twitter, son únicas y son totalmente confidenciales. Estas llaves permiten el acceso del proyecto a los servidores de Twitter. En la Figura 4 se muestra el código utilizado para conexión a Twitter.

```
public void Tweet() throws TwitterException{
      Twitter twitter;
       ConfigurationBuilder cb = new ConfigurationBuilder();
       cb.setDebugEnabled(true)
                  .setOAuthConsumerKey("Consumer Key")
                  .setOAuthConsumerSecret("Consumer Secret")
                  .setOAuthAccessToken("Access Token")
                  .setOAuthAccessTokenSecret("Access Token Secret");
       twitter = new TwitterFactory(cb.build()).getInstance();

   }
```

Figura 4. Clase usada para la conexión a Twitter

3.1.2 Extracción de tweets

En esta etapa se extraen los tweets, que coincidieron con el tema de búsqueda proporcionado por el usuario. A cada tweet se le da un formato especifico y es almacenado en un archivo de texto nombrado "Busqueda.txt". Solo pueden descargarse cien tweets por día, y este conteo es reiniciado a las 24 horas de cada día. Existen distintos planes de pago por parte de Twitter, en lo que permite la descarga de un mayor número de tweets o en algunos casos es ilimitado. En la Figura 5 se muestra la clase usada para extraer los tweets y darles formato.

```
public class TweetEx {
    private Twitter twitter;
    private SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

    public TweetEx() {
        twitter = TwitterFactory.getSingleton();
    }

    public List<Status> query(String query) throws TwitterException {
        QueryResult search = twitter.search(new Query(query));
        List<Status> tweets = search.getTweets();
        return tweets;
    }

    public void escribirTweet(Status status) throws IOException {
        File A = new File("/Resultado/Busqueda.txt");
        FileWriter w = new FileWriter(A);
        BufferedWriter bw = new BufferedWriter(w);
        PrintWriter wr = new PrintWriter(bw);
        wr.write(String.format(status.getUser().getScreenName()+" "+sdf.format(status.getCreatedAt())+"\n");
        wr.write(status.getText()+"\n\n");
    }

    public void escribirTweet(List<Status> status) throws IOException {
        for (Status tweet : status) {
            escribirTweet(tweet);
        }
    }

}
```

Figura 5. Clase usada para extracción de tweets

3.2 Módulo de lematización del tweet

Este módulo está compuesto de tres etapas: la primera que se encarga de identificar los metadatos de los tweets, la segunda que le da un formato a los tweets para distinguirlos unos de otros y la tercera que se encarga de procesar los tweets. Denominadas análisis, etiquetado y lematización del corpus, respectivamente. A continuación, se detallan dichas etapas.

3.2.1 Análisis del corpus

Al observar el corpus se identificaron los metadatos de los tweets los cuales están conformados por:

- Apodo o *Nickname* que el usuario elige y puede cambiar.
- Seguido por un nombre de usuario o *Username* el cual es único e inicia con el carácter "@".
- Posteriormente, se encuentra la fecha del día que se escribió el tweet y se presenta en el formato "dd mes aaaa".
- Por último, se encuentra el Tweet escrito por el usuario.

3.2.2 Etiquetado del corpus

Una vez identificados los metadatos de los tweets, hasta el momento el corpus en texto plano, el siguiente paso fue el etiquetamiento. Cada tweet fue etiquetado de la siguiente forma:

<INICIOTWEET> Tweet X</FINTWEET>

Donde la etiqueta **<INICIOTWEET>** es la marca de inicio del tweet, delimitado por el patrón de metadatos: apodo-usuario-fecha en la estructura general del tweet. La etiqueta **Tweet X** corresponde al texto escrito por el usuario. Finalmente, la etiqueta **</FINTWEET>** determina el final del tweet mediante el mismo patrón mencionado con anterioridad. Cabe mencionar que el tweet de inicio y el tweet de fin no contienen este patrón, a pesar de este hecho son considerados dado que son los que delimitan el texto plano.

3.2.3 Lematización del corpus

Una vez que el corpus ha sido etiquetado, se procede a identificar las palabras separadas por un espacio en blanco de cada oración en los tweets; este proceso se denomina *Tokenización*. El resultado de la *tokenización* es un nuevo corpus, el cual facilita la lematización[1] de los tweets. En la Figura 6 se puede observar un ejemplo del proceso de *tokenización*.

[1] Lematización: proceso que permite obtener el lema de una palabra.

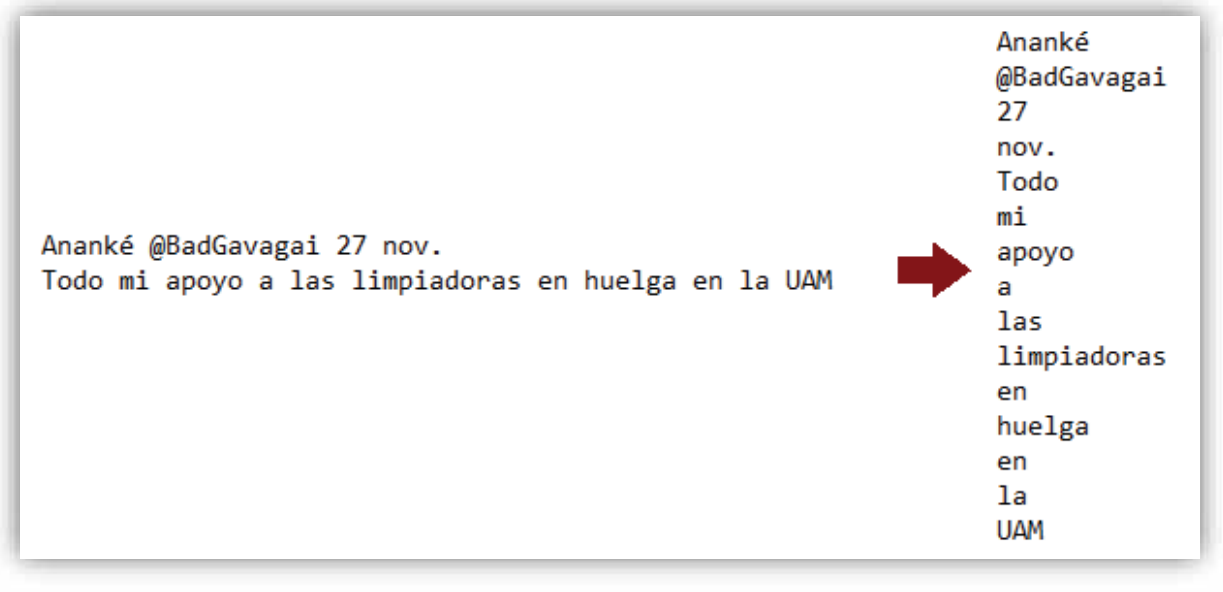

Figura 6. Tokenización de un Tweet

Una vez generado el nuevo corpus, se analizó cada token con TreeTager (descrito en la Sección 2.3 del Capítulo 2). El análisis devuelve tres columnas separadas por tabulaciones. Estas columnas son los atributos posicionales:

word /token Etiqueta POS Lemma

La primera columna contiene el token (word /token), la segunda columna contiene la categoría gramatical a base de etiquetas que se pueden encontrar en los tagset [1] de TreeTagger (Etiqueta POS) y la última columna representa el lema (Lemma).

En la Tabla 2 se muestra el tagset de TreeTagger para el idioma español.

Tabla 2.Documentación tagset para el idioma español

Etiqueta	Categoría gramatical
ACRNM	acronym (ISO, CEI)
ADJ	Adjectives (mayores, mayor)
ADV	Adverbs (muy, demasiado, cómo)
ALFP	Plural letter of the alphabet (As/Aes, bes)
ALFS	Singular letter of the alphabet (A, b)
ART	Articles (un, las, la, unas)

[1]Descargables desde : http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

Etiqueta	Categoría gramatical
BACKSLASH	backslash (\)
CARD	Cardinals
CC	Coordinating conjunction (y, o)
CCAD	Adversative coordinating conjunction (pero)
CCNEG	Negative coordinating conjunction (ni)
CM	comma (,)
CODE	Alphanumeric code
COLON	colon (:)
CQUE	que (as conjunction)
CSUBF	Subordinating conjunction that introduces finite clauses (apenas)
CSUBI	Subordinating conjunction that introduces infinite clauses (al)
CSUBX	Subordinating conjunction underspecified for subord-type (aunque)
DASH	dash (-)
DM	Demonstrative pronouns (ésas, ése, esta)
DOTS POS	tag for "..."
FO	Formula
FS	Full stop punctuation marks
INT	Interrogative pronouns (quiénes, cuántas, cuánto)
ITJN	Interjection (oh, ja)
LP	left parenthesis ("(", "[")
NC	Common nouns (mesas, mesa, libro, ordenador)
NEG	Negation
NMEA	measure noun (metros, litros)
NMON	month name
NP	Proper nouns
ORD	Ordinals (primer, primeras, primera)
PAL	Portmanteau word formed by a and el
PDEL	Portmanteau word formed by de and el
PE	Foreign word
PERCT	percent sign (%)
PNC	Unclassified word
PPC	Clitic personal pronoun (le, les)
PPO	Possessive pronouns (mi, su, sus)
PPX	Clitics and personal pronouns (nos, me, nosotras, te, sí)
PREP	Negative preposition (sin)
PREP	Preposition
PREP/DEL	Complex preposition "después del"
QT	quotation symbol (" ' `)

Etiqueta	Categoría gramatical
QU	Quantifiers (sendas, cada)
REL	Relative pronouns (cuyas, cuyo)
RP	right parenthesis (")", "]")
SE	Se (as particle)
SEMICOLON	semicolon (;)
SLASH	slash (/)
SYM	Symbols
UMMX	measure unit (MHz, km, mA)
VCLIger	clitic gerund verb
VCLIinf	clitic infinitive verb
VCLIfin	clitic finite verb
VEadj	Verb estar. Past participle
VEfin	Verb estar. Finite
VEger	Verb estar. Gerund
VEinf	Verb estar. Infinitive
VHadj	Verb haber. Past participle
VHfin	Verb haber. Finite
VHger	Verb haber. Gerund
VHinf	Verb haber. Infinitive
VLadj	Lexical verb. Past participle
VLfin	Lexical verb. Finite
VLger	Lexical verb. Gerund
VLinf	Lexical verb. Infinitive
VMadj	Modal verb. Past participle
VMfin	Modal verb. Finite
VMger	Modal verb. Gerund
VMinf	Modal verb. Infinitive
VSadj	Verb ser. Past participle
VSfin	Verb ser. Finite
VSger	Verb ser. Gerund
VSinf	Verb ser. Infinitive

Cabe mencionar que TreeTagger está entrenado para analizar muchos idiomas entre ellos el español. Sin embargo, para que TreeTagger pueda analizar de forma correcta un corpus, se debe descargar el archivo de parámetros o modelo del idioma deseado y señalizar en el código del programa la ruta de acceso al modelo como se muestra en la Figura 7.

```
tt.setModel("/treetagger/lib/spanish.par");
```

Figura 7. Selección del modelo de idioma

A continuación, la Figura 8 muestra un ejemplo de la lematizacion para la frase “Amo a la UAM” por medio de TreeTagger, dicha oración ya tiene el proceso de tokenización. La lematización proporciona una división en tres columnas: *token-etiqueta-lema*.

```
Amo     VLfin   amar
a       PREP    a
la      ART     el
UAM     NP      <unknown>
```

Figura 8. Archivo etiquetado con TreeTagger

3.3 Módulo de clasificación del tweet basado en la valoración

Este módulo está compuesto de dos etapas, la primera es la identificación del verbo principal de la búsqueda y la segunda la clasificación de los tweets a partir de este verbo.

3.3.1 Identificación del verbo principal

En este proceso se analizó la oración de búsqueda. El primer paso fue tokenizar la oración. Posteriormente, se analizan los token con TreeTagger para lematizarlos. Una vez lematizada la oración se identificó el verbo con ayuda de las etiquetas y se seleccionó el lema.

Tomando como ejemplo la figura 8, se puede observar en las columnas de atributos que el verbo de búsqueda corresponde al primer token y de acuerdo a la etiqueta “VLfin” (ver tabla 1) es un verbo léxico en infinitivo. Por lo que se selecciona el lema que es “amar” como verbo principal.

3.3.2 Clasificación de los tweets

En esta parte se clasificaron los tweets de acuerdo al verbo principal de búsqueda, resultado obtenido en una etapa preliminar, mediante la trasformación de este en su antónimo. Para la clasificación de los tweets se realizó una búsqueda sobre el corpus de tweets lematizados, en la que se deseaba encontrar el verbo original o su antónimo, obteniendo dos corpara para cada una de las búsquedas efectuadas en Twitter. De esta manera, se crean los corpora de tweets donde en el primer grupo se encuentran los tweets que contienen la valoración original (verbo principal) denominado "Corpus_Positivo" y en el segundo los tweets que contienen su antónimo denominado "Corpus_Negativo". En la Tabla 3 se observa un ejemplo de la clasificación de los tweets con la búsqueda "Amo a la UAM".

Tabla 3. Clasificación del verbo amar

Verbo original lematizado	Veces encontrado el verbo original	Veces encontrado el antónimo del verbo original
Amar	7	1

3.4 Módulo del análisis y visualización del resultado.

Este módulo se compone de tres etapas, la primera es el análisis de los dos corpus de Tweets, el segundo la representación gráfica del sistema y el tercero el funcionamiento del sistema.

3.4.1 Análisis del corpora

En esta parte se realizó un conteo de los tweets que se encontraban en cada corpora (Corpus_Positivo y Corpus:Negativo). Posteriormente, se hizo la sustracción del total de la suma de los dos grupos con el total de tweets descargados, esto para poder verificar el porcentaje de error. Es decir, cuántos tweets no se clasificaron. Este proceso, se puede expresar con la siguiente ecuación, denominada "Error de clasificación".

$$Erc = Td - (Tp+Tn)$$

Donde:

- `Erc` corresponde al resultado de los tweets erróneamente clasificados
- `Td` el total de tweets descargados
- `Tp` el número de tweets del Corpus_Positivo
- `Tn` el número de tweets del Corpus_Negativo

En la Tabla 4 se muestra un ejemplo de la distribución de la ecuación “Error de clasificación” siguiendo el ejemplo de la Tabla 3.

Tabla 4. Distribución de la ecuación "Error de clasificación"

Total de tweets descargados	Número de tweets del Corpus_Positivo	Número de tweets del Corpus_Negativo	Tweets erróneamente clasificados	Porcentaje de tweets erróneamente clasificados
30	17	4	9	30

3.4.2 Interfaz de usuario

En esta etapa se muestra la interfaz del sistema y sus componentes. En la Figura 9 se muestra la pestaña principal y en la cual está alojado todo el sistema. En la parte superior se encuentra una entrada de texto, donde el usuario escribe la oración de búsqueda. También se observa una leyenda debajo de la entrada de texto que indica el límite de tweets que se pueden descargar diariamente. A un costado esta un menú desplegable en el cual el usuario selecciona la cantidad de tweets que desea descargar. Posterior al menú desplegable se encuentra el botón de "buscar" que se encarga de activar el proceso de conexión y extracción de Tweets

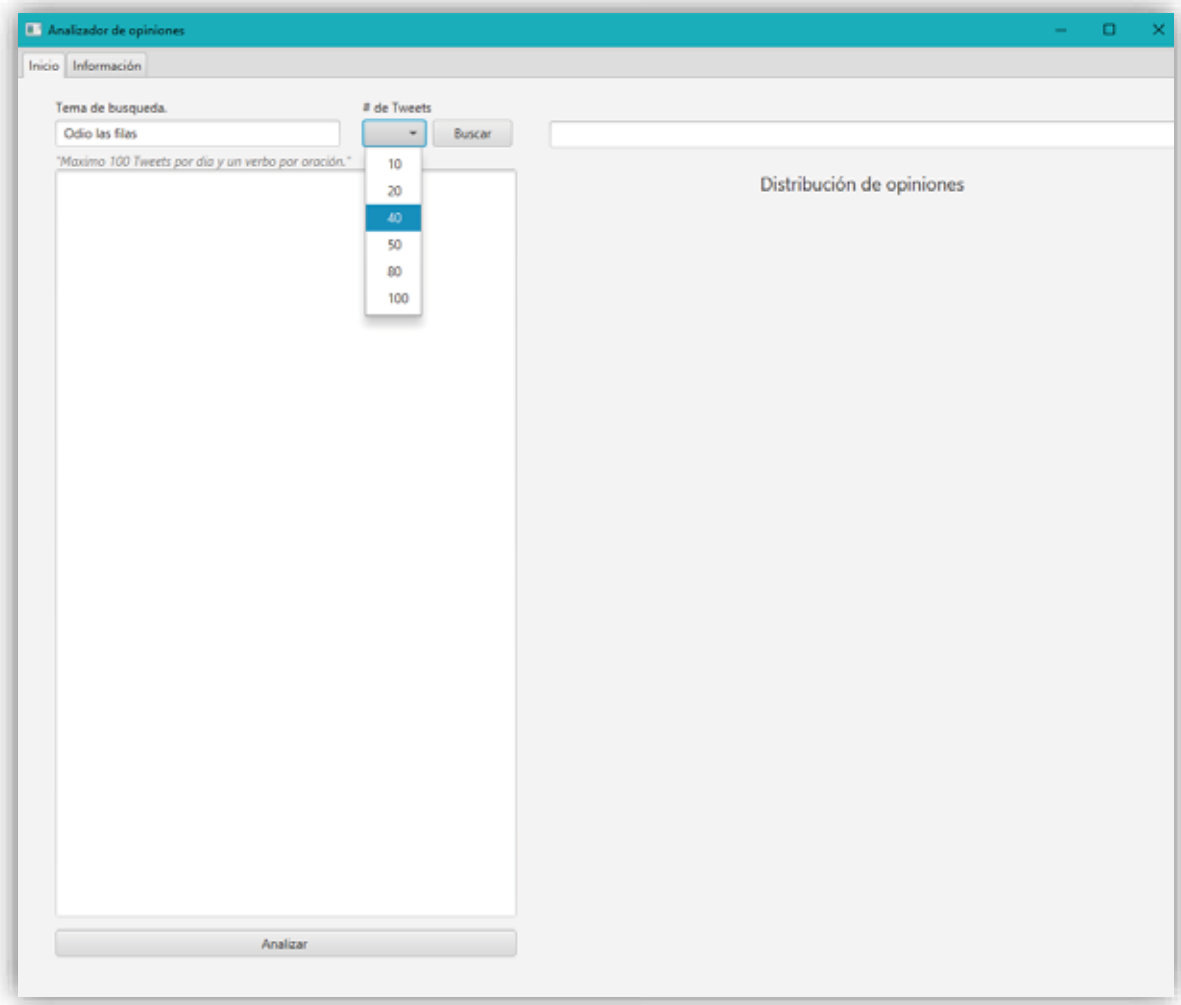

Figura 9. Pestaña de inicio

En la Figura 10 se visualizan los tweets descargados con el tema de la búsqueda previa y debajo de ellos el botón que analiza y presenta el resultado gráficamente.

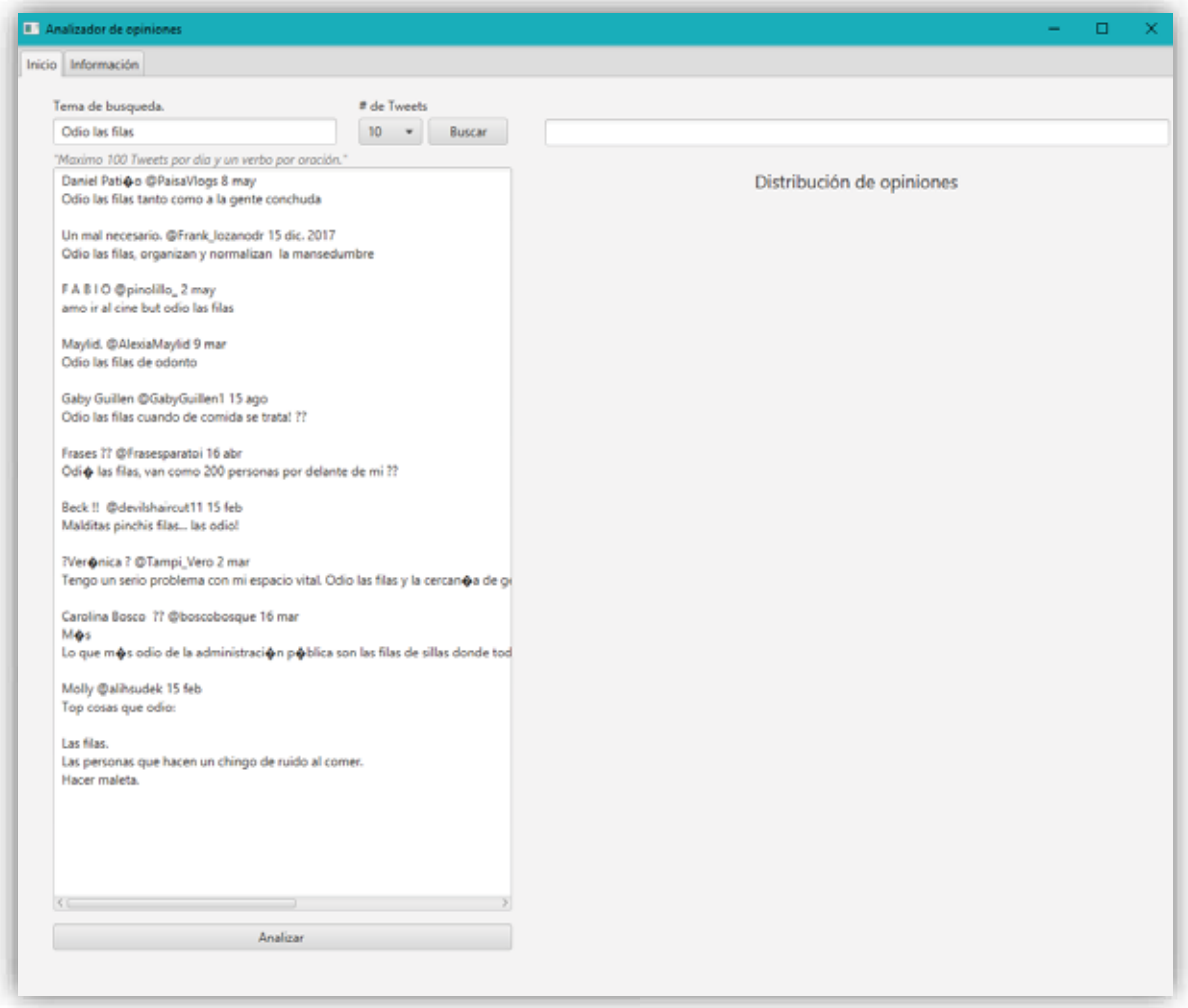

Figura 10. Búsqueda

En la Figura 11 se visualiza el resultado del análisis del corpus de tweets originales.

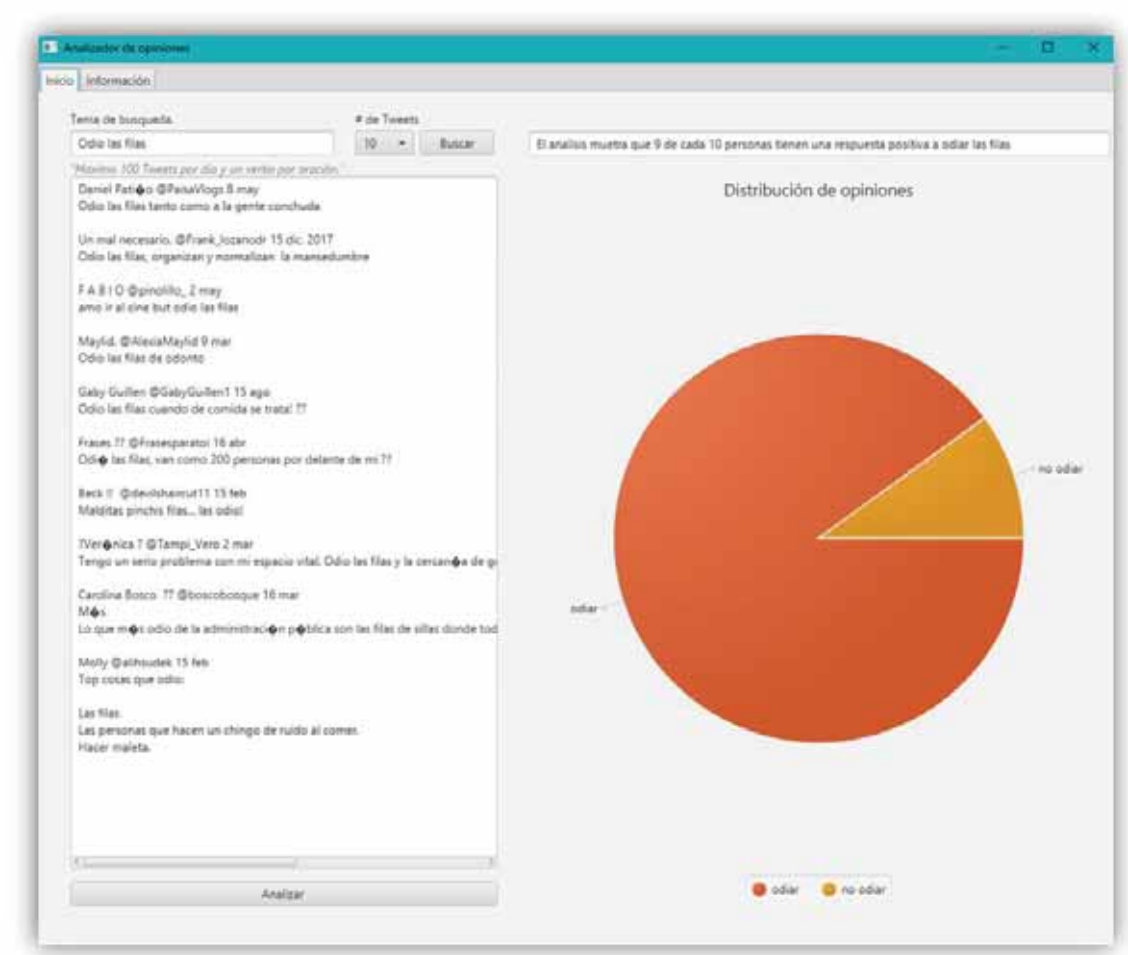

Figura 11. Resultado de análisis

En la Figura 12 se muestra la segunda pestaña llamada “Información”.

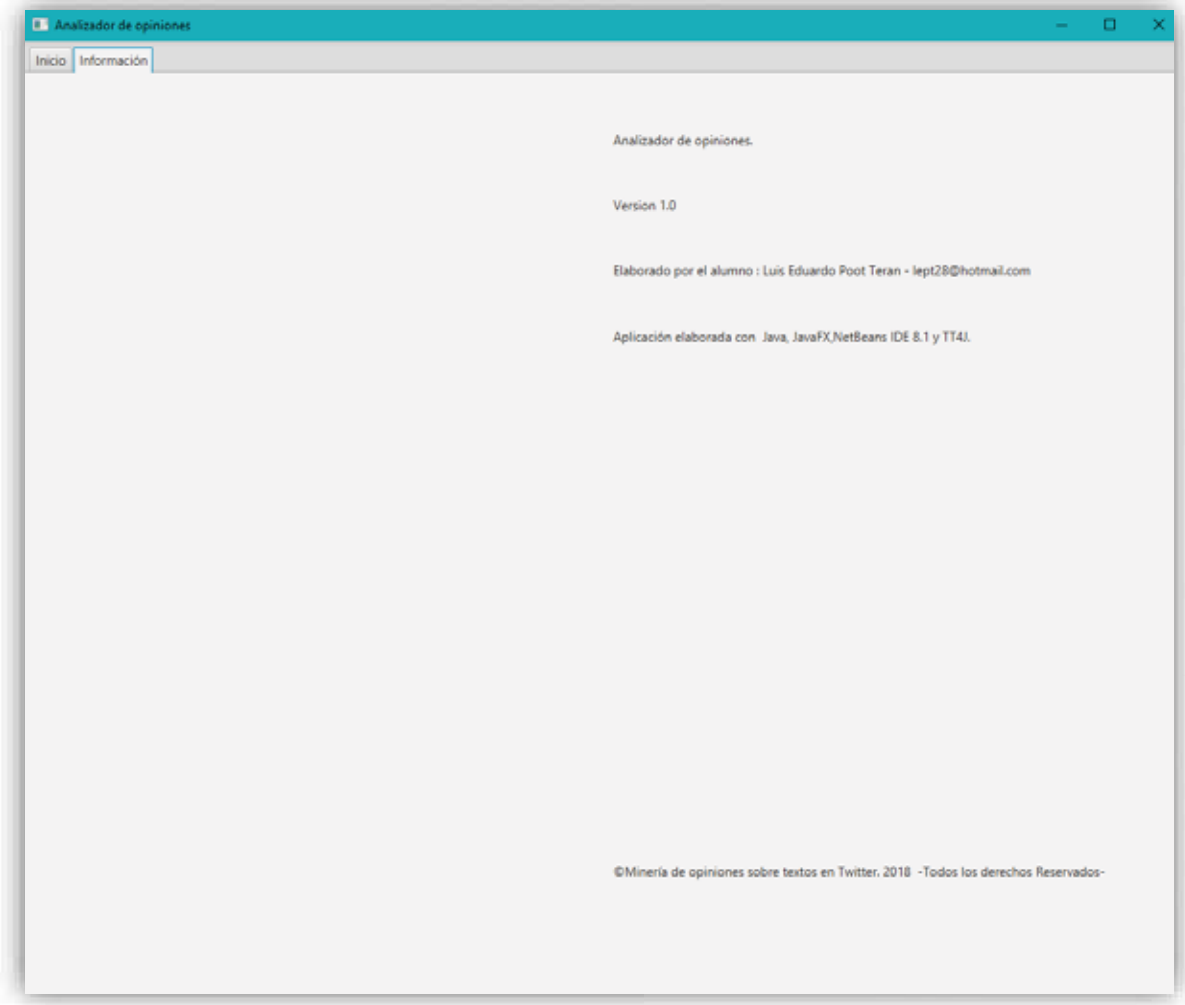

Figura 12. Pestaña información

3.4.3 Funcionamiento del sistema

Para el funcionamiento del sistema se implementaron una serie de archivos:

El primer archivo contiene *N* cantidad de tweets con el tema de búsqueda, correspondientes al número de tweets que usuario decidió descargar, denominado "Busqueda.txt", gráficamente se muestra en la Figura 13. Este mismo archivo se lee y se muestra en la interfaz de usuario.

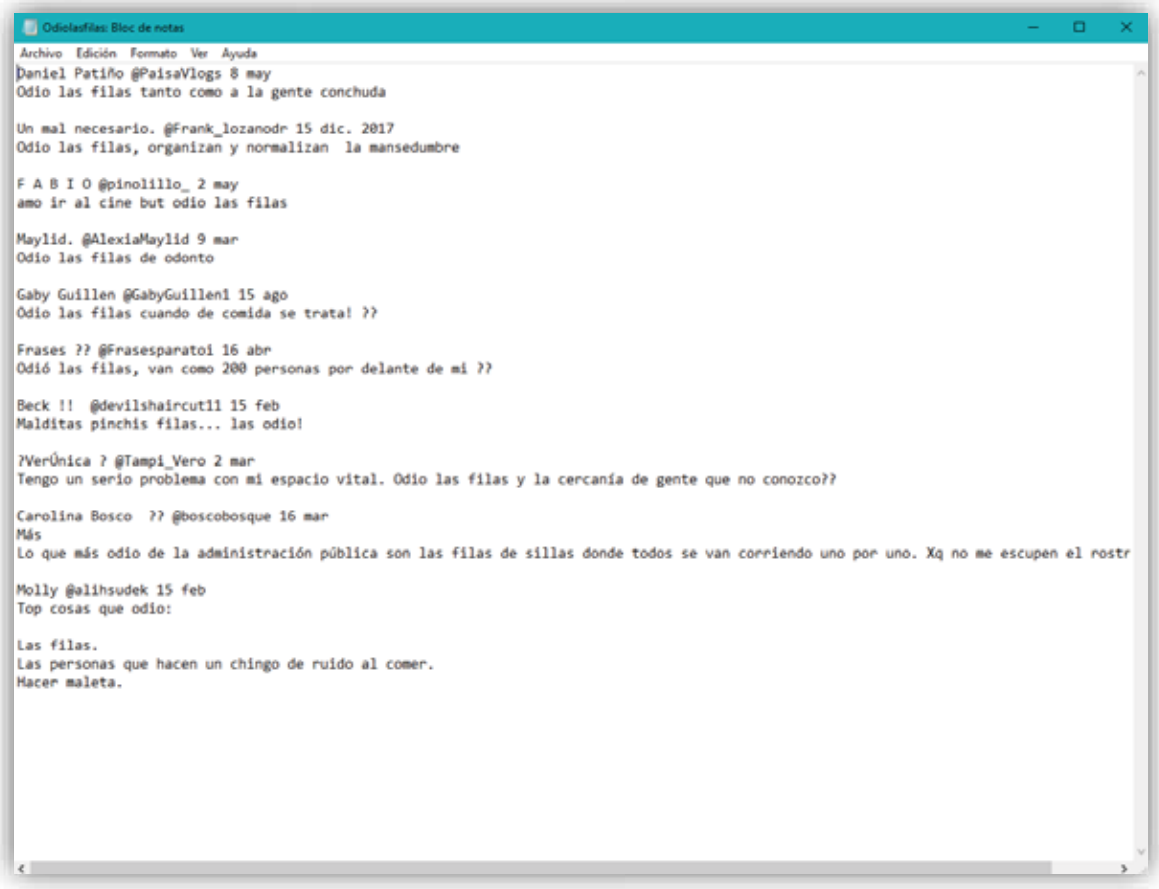

Figura 13. Archivo de Tweets descargados

- El segundo archivo contiene el verbo de la búsqueda.
- El tercer archivo contiene el predicado de la oración para poder presentarla al final.

El cuarto archivo llamado “Tokens.txt”, contiene el corpus de tweets ya tokenizados. Se puede observar en la Figura 14.

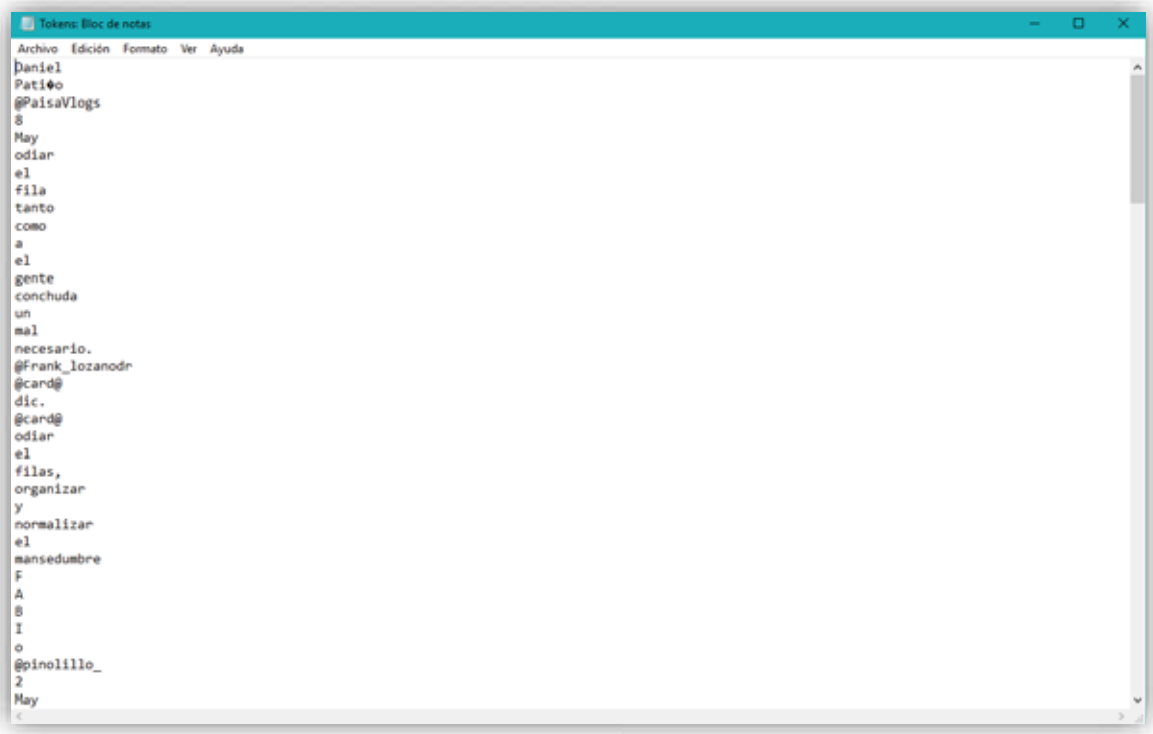

Figura 14. Corpus tokenizado

Para que el sistema no muestre errores al momento de que un usuario no seleccione el número de tweets a buscar o no escriba ninguna oración. Se agregó la siguiente sentencia de código que se muestra en la Figura 15. En la cual, si llegara a pasar algunos de los dos casos, mencionados con anterioridad, el sistema le mostrará un mensaje al usuario diciéndole que algunos de los campos está vacío.

```
String numero = (String) selection.getValue();//numero toma el valor de menu desplegable
String entrada = (String) textBusqueda.getText(); // entrada toma el valor de la entrada de texto
if (entrada.isEmpty())salida.setText("Escriba un tema de busqueda");//se verfica que no este vacio
 else if  (numero==null )salida.setText("Selecciona un numero");// se verifica que no sea nulo
```

Figura 15. Código de retroalimentación

Así mismo, se agregó un segmento de código en el módulo de clasificación, para evitar errores cuando se está buscando el verbo y no se encuentre. Es decir, cuando un usuario haya escrito una oración sin verbo. El cual se muestra en la Figura 16.

```
public void token(String token, String pos, String lemma)
{
        boolean resultado = pos.contains(verbo);
        if(resultado){
            //Si se encontro el verbo se escribe en un archivo externo
            wr.write(lemma);
        }else{
            //Si no se encuntra nigun verbo se muestra este mensaje
           System.out.println("Oración sin verbo");
        }

}
```

Figura 16. Verificación del verbo

CAPÍTULO 4. Resultados

4.1 Pruebas

En este apartado se muestra el correcto funcionamiento del sistema, realizando una serie de pruebas. Tomando como valores para *N*=10, 50, y 100 tweets. Se analizaran un total de 400 tweets.

4.1.1 Pruebas con corpus de tamaño diez tweets

De las Figuras 17-26 se muestran las pruebas realizadas con el corpus de tamaño 10. Para estas pruebas se utilizó como oración de búsqueda “Odio las filas”.

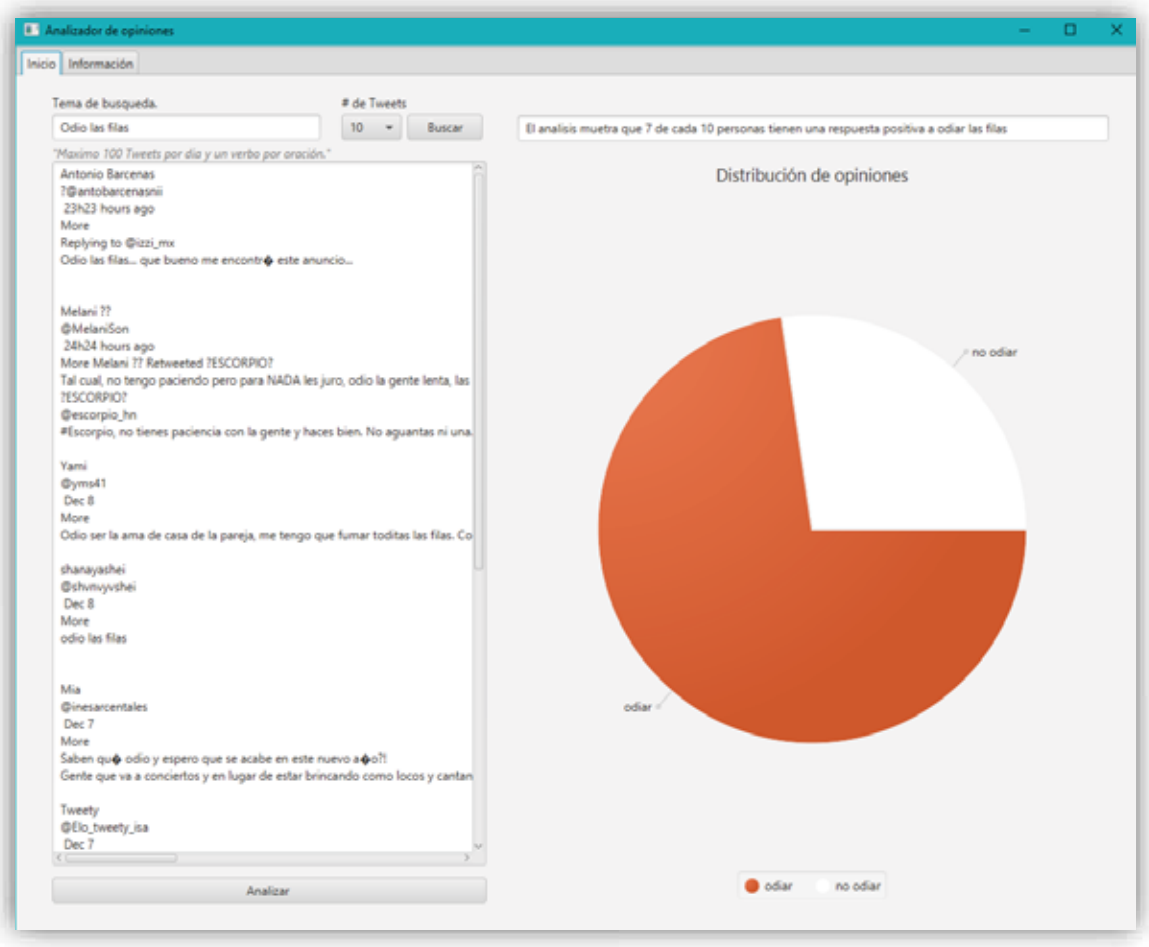

Figura 17. Prueba uno con un corpus de tamaño diez

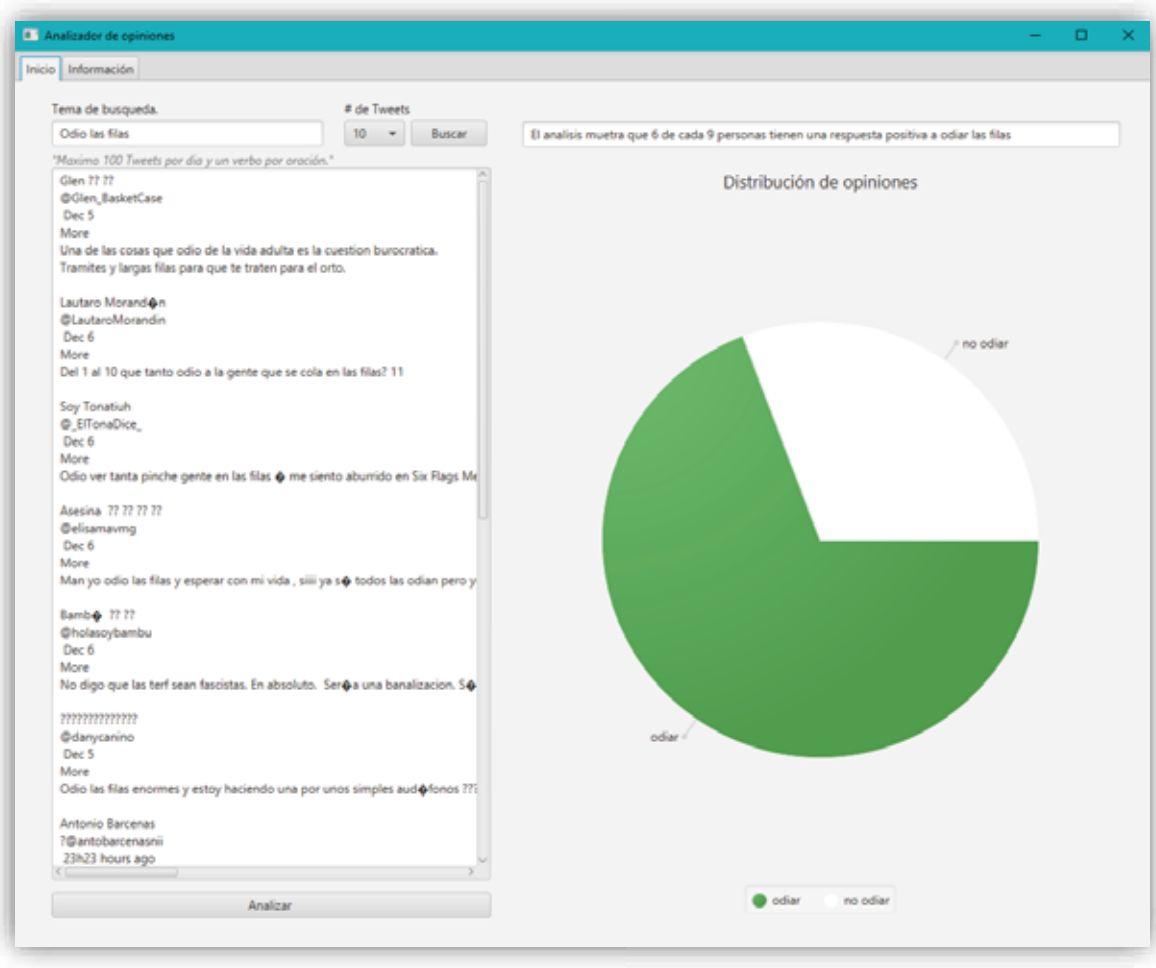

Figura 18. Prueba dos con un corpus de tamaño diez

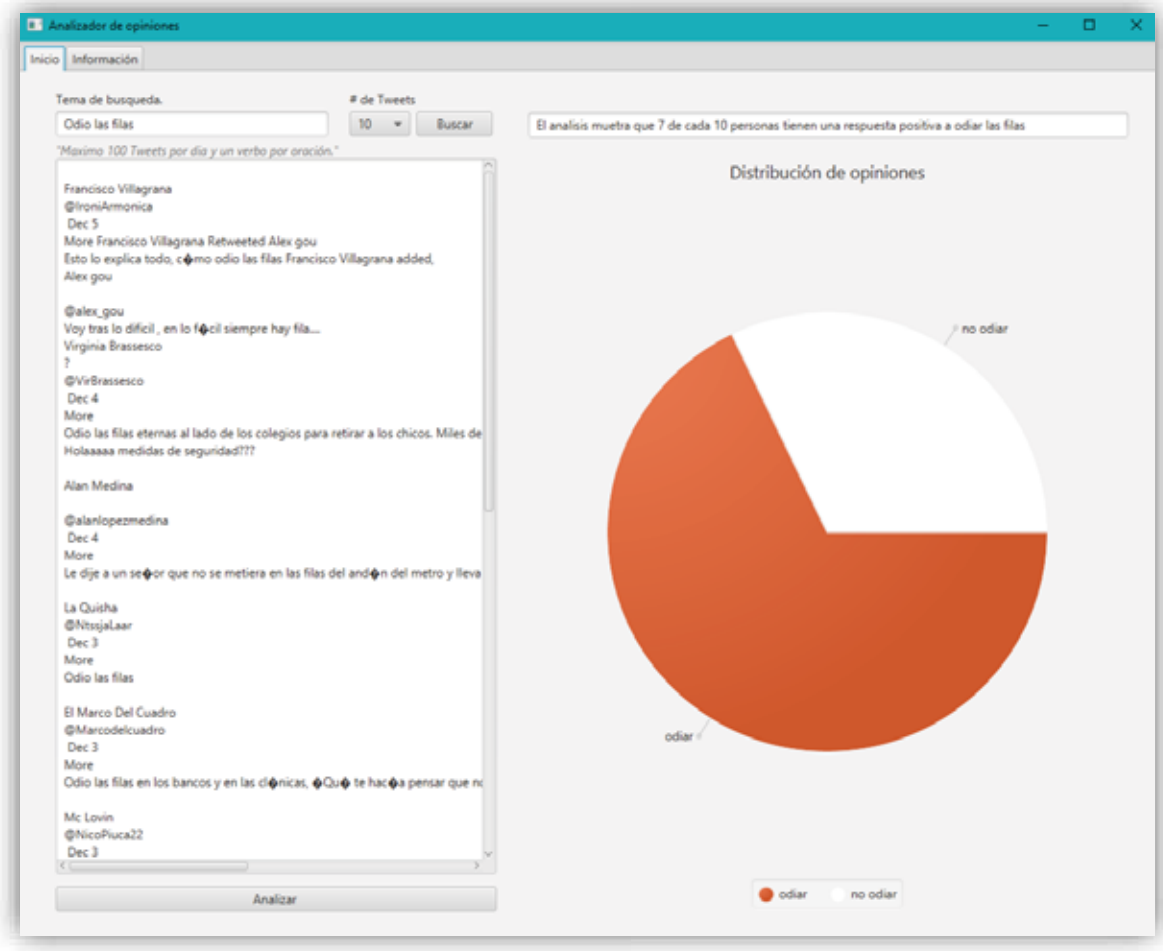

Figura 19. Prueba tres con un corpus de tamaño diez

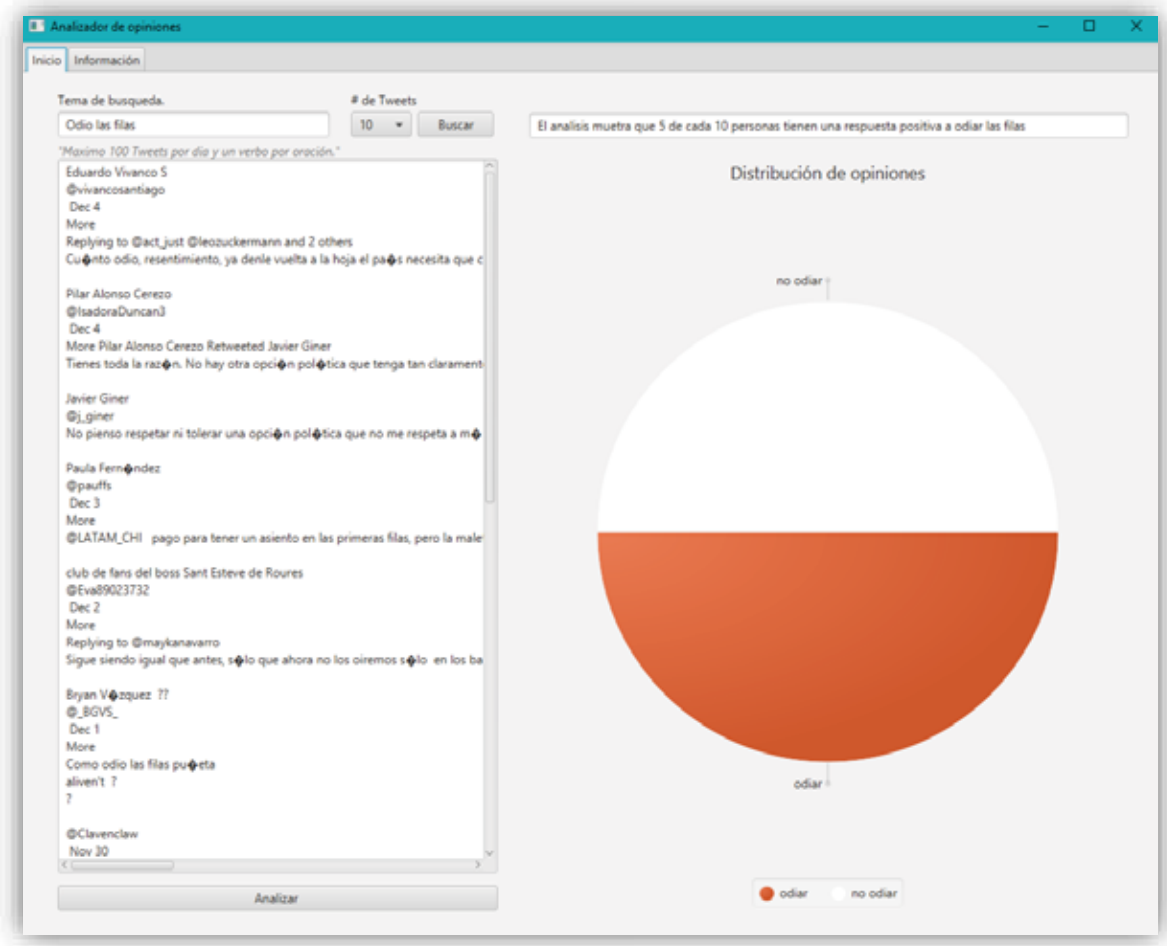

Figura 20. Prueba cuatro con un corpus de tamaño diez

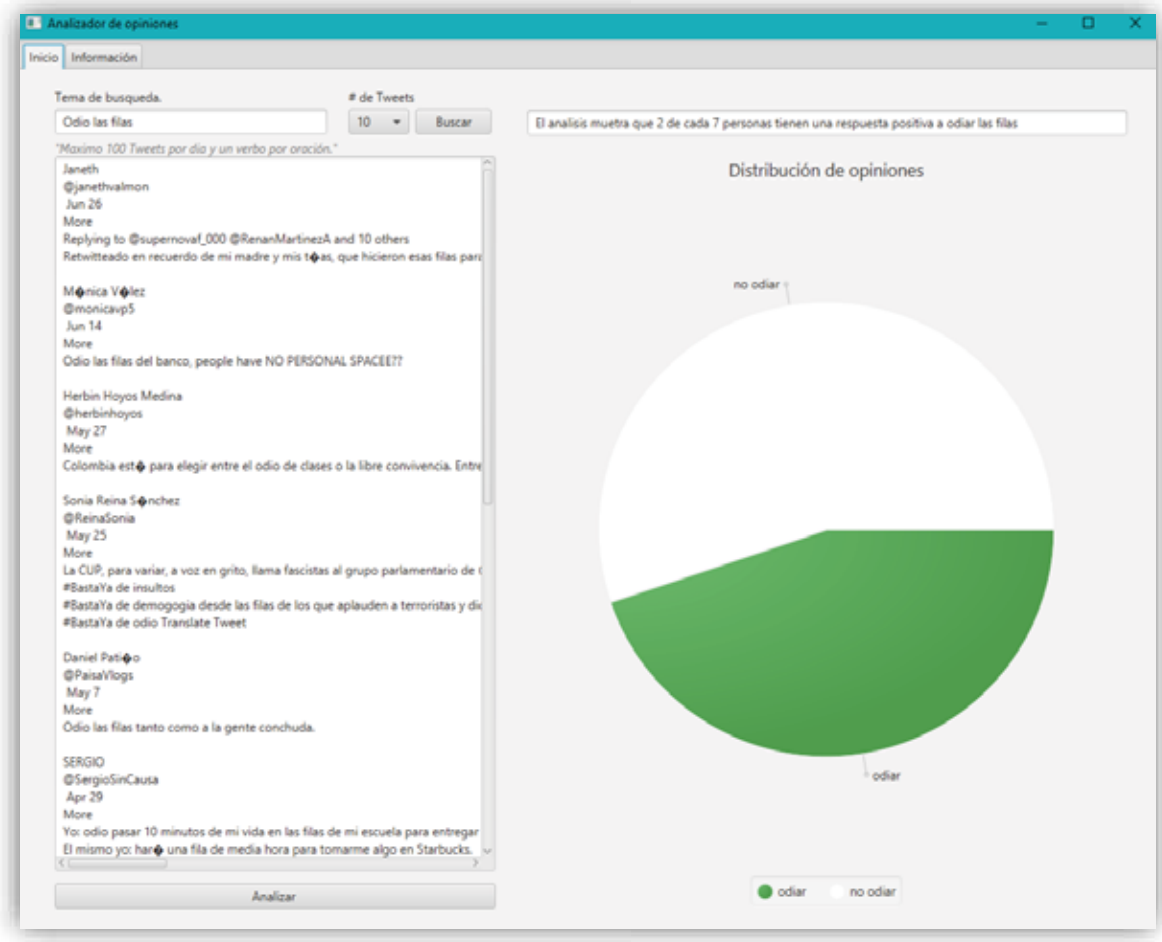

Figura 21. Prueba cinco con un corpus de tamaño diez

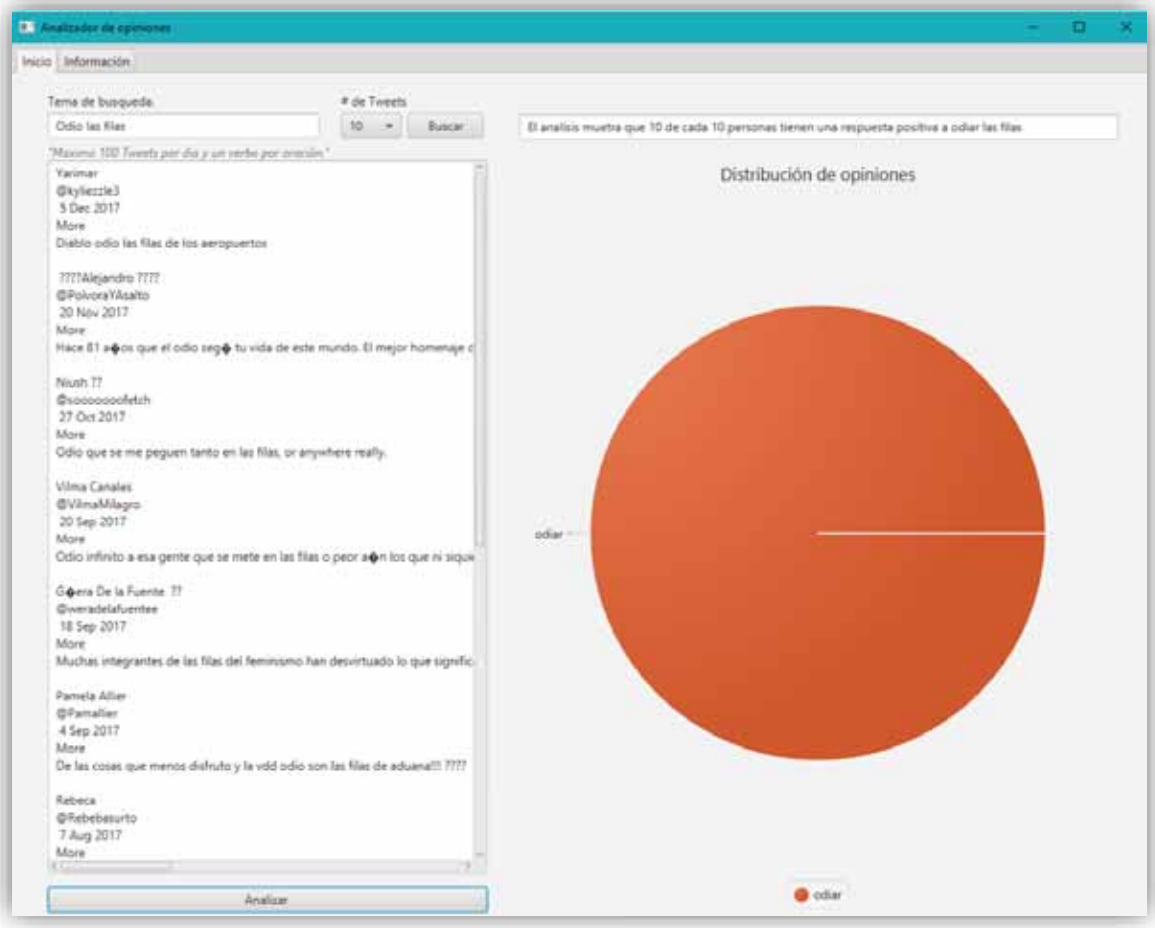

Figura 22. Prueba seis con un corpus de tamaño diez

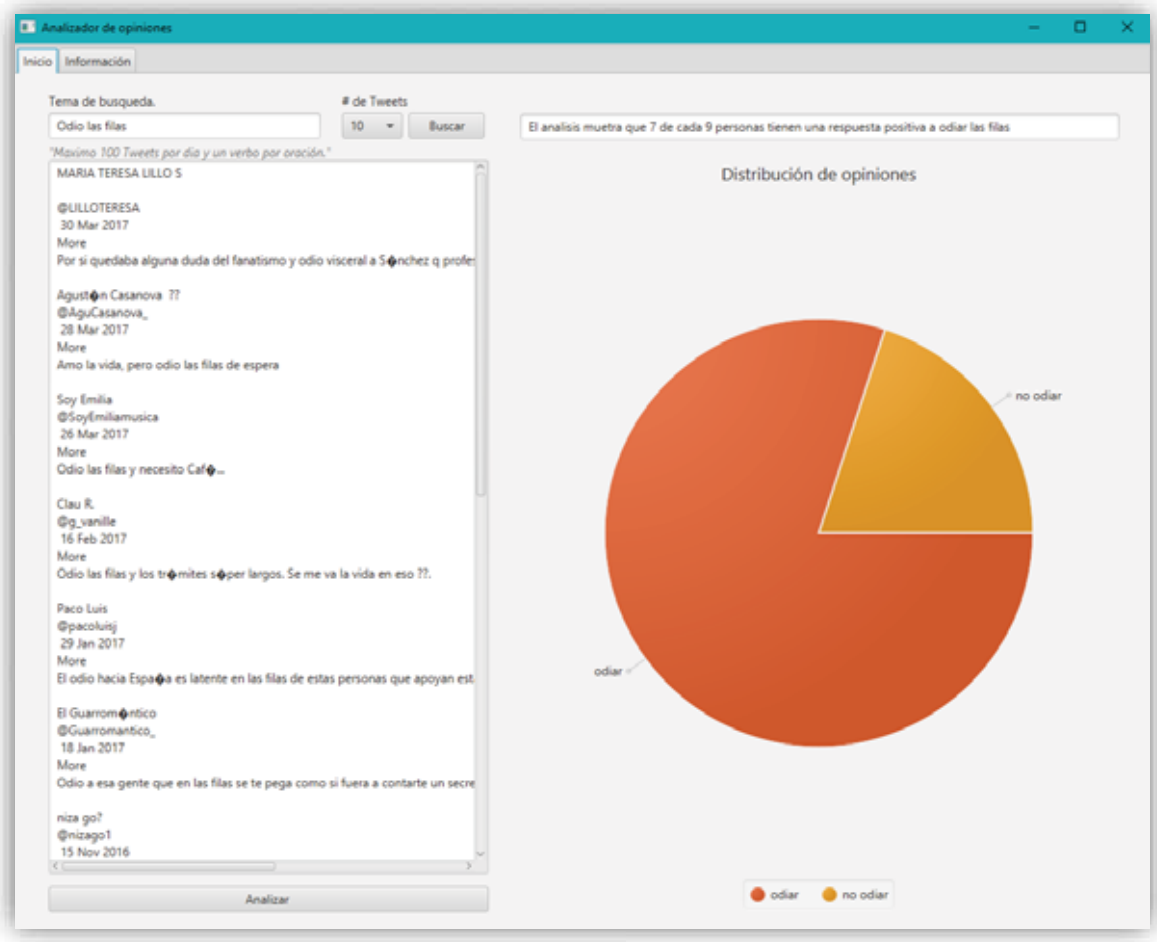

Figura 23. Prueba siete con un corpus de tamaño diez

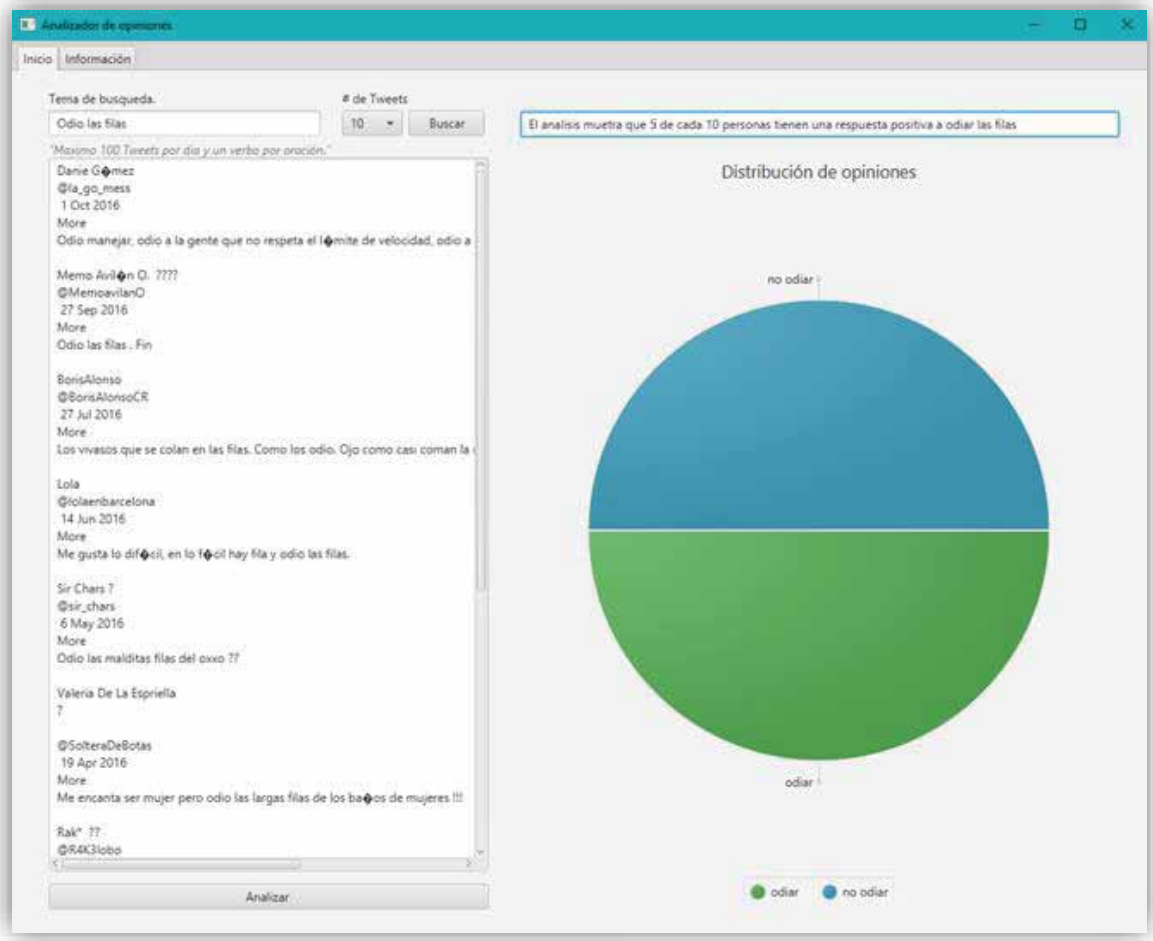

Figura 24. Prueba ocho con un corpus de tamaño diez

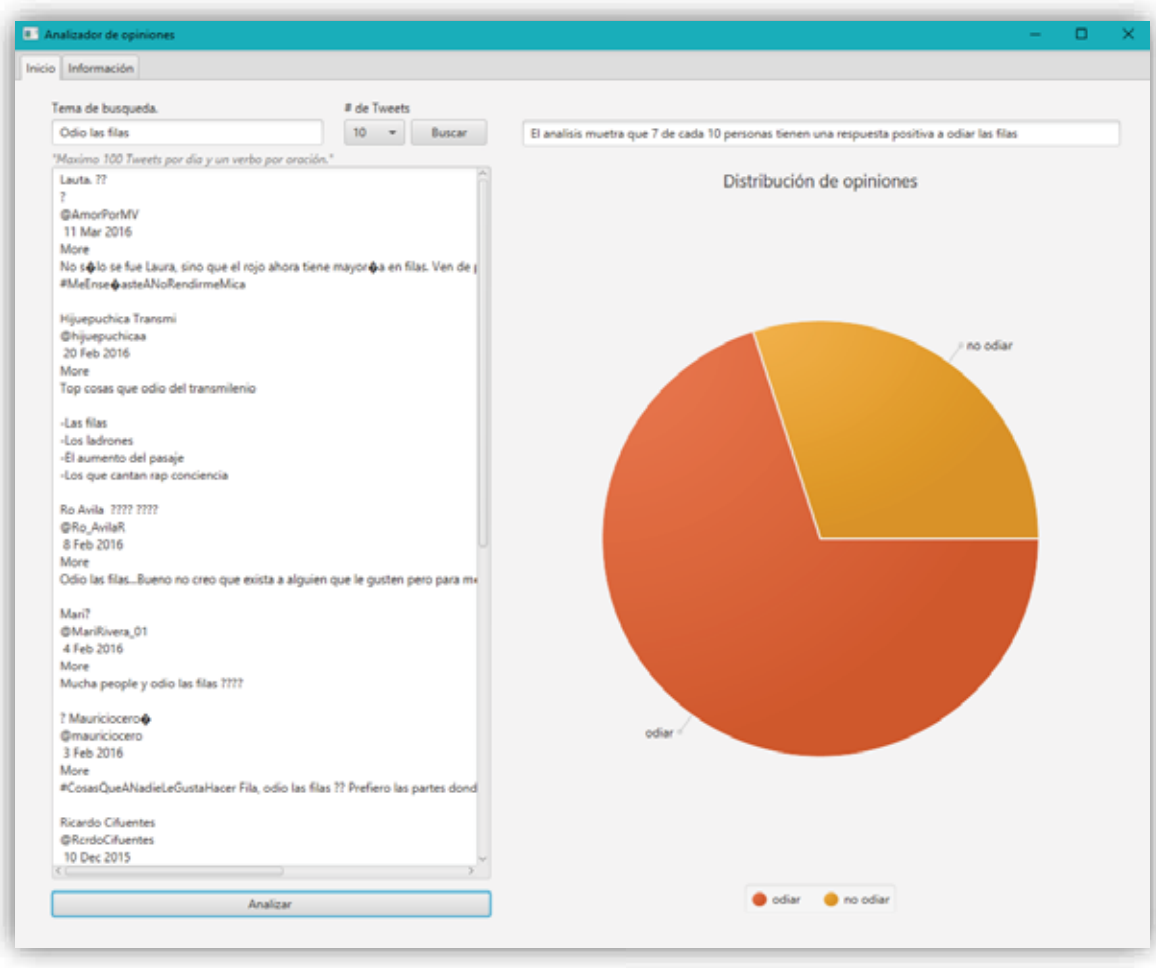

Figura 25. Prueba nueve con un corpus de tamaño diez

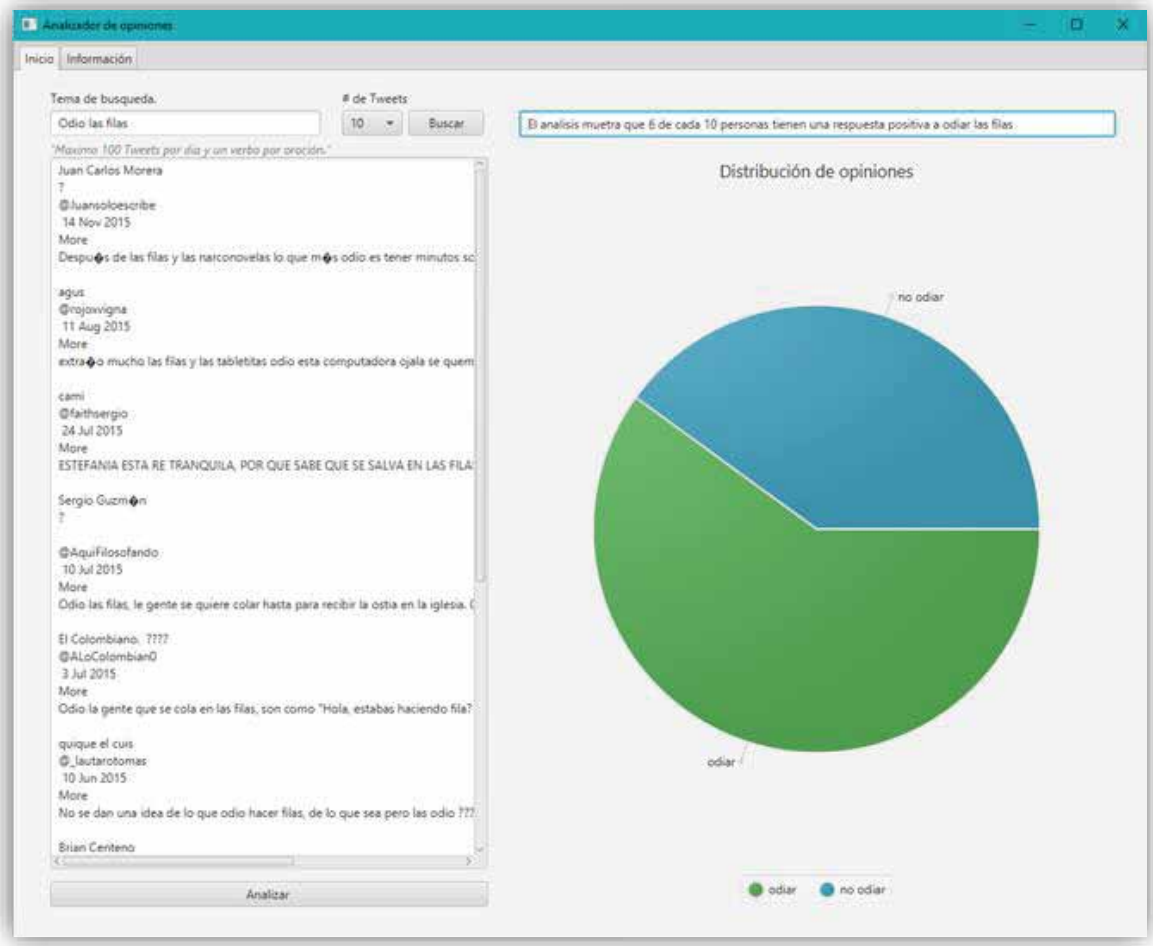

Figura 26. Prueba diez con un corpus de tamaño diez

4.1.2 Pruebas con corpus de tamaño cincuenta tweets

Las Figuras 27 y 28 muestran las pruebas realizadas con el corpus de tamaño cincuenta. Para estas pruebas se utilizó como oración de búsqueda “me gusta la navidad”.

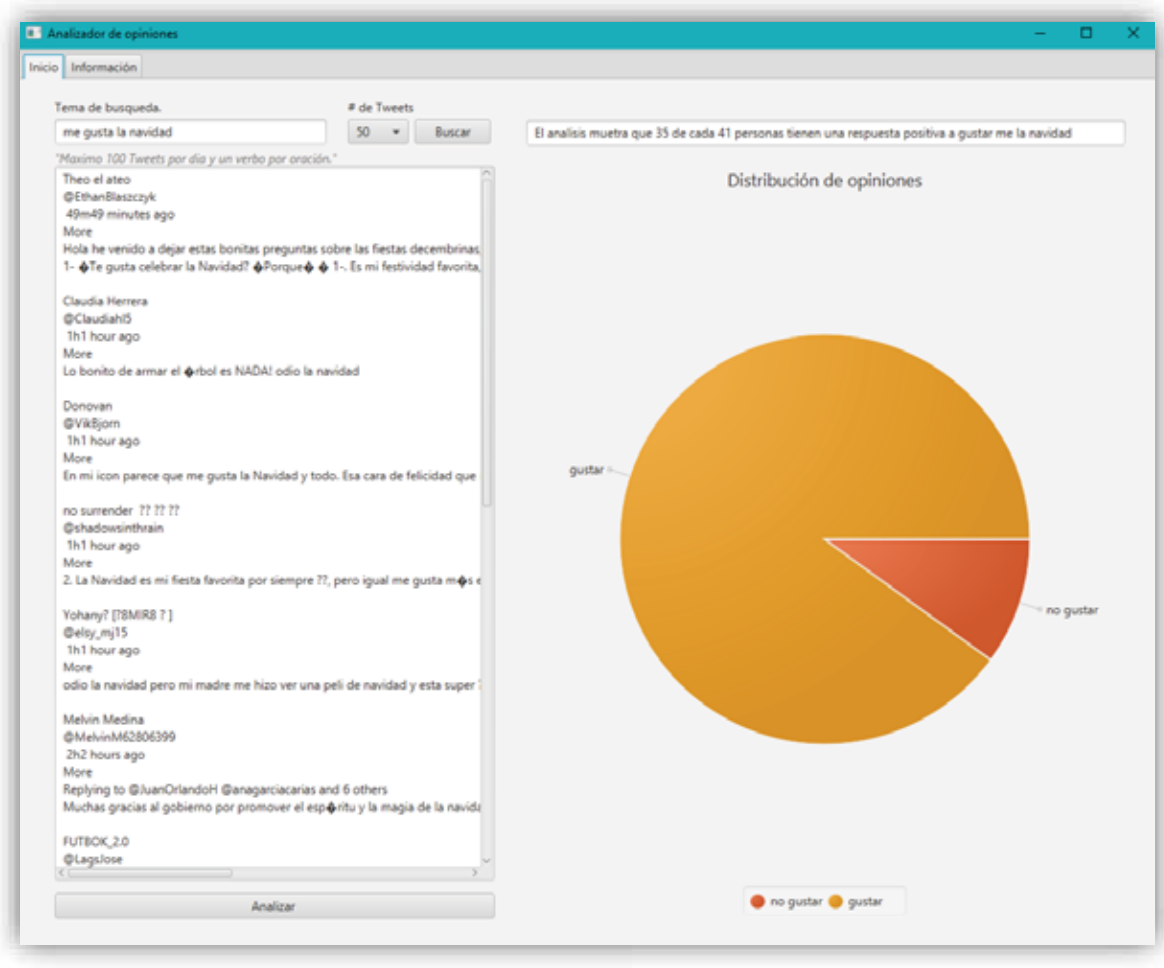

Figura 27. Prueba uno con un corpus de tamaño cincuenta

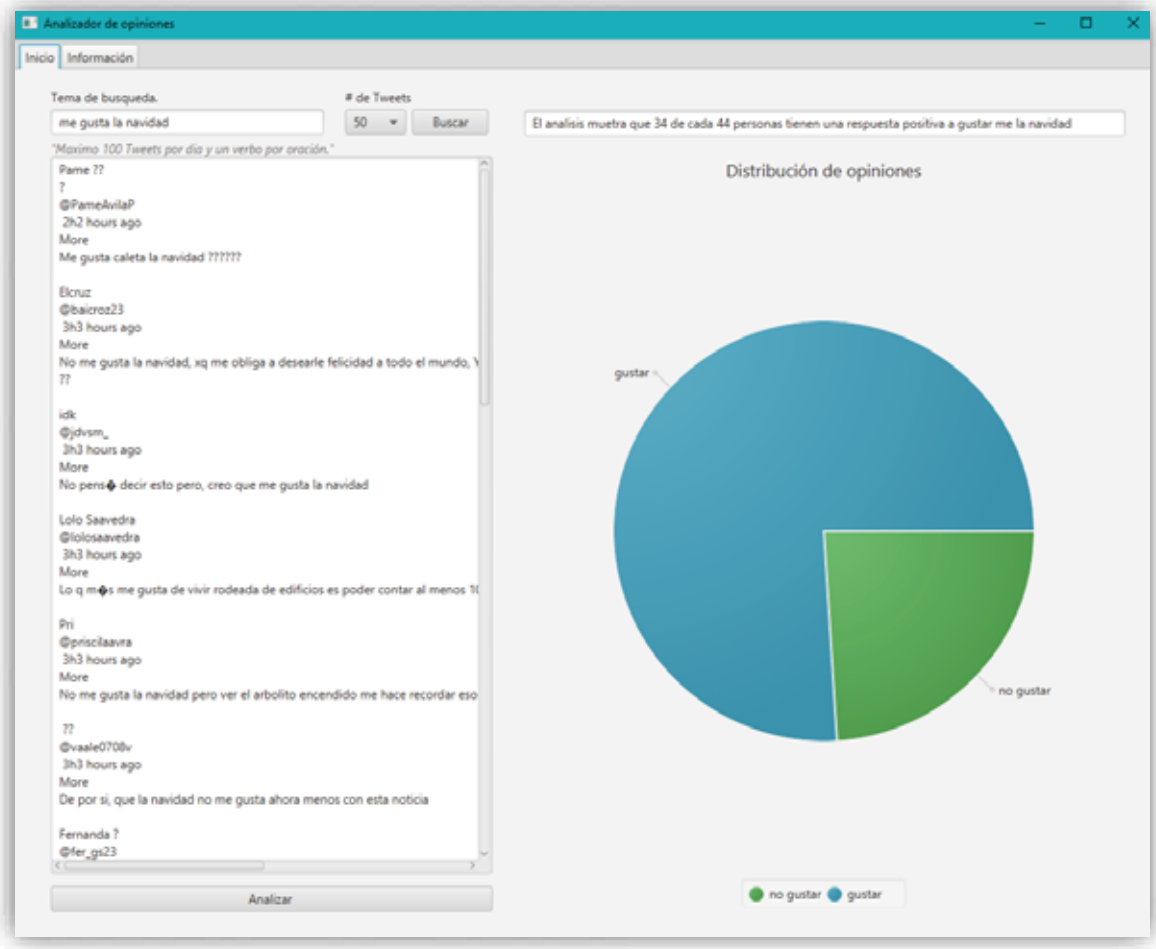

Figura 28. Prueba dos con un corpus de tamaño cincuenta

4.1.3 Pruebas con corpus de tamaño cien tweets

Las Figuras 29 y 30 muestran las pruebas realizadas con el corpus de tamaño cien. Para estas pruebas se utilizó como oración de búsqueda “Quiero que México cambie”.

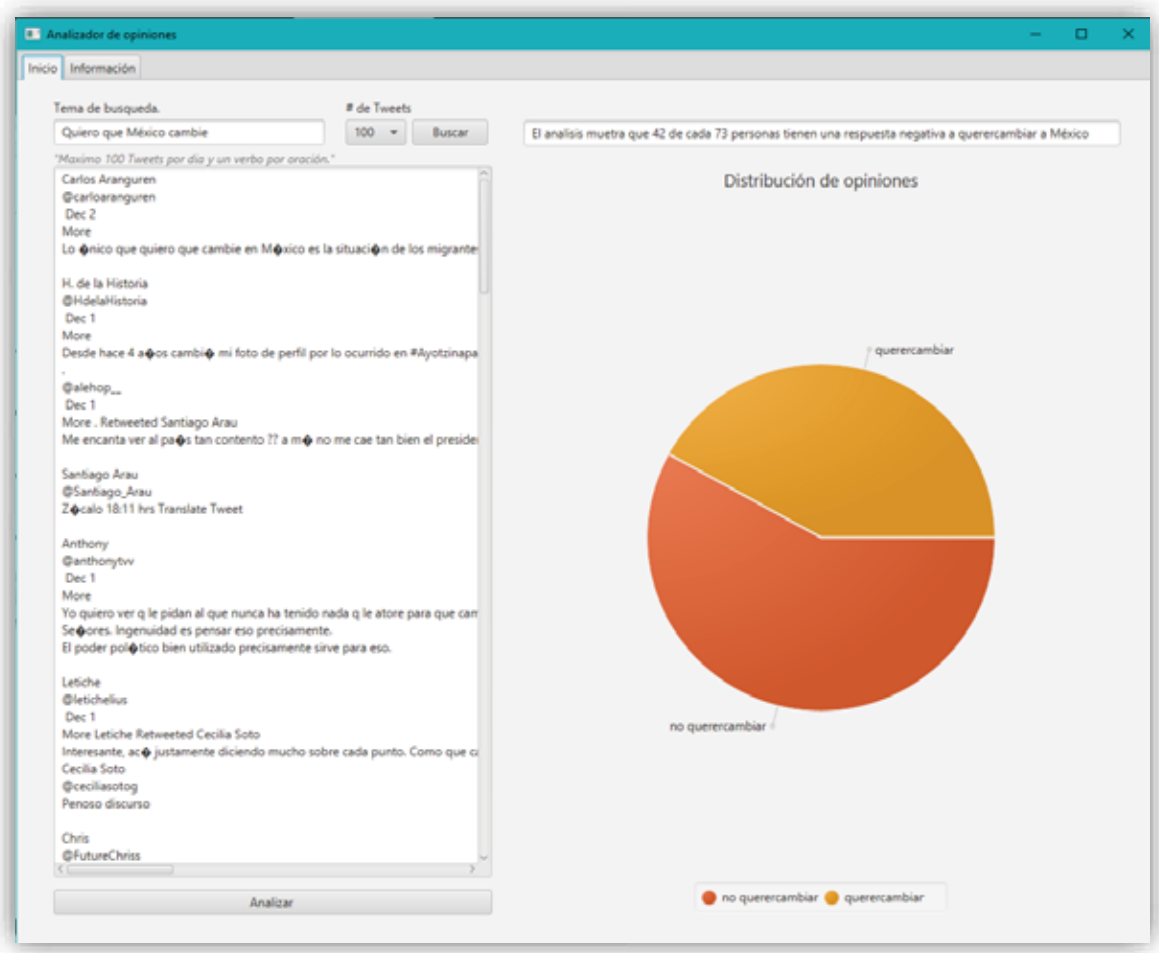

Figura 29. Prueba uno con un corpus de tamaño cien

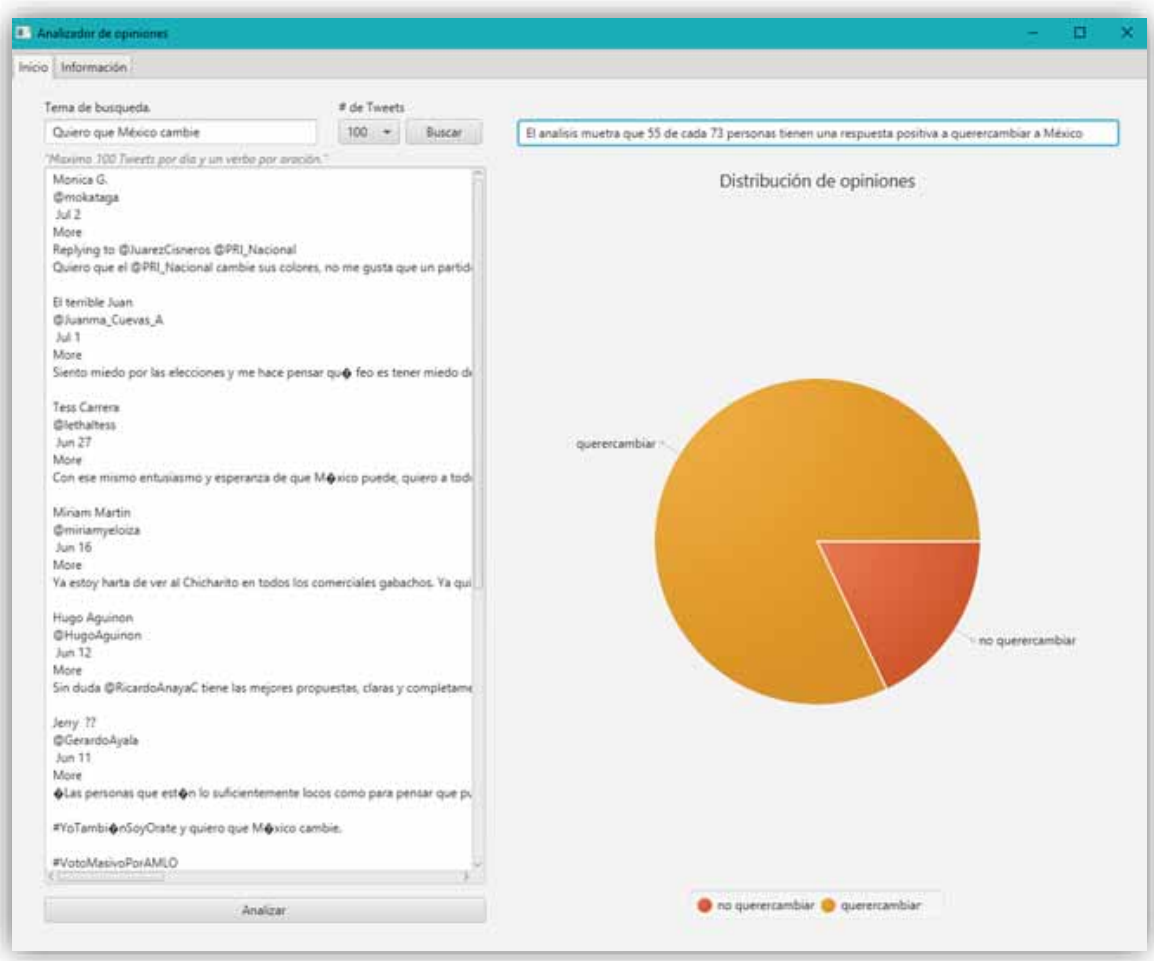

Figura 30. Prueba dos con un corpus de tamaño cien

4.2 Análisis y Resultados

En esta sección se analizarán los resultados del sistema. Para la realización del proyecto se utilizaron alrededor de mil tweets. En las pruebas realizadas se utilizaron cuatrocientos tweets, de los cuales se lematizaron alrededor de ocho mil tokens. En Tabla 6 se muestran los resultados de estas pruebas.

Tabla 5. Análisis de pruebas

Tamaño del corpus	No. de pruebas realizadas	No. de tokens lematizados	No. de tweets erróneamente clasificados	Porcentaje de tweets erróneamente clasificados
10	10	2000	5	5
50	2	2000	15	15
100	2	4000	54	27

En la Tabla 6 se observa que el porcentaje de tweets erróneamente clasificados tiene una relación directa con el tamaño del corpus, es decir si el tamaño del corpus aumenta o disminuye, el porcentaje de tweets erróneamente clasificados se ve afectado de la misma manera.

Este comportamiento puede deberse a varios factores como lo son:

- El módulo de extracción de tweets los almacena en codificación ANSI, provocando que los caracteres no identificados se conviertan en signos de interrogación. Cuando se lematiza, TreeTagger no logra reconocerlos y puede provocar un tweet sin clasificar. Un ejemplo de esto son los acentos como se puede ver en la Figura 31.

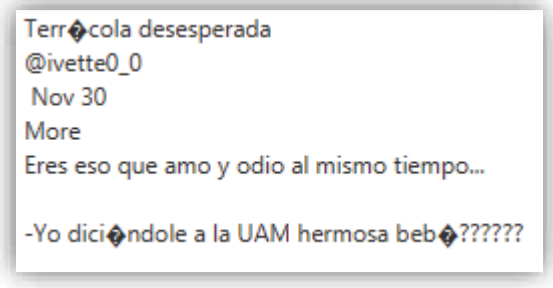

Figura 31. Ejemplo de caracteres no identificados

- Algunos usuarios no separan las palabras adecuadamente, provocando que en el proceso de lematizar, el lema no sea encontrado y devuelva <unknow>. En la figura 32 se puede observar un ejemplo de esto.

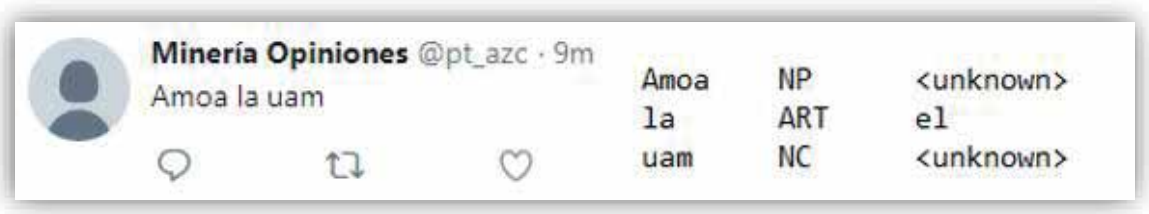

Figura 32. Ejemplo tweet mal escrito

- TreeTagger no reconoce la escritura de los tweets que no está hecha en español. Por lo que regresara la etiqueta de <unknow> como se mostró en el caso anterior.

Todas las pruebas se realizaron satisfactoriamente, ya que siempre se cumplió con la función de analizar y clasificar los comentarios, como puede ser observado en el capítulo 4.1.

Conclusiones

Se logró desarrollar satisfactoriamente el sistema para el análisis de opiniones en la red social Twitter, llegando a extraer tweets, lematizar tokens, clasificar corporas, analizar corporas y visualizar los resultados.

El sistema realiza el análisis adecuadamente, pero conforme aumenta el tamaño del corpus de tweets, los tweets erróneamente clasificados también lo hacen. El menor porcentaje de error en la clasificación de los tweets, se encontró en las pruebas realizadas con el corpus de diez tweets. Este porcentaje fue del 5%, sin embargo, el resultado no se puede considerar representativo por la muestra tan pequeña de opiniones. Se puede considerar guardar el resultado para búsquedas posteriores, no obstante ligar los resultados puede llevar a una conclusión errónea, ya que se pueden presentar variaciones de opiniones a través del tiempo.

La información del funcionamiento de TreeTagger en Java es muy escasa y poco descriptiva, lo que dificulta utilizar esta herramienta. Sin embargo, TreeTagger está completo con todos sus módulos, solo es necesario el ensayo y error para encontrar el modo adecuado de la aplicación deseada. Para mayor éxito en la lematizacion, se recomienda entrenar a TreeTagger o crear un archivo de parámetro con emoticones y abreviaturas. También, el almacenamiento hacerlo en formato UNICODE.

El procesamiento de lenguaje natural es campo con un gran campo de acción, el presente proyecto es una muestra de lo que se puede realizar. Aquellos que tengan el acceso a una suscripción de descarga ilimitada de tweets pueden monitorear la opinión publica del tema de su interés en tiempo real, esto visto desde un ámbito corporativo.

El presente proyecto puede ser ampliado al análisis de emociones sobre textos en Twitter, polarizando los tokens en el proceso de lematizacion y reincorporándolos a su oración original. Esto crearía oraciones polarizadas y con su correcto análisis se podría conocer el estado de ánimo del usuario.

Referencias bibliográficas

[1] García Menier, Everardo. (2006). Análisis De Textos Por Computadora. Boletín de Lingüística, 18(25), 121-134. Recuperado en 23 de mayo de 2018, de http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S0798-97092006000100005&lng=es&tlng=es.

[2] J. Paniagua "*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.

[3] D. Hernández Rubio, *"Extracción y Representación de servicios web escritos en SAWSDL mediante una ontología*", Proyecto Terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.

[4] C. Henríquez, "*Minería de Opiniones basado en la adaptación al español de ANEW sobre opiniones acerca de hoteles*", Procesamiento de Lenguaje Natural, Revista nº 56, pp. 25-32, marzo de 2016.

[5] J. Khairnar, M. Kinikar "*Machine Learning Algorithms for Opinion Mining and Sentiment Classification*". *IJSRP*, vol. 3, no. 6, pp. 1–6, Junio 2013.

[6] G. Sneka, "*Algorithms for Opinion Mining and Sentiment Analysis: An Overview*" International Journal of Advanced Research in Computer Science and Software Engineering, vol. 6, no. 2, pp.1-5, Febrero 2016.

[7] "NetBeans IDE", NetBeans. [*Online*]. Disponible: https://netbeans.org/features/index.html [Consultado: 01-Junio-2018].

[8] "Weka ", Weka. [*Online*]. Disponible: https://www.cs.waikato.ac.nz/ml/weka/ [Consultado: 24-Julio-2018].

[9] "TreeTagger ", TreeTagger. [*Online*]. Disponible: http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/ [Consultado: 24-Julio-2018].

Apéndices

Apéndice A. Clase para iniciar el StageScene

```
package tweet;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 *
 * @author lept28
 */
public class Tweet extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLVista.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.setTitle("Analizador de opiniones");
        stage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }

}
```

Apéndice B. Clase FXML para la visualización del sistema

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.effect.*?>
<?import javafx.scene.text.*?>
<?import javafx.scene.chart.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" focusTraversable="true" prefHeight="879.0" prefWidth="1111.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="tweet.FXMLVistaController">
   <children>
      <Label fx:id="label" layoutX="126" layoutY="120" minHeight="16" minWidth="69" />
     <TabPane prefHeight="879.0" prefWidth="1111.0" tabClosingPolicy="UNAVAILABLE">
       <tabs>
         <Tab text="Inicio">
           <content>
             <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="816.0" prefWidth="1054.0">
                 <children>
                   <GridPane alignment="CENTER" layoutX="36.0" layoutY="20.0" prefHeight="65.0" prefWidth="429.0">
                     <columnConstraints>
                       <ColumnConstraints hgrow="SOMETIMES" maxWidth="807.0" minWidth="10.0" prefWidth="288.0" />
                       <ColumnConstraints hgrow="SOMETIMES" maxWidth="434.0" minWidth="10.0" prefWidth="67.0" />
                       <ColumnConstraints hgrow="SOMETIMES" maxWidth="434.0" minWidth="10.0" prefWidth="79.0" />
                     </columnConstraints>
                     <rowConstraints>
```

<RowConstraints maxHeight="245.0" minHeight="0.0" prefHeight="15.0" vgrow="SOMETIMES" />
<RowConstraints maxHeight="688.0" minHeight="0.0" prefHeight="36.0" vgrow="SOMETIMES" />
<RowConstraints maxHeight="500.0" minHeight="5.0" prefHeight="17.0" vgrow="SOMETIMES" />
</rowConstraints>
<children>
<Button fx:id="buttonBuscar" mnemonicParsing="false" onAction="#buscar" prefHeight="25.0" prefWidth="74.0" text="Buscar" GridPane.columnIndex="2" GridPane.rowIndex="1" />
<ChoiceBox fx:id="selection" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="25.0" prefWidth="59.0" GridPane.columnIndex="1" GridPane.rowIndex="1">
<opaqueInsets>
<Insets />
</opaqueInsets>
</ChoiceBox>
<TextField fx:id="textBusqueda" maxWidth="-Infinity" prefHeight="25.0" prefWidth="265.0" promptText="Ejemplo: me gusta la nieve." GridPane.rowIndex="1" />
<Label text="Tema de busqueda." />
<Label prefHeight="7.0" prefWidth="187.0" text="# de Tweets" GridPane.columnIndex="1" />
<Label prefHeight="17.0" prefWidth="303.0" text=""Maximo 100 Tweets por dia y un verbo por oración."" textFill="#757575" GridPane.rowIndex="2">

<Font name="System Italic" size="12.0" />

</Label>
</children>
</GridPane>
<TextArea fx:id="salida" editable="false" layoutX="36.0" layoutY="85.0" prefHeight="697.0" prefWidth="429.0" />
<Button fx:id="analizar" layoutX="36.0" layoutY="794.0" mnemonicParsing="false" onAction="#analizarDatos" prefHeight="25.0" prefWidth="429.0" text="Analizar " />
<PieChart fx:id="graficaPie" layoutX="496.0" layoutY="82.0" prefHeight="739.0" prefWidth="582.0" title="Distribución de opiniones" />

```
                <TextField fx:id="opinion" editable="false" layoutX="496.0"
layoutY="40.0" prefHeight="25.0" prefWidth="584.0" />
              </children>
            </AnchorPane>
          </content>
        </Tab>
        <Tab text="Información">
          <content>
            <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0"
prefWidth="200.0">
              <children>
                <GridPane layoutX="25.0" layoutY="33.0" prefHeight="798.0"
prefWidth="1053.0">
                  <columnConstraints>
                    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
                    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
                  </columnConstraints>
                  <rowConstraints>
                    <RowConstraints maxHeight="163.0" minHeight="10.0"
prefHeight="59.0" vgrow="SOMETIMES" />
                    <RowConstraints maxHeight="235.0" minHeight="0.0"
prefHeight="59.0" vgrow="SOMETIMES" />
                    <RowConstraints maxHeight="332.0" minHeight="10.0"
prefHeight="62.0" vgrow="SOMETIMES" />
                    <RowConstraints maxHeight="332.0" minHeight="10.0"
prefHeight="59.0" vgrow="SOMETIMES" />
                    <RowConstraints maxHeight="399.0" minHeight="10.0"
prefHeight="387.0" vgrow="SOMETIMES" />
                    <RowConstraints maxHeight="332.0" minHeight="10.0"
prefHeight="165.0" vgrow="SOMETIMES" />
                  </rowConstraints>
                  <children>
                    <Label prefHeight="111.0" prefWidth="431.0" text="Analizador de
opiniones." GridPane.columnIndex="1" />
                    <Label prefHeight="116.0" prefWidth="425.0" text="Version 1.0"
GridPane.columnIndex="1" GridPane.rowIndex="1" />
```

```
                    <Label prefHeight="100.0" prefWidth="504.0" text="©Minería de opiniones sobre textos en Twitter. 2018  -Todos los derechos Reservados-" GridPane.columnIndex="1" GridPane.rowIndex="5" />
                    <Label prefHeight="86.0" prefWidth="450.0" text="Elaborado por el alumno : Luis Eduardo Poot Teran - lept28@hotmail.com" GridPane.columnIndex="1" GridPane.rowIndex="2" />
                    <Label prefHeight="81.0" prefWidth="544.0" text="Aplicación elaborada con  Java, JavaFX,NetBeans IDE 8.1 y TT4J." GridPane.columnIndex="1" GridPane.rowIndex="3" />
                  </children>
                </GridPane>
              </children>
            </AnchorPane>
          </content>
        </Tab>
      </tabs>
    </TabPane>
  </children>
</AnchorPane>
```

Apéndice C. Clase del modelo vista-controlador

```
package tweet;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventType;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.chart.PieChart;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;

/**
 *
 * @author lept28
 */
public class FXMLVistaController implements Initializable {


    ObservableList<String> ListaTweets = FXCollections.observableArrayList();
    ObservableList<PieChart.Data> graficaPieData = FXCollections.observableArrayList();

    @FXML
    private Label label;

    @FXML
    private TextField textBusqueda;
```

```
@FXML
private Button buttonBuscar;

@FXML
private Button analizar;

@FXML
private TextArea salida;

@FXML
private TextField opinion;

@FXML
private ChoiceBox selection;

@FXML
private PieChart graficaPie;

public static int cantidad;

private void cargarlista() {
   ListaTweets.removeAll(ListaTweets);
   String a="10";
   String b="20";
   String c="40";
   String d="50";
   String e="80";
   String f="100";
   ListaTweets.addAll(a,b,c,d,e,f);
   selection.getItems().addAll(ListaTweets);
}

@FXML
private void analizarDatos() throws Exception {
    graficaPieData.clear();
    Ttager lem = new Ttager();
    int i,j;
    lem.Lema("/Resultado/Odiolasfilas.txt");
    Clasificador clas = new Clasificador();
```

```
        i=clas.porcentaje();
        j=cantidad-i;
        Verbo v = new Verbo();
        String verb = v.verb();
        String pred = v.pred();
        graficaPieData.add(new PieChart.Data(verb,i));
        graficaPieData.add(new PieChart.Data("no "+verb,j));
        graficaPie.setData(graficaPieData);
        opinion.setText("El analisis muetra que "+i+" de cada "+cantidad+" personas tienen
una respuesta positiva a "+verb+" "+pred);
    }

    @FXML
    private void buscar() throws Exception {
      String numero = (String) selection.getValue();//numero toma el valor de menu
desplegable
      String entrada = (String) textBusqueda.getText(); // entrada toma el valor de la entrada
de texto
      if (entrada.isEmpty())salida.setText("Escriba un tema de busqueda");//se verfica que no
este vacio
       else if  (numero==null )salida.setText("Selecciona un numero");// se verifica que no
sea nulo
          else  {
                int numEntero = Integer.parseInt(numero);
                cantidad = numEntero;
                Lematizador tokens = new Lematizador();
                tokens.lematiza(entrada);
                tokens.predicado(entrada);
                String codigo = new String();
                String cadena;
                String archivo=("/Resultado/Odiolasfilas.txt");
                FileReader f = new FileReader(archivo);
                BufferedReader b = new BufferedReader(f);
                 while((cadena = b.readLine())!=null) {
                    codigo += cadena+"\r\n";
                 }
                 salida.setText(codigo);
                 b.close();
            }
```

```
    }


    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        cargarlista();

    }

}
```

Apéndice D. Clase para crear los tokens

```
package tweet;


import org.annolab.tt4j.TokenHandler;
import org.annolab.tt4j.TreeTaggerWrapper;
import java.util.StringTokenizer;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Locale;


public class Ttager
{

public static void Lema(String args) throws Exception, FileNotFoundException,
IOException
{
        System.setProperty("treetagger.home", "/treetagger");
            File A = new File("/Resultado/Tokens.txt");
            String cadena;
            String codigo = new String();
            String token;
            String archivo = args;
            FileReader f = new FileReader(archivo);
            BufferedReader b = new BufferedReader(f);
        TreeTaggerWrapper<String> tt = new TreeTaggerWrapper<String>();
            while((cadena = b.readLine())!=null) {
             codigo += cadena+"\r\n";
            }
            b.close();
            try {
```

```
                FileWriter w = new FileWriter(A);
                BufferedWriter bw = new BufferedWriter(w);
                PrintWriter wr = new PrintWriter(bw);
                StringTokenizer tokens=new StringTokenizer(codigo);
                while(tokens.hasMoreTokens()){
                      token=tokens.nextToken();
                try {
                   tt.setModel("/treetagger/lib/spanish.par");
                   tt.setHandler(new TokenHandler<String>()
                   {
                            public void token(String token, String pos, String lemma)
                            {

                                wr.write(lemma+"\r\n");

                            }
                   });

                  tt.process(new String[] {token});
                   }


               finally {
                    tt.destroy();
          }
             }
              wr.close();
              bw.close();

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

Apéndice E. Clase para lematizar los tokens y encontrar el verbo de búsqueda

```
package tweet;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.StringTokenizer;
import org.annolab.tt4j.TokenHandler;
import org.annolab.tt4j.TreeTaggerWrapper;

public class Lematizador
{
public static void lematiza(String args) throws Exception
{
            File A = new File("/Resultado/Verbo.txt");
            String tokeni;
            String verbo = "VL";

          System.setProperty("treetagger.home", "/treetagger");
          TreeTaggerWrapper<String> tt = new TreeTaggerWrapper<String>();
          try {
                FileWriter w = new FileWriter(A);
                BufferedWriter bw = new BufferedWriter(w);
                PrintWriter wr = new PrintWriter(bw);
                StringTokenizer tokens=new StringTokenizer(args," ");
                 while(tokens.hasMoreTokens()){
                     tokeni=tokens.nextToken();
                  tt.setModel("/treetagger/lib/spanish.par");
                  tt.setHandler(new TokenHandler<String>()
                  {
                            public void token(String token, String pos, String lemma)
                            {
                               boolean resultado = pos.contains(verbo);
                               if(resultado){
                                  //Si se encontro el verbo se escribe en un archivo externo
                                  wr.write(lemma);
                               }else{
```

```
                        //Si no se encuntra nigun verbo se muestra este mensaje
                        System.out.println("Oración sin verbo");
                    }

                }
            });
            tt.process(new String[] { tokeni });
        }  wr.close();
            bw.close();}


        finally {
            tt.destroy();
        }
}

    public static int bandera=0;

    public static void predicado(String args) throws Exception
{
        File A = new File("/Resultado/predicado.txt");
        String tokeni;
        String verbo = "VL";
      System.setProperty("treetagger.home", "/treetagger");
      TreeTaggerWrapper<String> tt = new TreeTaggerWrapper<String>();
      try {
          FileWriter w = new FileWriter(A);
          BufferedWriter bw = new BufferedWriter(w);
          PrintWriter wr = new PrintWriter(bw);
          StringTokenizer tokens=new StringTokenizer(args," ");
           while(tokens.hasMoreTokens()){
               tokeni=tokens.nextToken();
            tt.setModel("/treetagger/lib/spanish.par");
            tt.setHandler(new TokenHandler<String>()
            {
                public void token(String token, String pos, String lemma)
                {
                    boolean resultado = pos.contains(verbo);
                    if(resultado){
                     bandera=1;
```

```
                    }else if(bandera==1){
                      wr.write(token+" ");
                    }
                }
            });
            tt.process(new String[] { tokeni });
        }  wr.close();
            bw.close();}


        finally {
            tt.destroy();
        }
}
}
```

Apéndice F. Clase para conectarse a Twitter

```
package tweet;

public void TweetConect() throws TwitterException{
      Twitter twitter;
       ConfigurationBuilder cb = new ConfigurationBuilder();
       cb.setDebugEnabled(true)
            .setOAuthConsumerKey("Consumer Key")
            .setOAuthConsumerSecret("Consumer Secret")
            .setOAuthAccessToken("Access Token")
            .setOAuthAccessTokenSecret("Access Token Secret");
       twitter = new TwitterFactory(cb.build()).getInstance();

       Paging pagina = new Paging();

       Status tweetEscrito = twitter.updateStatus(Mensaje);

   }
```

Apéndice G. Clase para descargar los tweets

```
package tweet;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.List;

import twitter4j.Query;
import twitter4j.QueryResult;
import twitter4j.Status;
import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;

public class TweetEx {
   private Twitter twitter;
   private SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

   public TweetEx() {
      twitter = TwitterFactory.getSingleton();
   }

   public List<Status> query(String query) throws TwitterException {
      QueryResult search = twitter.search(new Query(query));
      List<Status> tweets = search.getTweets();
      return tweets;
   }

   public void escribirTweet(Status status) throws IOException {
      File A = new File("/Resultado/Busqueda.txt");
      FileWriter w = new FileWriter(A);
      BufferedWriter bw = new BufferedWriter(w);
      PrintWriter wr = new PrintWriter(bw);
      wr.write(String.format(status.getUser().getScreenName())+"
"+sdf.format(status.getCreatedAt())+"\n");
```

```
        wr.write(status.getText()+"\n\n");
    }

    public void escribirTweet(List<Status> status) throws IOException {
        for (Status tweet : status) {
            escribirTweet(tweet);
        }
    }

}
```

Apéndice H. Clase para clasificar los Tweets

```
package tweet;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author lept2
 */
public class Clasificador {

    public static int porcentaje() throws FileNotFoundException, IOException{
      int i=0;
      String cadena;
      String archivo=("/Resultado/Verbo.txt");
      FileReader f = new FileReader(archivo);
      BufferedReader b = new BufferedReader(f);
      cadena = b.readLine();
      String cadena2;
      String archivo2=("/Resultado/Tokens.txt");
      FileReader f2 = new FileReader(archivo2);
      BufferedReader b2 = new BufferedReader(f2);
      while((cadena2 = b2.readLine())!=null) {
          if(cadena.contains(cadena2)){
          i++;
          }
       }
       return i;
    }

}
```

Apéndice I. Clase para leer el verbo y el predicado

```
package tweet;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author lept2
 */
public class Verbo {

    public static String verb() throws FileNotFoundException, IOException{
        String cadena;
        String archivo=("/Resultado/Verbo.txt");
        FileReader f = new FileReader(archivo);
        BufferedReader b = new BufferedReader(f);
        return cadena = b.readLine();

    }

    public static String pred() throws FileNotFoundException, IOException{
        String cadena;
        String archivo=("/Resultado/predicado.txt");
        FileReader f = new FileReader(archivo);
        BufferedReader b = new BufferedReader(f);
        return cadena = b.readLine();

    }
}
```