UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ANÁLISIS DE SENTIMIENTOS EN REPOSITORIOS DE SOFTWARE Y SU REPERCUSIÓN EN LOS EQUIPOS DE DESARROLLO

T E S I S

QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

P R E S E N T A:
ANDRIC VALDEZ VALENZUELA

Director de Tesis:
Dra. Hanna J. Oktaba
Facultad de Ciencias

Co-Director de Tesis:
Dra. Helena M. Gómez Adorno
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Ciudad Universitaria, CD. MX. Diciembre 2020

A mis padres Eric Valdez y María Valenzuela...

Agradecimientos

> "Lo importante en la ciencia no es tanto obtener nuevos datos, sino descubrir nuevas formas de pensar sobre ellos."
>
> *William Lawrence Bragg*

Quiero agradecer principalmente a mi familia, padres y hermanos, por siempre mostrar apoyo y amor incondicional. A mis maestras y maestros que tuve a lo largo del posgrado, de cada uno aprendí bastante, especialmente a la Dra. Hanna Oktaba y Dra. Helena Gómez por sus aportes a este trabajo y siempre mostrar apoyo y disposición. A todas las amistades que formé a lo largo del posgrado, por su apoyo, por los trabajos en equipo y las incontables risas y buenos momentos. También, agradecer infinitamente a la UNAM (PCIC) y al Conacyt por su apoyo económico que me permitió estudiar este posgrado.

Ciudad Universitaria, IIMAS, 2020

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"No importa qué tan lento vayas
mientras no te detengas"

Confusio

En años recientes se ha experimentado un aumento en el desarrollo de sistemas de software de código abierto (OSS, por sus siglas en inglés), en los cuales, los profesionales del software desarrollan sus tareas de manera colaborativa y flexible. Esto ha derivado en la generación de datos (código, textos, archivos, logs, entre otros) de forma continua, acelerada y de acceso público. Es por esto que surge un nuevo campo de estudio llamado Minería en Repositorios de Software (MSR, por sus siglas en inglés), cuyo objetivo principal es extraer información valiosa de la gran cantidad de datos disponibles dentro de los repositorios de software [1].

La MSR está ganando cada vez más interés e importancia a nivel industrial y académico, debido a que las organizaciones e instituciones ya no necesitan depender únicamente de la intuición y experiencia de los equipos de trabajo para la toma de decisiones, sino que, además, pueden considerar los datos históricos almacenados en los repositorios. Pensando en esto, los proyectos/sistemas de software modernos pueden ser guiados e impulsados en función de la experiencia acumulada por los equipos de desarrollo y del nuevo conocimiento derivado del análisis de datos alojados en repositorios.

Al aplicar MSR se abre la posibilidad de realizar una gran cantidad de investigaciones y casos de estudio dentro de la Ingeniería de Software. Por ejemplo, establecer teorías generalizadas, validación de técnicas y metodologías, comprender mejor los procesos involucrados dentro del ciclo de vida del desarrollo de software (desde planificación y diseño hasta despliegue y mantenimiento de sistemas), e incluso, aplicaciones de análisis de sentimientos sobre los textos escritos en los canales de comunicación.

El análisis de sentimientos (también llamado minería de opiniones) es un campo de estudio que se encarga de extraer opiniones/sentimientos dentro de los textos utilizando técnicas computaciones, (como la Minería de Textos y el Procesamiento de Lenguaje Natural) con el fin de encontrar patrones, derivar nuevos conocimientos y tendencias. El rápido crecimiento en uso y popularidad que experimenta este campo de estudio es, en gran medida, debido los grandes volúmenes de datos generados por los usuarios (texto, código, vídeo, audio, etc) al interactuar con la gran cantidad de plataformas digitales disponibles en internet (redes sociales, correo electrónico, servicios de mensajería, repositorios de software, etc) [2].

Teniendo en cuenta lo anterior, en este trabajo de tesis se aplicará la MRS y el análisis de sentimientos con el fin de explotar la gran cantidad de datos alojados en los repositorios y estudiar los estados afectivos expresados por los profesionales del software.

En el resto de este capítulo se comenzará mencionando en la sección 1.1 la motivación y justificación de este trabajo. En la sección 1.2 se describen los objetivos generales y particulares que permitirán medir el alcance del trabajo. Posteriormente, en la sección 1.3 se plantearán las preguntas de investigación las cuales guiarán el desarrollo del trabajo. Y por último, en la sección 1.4 se define la organización y contenido del resto de la tesis.

1.1. Motivación

Los sistemas de seguimiento de incidencias (ITS, por sus siglas en inglés) son sistemas de software que se utilizan para administrar y almacenar datos acerca de las incidencias que pueden ocurrir durante el desarrollo de un proyecto de software; por ejemplo, corrección de errores, registro de tareas, control de versiones, tareas de mantenimiento, entre otras. Hoy en día, Jira[1], que almacena una gran cantidad de información de proyectos de código abierto, es el ITS más popular en el campo de la Ingeniería de Software y es utilizado en todo el mundo por empresas y profesionales de software. Para cada tarea o incidencia informada, Jira registra su descripción, su estado de resolución (resuelto o no resuelto), el tipo de problema (*Bugs*, Tareas, Mejoras, etc.), su nivel de prioridad, su secuencia de comentarios escritos, fecha y localización geográfica donde se publicó cada comentario, el número de desarrolladores involucrados, entre otros [3]. El éxito de un proyecto, producto o cualquier tarea de software, puede depender de muchos factores, como la satisfacción del cliente, la experiencia y productividad de cada miembro del equipo, o incluso, del entorno de trabajo. Sin embargo, es interesante preguntarse de qué forma los estados afectivos (conjunto de sentimientos y emociones), que llegan a expresar los profesionales del software, influyen en el éxito/fracaso de sus tareas o bien en el rendimiento de sus actividades diarias.

En los últimos años ha habido una creciente atención en los estados afectivos que experimentan los profesionales del software, principalmente porque existe una mayor conciencia del impacto que estos pueden llegar a tener en el rendimiento de sus tareas o en el trabajo en equipo [4]. Esto ha motivado el desarrollo de diversos trabajos de investigación y estudios empíricos con el fin de profundizar en el tema. Por ejemplo, M. Ortu [5] mostró cómo los estados afectivos de los desarrolladores afectan su productividad, encontrando que cuando se expresan más emociones positivas (como el amor y la alegría), menor es el tiempo que tardan en resolver sus tareas; mientras que, al expresar más emociones negativas (como tristeza), mayor es el tiempo que tardan en resolver las tareas. En otro estudio, Guzman [6] mostró un análisis de sentimientos en los comentarios de *commits* para proyectos en repositorios alojados en GitHub[2], encontrando que los proyectos desarrollados en lenguaje de programación Java, tienden a tener más comentarios negativos cuando se comparan con otros lenguajes; también demostraron que entre mayor sea la distribución geográfica de los equipos de desarrollo, mayor es el sentimiento positivo expresado en sus comentarios.

Por otro lado, al ser un campo de estudio reciente, aún falta hacer más investigaciones dentro del área. Es por esto que la principal motivación de este trabajo de tesis es aportar nuevos conocimientos a la Ingeniería de Software con el fin de ayudar a mejorar sus procesos. En este trabajo se propone aplicar minería en repositorios de software (Jira) haciendo uso de métodos y herramientas del campo de la Inteligencia Artificial con el fin de conocer más acerca de los estados afectivos (sentimientos o emociones) que llegan a experimentar los profesionales del software y cómo estos influyen en sus actividades y rendimiento.

[1] https://www.atlassian.com/es/software/jira

[2] https://github.com/ es una plataforma en donde se alojan proyectos utilizando el sistema de control de versiones Git.

1.2. Objetivos

El objetivo general de este trabajo consiste en:

> Implementar minería en repositorios de software con el fin de analizar los sentimientos expresados en los entornos de comunicación colaborativos y estudiar cómo estos se relacionan y repercuten en el rendimiento de los equipos de desarrollo.

Los objetivos particulares consisten en:

- Diseñar y desarrollar un modelo de clasificación de sentimientos basado en aprendizaje de máquina
- Extraer diversos tipos de datos de repositorios de software alojados en sistemas de seguimiento de incidencias.
- Aplicar un análisis de sentimientos en los repositorios de software y estudiar el efecto de los sentimientos en los equipos de desarrollo.

1.3. Preguntas de investigación

Las preguntas de investigación que se plantean en esta tesis y que que servirán como guía en el desarrollo de este trabajo, son las siguientes:

1. ¿De qué forma se expresan y cómo se distribuyen los sentimientos en los repositorios de software Jira?
2. ¿Cómo es la relación entre los sentimientos y el tipo/estado de resolución de las incidencias[3] (resuelta o no resuelta)? Esto es, cómo se expresan los sentimientos cuando las tareas se resuelven y cuando no se resuelven.
3. ¿Cómo se expresan los sentimientos de acuerdo a la hora del día y el día de la semana en la que se trabajaron las incidencias?
4. ¿Cómo es la relación entre los sentimientos y el tiempo de resolución de las incidencias? Esto es, en qué grado y sentido se relacionan los sentimientos con el tiempo en que tardan en resolverse las incidencias (en días).

1.4. Organización de la tesis

A continuación se mencionan los capítulos que componen este trabajo de tesis y se describe brevemente su contenido:

- El **capítulo 2** (Marco teórico) aborda todos los conceptos y definiciones relacionados al trabajo de investigación. Se mencionan los principales tipos de repositorios de software y se habla del campo de la MRS y sus aplicaciones. Se describe el concepto de análisis de sentimientos y se definen los principales niveles y enfoques para su implementación.

[3] Una incidencia (o *issue* en inglés) es cualquier tipo de tarea o problema a resolver dentro de un proyecto de software, por ejemplo, resolver un error (*bug*), implementar alguna mejora, desarrollar una nueva funcionalidad, entre otros

Se expone acerca del campo de la Minería de Textos describiendo sus tareas y procesos principales. Y por último, se describen los conceptos y los distintos enfoques alrededor del aprendizaje de máquina.

- El **capítulo 3** (Trabajos relacionados) describe los antecedentes y trabajos relacionados en torno al análisis de sentimientos en la Ingeniería de Software, además, se mencionan las características principales de las herramientas disponibles públicamente para la clasificación de sentimientos dentro del área de investigación, así como los distintos *datasets*/corpus disponibles para su entrenamiento.

- El **capítulo 4** (Metodología de investigación) explica el marco metodológico propuesto para llevar a acabo un análisis de sentimientos en los repositorios de software. Se muestran diagramas, códigos y distintas configuraciones en torno a las etapas que componen la metodología, que comienza con una etapa de desarrollo y entrenamiento de un modelo de clasificación de sentimientos, seguido de una etapa de extracción de datos de los repositorios Jira, después de una etapa de evaluación y predicción de sentimientos y por último de una etapa de exploración y análisis.

- El **capítulo 5** (Resultados) muestra los resultados obtenidos después de aplicar la metodología propuesta, donde cada sección del capítulo presenta los resultados para cada una de las preguntas de investigación planteadas, esto es, se expone acerca de los sentimientos en repositorios de software, los sentimientos y resolución de incidencias, los sentimientos y la hora/día de la semana y por último los sentimientos y tiempo de resolución de incidencias.

- El **capítulo 6** (Conclusiones y trabajo a futuro) menciona en forma resumida los resultados, contribuciones y lecciones aprendidas en el desarrollo de esta tesis, además, se proponen las distintas direcciones que se pueden tomar como trabajo a futuro.

Capítulo 2

Marco Teórico

"La simplicidad es la mayor sofisticación"

Leonardo da Vinci

En este capítulo se describen los diferentes conceptos y términos que se utilizarán en el desarrollo de esta tesis. Se comienza con la **sección 2.1** en donde se mencionan los principales tipos de repositorios de software en la actualidad (de incidencias, código, control de versiones, etc.) y se introduce al campo de la MRS describiendo algunas de sus principales áreas de aplicación (como la Predicción de Defectos, Minería de textos, etc). Después, en la **sección 2.2** se describe e introduce el concepto de análisis de sentimientos y se definen sus principales niveles (Documento, Oración, Aspecto) y enfoques (Aprendizaje de máquina, Lexicones) para su implementación. Posteriormente, en la **sección 2.3** se expone acerca del campo de la Minería de Textos describiendo sus tareas (como clasificación de textos) y procesos principales. Por último, en la **sección 2.4** se describen los conceptos y los distintos enfoques alrededor del aprendizaje de máquina, abordando temas relacionados al aprendizaje supervisado, a los modelos/algoritmos de clasificación y distintas métricas para la evaluación de rendimiento.

2.1. Repositorios en Ingeniería de Software

En términos generales se puede definir un repositorio como un lugar en donde se almacenan o guardan cualquier tipo de objetos. En términos informáticos, un repositorio representa un espacio centralizado donde se almacena información digital como archivos de textos, imágenes, videos, entre otros. Desde el punto de vista de la Ingeniería de Software, un repositorio representa una pieza fundamental para los procesos de desarrollo de software, debido a que en ellos se almacena información valiosa y útil acerca de los proyectos o tareas (código, configuraciones, comunicaciones, etc) [1].

El aumento acelerado en el desarrollo de sistemas de software de código abierto (OSS) ha derivado en la generación de grandes cantidades de datos (como códigos, textos, archivos) de forma continua, los cuales en su mayoría, son almacenados en repositorios de software públicos. Debido a esto, un nuevo campo de estudio llamado Minería en Repositorios de Software (MSR), nace con el fin de extraer información valiosa de esta gran cantidad de datos disponibles dentro de los repositorios de software [1](sección 5.1).

Mediante la MSR se pueden extraer una gran cantidad de datos que pueden ser usados para llevar a cabo estudios e investigación que sirvan para la validación de técnicas/métodos

o establecer teorías con sustento y generalizadas dentro de la Ingeniería de Software. Esto puede resultar muy útil para las organizaciones e instituciones porque les permite generar nuevos conocimientos o tomar decisiones sustentadas con el fin de mejorar sus procesos de desarrollo. Por ejemplo, es posible responder a preguntas del tipo: ¿Es el proceso X mejor que el proceso Y? ¿Cuál es la probabilidad de que ocurra un defecto? ¿Cuál es el tiempo estimado de corrección de un *Bug*?, etc [1](sección 5.2). Algunos de los beneficios potenciales al aplicar MSR son:

- Mejorar el mantenimiento de los sistemas de software.
- Validación empírica de técnicas y métodos.
- Apoyo a la reutilización de software.
- Asignación adecuada de recursos de prueba y mantenimiento.

Debido a que los repositorios almacenan una variada cantidad de información, existen diversos tipos de ellos. En la figura 2.1 se muestran los tipos de repositorios de software más comunes usados en la Ingeniería de Software.

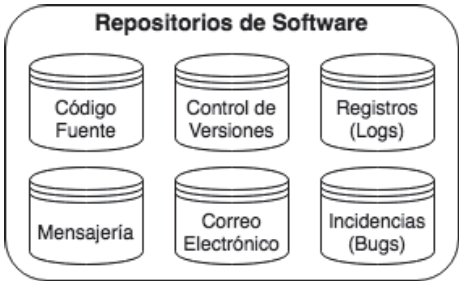

Figura 2.1: Repositorios de software más comunes.

A continuación describiremos dos de los repositorios más usados: repositorios de Control de Versiones y repositorios de Seguimiento de Incidencias [1](sección 5.3, 5.4, 5.6).

- **Sistemas de Control de Versiones**. Los VCS (por sus siglas en inglés), son sistemas que registran y mantienen los rastros en el desarrollo de un proyecto de software. Dan seguimiento de todos los cambios que surgen en cualquier artefacto dentro de un sistema, como el código fuente, los manuales de documentación, etc. Estos repositorios son los más empleados y más fácilmente disponibles en los proyectos de software. Git, CVS y Perforce son algunos ejemplos de repositorios de control de versiones más populares que se utilizan en la práctica. Todos los cambios, sin importar su tamaño (grandes o pequeños), se registran a lo largo del tiempo para que sea posible recordar revisiones o versiones específicas de los artefactos del sistema en un futuro.
- **Sistemas de Seguimiento de Incidencias**. Los ITS (por sus siglas en inglés), rastrean y mantienen el historial de informes de incidencias, brindando información valiosa sobre los proyectos de software, así como de los profesionales involucrados. Se utilizan generalmente en entornos colaborativos y distribuidos, permitiendo a los equipos dar seguimiento del progreso de las incidencias de un sistema en desarrollo. Una incidencia puede referirse a cualquier asunto que afecta/modifica en cualquier forma el desarrollo de un sistema de software; puede tomar la forma de *bugs*, cambios, mejoras o nuevas

implementaciones. Cada incidencia reportada en los ITS, contiene información como su descripción, su estado de resolución (Resuelta o No Resuelta), el tipo de incidencia (*Bug, Task, Improvement,* etc), cronología de comentarios publicados por los desarrolladores, fecha y zona geográfica donde se publicó el comentario, el número de desarrolladores implicados, entre otros. Jira, Trello y Asana son algunos ejemplos de repositorios de seguimiento de incidencias más populares que se utilizan en la práctica.

Como ya mencionamos, los repositorios de software registran varios tipos de información sobre la evolución y el progreso de un proyecto de software. Son diversas las aplicaciones/investigaciones que se pueden llevar a cabo al aplicar la MRS y sacar provecho de esta información. La figura 2.2 muestra algunas de ellas, como aplicaciones basadas en Minería de Textos, Predicción de Defectos, Estimación de Esfuerzo, entre otras.

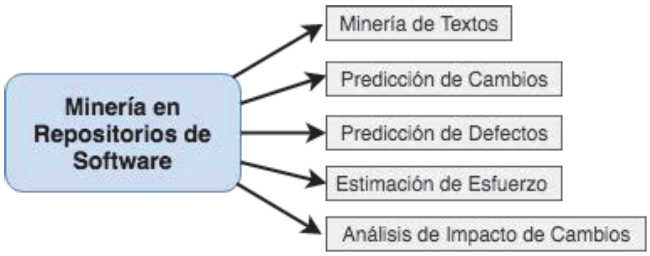

Figura 2.2: Aplicaciones de la Minería en Repositorios de Software.

A continuación daremos una descripción de las características principales de cada una de las aplicaciones [1](sección 5.9):

- **Minería de Textos**. Pertenece al campo de la Inteligencia Artificial y su objetivo principal es la exploración de grandes cantidades de datos textuales y derivar nuevos conocimientos a partir de ellos. La Ingeniería de Software es un área donde el trabajo en equipo es esencial, por lo que existen diversos canales/medios por los cuales los profesionales del software se comunican y trabajan de forma colaborativa. Son diversos los repositorios de software en donde se pueden encontrar rastros de estas comunicaciones textuales, como por ejemplo, repositorios de Mensajería (chats en línea), repositorios de Correo Electrónico, e incluso en los ITS y los VCS (descripciones de incidencias, comentarios de las incidencias, etc). Estos datos textuales pueden utilizarse para estudiar las relaciones sociales entre los desarrolladores, o bien, aplicar análisis de sentimientos y observar cómo las emociones influyen en el rendimientos de los equipos de desarrollo.

- **Predicción de Cambios**. La propensión al cambio puede definirse como la probabilidad de que cambie un componente de software. Al igual que la propensión a defectos, la propensión al cambio es muy importante y debe evaluarse con precisión. A partir del análisis histórico de cambios en los componentes, es posible construir modelos predictivos que ayuden a descubrir cambios probables en un sistema de software.

- **Predicción de Defectos**. La propensión a defectos (o predicción de defectos) es una de las áreas más activas de la investigación en ingeniería de software. Los repositorios mantienen registro sobre los defectos, los cuales se pueden analizar y derivar conocimiento útil. La información disponible contiene los detalles del error, como si fue corregido o no, la persona encarga de la corrección y los componentes afectados. Con esto es posible crear modelos de predicción de defectos basados en aprendizaje de maquina, cuyos resultados pueden influir en la toma de decisiones.

- **Estimación de Esfuerzo**. La estimación de recursos (como equipos, mano de obra, etc.) es fundamental para poder monitorear y controlar el que un sistema de software se finalice en tiempo y forma. Además, la estimación de esfuerzo del software puede ayudar a que los procesos de planificación sean más precisos de acuerdo a las necesidades. La sobreestimación del esfuerzo puede conducir al riesgo de que se asignen demasiados recursos al proyecto. Por otro lado, la subestimación del esfuerzo puede llevar a calendarios ajustados y poner en riesgo las entregas en tiempo. En años recientes, la importancia de la estimación del esfuerzo de desarrollo de software ha motivado a la construcción de modelos (basados en aprendizaje de máquina) para predecir costos y esfuerzos en el desarrollo de sistemas de software.

- **Análisis de Impacto de Cambios**. En la gestión de la configuración de software, uno de los aspectos importantes es conocer el impacto del cambio solicitado en el sistema, por lo que, al aplicar MRS, es posible diseñar y construir modelos predictivos a partir de datos históricos que ayuden en la toma de decisiones relacionadas con la implementación del cambio.

2.2. Análisis de sentimientos

Debido al avance tecnológico que hemos experimentado en las últimas décadas, día con día se generan grandes volúmenes de datos en forma de textos, imágenes, audios, vídeos, etc. Según estimaciones de la industria, el 80 % de los datos generados vienen en un formato no estructurado, esto es, no está organizada de una manera definida o bien no cuenta con un modelo pre-definido; además, más del 50 % de estos datos no estructurados se encuentran en forma de texto. Estos datos se generan mientras interactuáramos con otras personas y/o sistemas mediante la gran cantidad de plataformas digitales disponibles en internet, por ejemplo, redes sociales, plataformas de mensajería y vídeo, comercios electrónicos, asistentes virtuales, entre otros [7].

El gran volumen de contenido generado por los usuarios ha permitido que diversas áreas de estudios puedan explotar y analizar estos datos. Una de ellas es el Análisis de Sentimientos, también conocido como Minería de Opiniones, cuyo objetivo principal es el análisis del lenguaje escrito con el fin de extraer opiniones, sentimientos, actitudes y emociones expresados por las personas hacia productos, servicios, individuos, temas en general, entre otros [8].

En años recientes el Análisis de Sentimientos ha tenido un gran crecimiento en uso y popularidad a nivel comercial, académico y social, debido a que juegan un papel muy importante en la toma de decisiones para su beneficio. Por ejemplo, cuando una organización necesita conocer los sentimientos/opiniones que sus clientes o el público en general expresan acerca de sus productos o servicios, es posible que, mediante un sistema automatizado, se haga la extracción y análisis de los datos en forma de comentarios/reseñas que se publican en redes sociales o en su página web; esto les permite tener un mejor conocimiento del mercado y mejorar sus estrategias comerciales [9]. La figura 2.3 representa los datos de *Google Trends* referentes a la palabra clave "Análisis de Sentimientos" desde el 2010 a la fecha, lo que demuestra claramente el interés continuo y creciente en este campo.

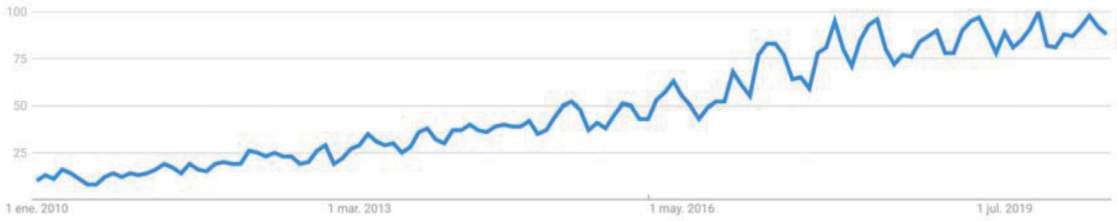

Figura 2.3: Datos de *Google Trends* referentes a la palabra clave Análisis de Sentimientos

Otros ejemplos interesantes en donde la tecnología central sea el Análisis de Sentimientos, se encuentran las aplicaciones desarrolladas por agencias del gobierno, en donde monitorean las redes sociales para descubrir los sentimientos del público y las preocupaciones de los ciudadanos sobre políticas gubernamentales. Por otro lado, diversos estudios han analizado los sentimientos de las opiniones públicas en el contexto de la política electoral; en un estudio O'Connor et al. [10] calcularon un puntaje de sentimiento en función del recuento de palabras con sentimiento positivo y negativo, en donde encontraron una correlación positiva con la aprobación presidencial y las encuestas de elecciones. Otra área de aplicación popular es la predicción del mercado de valores; en un estudio Zhang et al. [11] identificaron los estados de ánimo positivos y negativos en *Twitter* para predecir el movimiento de índices bursátiles. Demostraron que cuando la gente expresa más esperanza, miedo o preocupación, algunos índices bursátiles bajan, mientras que, los índices bursátiles suben cuando la gente tiene menos esperanza, miedo o preocupación.

El Análisis de Sentimientos se puede implementar principalmente en tres niveles de granularidad: nivel Documento, nivel Oración, nivel Aspecto [12]. A continuación mencionamos los aspectos generales de cada uno:

- **Nivel Documento**. Este nivel de Análisis de Sentimientos es el más usado y estudiado del campo. Su objetivo es clasificar un documento (por ejemplo una reseña de algún producto) como una opinión/sentimiento positivo o negativa (conocido como orientaciones o polaridades de sentimiento). Además, se considera cada documento como un todo, sin tomar en cuenta entidades o aspectos dentro del documento ni los sentimientos expresados sobre ellos.

- **Nivel Oración**. El objetivo de este nivel es clasificar cada oración en un documento como una opinión/sentimiento positivo, negativa o neutral. La clasificación en este nivel es muy similar a la clasificación a nivel de documentos porque las oraciones pueden considerarse como documentos breves. Sin embargo, la clasificación a nivel oración suele ser más difícil porque la información contenida en una oración es mucho menor que la contenida en un documento debido a su diferencia de longitud.

- **Nivel Aspecto**. En este nivel se intenta cubrir las desventajas de los dos niveles anteriores, los cuales son insuficiente para la mayoría de las aplicaciones principalmente porque no identifican el objetivo o la entidad a la cual el sentimiento/opinión va dirigida. Un documento con sentimiento positivo o negativo hacia una sola entidad no significa que el autor sea positivo o negativo sobre todos los aspectos de la entidad. Es por esto que este nivel tiene el objetivo de realizar un análisis más completo en donde identifique el sentimiento positivo, negativo o neutral sobre cada aspecto.

Por otro lado, existen dos enfoques principales para aplicar un Análisis de Sentimientos (ver Figura 2.4), uno basado en Aprendizaje de Máquina y otro basado en Lexicones (y un tercer método Híbrido que combina ambos) [13]. A continuación mencionamos los aspectos generales de cada uno:

- **Enfoque de Aprendizaje de Máquina**.
  Este enfoque se divide en dos categorías, una A) Basada en Aprendizaje Supervisado y otra B) Basada en Aprendizaje No Supervisado. En el tipo A se debe contar con un *dataset* etiquetado (es decir, texto con su sentimiento asociado) para entrenar un modelo de clasificación y después realizar la predicción sobre nuevos datos (es decir, asociar un sentimiento sobre nuevos textos). Algunos algortimos de clasificación supervisado son *Naïve Bayes*, *Support Vector Machine* o *Decision Tree*. Por otro lado, el tipo B se utiliza cuando no se cuenta con datos etiquetados y se hace uso de algoritmos de agrupación.

- **Enfoque de Lexicones**.
  Es considerado como un enfoque de clasificación de sentimientos no supervisado y de los más sencillos de implementar. Este enfoque implica calcular la orientación-sentimiento de un documento a partir de la orientación semántica de las palabras o frases del documento (texto). A cada expresión positiva (una palabra o frase) se le asigna un valor de orientación positivo y a cada expresión negativa se le asigna un valor orientación negativo. Con esto es posible obtener una puntuación numérica calculada por el algoritmo y determinar el sentimiento general del documento (Positivo, Negativo o Neutral).

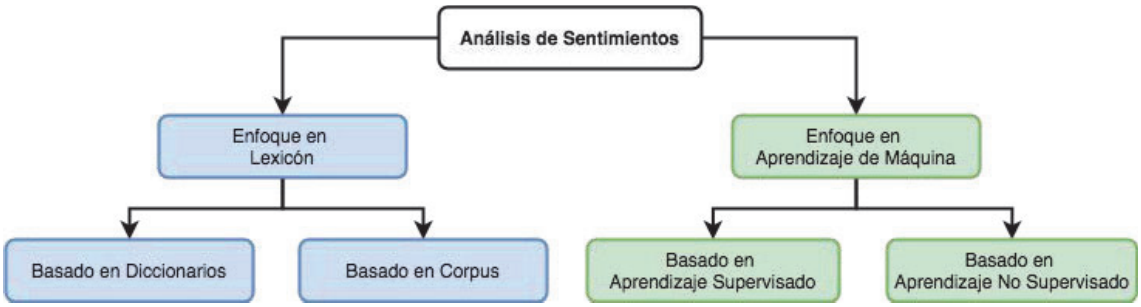

Figura 2.4: Principales enfoques utilizados para el Análisis de Sentimientos

El Análisis de Sentimientos reúne varias áreas de investigación, como la Minería de Textos y el Procesamiento del Lenguaje Natural, las cuales permiten extraer información significativa a partir de los datos en forma de texto y con ello generar conocimiento estructurado y procesable.

Con el fin aplicar una clasificación de sentimientos supervisados es necesario abordar conceptos relacionados al campo de estudio de la Minería de Textos y el Procesamiento de Lenguaje Natural (PLN), en la siguiente sección hablaremos de ellos.

2.3. Minería de textos

La Minería de Textos es un campo de estudio que hace uso de métodos de investigación y herramientas de software con el fin de encontrar patrones, derivar nuevos conocimientos y tendencias a partir del análisis de grandes cantidades de texto. En tiempos pasados, las investigaciones o estudios sociales analizaban la información contenida en los datos textuales (documentos, revistas, encuestas, etc) de forma manual, lo que representaba una tarea de gran consumo de tiempo y propenso a errores humanos. Hoy en día, es común hacer uso de técnicas de Minería de Textos para automatizar los procesos de extracción y análisis de los datos textuales. Por ejemplo, es posible hacer predicciones de cualquier clase de tarea/proyecto de una forma más rápida y precisa, como predecir la dirección del mercado de valores o la ocurrencia de protestas políticas, o bien, aplicarlo en áreas comerciales, comunicaciones, educación y economía [14].

La Minería de Textos, junto con técnicas de PLN y métodos estadísticos, permite la aplicación de una amplia variedad de tareas que pueden aportar información sobre diferentes aspectos de los textos [15]. Algunas de sus tareas principales son:

- **Clasificación de Textos**. También conocida como Categorización de Documentos, se encarga de asignar un documento a una o más categorías predefinidas. Ejemplos donde se implementa sería en la identificación de correos electrónicos basura (spam) y el análisis de sentimientos (clasificación de textos como positivos, negativos o neutrales).

- **Recuperación de Información**. Se utilizan métodos para la adquisición/recuperación de documentos que coinciden con una consulta a partir de una gran colección de documentos.

- **Resumen (Summarization)**. Encuentra las partes más importantes en uno o más documentos y genera un texto que sea significativamente más corto que el original.

- **Extracción de Información**. Se encarga de extraer información estructurada o encontrar relaciones en textos no estructurados. El Reconocimiento de Entidades (identificar nombres de personas, organizaciones, lugares, etc) o Reconocimiento de Términos (extracción de términos relevantes en textos) son aplicaciones típicas.

Como se mencionó anteriormente, el Análisis de Sentimientos se refiere al proceso de identificar el sentimiento/opinión expresado en los textos. Esto implica clasificar sentimientos en una de las clases predefinidas: positiva, negativa o neutral. Dado que uno de los objetivos de este trabajo es aplicar un Análisis de Sentimientos en repositorios de software, nos centraremos en la tarea de Clasificación de Textos en el proceso de Minería de Textos.

La tarea de Clasificación de Textos se define como un proceso en el que, dado un texto de entrada, se le asignan una o más categorías predefinidas. Existen principalmente tres tipos de enfoques para la clasificación de textos [16][17]: **A)** *Clasificación Binaria*, es la tarea de clasificación en su forma más simple y la cual consiste en clasificar un texto en una de dos categorías o clases existentes, donde cada texto pertenece exclusivamente a una clase (no puede pertenecer a ambas); tomando como ejemplo el sistema de detección de correo basura, en donde, dado un nuevo correo de entrada solamente puede ser asignado una de estas dos clases: *spam* o *nospam*. Por otro lado se encuentra **B)** *Clasificación Múlticlase*, cuya diferencia radica en que ahora se cuentan con dos o más clases para clasificar un texto (de igual manera cada texto pertenece exclusivamente a una clase); por ejemplo, considerando un conjunto de artículos que fueron escritos para distintos periódicos, se puede contar con un sistema que

ayude a clasificarlos de acuerdo a su tema principal, como: *Deportes, Tecnología, Política y Negocios*, siendo estas las clases de asignación. Por último está **C)** *Clasificación Múltietiqueta*, en este caso existen múltiples clases, en donde una o más de estas pueden ser asignadas a un texto de entrada; por ejemplo, asignar un conjunto de emociones básicas a una serie de comentarios de *twitter* sobre las elecciones presidenciales, siendo las clases de asignación: *Alegría, Ira, Tristeza, Sorpresa y Miedo*, en donde cada comentario puede pertenecer (expresar) a una o más de estas clases de emociones a la vez.

Por otro lado, para lograr implementar alguno de los enfoques mencionados en el párrafo anterior, se pueden seguir los pasos que se muestran en la figura 2.5, los cuales describen el proceso general de aplicación de Minería de Textos para la tarea de clasificación.

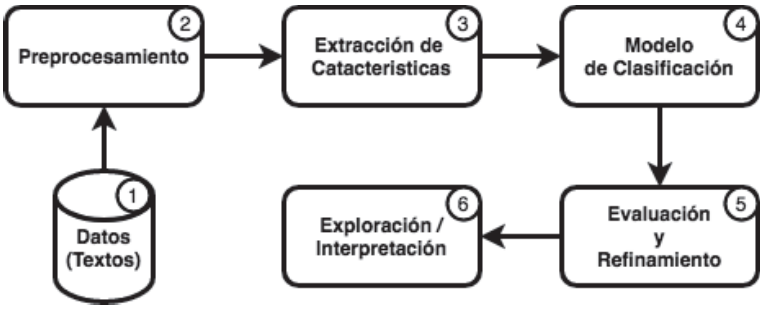

Figura 2.5: Proceso de Minería de Textos para la tarea de clasificación de textos.

Este proceso consta principalmente de seis etapas descritas a continuación [18]:

1. **Recopilación de Datos**. En esta etapa se realiza la identificación, extracción y almacenamiento de los datos. Esto es, primero se identifica las fuentes de los datos que contienen la información deseada (bases de datos, repositorios, páginas web, etc). Después se debe aplicar alguna técnica o herramienta para la extracción de los datos (API's, Consultas SQL, Web Scraping, etc). Por último, se almacenan los datos recopilados en el formato adecuado para su posterior análisis.

2. **Preprocesamiento de Datos**. En esta etapa se aplican técnicas de limpieza de datos y de PLN con el fin de tener textos más representativos en el campo de aplicación. Algunas de las técnicas más comunes son: Manejo de cadenas (uso de regex), Remoción de *StopWords*, Normalización de textos (*Stemming* y *Lemmatization*), Tokenización (uso de n-gramas), entre otros.

3. **Extracción de Características**. También llamado Vectorización, esta etapa se encarga de la transformación de los textos en representaciones vectoriales de modo que puedan ser procesados por los modelos de clasificación. Esto debido a que los algoritmos de aprendizaje automático operan en un espacio de características numéricas, los cuales esperan como entrada (una matriz bidimensional mxn) un conjunto de instancias (m filas) junto con sus características (n columnas).

4. **Modelo de Clasificación**. Con el fin de construir un clasificador de textos bajo un enfoque de aprendizaje supervisado, primero es necesario seleccionar el algoritmo de clasificación (como Regresión Logística, Máquinas de Soporte de Vectores o Potenciación del Gradiente), se realizan ajustes de parámetros y por último se entrena el modelo (mediante instancias de entrenamiento).

5. **Evaluación y Refinamiento del Modelo**. El proceso de construcción de un modelo es un ciclo iterativo donde constantemente se realizan diversos ajustes con el fin de mejorar su desempeño. Es en esta etapa donde se evalúa el rendimiento del modelo (utilizando métricas como *Precision*, *Recall*, *F1Score*, *Accuracy*, etc) y se determina si es necesario hacer cualquier ajuste en cualquiera de las etapas anteriores del proceso (preprocesamiento, vectorización, etc) para refinarlo. Una vez obtenido el desempeño apropiado, el modelo está listo para realizar predicciones sobre nuevos datos.

6. **Exploración-Interpretación**. Esta última etapa del proceso se encarga de explorar e interpretar los resultados utilizando herramientas de visualización (tablas, reportes, gráficos). Se extrae información útil, se identifican patrones y se derivan nuevos conocimientos.

En las siguientes secciones se dará más detalle de cada una de las etapas, describiendo las técnicas y metodologías más utilizadas dentro del proceso de Minería de Textos para la tarea de clasificación.

2.3.1. Recopilación de datos

Existen diversas técnicas que permiten la extracción de información desde una fuente de datos (como las bases de datos, repositorios, páginas web, etc). El tipo de fuente y el formato de los datos, indica que tipos de técnicas de extracción implementar. Entre las ténicas más comunes se encuentran el uso de Web API's, consultas SQL o Web Scraping. A continuación describimos cada una de ellas [19][20]:

- **Aplication Programming Interface (Web API)**. Es un conjunto de reglas/funciones que permite la conexión e intercambio de información entre dos componentes de software en la web. Una API permite agregar funcionalidades extra al componente que la consume, como almacenar datos en una base, ejecutar rutinas de código en servidores externos, o bien, extraer cualquier tipo de información que sea de utilidad. Una de las arquitecturas más comunes son las llamadas *API REST*, la cual trabaja bajo un enfoque cliente-servidor, se consumen recursos utilizando el protocolo *http* mediante sus métodos *get, post, put, delete* y se intercambia información en un formato *JSON*[1].

- **Consultas SQL**. Es un lenguaje utilizado para realizar consultas sobre bases de datos estructuradas (como *MySQL, PostgreSQL, etc*). Es de los lenguajes de consulta más populares y usados hoy en día, y se pueden implementar utilizando cualquier lenguaje de programación de alto nivel. En este caso se deben contar con los permisos de accesos necesarios a las bases de datos y, además, conocer previamente los esquemas (nombres/formas de las bases y sus tablas) para su correcto consumo.

- **Web Scraping**. También conocido como Extracción de Datos Web, es una técnica que permite extraer datos no estructurados de los sitios web para después transfórmalos en datos estructurados y almacenarlos en algún repositorio centralizado (en un formato de base de datos, hoja de cálculo, JSON, etc) para su posterior análisis. Existen herramientas de software libre que implementan esta técnica de forma automatizada (como *Scraper API y ScrapeSimple*).

[1]En este trabajo de tesis esta es la técnica que se utiliza para extraer datos de los repositorios de software, en donde el sistema de Jira expone una API REST para el consumo de sus datos. Más adelante, en el capítulo 4, se describirán a detalle los pasos para la recopilación de datos

2.3.2. Preprocesamiento de textos

El preprocesamiento de textos consiste en aplicar una serie de métodos al lenguaje escrito con el fin de obtener una versión normalizada del mismo y así mejorar el rendimiento de los clasificadores de textos. Algunos de estos métodos incluyen: remover etiquetas HTML (documentos de la web), eliminar palabras vacías (como artículos, preposiciones, conectores), expandir abreviaciones, tokenización, lemmatization, stemming, etc.

A continuación se describen algunas de las principales técnicas/métodos incluyen [14][21]:

- **Limpieza de datos**. Esta parte se encarga de implementar técnicas para el manejo de cadenas con el fin de aplicar limpieza a los datos, por ejemplo, para convertir todas las letras a minúsculas o mayúsculas, convertir números en palabras o eliminar números, eliminar signos de puntuación/acentos, eliminar espacios en blanco, expandiendo abreviaturas. Los lenguajes de programación de alto nivel (como Python o R) permiten hacer uso de funciones integradas o bibliotecas para facilitar este proceso. Además, se puede hacer uso de expresiones regulares (*regex*, por abreviación en ingles), las cuales se componen de secuencias de caracteres que definen un patrón de búsqueda; esto permite encontrar patrones en los textos de una manera muy rápida y flexible.

- **Tokenización**. Es un proceso de segmentación de los textos en *tokens* mediante un espacio en blanco o signos de puntuación. Los *tokens* son componentes textuales mínimos e independientes que tienen alguna sintaxis y semántica definidas. La lista de *tokens* generada se convierte en la entrada de los siguientes pasos de normalización de texto: Remoción de Palabras Vacías y *Stemming/Lemmazitation*. La figura 2.6 muestra un ejemplo de tokenización de un texto mediante espacios en blanco.

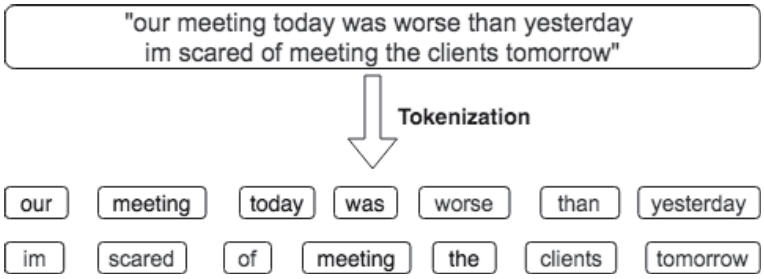

Figura 2.6: Proceso de tokenización de un texto por espacios en blanco.

- **Remoción de Palabras Vacías**. En inglés se conocen como *StopWord* y se refiere al proceso de eliminar las palabras gramaticales que son irrelevantes o aportan poca información en el contenido del texto. Generalmente estas palabras se repiten con frecuencia, como los pronombres (*i, we, us*), las preposiciones (*in, on, at*), los artículos (*a, an, the*), entre otros. Para la gran mayoría de las tareas resulta útil y eficiente la remoción de las palabras vacías debido a que permite enfocarse en el contenido de las palabras como sustantivos y verbos. Para aplicar este método se debe contar con una lista de palabras vacías, en donde, dado un texto como entrada, se compara cada palabra con el contenido de la lista y se remueven las registradas.

- **Stemming**. Una vez generada la lista de *tokens*, en este proceso se realiza un mapeo de cada *token* en su forma raíz o base. El objetivo de este procedimiento es convertir

todas las formas diferentes de una palabra en una forma común conocida como *stem* (raíz). El significado semántico de algunas palabras se crean a partir de la misma raíz, siendo muy similares (por ejemplo, las palabras *fishing, fished, fisher*, tienen su raíz en la palabra *fish*). Esto permite reducir las ocurrencias de palabras con el mismo significado lo que deriva en textos más significativos. Una desventaja de este método es que recurre a reglas de eliminación de prefijos y sufijos (como *s*, *es*, *er*) para poder encontrar la raíz de una palabra, produciendo palabras que no son válidas/correctas, o bien, pierden el significado en el contexto original de la oración

- **Lemmatization**. Este procedimiento cubre las desventajas que presenta el procedimiento Stemming. De igual manera, partiendo de una lista de *tokens*, implementa un análisis morfológico de las palabras obteniendo su lema (su forma canónica o de diccionario). Por ejemplo, las palabras *better* y *thoughts* son convertidas en su lema *good* y *think*, respectivamente. Además, utiliza lo que se conoce como etiquetado gramatical (Part Of Speech o POS, por sus siglas en inglés) en donde se marca cada palabra de un texto según su definición y su contexto (asignando etiquetas según sean verbos, sustantivos, pronombres, adjetivos, etc). En la práctica es más complejo implementar un procedimiento de Lemmatization que uno de Stemming, pero el primero suele dar mejores resultados

2.3.3. Extracción de características

La falta de una estructura definida (sin un formato ordenado) y su naturaleza ruidosa de los datos textuales dificultan que los modelos de aprendizaje automático trabajen directamente con datos de texto sin procesar. Es por esto que los modelos de aprendizaje solo permiten como entrada una representación vectorial (numéricas) de los textos. Al proceso de transformación de un texto a un vector se le conoce como extracción de características o vectorización [14].

En esta sección describiremos algunas de las estrategias más populares y efectivas para extraer características significativas de los datos textuales, como: ***One-Hot Encoding***, ***Frecuency***, ***TF-IDF***. Estas estrategias se basan en un enfoque llamado ***Bag-Of-Words*** (BOW), el cual a partir de un corpus construye un vocabulario que contiene palabras únicas (sin repetición), esto es, cada palabra del corpus aparece solo una vez en el vocabulario [21][22]. Con esto se realiza la etapa de vectorización, codificando cada documento (texto) de entrada de acuerdo a la frecuencia con la que aparecen cada una de las palabras del vocabulario en el documento. Como ejemplo, utilizaremos el siguiente texto como corpus "*IA is future, I like it, I will learn IA in two months*", el vocabulario o bolsa de palabras generado (y tokenizado) se observa en la figura 2.7.

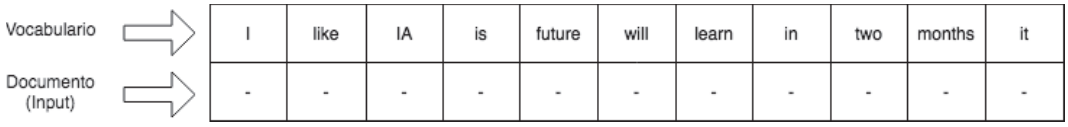

Figura 2.7: Vocabulario (BOW) generado a partir del corpus.

- **One-Hot-Encoding (OHE)**. Este método genera un vector binario (1's y 0's) a partir de un documento (texto) de entrada, en donde, cada palabra (token) del vocabulario toma el valor de 1 si existe en el documento, de lo contrario toma el valor de 0. En otras palabras, cada elemento en el vector OHE refleja la presencia o ausencia de la palabra en el vocabulario. Un ejemplo se muestra en la figura 2.8, considerando como base el vocabulario de la figura 2.7 y como documento de entrada "*IA is future*".

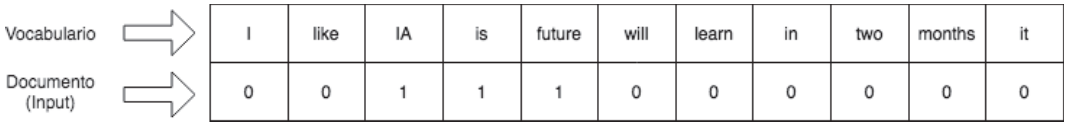

Figura 2.8: Vectorización mediante One-Hot-Encoding

- **Frecuency**. Este método es muy similar al método OHE, la única diferencia es que, además de checar si cada palabra del vocabulario está presente o no en el documento, realiza el conteo/ocurrencia de cada palabra. Un ejemplo se muestra en la figura 2.9, considerando como base el vocabulario de la figura 2.7 y como documento de entrada "*I like IA and I will learn IA in two months*". Se observa que las palabras "I" y "IA" aparecen dos veces en el documento, por lo que la representación vectorial contiene dos ocurrencias de cada una.

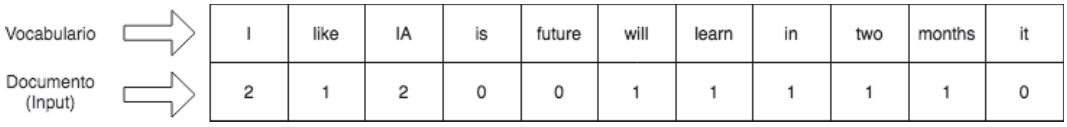

Figura 2.9: Vectorización mediante conteo de frecuencias.

- **TF-IDF**. Las técnicas anteriormente mencionadas (OHE y Frequency) tienen la desventaja que, si una palabra en particular está apareciendo en todos los documentos del corpus, entonces adquirirá mayor importancia, lo que podría derivar en un mal análisis. Es por ello que surge Term Frequency–Inverse Document Frequency o TF-IDF, la cual es una técnica común para normalizar la frecuencia de tokens en un documento con respecto al contexto del corpus, es decir, su objetivo es dar un mayor peso a cualquier término que aparezca con frecuencia en un documento en particular, pero no en muchos documentos del corpus. Como su nombre lo indica TF-IDF es el producto de dos componentes: 1) ***Term Frecuency (TF)***, el cual indica con qué frecuencia aparece un término (t) en un documento (d) y 2) ***Inverse Document Frequency (IDF)***, el cual mide la importancia de un término, reduciendo el valor de palabras comunes que aparecen en diferentes documentos. Por lo que peso de un término (t) en un documento (d), se calculo como:

$$tfidf(t,d) = TF * IDF = tf_{t,d} * log\left(\frac{N}{df_t}\right)$$

  Donde N es el número total de documentos en el corpus, df_t es el número de documentos en donde aparece la término (palabra) t, $tf_{t,d}$ indica el número de veces o la frecuencia con la que el término t aparece en el documento d .

Por otro lado, estas estrategias de vectorización se basan en frecuencias (conteo de palabras) sin tomar en cuenta la estructura y contexto del texto. Existen estrategias de vectorización más avanzados como ***Word Embeddings*** que cubren las limitantes de las técnicas BOW y se encargan de capturar el contexto y las relaciones semánticas de las palabras. Son estrategias basadas en predicción y utilizan Redes Neuronales para entrenar modelos para generar las representaciones vectoriales del texto (el uso de está técnica queda fuera del alcance de esta tesis, y se pretende aplicar en trabajos a futuro).

2.4. Aprendizaje de máquinas

El Aprendizaje de Máquina (*Machine Learning*) es un campo de investigación dentro del área de la Inteligencia Artificial cuyo objetivo principal es dotar a las máquinas con la habilidad de aprender a partir de un conjunto de datos, apoyándose en técnicas y métodos de áreas como la Estadística e Informática (también se le conoce como análisis predictivo o aprendizaje automático). Las aplicaciones que hacen uso de métodos de aprendizaje de máquina han tenido un crecimiento exponencial en los últimos años. Desde aplicaciones para uso comercial (traductores, sistemas de recomendación, detección de fraude, etc) hasta investigaciones y estudios basados en datos (estudios sobre enfermedades, autos autónomos, etc) [17][22]. Entre los enfoques principales y más utilizados se encuentran:

- **Supervisados**. Los datos de entrenamiento que alimentan al modelo de aprendizaje incluyen las soluciones deseadas (etiquetas). Dos de las tareas más comunes son las de clasificación (predicen un valor categórico, como un filtro de correo *spam* o el análisis de sentimientos) y regresión (predicen un valor numérico, como predecir el valor de casas/coches).

- **No Supervisados**. Al contrario del enfoque supervisado, los datos de entrenamiento no incluyen las soluciones deseadas, es decir, no están etiquetados. Entre las tareas más comunes se encuentran las de *Clustering* (identifica grupos de tal manera que sus miembros compartan características similares), *Dimensionality Reduction* (simplifica los datos sin perder mucha información) y *Anomaly Detection* (identifica datos inusuales/atípicos en un conjunto de datos).

- **Semi-Supervisados**. Es una combinación de aprendizaje supervisado y no supervisado, en donde algunos de los datos de entrenamiento están etiquetados (minoría) y otros no etiquetados (mayoría). Un ejemplo de este enfoque es cuando se cuenta con un archivo de fotos donde solo algunas de las imágenes están etiquetadas (personas, lugares, objetos, etc) y la mayoría no están etiquetadas.

En las siguientes secciones nos centraremos particularmente en el aprendizaje supervisado debido a que será el que se implemente en este trabajo de tesis.

2.4.1. Aprendizaje supervisado

Como ya se mencionó anteriormente, en este tipo de aprendizaje los datos de entrenamiento que se proporcionan al algoritmo incluyen las soluciones deseadas, llamadas etiquetas [17]. En la figura 2.10 se muestra el proceso de entrenamiento general para un sistema basado en aprendizaje supervisado. Se observa que, a partir de un conjunto de datos etiquetados (azul) y un algoritmo de aprendizaje (rojo), se construye y entrena un modelo (blanco), el cual es usado para predecir valores (verde) para nuevos datos de entrada (naranja).

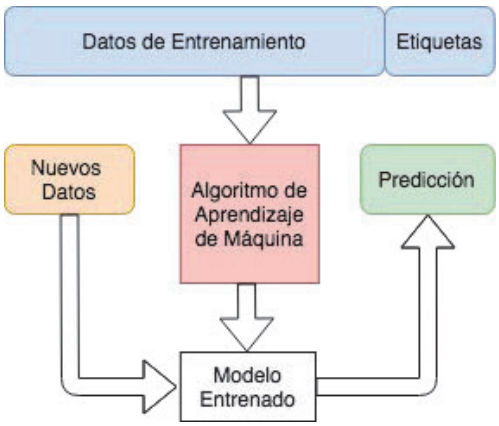

Figura 2.10: Proceso de entrenamiento bajo un enfoque de aprendizaje supervisado.

Por otro lado, existen dos tareas principales dentro del aprendizaje supervisado: de clasificación y de regresión. A continuación describimos cada una de estas.

1. **Clasificación**. Este enfoque implica la predicción de un valor categórico o asignación de una clase definida para una instancia de entrada. En la figura 2.11 (A) se muestra un clasificador binario, el cual se conforma de un modelo entrenado y un conjunto de datos etiquetados con dos clases predefinidas. Un ejemplo práctico es un filtro de *spam* en correos electrónicos: se construye y se entrena un modelo (línea roja) con ejemplos de correos etiquetados como *spam* (triangulo) y correos etiquetados como no *spam* (círculos) y el modelo debe aprender a clasificar nuevos correos electrónicos.

2. **Regresión**. Este enfoque implica la predicción de un valor numérico objetivo, dado un conjunto de características (features o predictores). En la figura 2.11 (B) se muestra un modelo de regresión lineal, el cual se conforma de un modelo entrenado y un conjunto de datos etiquetados. Un ejemplo práctico es la predicción del precio de automóviles, que de igual manera que para el enfoque de clasificación, se entrena un modelo (línea roja) a partir de ejemplos de precios (círculos) incluidos sus características (kilometraje, edad, marca, etc.).

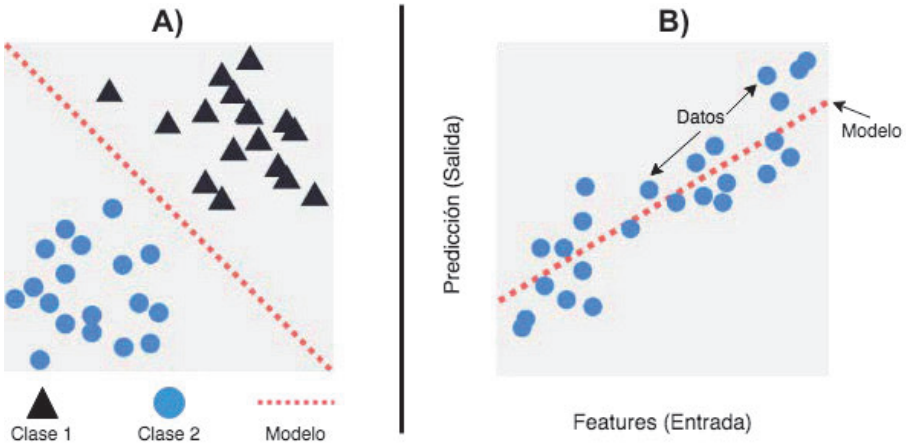

Figura 2.11: Modelos de A) Clasificación B) Regresión supervisados.

2.4.2. Algoritmos de clasificación

Existen diferentes algoritmos para construir modelos de clasificación, ente los principales se encuentran [17][23][24]:

- **Regresión Lógistica** (*LogisticRegression*). Este algoritmo estima la probabilidad de que una instancia (nuevo dato) pertenezca a una clase en particular. Si la probabilidad estimada es superior a 0.5 (o 50 %), el modelo predice que la instancia pertenece a una clase positiva (etiquetada como "1"), o bien, pertenece aa una clase negativa (etiquetado como "0") cuando la probabilidad es menor a 0.5. El modelo se representa por la siguiente expresión vectorial $p = h_\theta(x) = \sigma(X^T\theta)$, donde σ es una función sigmoide que genera como salida un número entre 0 y 1, la cual se define como: $\sigma(t) = \frac{1}{1+exp(-t)}$

- **Maquinas de Vectores de Soporte** (*LinearSVC*). El objetivo de este algoritmo (en su representación lineal) es encontrar un hiperplano en un espacio n-dimensional (siendo n el número de características) que clasifique claramente los puntos de datos. Con el fin de separar en dos clases (de acuerdo a la distribución de los datos) existen varios hiperplanos posibles que podrían elegirse, por lo que se debe encontrar un plano que tenga el margen máximo, es decir, la distancia máxima entre puntos de datos de ambas clases.

- **Potenciación del Gradiente** (*GradientBoostingClassifier*). Se compone de un grupo de algoritmos de aprendizaje automático que combinan varios modelos de considerados "débiles" con el fin de crear un modelo de clasificación más sólido. Para implementar este algoritmo se deben considerar tres componentes: 1) Una función de costo, la cual se selecciona dependiendo del problema a resolver, 2) Un modelo "débil", en donde generalmente se utiliza el algoritmo de árboles de decisión y un 3) modelo aditivo en el que se van agregando árboles uno a uno, en donde los árboles ya existentes en el modelo se van modificando y, además, se utiliza un procedimiento de descenso de gradiente para minimizar el error al agregar árboles.

- **Bosques Aleatorios** (*RandomForestClassifier*). Este algoritmo utiliza un método llamado Aprendizaje en Conjunto, en el que se construyen múltiples árboles de decisión para el entrenamiento del modelo. Se toman las predicciones de cada árbol de decisión y se calcula su promedio para obtener un valor de predicción final, lo que da como resultado un mejor rendimiento que si solo se utilizara un solo árbol en el modelo.

Otros algoritmos de clasificación importantes son:

- **Árboles de Decisión** *(DecisionTreeClassifier).*
- **Descenso del Gradiente Estocástico** *(SGDClassifier).*
- **K Vecinos más cercanos** *(KNeighborsClassifier).*
- **Naive Bayes** *(MultinomialNB, GaussianNB, BernoulliNB).*

2.4.3. Evaluación de rendimiento

Existen diversas técnicas que permiten evaluar el rendimiento de un modelo de clasificación, entre las principales se encuentran [17][25]:

- **Matriz de Confusión**.

  El objetivo de una matriz de confusión es el de observar/visualizar de forma rápida y concisa el número de veces en el que modelo de clasificación acertó y en el que confundió las diferentes clases. Por ejemplo, el número de veces en el que las instancias de la clase A se clasificaron correctamente como clase A y en las que se clasificaron incorrectamente como clase B (y viceversa). En la figura 2.12 se muestra una forma gráfica de la matriz de confusión para dos clases (Positivo y Negativo), en sus columnas se muestran los valores actuales y en las filas los calores de predicción para cada instancia. Todas las instancias de prueba se distribuyen en las diferentes celdas de la matriz, las cuales se interpretan de los siguiente forma:

  - **Verdaderos Positivos** (TP), contiene aquellas instancias en las que su clase actual es positiva y su clase de predicción es positiva.
  - **Verdaderos Negativos** (TN), contiene aquellas instancias en las su clase actual es negativa y su clase de predicción es negativa.
  - **Falsos Positivos** (FP), contiene aquellas instancias en las que su clase actual es negativa y su clase de predicción es positiva.
  - **Falsos Negativos** (FN), contiene aquellas instancias en las que su clase actual es positiva y su clase de predicción es negativa.

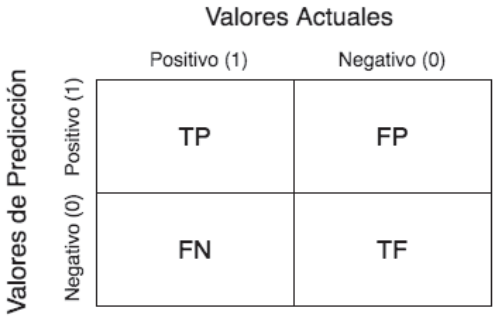

Figura 2.12: Matriz de Confusión.

- **Accuracy**. De todas las clases, cuántas predicciones fueron correctamente realizadas por el modelo. Se calcula como:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**. De todas las clases positivas que se predijeron correctamente (valor de predicción), cuántas son realmente positivas (valor actual). Se calcula como:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**. De todas las clases positivas (valor actual), cuantas se predijeron correctamente (valor de predicción). Se calcula como:

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

- **F1Score**. Se define tomando en cuenta las métricas Precision y Recall, calculando una media armónica, esto es:

$$F1Score = \frac{2 * Recall * Precision}{Recall + Precision}$$

- **Micro Average** Es una medida que considera los TP, TN, FP, FN para cada una de las clases existentes. Es útil cuando se cuenta con datasets desbalanceados debido a que toma en cuenta la proporción de cada clases. Se calcula como (donde c es cada clase):

$$MicroAveragePrecision = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c}$$

$$MicroAverageRecall = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FN_c}$$

- **Macro Average** Es una medida que considera el promedio de las métricas *Precision* y *Recall* de las clases existentes. Es útil para observar el rendimiento del modelo en general y se calcula como (donde c es cada clase y N es el número total de clases):

$$MacroAveragePrecision = \frac{\sum_c Precision_c}{N}$$

$$MacroAverageRecall = \frac{\sum_c Recallc}{N}$$

- **Curvas AUC-ROC**. Como se muestra en la figura 2.13, estas curvas se generan al graficar la tasa de FP (FPR en eje x) contra la tasa de TP (TPR en eje y) mientras se va variando el límite/umbral de decisión en el modelo de clasificación (línea verde). Mientras que, AUC, mide el área bajo la curva generada (área gris, toma valores entre 0 y 1), la cual nos indica qué tan bueno es el modelo en la distinción entre clases; entre más grande sea el área bajo la curva mejor es el rendimiento del modelo.

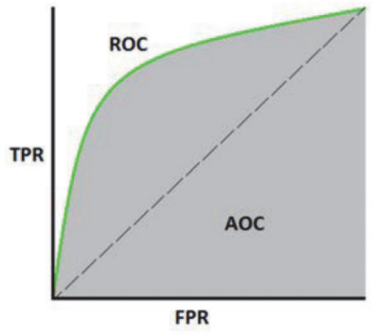

Figura 2.13: Curva AUC-ROC.

Capítulo 3

Trabajos Relacionados

> "Juntarse es un comienzo, seguir juntos es un progreso y trabajar juntos es un éxito"
>
> *Henry Ford*

Como ya mencionamos anteriormente la Minería en Repositorios de Software (MSR) es un campo de estudio que se encarga de extraer información valiosa de la gran cantidad de datos disponibles dentro de los repositorios de software, como los Sistemas de Seguimiento de Incidencias y los Sistemas (Jira) de Control de Versiones (GitHub). Esta información acerca de los sistemas y proyectos puede ser explotada de distintas maneras a fin de derivar nuevos conocimientos, encontrar tendencias, mejorar procesos y tener una mejor comprensión del desarrollo y evolución de la Ingeniería de Software.

Diversos trabajos de investigación y estudios empíricos se han desarrollado con el fin de contribuir al campo de la MRS. Estos van desde estudios sobre la estimación de costo y esfuerzo de proyectos de software hasta estudios sobre el análisis de sentimientos en repositorios de software. Siendo esta última, un área que en años recientes ha tomado mayor interés en la comunidad científica y ha experimentado un crecimiento importante. Principalmente porque existe una mayor conciencia del efecto de los estados afectivos (sentimientos y emociones) sobre el desempeño laboral y los equipos de desarrollo.

En este capítulo se presenta un panorama global del estado del arte relacionado al análisis de sentimientos en la Ingeniería de Software. En la **sección 3.1** presentamos los trabajos más relevantes dentro del área, se muestran estudios donde se aplica el análisis de sentimientos en repositorios Jira/GitHub, estudios en donde se relacionan los sentimientos y la productividad de los desarrolladores, estudios en donde se exponen las limitaciones y malos resultados obtenido dentro del área, entre otros. En la **sección 3.2** se presentan los distintos datasets/corpus utilizados para estudiar el análisis de sentimientos en la Ingeniería de Software, describiendo su composición y enfoque de construcción. Y por último, en la **sección 3.3** se presentan las herramientas de uso público más relevantes (y usadas hoy en día) para la clasificación de sentimientos en la Ingeniería de Software, describimos su enfoque de desarrollo, sus características principales y su rendimiento.

3.1. Análisis de sentimientos en Ingeniería de Software

El Análisis de Sentimientos en la Ingeniería de Software es un área de estudio particular dentro de la MSR. El objetivo principal es identificar los sentimientos y emociones que experimentan los profesionales del software al analizar los canales de comunicación en entornos de desarrollo colaborativos, por ejemplo, en comentarios escritos en los sistemas de seguimiento de incidencias (Jira/Backlog), sistemas de control de seguimientos (Github/Bitbucket) y sitios técnicos de preguntas y respuestas (StackOverflow). A continuación mencionaremos algunos de los trabajos/estudios más recientes y relevantes dentro del área.

M. Ortu [5] estudió la relación entre los estados afectivos de los desarrolladores (sentimientos, emociones y amabilidad) y el tiempo de resolución de incidencias. El análisis se realizó sobre 560 mil comentarios de incidencias de proyectos de Apache alojados en repositorios de Jira. Sus resultados mostraron que cuando se expresan más emociones positivas (como el amor y la alegría), menor es el tiempo que tardan en resolver sus tareas; mientras que, al expresar más emociones negativas (como tristeza), mayor es el tiempo que tardan en resolver las tareas.

En otro estudio M. Ortu [26] investigó los aspectos sociales entre los equipos de desarrolladores y el atractivo de un proyecto de software desarrollado con el soporte de herramientas ágiles (como los tableros *Agile*). Para el estudio se consideraron 14 proyectos de código abierto, los cuales fueron desarrollados utilizando tableros *Agile* del sistema Jira. Se analizaron todos los comentarios escritos por los desarrolladores involucrados en los proyectos y se estudió si la cortesía (*politeness*) de los comentarios afectó la cantidad de desarrolladores involucrados a lo largo del tiempo requerido para solucionar cualquier incidencia. Los resultados mostraron que el nivel de cortesía en el proceso de comunicación entre los desarrolladores, sí tiene un efecto en el tiempo requerido para solucionar las incidencias, ya que en la mayoría de los proyectos analizados, existe una correlación positiva con el atractivo del proyecto tanto para los desarrolladores activos como para los potenciales. Esto es, cuanto más educados eran los desarrolladores, menos tiempo se tardaba en solucionar una incidencia y, en la mayoría de los casos analizados, cuanto más querían los desarrolladores formar parte del proyecto, más estaban dispuestos a seguir trabajando en el proyecto a lo largo del tiempo.

E. Guzman [6] mostró un análisis de sentimientos en los comentarios de *commits* de diferentes proyectos de código abierto alojados en repositorios de GitHub. Se estudió la relación de los sentimientos con diferentes factores como el lenguaje de programación utilizado, la hora y el día de la semana en que se realizó el *commit*, la distribución del equipo y la aprobación del proyecto. Los resultados mostraron que los proyectos desarrollados en lenguaje de programación Java tienden a tener más comentarios negativos cuando se comparan con otros lenguajes; también demostraron que entre mayor sea la distribución geográfica de los equipos de desarrollo, mayor es el sentimiento positivo expresado en sus comentarios. Además, mostraron que los comentarios escritos los días lunes tienden a expresar un sentimiento más negativo respecto a los demás días de las semana.

V. Sinha [27] presentó un análisis de sentimientos en los registros de *commits* para proyectos de GitHub. Se extrajeron un total de 28,466 proyectos que fueron desarrollados dentro de un período de siete años. Se plantearon analizar el sentimiento general del desarrollador en los mensajes de *commits*, estudiar la relación entre el sentimiento del desarrollador en los mensajes de *commits* y el día de la semana en que se realizó el *commit* y, también, analizar si existe una correlación entre el número de archivos modificados y sentimiento del desarrollador. Sus resultados mostraron que la mayoría del sentimiento expresado en todos los comentarios de *commits* fue neutro (74.7 %), seguido del sentimiento negativo (18.1 %) y por último el sentimiento positivo (7.2 %). Además, mostraron que los días martes es cuando mayor se expresan

los sentimientos negativos respecto a los demás días de la semana. Por último, encontraron una fuerte correlación (utilizando la prueba de correlación de Pearson >0.95) entre la cantidad de archivos modificados y el sentimiento expresado por los *commits* de las que formaban parte los archivos.

A. Murgia [28] realizó un estudio en el que analizaron si los artefactos involucrados en el desarrollo, como los reportes de incidencias, contienen información emocional sobre el desarrollo de software. La finalidad del estudio fue verificar la viabilidad de una herramienta automática para la extracción de emociones en artefactos de desarrollo de software. El análisis se realizó para 117 proyectos del software de Apache alojados en Jira, de los cuales se extrajeron 81,523 incidencias correspondientes a 271,416 comentarios (y un total 20,537 usuarios involucrados). Posteriormente, utilizando el marco de trabajo *Parrots*, se identificaron seis emociones primarias en los comentarios de incidencias: Amor, Alegría, Sorpresa, Ira, Tristeza y Miedo. Sus resultados muestran que los desarrolladores sí llegan a expresar emociones durante el desarrollo, en particular Gratitud (Amor), Alegría y Tristeza. Sin embargo, concluyen que es necesario hacer más investigación en el área antes de construir una herramienta de extracción de emociones completamente automática.

En un estudio presentado por D. Gachechiladze [29] se investigaron las emociones que experimentan los desarrolladores en entornos colaborativos, centrándose principalmente en la Ira (*Anger*). Desarrollaron un modelo basado en aprendizaje supervisado para identificar la dirección de la Ira en los comentarios de incidencias para proyectos alojados en repositorios de Jira. Esto es, distinguir entre la Ira dirigida hacia uno mismo, hacia los demás o hacia los objetos. Esto con el argumento de que detectar la Ira hacia uno mismo podría ser útil para ayudar a los desarrolladores que presentan dificultades en sus tareas, la detección de la Ira hacia los demás puede ser útil para la gestión de los equipos y detectar la Ira hacia los objetos puede ser útil para recomendar y priorizar mejoras. El clasificador desarrollado fue entrenado con un estándar de oro (*Gold Standard*) etiquetado manualmente; este consiste de 723 oraciones extraídas de comentarios de informes de incidencias para proyectos de Apache. Los resultados que obtuvieron confirman que las tres direcciones de la Ira antes mencionadas están presentes en los comentarios de incidencias, además, concluyen que el clasificador desarrollado mostró un rendimiento razonable en la detección automática de la dirección de las emociones.

W. Robinson [30] realizó un estudio sobre 124 proyectos alojados en repositorios de GitHub. El objetivo fue comprender la relación entre el sentimiento y el cambio en el comportamiento (como secuencia de acciones) de los desarrolladores a lo largo del tiempo. Para el modelo de comportamiento, desde los repositorios, se extrajeron datos de ciertos eventos con el fin de identificar y analizar secuencias de acciones comunes por parte de los desarrolladores, por ejemplo, estudiar patrones que ocurren con frecuencia en su flujo de actividades, como dada la Incidencia Z, le sigue un Comentario Z y concluye con un *Commit* Z. Para aplicar el análisis de sentimientos en comentarios de incidencias utilizaron un enfoque basado en lexicones (NRC Word-Emotion v0.92). Después de aplicar un modelo de regresión, sus resultados mostraron que un cambio de comportamiento (cambio de rutina o acciones) está asociado positivamente con el cambio de sentimiento.

D. Graziotina [31] realizó un estudio para explorar qué sucede con los desarrolladores cuando están contentos y cuando no lo están. Para esto, diseñaron un cuestionario que les permitió conocer las experiencias positivas y negativas que experimentan los desarrolladores en su día a día. Por ejemplo, se les pidió a los participantes que describieran alguna situación positiva/negativa y que dieran detalles sobre lo que pudo haberlos hecho experimentar ciertos sentimientos, demás, mencionar si influyó en el desempeño de sus tareas y de qué manera. Después de aplicar el cuestionario a 317 participantes, sus resultados mostraron que el mayor

impacto de la felicidad y no felicidad se da en la productividad y la calidad del desarrollo. Además, el hecho de que no estén felices tiene efectos adversos tanto en el trabajo como en su salud mental (como agotamiento, ansiedad, depresión, etc); y por otro lado, encontraron que procurar una atmósfera positiva en el trabajo, una mayor realización personal, compromiso laboral y creatividad, es beneficioso para el bienestar de los desarrolladores. Concluyen con que los desarrolladores experimentan la felicidad más como un beneficio para ellos mismos y la no felicidad más como perjudicial para los demás.

En una investigación realizada por R. Jongeling [32] se estudió el impacto en la elección de distintas herramientas para el análisis de sentimientos dentro del dominio de la Ingeniería de Software. Las herramientas seleccionadas fueron SentiStrength [33], Stanford NLP [34] y NLTK [35], las cuales se desarrollaron para implementarse en dominios relacionadas a la industria y al entretenimiento (entrenadas con comentarios acerca de películas y productos), pero, han sido utilizadas en diversos estudios enfocadas en extraer sentimientos en el contexto del desarrollo de software. El objetivo fue estudiar el impacto en la elección de estas herramientas para realizar estudios relacionados al análisis de sentimientos dentro de la Ingeniería de Software. Investigaron si las herramientas concuerdan con el sentimiento reconocido/detectado por los evaluadores humanos y, además, si concuerdan entre ellas mismas. Sus resultados mostraron que no solo las herramientas consideradas no coinciden con el etiquetado manual (no hay coincidencia con evaluadores humanos), sino que tampoco coinciden entre sí. Llegan a la conclusión que este desacuerdo puede llevar a resultados contradictorios/erróneos y que existe la necesidad de construir herramientas para el análisis de sentimientos dirigidas especialmente al dominio de la Ingeniería de Software.

En el estudio realizado por B. Lin [36] se argumenta que varios trabajos relacionados al análisis de sentimientos en la Ingeniería de Software brindan resultados poco confiables debido a que las herramientas usadas no fueron construidas/entrenadas para aplicarse en este dominio. Es por ello que llevaron a cabo un re-entrenamiento de la herramienta Stanford CoreNLP (construida sobre una red neuronal recursiva) con un dataset de 4,000 oraciones etiquetados manualmente y extraídas de StackOvewrflow. Además, evaluaron el rendimiento de diferentes herramientas (como NLTK, SentiStrength, SentiStrength-SE, EmoTxt) utilizando datasets del dominio de la Ingeniería de Software (como App Reviews, JIRA Issues y StackOvewrflow). Sus resultados muestran que ninguna herramienta analizada está lista aún para el análisis de sentimientos dentro de la Ingeniería de Software, incluidas aquellas re-entrenadas en el contexto de aplicación. Concluyen con la advertencia a la comunidad investigadora sobre las fuertes limitaciones de las herramientas actuales y que la extracción de sentimientos en este dominio aún no está lo suficientemente madura.

Por último, N. Novielli [37] muestra un estudio de referencia (*Benchmark*) sobre el análisis de sentimientos para la investigación en Ingeniería de Software. Se presentan las herramientas más comunes y relevantes en el área, como SentiStrength, SentiStrengthSE, SentiCR y SentiSD, describiendo sus características principales. Además, se muestran los dataset de uso público más utilizados en la comunidad científica, como Jira, StackOverflow, Java Libraries y Code Reviews. Realizan un re-entrenamiento de las herramientas con los dataset antes mencionados con el fin de comparar su rendimiento y estudiar la concordancia entre herramientas y evaluadores. Sus resultados muestran que existe una buena concordancia entre herramientas (entre sí) y también entre herramientas y los evaluadores (con respecto al etiquetado manual). Además, concluyen en que un re-entrenamiento de la herramienta puede mejorar en gran medida el rendimiento de los clasificadores de sentimientos (particularmente con los dataset de Jira y StackOverflow).

3.2. Datasets para el análisis de sentimientos

En esta sección mencionaremos distintos datasets utilizados para estudiar el análisis de sentimientos en la Ingeniería de Software. Estos datasets van a permitir, principalmente, entrenar modelos de clasificación de sentimientos basados en aprendizaje de máquina (supervisados) con el fin de extraer los sentimientos que experimentan los desarrolladores de software. A continuación describimos los datasets más relevantes en el área y que son disponibles al público[1].

- **Jira**. Este dataset, considerado un estándar de oro (*Gold Standard*), fue desarrollado y publicado por M Ortu y A. Murgia [38] con el fin de contribuir a la falta de datos en el área y aportar a la investigación de las emociones en los artefactos de software. Está compuesto por casi 6,000 instancias (2 mil comentarios y 4 mil oraciones) extraídas de repositorios de código abierto (como Apache, Spring y JBoss) alojados en Jira y vinculadas a mil proyectos, 70 mil incidencias y 2 millones de comentarios aproximadamente. La tabla 3.2 muestra algunos ejemplos de comentarios junto con su etiqueta.

  Como se muestra en la tabla 3.1, el dataset se compone de tres grupos: **1)** Consta de 392 comentarios de incidencias etiquetadas con emociones de amor, alegría, sorpresa, ira, miedo y tristeza, **2)** Consta de 1,600 comentarios de incidencias etiquetada con emociones de amor, alegría y tristeza y **3)** Consta de 4,000 oraciones de incidencias etiquetadas con emociones de amor, alegría, ira y tristeza. El etiquetado fue realizado por 22 evaluadores, entre investigadores de institutos y estudiantes de maestría y PhD. Se utilizó el marco de trabajo Shaver [39], el cual define un conjunto de reglas de etiquetado y se basa en una clasificación jerárquica de las emociones (basada en un árbol estructurado) en donde cada nivel refina la granularidad del anterior.

  Para medir el nivel de acuerdo/concordancia entre evaluadores para el etiquetado del dataset, se utilizó el valor Kappa (k) de Cohen [40], el cual indica que si el valor $k < 0$ el acuerdo es casualidad, si $0.01 \leq k \leq 0.20$ es leve, si $0.21 \leq k \leq 0.40$ es regular, si $0.41 \leq k \leq 0.60$ es moderado, si $0.61 \leq k \leq 0.80$ es sustancial y si $0.81 \leq k \leq 1$ es casi perfecto. Los resultados indicaron que para las emociones de amor, alegría y tristeza se obtuvo un acuerdo regular, mientras que la mayoría de los evaluadores coincidió cuando hubo ausencia de emoción (neutral).

  El dataset de Jira ha sido implementado en diversos estudios relacionados a los estados afectivos/emocionales que experimentan los profesionales del software. Por ejemplo, para estimación del tiempos de corrección de incidencias considerando las emociones en los desarrolladores, para estudiar el impacto de las emociones en la productividad/curva de aprendizaje/atractivo de un proyecto para los nuevos desarrolladores, para investigar el impacto de las emociones en la calidad del software y para analizar la deuda técnica y social en el desarrolla de software.

- **Stack Overflow**. Este dataset fue publicado en un trabajo realizado por N. Novielli [41] con el fin de contribuir al estudio de las emociones en el desarrollo de software. Consta de 4,800 publicaciones extraídas de proyectos alojados en Stack Overflow (publicados desde julio de 2008 a septiembre de 2015), las cuales se encuentran en forma de preguntas, respuestas y comentarios. Al igual que el dataset de Jira, para la construcción de este *Gold Standard*, se siguieron un conjunto de reglas de etiquetado basadas en el marco de trabajo de Shaver.

[1] Para fines prácticos, la mayoría de los datasets mostrados llevan el mismo nombre del repositorio de donde fueron recolectados los datos.

Grupo	Granularidad	# Instancias	# Evaluadores	Emociones
1	comentario	392	16	amor, alegría, sorpresa, ira, miedo, tristeza
2	comentario	1600	3	amor, alegría, tristeza
3	oración	4000	3	amor, alegría, ira, tristeza

Tabla 3.1: Distribución/Grupos del etiquetado de emociones para el dataset de Jira.

Texto (Comentario/Oración)	Etiqueta (Emoción)
This has been in RESOLVED state for over one year so now closing; if it turns out to not be fixed please re-open.	Ira (Anger)
The changes looks good for me. Reviewed with no commentaries.	Alegría (Joy)
After the changes the error no longer appears in nightly build results. Closing as Fixed.	Tristeza (Sadness)

Tabla 3.2: Ejemplos de etiquetado de comentarios/oraciones de incidencias para el dataset de Jira.

El dataset fue etiquetado por 12 estudiantes (de Ciencias de la Computación) en donde se les pidió que indicaran la presencia/ausencia de emociones en cada instancia (se consideraron seis emociones básicas: amor, alegría, ira, tristeza, miedo y sorpresa). Cada publicación fue etiquetada por tres evaluadores y en el caso de que hubiera desacuerdo, estos fueron resueltos aplicando un criterio de mayoría de votos. En la 3.3 se muestra la distribución del etiquetado para cada una de las emociones mencionas, mientras que en la figura 3.4 se muestran ejemplos de publicaciones junto con su etiqueta.

Una vez hecho hecho el etiquetado, se observó el nivel de acuerdo/concordancia entre evaluadores (porcentaje de casos con la misma etiqueta), obteniendo desde un 86 % para Alegría hasta un 96 % para Miedo, lo que demuestra una alta confiabilidad. Además, según la medida de concordancia Kappa (k), el dataset muestra valores más bajos, desde 30 % para Sorpresa (justo/moderado) hasta 66 % para Amor (substancial/importante), esto debido a la distribución (no balanceada) de las emociones en el dataset (por ejemplo se observan valores más bajos de concordancia para las emociones menos frecuentes como sorpresa, tristeza y miedo).

Por otro lado, el dataset de StackOverflow es, actualmente, uno de lo más usados y confiables para realizar estudios enfocados al análisis de sentimientos en el dominio de la Ingeniería de Software. Por ejemplo, se ha usado para validar y entrenar herramientas basadas en aprendizaje de máquinas (como SentiCR y Senti4SD) o para el desarrollo de lexicones (como SentiStrengthSE), para investigar el impacto de los sentimientos en el desempeño de los desarrolladores, para tener puntos de comparación con otros datasets de uso público, entre otros.

Posts	Emociones						Total
	Amor	*Alegría*	*Sorpresa*	*Ira*	*Tristeza*	*Miedo*	
#	1220	491	45	882	230	106	4800
%	25 %	10 %	1 %	18 %	5 %	2 %	

Tabla 3.3: Distribución del etiquetado de emociones para el dataset de StackOverflow.

Texto (Publicación)	**Etiqueta (Emoción)**
Absolutely terrible API design	Ira (Anger)
I'm happy with the approach, the code looks good	Alegría (Joy)
Thanks for your input! You're, like, awesome!	Amor (Love)

Tabla 3.4: Ejemplos de etiquetado de publicaciones para el dataset de StackOverflow.

- **Code Review**. Este dataset fue creado para evaluar y entrenar la herramienta SentiCR [42]. Se compone de 2,000 comentarios de reseñas extraídos de 20 proyectos de código abierto alojados en Gerrit. Para su etiquetado se desarrolló una aplicación web, en donde tres evaluadores asignaban (de forma manual) a cada comentario el sentimiento asociado (positivo, negativo o neutral) de acuerdo a su criterio y experiencia.

  El nivel de concordancia observado entre los tres evaluadores fue correcto en 1,239 (62.5 %). Mientras que el valor Kappa obtenido fue de $k = 0.408$ lo cual indica una concordancia moderada entre evaluadores. Por otro lado, la distribución final del dataset fue: 7.7 % comentarios positivos, 19.9 % comentarios negativos y 72.4 % comentarios neutrales. Por último, se pasó de un dataset de tres clases a uno de dos clases, en donde se realizo un ajuste en las etiquetas, transformando los comentarios positivos y neutros como "no-negativos"; esto con el fin de mejorar el rendimiento del clasificador de sentimientos (SentiCR).

- **Java Libraries**. Fue desarrollado y publicado por un estudio de B. Lin [36]. Este dataset se compone de 1,500 oraciones extraídas de proyectos alojados en StackOverflow (julio 2017), seleccionando solo aquellas que contuvieran las palabras: *Java, Library, API.* El proceso de etiquetado se llevo a cabo desde una aplicación web por cinco evaluadores; se les mostraban las oraciones y debían asignar un sentimiento en forma de puntaje/valor, desde -2 a +2, en donde -2 implica un fuerte negativo, -1 débil negativo, 0 neutral, +1 débil positivo y +2 fuerte positivo. Posteriormente se hizo un mapeo, en cual las oraciones etiquetadas con -2 y -1 se consideraron negativas, las etiquetadas con 0 como neutral y las etiquetadas con +1 y +2 como positivas. Por último, después de resolver conflictos en el etiquetado, se obtuvo una distribución de 178 comentarios positivos (9 %), 131 comentarios negativos (12 %) y 1,191 comentarios neutrales (79 %).

3.3. Herramientas para el análisis de sentimientos

Existen una gran variedad de herramientas para el Análisis de Sentimientos aplicadas en ámbitos como las redes sociales o en reseñas de servicios/productos. Es posible hacer uso de ellas con el fin de aplicar un análisis de sentimientos en los repositorios de software, pero al no estar entrenadas con el lenguaje técnico y particular que se desenvuelve dentro de la Ingeniería de Software, es muy probable que se obtengan resultados poco confiables o erroneos. Para dar solución a esta limitación, la comunidad científica recientemente ha desarrollado herramientas de código abierto específicas para el dominio de la Ingeniería de Software. Entre las más utilizadas y más aceptación han tenido por la comunidad se encuentran: SentiCR, Senti4SD y SentiStrengh-SE. A continuación describiremos las características principales de cada una de ellas.

- **SentiCR**. Construida por T. Ahmed [42], está basada en aprendizaje supervisado y específicamente entrenada y evaluada en comentarios de revisión de código (dataset Code Reviews). La herramienta está desarrollada en lenguaje Pyhton, utilizando librerías como NLTK para el procesamiento de lenguaje y Scikit-Learn para implementar algoritmos de clasificación. Para su desarrollo, se implementaron tres etapas principales: una de preprocesamiento de datos, una de extracción de características y una de evaluación. Estas se resumen en los siguientes pasos:

  1. Se removieron URL's utilizando expresiones regulares, debido a que no aportan información relevante en el análisis de sentimientos.
  2. Se realizó un manejo de los emoticones contenidos en los textos debido a que aportan gran información en la detección de sentimientos. Para su interpretación, cuando se encuentra un emoticón en el texto, como ":)", se mapea/traduce a la palabra "PositiveEmoticon", y uno ":(", a la palabra "NegativeEmoticon".
  3. Se implementó un manejo de negaciones (haciendo uso del método POS) con el fin de evitar las clasificaciones erroneas en los comentarios que incluyen palabras negativas (por ejemplo *not, never, nobody*, etc).
  4. Se aplicó el método *Stemming* para convertir cada palabra tokenizada en su palabra raíz.
  5. Se removieron palabras (StopWord) repetitivas (como artículos y preposiciones) que no aportan mucho significado a la hora de identificar sentimientos (por ejemplo *what, on, in, and*, etc)
  6. Para la extracción de características o vectorización se implementa el método TF-IDF.
  7. Para abordar el problema del desequilibrio de los datasets de entrenamiento, se aplica la técnica de sobre muestreo SMOTE.
  8. Por último, una vez entrenado el modelo, se evaluó utilizando ocho algoritmos de clasificación (como SVC, GBT, RF, NB, etc) mediante validaciones cruzadas.

  El mejor rendimiento que presentó el modelo, fue al entrenarlo con el algoritmo *Gradient Boosting* (GBT), en donde se obtuvieron las siguientes medidas de desempeño: 67.84 % de *Precision*, 58.35 % de *Recall*, 62 % de *F1Score* y 83.3 % de *Accuracy*. Por otro lado, se hizo la comparación entre SentiCR y otras herramientas basadas en lexicones (como NLTK Vader, SentiStrength y TextBlog), siendo SentiCR (en general) la que mejor desempeño tuvo de acuerdo a las métricas antes mencionadas.

- **Senti4SD**. Presentada en un estudio por F. Calefato [43], fue construida específicamente para la aplicación del análisis de sentimientos en los canales de comunicación de los desarrolladores de software, y se encuentra disponible públicamente para fines de investigación. Esta herramienta fue entrenada y validada con el dataset de StackOverflow antes mencionado (*Gold Standard* de 4,000 publicaciones). Además, fue desarrollada en lenguaje R y se utilizaron librerías como *caret* para implementar distintos algoritmos de clasificación y métodos de evaluación de rendimiento.

  Senti4SD, al igual que SentiCR, está desarrollada bajo un enfoque de aprendizaje supervisado. Entre sus principales características se encuentran:

- Implementa un preprocesamiento de datos, en donde aplica: Remoción de URL's, de fragmentos de código y de etiquetas HTML (no implementa manejo de *StopWords*, ni técnias como *Stemming* ni *Lmmeatization*).
- Aplica un método de tokenización utilizando la herramienta Stanford NLP.
- Aprovecha un conjunto de características basadas en n-gramas.
- Utiliza lexicones de sentimientos y caracteristicas semánticas basadas en *Word Embeddings* (implementando el algoritmo *Word2Vec*)
- Fue entrenada utilizando el algoritmo de clasificación de máquinas de soporte de vectores (SVM) en su versión lineal. Para la selección del parámetro C del modelo se utilizaron validaciones cruzadas (en 10 grupos).
- Se entrenó y evaluó utilizando una estrategia 70-30, en donde se utiliza 70 % del dataset como datos entrenamiento y un 30 % como datos de prueba/evaluación.

El rendimiento (general) reportado del modelo (considerando el promedio del rendimiento obtenido para las clases Positiva, Negativa y Neutral) fue: 87 % de *Precision*, 87 % de *Recall* y 87 % de *F1Score*.

Por otro lado, se comparó el rendimiento de Senti4SD con las herramientas SentiStrength y SentiStrength-SE (utilizando el mismo dataset de StackOverflow en todos los casos), en donde Senti4SD presentó un mejor rendimiento, además, se observó que reduce la clasificación errónea de publicaciones neutrales/positivas como publicaciones negativas.

- **SentiStrengh-SE**. Desarrollada por M. Rakibul [33], es una herramienta especializada para el análisis de sentimientos en la Ingeniería de Software. Es un clasificador basado en lexicones (hacen uso de diccionarios) y está construida sobre la API de SentiStrength [44], donde esta última ha sido implementada en diversos estudios relacionados a la predicción de sentimientos en el desarrollo de software, a pesar de que fue construida para redes sociales. La figura 3.1 muestra la interfaz gráfica para utilizar la herramientas SentiStrengh-SE, la cual está desarrollada en lenguaje Java. Esta permite como entrada un archivo de texto, con múltiples oraciones/documentos, y como salida entrega el mismo archivo pero con el sentimiento asociado para cada texto.

  SentiStrengh-SE asigna a cada palabra negativa una puntuación de sentimiento que va de -2 a -5, que representa la polaridad del término fuera de su contexto. Del mismo modo, las palabras positivas se asocian con una puntuación entre $+2$ y $+5$, mientras que las palabras neutrales reciben puntuaciones iguales a ± 1. Las puntuaciones de sentimiento positivo varían de $+1$ (ausencia de sentimiento positivo) a $+5$ (extremadamente positivo) mientras que las puntuaciones de sentimiento negativo varían de -1 (ausencia de sentimiento negativo) a -5 (extremadamente negativo). En función de su suma algebraica, se puede informar el puntaje de sentimiento general, es decir, positivo (puntaje $= 1$), negativo (puntaje $= -1$) o neutral (puntaje $= 0$). Los emoticones positivos y negativos también se incluyen en el diccionario. Como ejemplo, consideremos la oración "*It's a good feature*", la cual contiene la palabra "*good*", que de acuerdo al diccionario (lexicón), tiene asociado un valor de $+2$, por lo que se clasifica como positiva (el resto de las palabras se consideran neutrales).

  Por otro lado, al evaluar el rendimiento de SentiStrengh-SE utilizando el dataset de Jira mencionado anteriormente (compuesto de 6,000 comentario de incidencias) se obtuvieron las siguientes métricas generales: 73.85 % de *Precision*, 85 % de *Recall* y 77.48 % de

F1Score; mientras que para SentiStrengh se obtuvo: 61.69 % de *Precision*, 78.54 % de *Recall* y 62.02 % de *F1Score*. Lo que indica una mejora notable en el rendimiento de SentiStrengh-SE sobre SentiStrengh.

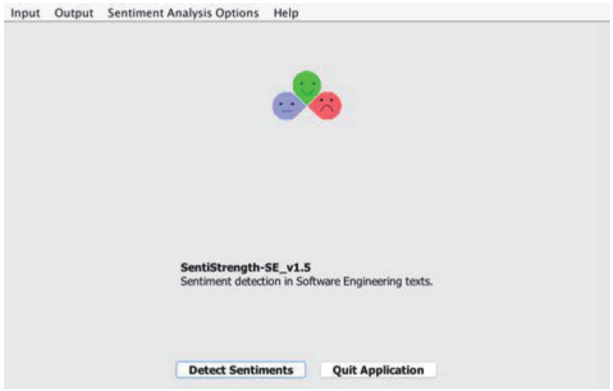

Figura 3.1: Interfaz gráfica de la herramienta SentiStrengthSE.

Capítulo 4

Metodología de investigación

> "El valor de una idea radica en el uso de la misma"
>
> *Thomas A. Edison*

En este capítulo se describe el marco metodológico de este trabajo de tesis, el cual define las etapas/procesos necesarios para realizar un análisis de sentimientos en repositorios de software. En la **sección 4.1** se describe de forma general la metodología de investigación propuesta y se definen los pasos que la componen. Posteriormente se dará una explicación detallada para cada una de las etapas, comenzando con la **sección 4.2** (Entrenamiento), relacionado al diseño, construcción y entrenamiento del modelo clasificación de sentimientos (al que nombramos ASIS, por Análisis de Sentimientos en Ingeniería de Software); seguido de la **sección 4.3** (Extracción de Datos), en donde se describe el proceso de extracción y almacenamiento de los datos alojados en repositorios de software (Jira); después, en la **sección 4.4** (Evaluación y Predicción), se reportan las métricas de rendimiento para el modelo ASIS y se muestra una comparativa con las herramientas actuales para la tarea de clasificación de sentimientos (como SentiCR, Senti4SD y SentiStrenghtSE descritas en el capítulo 3) entrenadas con el dataset de Jira, además, se presenta el proceso para la predicción de sentimientos sobre los nuevos datos extraídos; por último, en la **sección 4.5** (Exploración), se describe el proceso de análisis de los datos/sentimientos, lo cual nos permitirá responder a las preguntas de investigación planteadas.

4.1. Metodología propuesta

Como ya se mencionó anteriormente, el objetivo principal de este trabajo de tesis es aplicar un análisis de sentimientos en repositorios de software y estudiar el impacto/efecto de las emociones en los procesos de desarrollo y profesionales del software. Para esto, se propone desarrollar y entrenar un modelo de clasificación de sentimientos basado en aprendizaje de máquina (de tipo supervisado) que permita identificar los sentimientos presentes en los comentarios escritos de incidencias/tareas dentro de un proyecto de software.

Para el entrenamiento de este modelo de clasificación se hará uso de los datasets descritos en el capítulo 3, principalmente el dataset de Jira, debido a que, además de ser considerado un *gold standard*, todo nuestro estudio se centra en datos alojados en repositorios Jira, lo que va a permitir tener un modelo más representativo y ajustado al contexto de aplicación.

Pensado en lo anterior, la metodología a seguir en este trabajo consta de cuatro etapas, basadas principalmente en el proceso de Minería de Textos para la tarea de clasificación, tal como se describe en el capítulo 2 (ver figura 2.5). Por otro lado, se hará uso del lenguaje de programación Python a lo largo de todo el proceso, debido a que, además de ser un lenguaje muy flexible y que cuenta con gran soporte por la comunidad de desarrolladores, presenta varias bibliotecas de código abierto que nos van a permitir resolver tareas e implementar técnicas de una forma más sencilla y rápida; por ejemplo, se hará uso recurrente de la biblioteca Scikit-learn [45] para implementar algoritmos de aprendizaje de máquina, NLTK [35] para aplicar métodos de Procesamiento de Lenguaje Natural, entre otras [1].

En el resto de esta sección definiremos cada una de las etapas del proceso, dando una descripción general de sus componentes y funcionalidades, en las secciones posteriores nos centraremos en cada una de ellas, dando una descripción más particular y detallada.

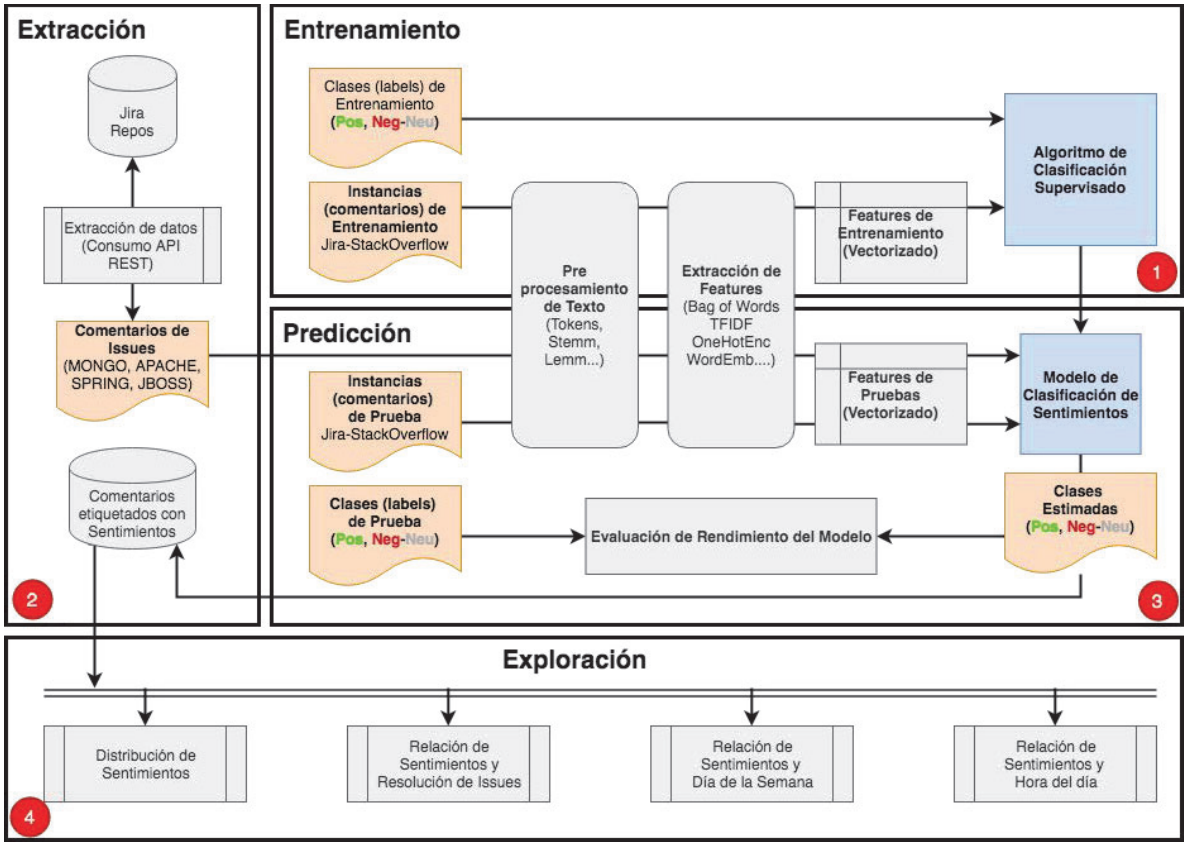

Figura 4.1: Diagrama de la metodología propuesta para el Análisis de Sentimientos en repositorios de software Jira

1. **Etapa de desarrollo y entrenamiento del modelo**. En esta primera etapa se realiza la parte del desarrollo y entrenamiento del modelo de clasificación. En la figura 4.1 (1) se muestra el diagrama de esta primera etapa, el cual se compone de tres procesos principales que se llevan a cabo de forma secuencial (donde la salida del primer proceso es la entrada del segundo y así sucesivamente). Por un lado, **1)** Preprocesamiento de Textos, en donde se aplican técnicas para la normalización de textos como limpieza y manipulación de cadenas de caracteres y métodos como *Tokenization/Lemmatization*; por otro

[1]Código del modelo completo: https://github.com/andricValdez/ASIS/

lado, **2)** Extracción de Características (*features*), en donde se obtiene una representación vectorial (numérica) de las instancias de entrenamiento (textos) al aplicar técnicas como OHE o TFIDF. Por último, **3)** Algoritmo Supervisado, en donde se selecciona el/los algoritmos de clasificación (candidatos) a utilizar en el entrenamiento del modelo. Una vez implementados estos procesos, se pasa a la parte de entrenamiento, en la que, al tener un dataset/corpus etiquetado (por ejemplo el dataset de Jira que se compone de comentarios de incidencias etiquetados con tres clases de sentimientos: positivos, negativos y neutrales) se aplican los tres procesos antes mencionados para obtener como resultado un modelo de clasificación de sentimientos.

2. **Etapa de extracción de datos**. Es necesario contar con nuevos datos (comentarios textuales) para realizar la predicción de sentimientos sobre estos utilizando el modelo de clasificación antes desarrollado/entrenado y posteriormente pasar a una etapa de exploración y análisis. En la figura 4.1 (2) se muestra el diagrama correspondiente, en donde se realiza la extracción de datos desde los repositorios de software (Jira) mediante el consumo de una API REST expuesta para desarrolladores por la empresa de software Atlassian [46]. Los datos a extraer corresponden a proyectos de código abierto de Mongo, Apache, JBoss y Spring, debido a que cuentan con gran actividad y colaboración, además, se encuentran entre los proyectos líderes en el mercado. Más adelante describiremos a detalle este proceso, mencionando los distintos tipos de datos, sus formatos y sus funciones.

3. **Etapa de evaluación y predicción de sentimientos**. En esta tercera etapa, teniendo el modelo de clasificación entrenado y los datos extraídos, se prosigue a evaluar el rendimiento del modelo y a la predicción de sentimientos. En la figura 4.1 (3) se muestra el diagrama correspondiente; para su evaluación se toman las instancias de prueba del dataset (utilizando una partición 70 % *Train* - 30 % *Test*) junto con su actual etiqueta de sentimiento y se someten a los procesos descritos en la etapa 1, posteriormente el modelo realiza la predicción del sentimiento para cada instancia y por último se compara el sentimiento actual con el predicho. Esta evaluación se interpreta utilizando métricas como *Matriz de Confusión*, *F1Score*, *Curvas ROC*, etc (descritas en el capítulo 2), las cuales nos van a servir para mejorar el modelo y compararlo con otras herramientas. Además, es importante mencionar que el entrenamiento y evaluación de un modelo es un proceso iterativo en el que constantemente se deben realizar diversas configuraciones y modificaciones con el fin de mejorar su rendimiento, por ejemplo, probando distintos algoritmos de clasificación, modificando parámetros del algoritmo, cambios en el preprocesamiento o en la extracción de *features*, etc. Después de que el rendimiento del modelo es el deseado, le sigue el proceso de predicción de sentimientos sobre los nuevos datos extraídos de los repositorios, cuya salida es almacenada para su posterior análisis.

4. **Etapa de exploración y análisis**. En esta última etapa se realiza la exploración y análisis de los sentimientos encontrados en los comentarios de incidencias (extraídas de los repositorios) con el fin de responder a la preguntas de investigación planteadas. En la figura 4.1 (4) se muestra el diagrama correspondiente; se llevan a cabo distintos análisis relacionando estos sentimientos con otras variables/datos, por ejemplo, con la hora y día de la semana en que se escribieron los comentarios, con la resolución de la incidencia (si fue resuelta o no), con el tiempo que tardó en resolverse una incidencia, entre otros. Para esto se utilizaron diferentes métricas estadísticas como el coeficiente de correlación de *Pearson*, diagramas de caja, entre otras.

4.2. Desarrollo y entrenamiento del modelo de clasificación

En esta sección describiremos el proceso de desarrollo y entrenamiento del modelo. Este corresponde a un modelo de clasificación basado en aprendizaje de máquina bajo un enfoque supervisado, en el cual se provee como entrada los datos junto con su resultados deseados (instancias de entrenamiento). En este caso de estudio, el modelo desarrollado es un clasificador de sentimientos, donde los datos de entrenada son comentarios de textos junto con su etiqueta de sentimiento (positivo, negativo o neutral).

Diferentes ideas, técnicas y métodos de las que aplica y se compone este modelo tienen sustento en las herramientas existentes (presentadas en la 3.3 del capítulo anterior), libros de consulta [22] [17] [18] y blogs/foros web como *StackOverflow, Medium, MachineLearningMastery*, etc. Por otro lado, el objetivo de presentar este clasificador es obtener un buen desempeño en la predicción de sentimientos en comentarios de incidencias, particularmente para repositorios de Jira y que, además, sirva para futuros estudios dentro del área.

A continuación se muestra una versión resumida del código del modelo dando una descripción de sus características principales.

```
# import libraries (sklearn, nltk, pandas, Pipeline, matplotlib, etc)
class TextNormalizer(BaseEstimator, TransformerMixin):
    def preprocessing(self, issueCom, norm='stem'):
        response = []
        comment = self.replaceText(issueCom)
        comment = self.handleEmoticons(comment, self.emodict)
        comment = self.handleContractions(comment)
        comment = self.handleStopw(comment)
        comment = self.removeSpecialCharacters(comment)
        comment = [word for word in nltk.word_tokenize(comment)]
        comment = nltk.pos_tag(comment)

        for (word, tag) in comment if comment is not None else '':
            if norm == 'lem': ...
            if norm == 'stem': ...
        return response

    def transform(self, issuesData):
        response = []
        for issueCom in issuesData:
            prepCom = self.preprocessing(issueCom, 'stem')
            response.append(prepCom)
        return response


```

```
26      def createPipeline(vectorize, oversampling, classifier):
27          steps = [('normalize', TextNormalizer()),
28                   ('vectorize', vectorize),
29                   ('oversampling', oversampling),
30                   ('classifier', classifier)]
31          return Pipeline(steps)
32
33      def trainModel(vectorize, oversampling, classifier, dset):
34          dataset = readDataset(dset)
35          X, y = dataset['Sentence'], dataset['Label']
36          pipe = createPipeline(vectorize, oversampling, classifier)
37          return pipe.fit(X,y)
38
39      trained_model = trainModel(TfidfVectorizer(params), SVMSMOTE(params),
        ↪  GradientBoostingClassifier(params), 'datasetJIRA.csv')
```

El código completo se encuentra en el archivo *ASIS.py* del proyecto[2]. En este, se comienza declarando todos los módulos utilizados en el desarrollo del modelo, como sklearn, numpy, etc (línea 1). Posteriormente, se llama a la función *trainModel* (línea 39 y definida en las líneas 33 a 37) la cual se encarga de llevar a cabo todo el proceso de desarrollo y entrenamiento; recibe como parámetros el tipo de vectorización a implementar (*vectorize*), el sobremuestreo (*oversampling*), el algoritmo de clasificación (*classifier*) y el dataset de entrenamiento (*dset*). Se prosigue con la lectura del dataset (línea 34) para extraer sus comentarios con sus correspondientes etiquetas de sentimiento (línea 35), después se llama a la función *createPipeline* (línea 36 y definida en las líneas 26 a 31) la cual recibe los tres primeros parámetros antes mencionados y se encarga de definir la serie de pasos de entrenamiento haciendo uso del módulo *Pipeline* de *imblearn*[3]. Este *pipeline* consta de cuatro pasos definidos a continuación:

1. ***normalize*** (línea 27), llama a la clase *TextNormalizer* (líneas 2 a 25) en donde se ejecuta el método *transform* (líneas 18 a la 23), que corresponde al punto de entrada de la clase (en donde se lleva a cabo la transformación de los datos) y recibe como parámetro los comentarios de incidencias. Cada comentario pasa como argumento al método *preprocessing* (línea 21), en el cual se ejecutan una serie de métodos con el fin de aplicar una limpieza sobre el comentario, retornando este mismo pero normalizado (líneas 3 a la 16). En el método *replaceText* (línea 5), se reemplazan partes del texto que en su forma natural no son muy relevantes, por ejemplo, los enlaces (*http*) se reemplazan por la palabra URL, los nombres de usuario por USER, las versiones de software por VERSION, las partes de código por CODE, entre otros. Es común que en el lenguaje informal se utilicen emoticones para expresar algún sentimiento, los cuales aportan información valiosa en nuestro contexto de aplicación, para manejarlos se implementa

[2]https://github.com/andricValdez/ASIS/blob/master/ASIS.ipynb

[3]Construir el modelo bajo este enfoque (con *Pipelines*) permite definir una interfaz que encapsula los pasos de entrenamiento (normalización, vectorización, etc), los cuales se realizan en secuencia y en cada uno se lleva a cabo una transformación de los datos, donde la salida de un paso es la entrada del siguiente; esto tiene la ventaja de definir un flujo de trabajo más flexible, configurable y legible.

la función *handleEmoticons* (línea 6), la cual realiza la búsqueda de algún emoticón en el texto (que exista en el diccionario *emodict*), por ejemplo, ":)" y ": (" se reemplazan por las palabras *"PositiveSentiment" y "NegativeSentiment"*, respectivamente. Se implementa una expansión de contracciones (método *handleContractions*, línea 7), en donde las abreviaturas de palabras como *don't, I'd, it's* se conviertan en *do not, I would, It is*, respectivamente. Se remueven palabras vacías (o *Stop Words*) las cuales no aportan significado al texto, ejecutando el método *handleStopw* (línea 8) se eliminan artículos (*a, an, the*), preposiciones (*on, in, to*), entre otros. Se aplica el método *removeSpecial-Characters* (línea 9) para remover caracteres especiales como *@, *, !, ?*, etc. Por último, se lleva a cabo un proceso de *Lemmatization* o *Stemming* (de acuerdo al valor de la variable *norm*, pasada como argumento). Una vez que se lleva a cabo la transformación de los datos (normalización), la salida de este proceso pasa a ser entrada del siguiente: *vectorize*.

2. ***vectorize*** (línea 28), ejecuta el proceso de extracción de características, en donde se aplican método como One-Hot-Encodingo o TF-IDF, retornando una forma vectorizada (numérica) para cada comentario de incidencia (texto), con el fin de que pueda ser procesado por los algoritmos de clasificación. Cada método de vectorización cuenta con ciertos parámetros de configuración que permiten extraer características del texto de diferentes formas, por ejemplo, hablando particularmente de TF-IDF (el cual se implementa en el modelo final), contiene el parámetro *ngram_ range = (1,n)* que permite la extracción de subsecuencias de palabras en el rango de 1 a n (unigramas, bigramas,..., ngramas), los parámetros *max_ df* y *min_ df*, que permiten remover términos que aparecen muy frecuente y muy poco frecuente en el texto, respectivamente, y entre otros parámetros (esta selección de parámetros y sus valores se realiza mediante un proceso llamado optimización/ajuste de hiperparámetros, descrito más adelante en la sección 4.4). Una vez que se lleva a cabo la transformación de los datos (vectorización), la salida de este proceso pasa a ser entrada del siguiente: *oversampling*.

3. ***oversampling*** (línea 29), implementa un ajuste de balance sobre el dataset, debido a que, al ser muy común que las clases no estén equitativamente distribuidas, esto conlleve a problemas de sesgo o tendencias erróneas. Este es el caso del dataset de Jira utilizado en el entrenamiento, el cual cuenta con 5,888 instancias donde el 20 % pertenecen a la clase positivas, 14 % a la negativa y 66 % a la neutral. Para contrarrestar esto, en esta etapa se implementan métodos de *Oversampling* (como SMOTE y SVSMOTE de la librería *imblearn*) que permiten obtener un dataset mejor equilibrado. Esto se logra mediante la creación de nuevas instancias (artificiales) basadas en las existentes y solo para las clases que cuentan con menos instancias (en nuestro caso para la clase positiva y negativa). Una vez que se lleva a cabo la transformación de los datos (sobremuestreo), la salida de este proceso pasa a ser entrada del siguiente: *classifier*.

4. ***classifier*** (línea 30), en este último paso se termina de crear y establecer el *pipeline* al seleccionar el algoritmo/estimador a utilizar en el modelo; algunos algoritmos de clasificación más comunes son *LogisticRegression, LinearSVC o RandomForestClassifier*, cuya selección final depende de la evaluación del rendimiento del modelo, la cual se muestra en la sección 4.4.

Por último, el *pipeline* resultante se retorna (línea 31) a la función incial *trainModel* (línea 36), donde finalmente se obtiene el modelo de clasificación entrenado (línea 37), listo para ser evaluado y posteriormente llevar a cabo la predicción de sentimientos sobre nuevos datos.

4.3. Extracción de datos de repositorios

La extracción de datos de repositorios Jira se lleva a acabo consumiendo su API (para facilitar este proceso se hará uso de la biblioteca jira 2.0.0 [47]). Cada servidor alojado en Jira almacena varios proyectos de acceso público y estos a su vez contienen todas las incidencias (*issues*) que se van registrando a lo largo del ciclo de vida de desarrollo. En la tabla 4.1 se muestran distintos servidores junto con su dirección URL de acceso (*endpoint*) y sus proyectos almacenados.

Servidor	**Dirección URL**	**Proyectos**	**# Proyectos**
SPRING	*https://jira.spring.io*	GRADLE, STS, BATCH, ...	89
MONGO	*https://jira.mongodb.org*	PYTHON, SERVER, TOOLS, ...	33
APACHE	*https://issues.apache.org/jira*	CASSANDRA, MAPREDUCE, ...	658
JBOSS	*https://issues.jboss.org*	AEROGEAR, FUSETOOLS, ...	429

Tabla 4.1: Servidores alojados en Jira con su respectiva dirección URL de acceso y proyectos.

A continuación se muestra un resumen del código implementado en esta etapa, el cual implementa la extracción y almacenamiento de los datos para cada servidor antes mencionado (correspondiente al script *dataExtraction.py* del proyecto[4]).

```
1        from jira import JIRA
2
3        jira = JIRA({"server": APACHE})
4        projects = jira.projects()
5
6        def getIssues(project)
7        def getIssuesData(issues)
8        def saveIssuesData(data, project, server)
```

Se comienza declarando el módulo de *jira* para acceder a la API (línea 1). Posteriormente se crea/instancia el objeto (línea 3) indicando el servidor al cual se hará la conexión (por ejemplo al servidor de APACHE) y se obtienen todos sus proyectos correspondientes[5] (línea 4). Después, se ejecutan tres funciones principales: *getIssues* (línea 6), para cada proyecto se extraen todas las claves (*keys*) que corresponden a todas sus incidencias reportadas. Posteriormente, estas incidencias se pasan como parámetro a la función *getIssuesData* (línea 7) en donde se extraen todos sus datos correspondientes (comentarios, autor, etc). Por último, se ejecuta *saveIssuesData* (línea 8), la cual se encarga de almacenar los datos en un archivo de formato *.csv*, recibiendo como parámetro los datos y su respectivo proyecto y servidor al que pertenecen. Los datos se almacenan dentro del directorio */data* y llevan el siguiente patrón de nombramiento (ver figura 4.2): *'ISSUES-COMMENTS-' + server + project + '.csv'*.

[4] https://github.com/andricValdez/ASIS/blob/master/dataExtraction.ipynb

[5] Todos los datos se extraen en formato JSON y son transformados y manipulados utilizando la biblioteca de *Pandas*.

Figura 4.2: Ejemplos de formato de archivos almacenados para cada servidor y proyecto.

La figura 4.3 es un *dataframe* donde se muestra el tipo y la forma de los datos extraídos y almacenados en los archivos antes mencionados; los presentados corresponden al proyecto de *Cassandra* (el cual cuenta con más de 100,000 comentarios repartido en 16,000 incidencias). Cada columna representa el tipo de dato y cada fila representa un comentario con distintos valores asociados; a continuación describimos cada uno de ellos:

- ***comment***. Es un comentario escrito de extensión indefinida.
- ***author***. Indica el nombre del desarrollador que escribió el comentario.
- ***issueComDate***. Indica la fecha en que se publicó el comentario.
- ***key***. Indica la incidencia a la que pertenece el comentario.
- ***issueCreated***. Indica la fecha de creación de la incidencia.
- ***resolutionDate***. Indica la fecha de resolución de la incidencia.
- ***issueType***. Indica el tipo de la incidencia (*Bug, New Feature, Improvement, etc*).
- ***issueResolution***. Es el estado de resolución de la incidencia (*Fixed, Don't Fixed, etc*).
- ***summary***. Es un resumen que describe el objetivo/intención de la incidencia.
- ***timeZone***. Indica la zona horaria desde donde se escribió el comentario.

author	comment	issueComDate	issueCreated	issueResolution	issuetype	key	resolution.date	summary	timeZone
jbellis	This patch makes changes that make remove supp...	2009-03-07T03:38:38.982+0000	2009-03-07T03:37:05.000+0000	Fixed	New Feature	CASSANDRA-1	2009-03-27T02:22:45.000+0000	Remove support	America/Chicag
jbellis	This patch provides the actual remove support ...	2009-03-07T03:39:49.299+0000	2009-03-07T03:37:05.000+0000	Fixed	New Feature	CASSANDRA-1	2009-03-27T02:22:45.000+0000	Remove support	America/Chicag
arsson	This is an initial version that should require...	2009-03-10T18:35:55.947+0000	2009-03-10T18:30:26.000+0000	Fixed	Task	CASSANDRA-2	2009-03-26T21:17:40.000+0000	Set up a cassandra website	America/Los_Angele
jbellis	This fixes the CF deserialization in SequenceF...	2009-03-10T18:44:58.262+0000	2009-03-07T03:37:05.000+0000	Fixed	New Feature	CASSANDRA-1	2009-03-27T02:22:45.000+0000	Remove support	America/Chicag

Figura 4.3: Tipos de datos extraídos de Jira (proyecto: Cassandra).

4.4. Evaluación y predicción de sentimientos

Como ya se mencionó en esta etapa se lleva a cabo la evaluación de modelos y la predicción sentimientos sobre los nuevos comentarios extraídos en la etapa anterior. A continuación se muestra un resumen del código implementado para la evaluación:

```
1   def evaluateModels(folds=10, shuffle=False, dset, models):
2       kf = KFold(n_splits=folds, shuffle=shuffle)
3       dataset = readDataset(dset)
4       for model in models:
5           for train, test in kf.split(dataset):
6               # Entrenamiento del modelo en cada fold
7               # Calcular: 'precision','recall','f1','micro_avg',...
8
9   models = []
10  vectorize = TfidfVectorizer(params)
11  oversampling = SVMSMOTE(params)
12  classifiers = [LogisticRegression(params), LinearSVC(params),...]
13  for classifier in classifiers:
14      models.append(createPipeline(vectorize, oversampling, classifier))
15
16  evaluateModels(10, True, 'datasetJIRA.csv', models)
17  sns.heatmap(confusion_matrix(y_test, pipe.predict(X_test)),...)
18  plot_multiclass_roc(pipe, X_test, y_test,...)
```

Se comienza con la creación de los distintos modelos que serán sometidos a la evaluación de rendimiento (líneas 9 a la 14), que como mencionamos anteriormente, por cada uno se crea un *pipiline* ejecutando la función *createPipeline* (línea 14), pasando como argumento un método de vectorización (línea 10), un método de sobremuestreo (línea 11) y distintos algoritmos de clasificación (línea 14). Después de esto, este arreglo de modelos se pasan como argumento a la función *evaluateModels* (línea 16 y definida en las líneas 1 a 7) en donde se aplica la técnica de validaciones cruzadas de k iteraciones (10 por defecto), en la que se particiona el dataset en k subconjuntos. En cada iteración se toman como datos de entrenamiento $k-1$ subconjuntos y el subconjunto restante como datos de prueba sobre los cuales se realiza la predicción (línea 5 y 6), siendo con estos resultados, que se calculan diferentes métricas de rendimiento, como: *Precision (P), Recall (R), F1 Score (F1), Micro Average (micro), Macro Average (macro) y Accuracy* (línea 7). Este proceso se repite hasta completar las k iteraciones (esto es, en cada iteración se entrena, prueba y calculan las métricas sobre los subconjuntos de datos), donde finalmente se calcula la media aritmética de los resultados de cada iteración para cada métrica de rendimiento, obteniendo así un resultado final para cada una.

Los resultados de esta evaluación se muestran en la tabla 4.2, en donde se reportan las métricas antes mencionadas para siete modelos diferentes, entrenados con distintos algoritmos de clasificación (se remarcan con negro la métricas con el valor más alto entre todos los

modelos). Se observa que los modelos entrenados con los algoritmos de *RandomForest* y *GradientBoosting* obtuvieron los mejores rendimientos, siendo este último, en general, el mejor de los siete para las tres clases de sentimientos, por lo que fue el seleccionado como modelo final y utilizado en etapas posteriores. Por otro lado, los resultados de estos últimos dos modelos, fueron los obtenidos después de someterse a una optimización/ajuste de hiperparámetros, tanto para los parámetros del algoritmo/clasificador, como para los métodos de vectorización y sobremuestreo (todo el *pipeline*); esto con ayuda del módulo de *GridSearchCV* de la biblioteca *sklearn*, el cual realiza una búsqueda exhaustiva de los mejores parámetros de entrada para cada modelo.

Dataset	Clase	GradientBoosting Classifier			RandomForest Classifier			Logistic Regression			LinearSVC			KNeighbors Classifier			DecisionTree Classifier			BernoulliNB		
		P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
Jira	*Positiva*	0.77	**0.87**	**0.82**	**0.78**	0.85	0.81	**0.78**	0.78	0.70	0.72	0.70	0.71	0.65	0.62	0.63	0.74	0.82	0.78	0.71	0.81	0.76
	Negativa	**0.85**	0.64	**0.73**	*0.80*	0.66	0.72	0.75	*0.66*	0.70	0.69	0.65	0.67	0.33	**0.73**	0.44	0.64	0.67	0.65	0.67	0.60	0.63
	Neutral	**0.90**	**0.90**	**0.90**	0.89	**0.90**	0.89	0.87	0.89	0.88	0.85	0.86	0.85	0.83	0.59	0.68	0.88	0.84	0.86	0.87	0.85	0.86
	Micro	**0.86**	**0.86**	**0.86**	0.85	0.85	0.85	0.84	0.84	0.84	0.80	0.80	0.80	0.61	0.61	0.61	0.81	0.81	0.81	0.81	0.81	0.81
	Macro	**0.84**	0.80	**0.81**	0.82	0.80	**0.81**	0.80	0.78	0.79	0.75	0.74	0.74	0.60	0.65	0.58	0.75	**0.82**	0.78	0.75	0.76	0.75
	Accuracy		**0.86**			0.85			0.83			0.80			0.61			0.81			0.81	

Tabla 4.2: Métricas de rendimiento para distintos modelos entrenados con el dataset de Jira y utilizando diferentes algoritmos de clasificación. Se remarcan con negro la métricas más altas.

Por otro lado, en la figura 4.4 (línea 16) y en la figura 4.5 (línea 17) se muestran la Matriz de Confusión y las Curvas ROC, respectivamente (para el modelo de clasificación final). De la matriz podemos destacar que, evaluando al modelo sobre el conjunto de prueba (30 % dataset), este clasificó de forma errónea 99 comentarios negativos (etiqueta actual) como comentarios neutros (etiqueta de predicción) y, además, confundió 105 comentarios neutros como positivos. En total, se clasificaron correctamente 1,649 comentarios (suma de los cuadros de la diagonal) y de forma incorrecta 295 (resto de los cuadros), entre las tres clases. Mientras que en las Curvas ROC, se observan los valores del área bajo la curva para las diferentes clases, siendo de 0.90 para la clase negativa, 0.89 para la neutral y 0.94 para la positiva.

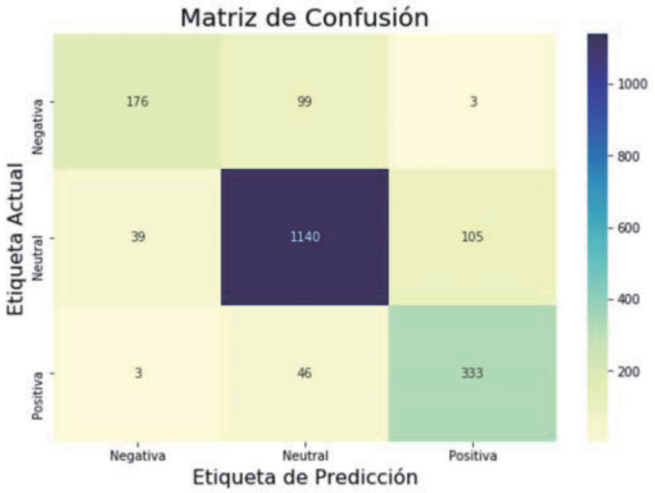

Figura 4.4: Matriz de confusión para el modelo de clasificación final.

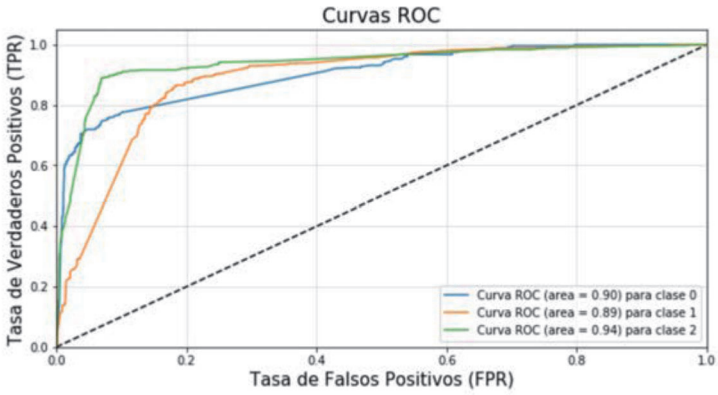

Figura 4.5: Curvas AUC-ROC del modelo de clasificación final. Clase 0: Negativo, Clase 1: Neutral, Clase 2: Positiva.

Antes de concluir con la etapa de evaluación, pasaremos a comparar el modelo resultante antes descrito (*ASIS*) con las principales herramientas de clasificación de sentimientos en la actualidad (descritas en el capítulo 3): *SentiCR, Senti4SD y SentiStrengthSE.* La tabla 4.3 muestra las métricas de rendimiento para cada herramienta entrenada con el dataset de Jira; se observa que, en comparación con las demás, el modelo ASIS desarrollado presenta los valores más altos para la gran mayoría de las métricas de rendimiento en las tres clases sentimientos.

Dataset	Calse	ASIS			SentiCR			Senti4SD			SentiStrengthSE		
		Precision	*Recall*	*F1Score*	*Precision*	*Recall*	*F1Score*	*Precision*	*Recall*	*F1Score*	*Precision*	*Recall*	*F1Score*
Jira	*Positiva*	**0.77**	0.87	**0.82**	0.76	0.89	**0.82**	0.76	0.79	0.78	0.50	**0.91**	0.65
	Negativa	**0.85**	**0.64**	**0.73**	*0.81*	0.61	0.70	0.72	*0.57*	0.64	0.41	**0.64**	0.50
	Neutral	**0.90**	**0.90**	**0.90**	0.89	0.89	0.89	0.86	0.89	0.88	0.89	0.59	0.71
	Micro	**0.86**	**0.86**	**0.86**	0.85	0.85	0.85	0.83	0.83	0.83	0.66	0.66	0.66
	Macro	**0.84**	**0.80**	**0.81**	0.82	**0.80**	0.80	0.78	0.75	0.76	0.60	0.71	0.62

Tabla 4.3: Métricas de rendimiento para las distintas herramientas de clasificación de sentimientos (Clases: Positivos, Negativos y Neutrales) entrenadas con el dataeset de Jira. Se remarcan con negro las métricas más altas.

Una vez que el modelo fue entrenado y evaluado, y que se obtuvo la mejor evaluación posible o la deseada, se pasa a la predicción de sentimientos sobre los nuevos datos (comentarios) extraídos de los repositorios. A continuación se muestra una versión resumida del código implementado en la predicción:

```
1     def predictSentiments(issuesData, trained_model):
2         sentimentScores = []
3         for index, row in issuesData.iterrows():
4             sent = int(train_model.predict([str(row.comment)]))
5             sentimentScores.append(sent)
6         return sentimentScores
7
```

```
8       trained_model = trainModel(TfidfVectorizer(params),
        ↪  SVMSMOTE(params),GradientBoostingClassifier(params),'datasetJIRA.csv')
9       servers_projects = ['APACHE-MAPREDUCE', 'MONGO-PYTHON',...]
10      for s_p in servers_projects:
11          issuesData = pd.read_csv('repoData/ISSUES-COMMENTS-'+s_p+'.csv')
12          issuesData["sentiment"]=predictSentiments(issuesData,trained_model)
13          issuesData.to_csv('repoDataSA/ISSUES-COMMENTS-'+s_p+'.csv')
```

Se comienza entrenando el modelo (línea 8) llamando a la función *trainModel* (antes mencionada). Posteriormente se define una lista (línea 9) con el nombre de todos los proyectos en los cuales se hará la predicción (cada elemento debe ser seguir el patrón de nombramiento: "SERVIDOR-PROYECTO"). Se itera sobre la lista de proyectos (líneas 10 a 13), en donde se realiza la lectura de los archivos *.csv* (línea 11) de cada uno, los cuales contienen todas sus incidencias y respectivos datos. Después se ejecuta la función *predictSentiments* (línea 12 y definida en líneas 1 a 6), la cual recibe como parámetros los datos de las incidencias y el modelo entrenado; con esto, para cada comentario de incidencia se pasa a predecir/clasificar su sentimiento asociado (línea 3 y 4). Finalmente, estos sentimiento se retornan (línea 6) y se asignan (línea 12) como un nuevo valor para cada fila/comentario en una nueva columna de nombre *sentiment* y por último se guardan los cambios (línea 13). En la figura 4.6 se muestra esta nueva columna que almacena un número entero (remarcada con color rojo), que puede tomar tres valores: el valor de 1 indica que el sentimiento asociado al comentario es positivo, el valor de -1 indica que es negativo y un valor de 0 indica que es neutral.

comment	issueComDate	issueCreated	issueResolution	issuetype	key	resolution.date	summary	timeZone	sentiment
fixes the problem by onnecting to NameNod...	2005-07-24T23:51:00.000+0000	2005-07-24T23:46:18.000+0000	Won't Fix	Bug	HADOOP-8	2006-02-24T08:13:30.000+0000	NDFS DataNode advertises localhost as it's add...	Etc/UTC	0
sh there was "edit" option IRA. Obvio...	2006-01-11T02:46:51.000+0000	2006-01-11T02:45:05.000+0000	Invalid	Bug	HADOOP-18	2006-03-07T06:30:32.000+0000	Crash with multiple temp directories	Etc/UTC	0
I have sucessfully used ored.local.dir with ...	2006-01-12T03:49:20.000+0000	2006-01-11T02:45:05.000+0000	Invalid	Bug	HADOOP-18	2006-03-07T06:30:32.000+0000	Crash with multiple temp directories	America/Los_Angeles	1
Say! Here's a patch that plements the JobTra...	2006-01-21T05:42:08.000+0000	2006-01-21T05:40:53.000+0000	Fixed	Bug	HADOOP-7	2006-02-07T03:38:43.000+0000	MapReduce has a series of problems concerning ...	Etc/UTC	0

Figura 4.6: Nueva columna de nombre *sentiment* que indica el sentimiento asociado para cada comentario (columna *comment*), siendo 1 positivo, -1 negativo y 0 neutral.

Teniendo el modelo entrenado y evaluado, y posteriormente haber hecho la predicción de sentimientos sobre los datos extraídos, se pasa a la última y más importante etapa del proceso (etapa 4) en la que se realiza el análisis y exploración de los datos.

4.5. Exploración y análisis

En esta última etapa del proceso se lleva a cabo un análisis sobre los datos con el fin responder a las preguntas de investigación planteadas en esta tesis. A continuación se muestra una versión resumida del código que se implementó en esta etapa (el código completo se encuentra en el archivo *analytics.py* del proyecto[6]):

```
1        # import modules (pandas, numpy, statistics, scipy.stats, wilcoxon,
         ↪  pearsonr, matplotlib, etc)
2        issues = pd.read_csv()
3        def sentimentsRepos(issues)
4        def sentimentsAndIssueResolutionType(issues)
5        def sentimentsAndHourOfDay(issues)
6        def sentimentsAndDayOfWeek(issues)
7        def sentimentsAndIssueResolutionTime(issues)
```

Se comienza con la declaración de los módulos necesarios para el análisis (línea 1) y después se obtienen todas las incidencias (y sus datos) leyendo los archivos para todos los proyectos almacenados (línea 2). Posteriormente, desde la línea 3 a la 8 se definen las funciones principales, donde cada una realiza el análisis correspondiente con el fin de responder a cada una de las preguntas de investigación (PI), esto es:

- La función *sentimentsRepos* (PI 1) realiza el análisis de sentimientos en comentarios de incidencias para los proyectos de software considerados; muestra ejemplos de comentarios junto con su sentimiento expresado y la distribución general de sentimientos positivos, negativos y neutrales.
- La función *sentimentsAndIssueResolutionType* (PI 2) realiza un análisis entre los sentimientos en comentarios con el tipo de resolución de la incidencia (resuelta o no resuelta); se observa en particular si aquellas incidencia resueltas contienen mayor sentimiento positivo y las no resueltas mayor sentimiento negativo.
- En las funciones *sentimentsAndHourOfDay* y *sentimentsAndDayOfWeek* (PI 3) se estudia la relación entre sentimientos en comentarios y la hora/día de la semana en que se escribió el comentario; observando cómo varían los sentimientos durante el día y a lo largo de la semana y ver cuándo es que los desarrolladores expresan más/menos sus sentimientos.
- En la función *sentimentsAndIssueResolutionTime* (PI 4) se analiza la relación entre sentimientos en comentarios y el tiempo de resolución de una incidencia; se busca encontrar alguna correlación entre estas dos variables, estudiando cómo varían los sentimientos con respecto al tiempo (en días) en que se tardaron en finalizar las incidencias (resueltas y no resueltas).

En siguiente capítulo 5 se muestran los resultados de estos análisis, presentando las configuraciones, restricciones, gráficos, tablas y discusiones acerca de ellos.

[6]https://github.com/andricValdez/ASIS/blob/master/analytics.ipynb

Capítulo 5

Resultados

> "Mientras me quede algo por hacer, no habré hecho nada."
>
> *Julio César*

En este capítulo se mostrarán y discutirán los resultados obtenidos al aplicar la metodología descrita en el capítulo anterior. Cada una de las secciones tienen la finalidad de responder a las preguntas de investigación planteadas en este trabajo de tesis. Se comienza con la **sección 5.1** en donde se muestra la distribución general de sentimientos en los repositorios y ejemplos de comentarios clasificados como positivos, negativos y neutrales. En la **sección 5.2** se estudia la relación entre los sentimientos y el estado de resolución de las incidencias, esto es, cómo se expresan los sentimientos cuando las incidencias son resueltas y cuando no lo son. En la **sección 5.3** se muestra cómo se expresan los sentimientos de acuerdo a la hora del día y día de la semana en el que se trabajó en las incidencias. Y por último, en la **sección 5.4** se estudia la relación entre el tiempo de resolución de las incidencias (en días) y los sentimientos expresados.

Por otro lado, para este estudio se consideraron un total de 30 proyectos alojados en servidores de *Mongo, Apache, JBoss y Spring*; en la tabla 5.1 se muestran los proyectos seleccionados, siendo los de mayor actividad (reportando desde el año 2002 al 2020) y con mayor cantidad de datos disponibles (en incidencias y comentarios) respecto de los demás.

Servidor	**Proyectos**	**Total**			**Incidencias >10 comentarios**	
		# Proyectos	**# Incidencias**	**# Comentarios**	**# Incidencias**	**# Comentarios**
Mongo	SERVER, CSHARP, JAVA, NODE, TOOLS, PYTHON, MONGOID PHP, RUBY, WT	10	62 008	224 511	3 875	66 745
Apache	ACCUMULO, CASSANDRA, MAPREDUCE, IGNITE, STORM, ATLAS, BEAM, DIRSERVER, FELIX	9	52 714	319 114	8 086	174 221
JBoss	AEROGEAR, DROOLS FUSETOOLS, JBPAPP, RHBPMS, AS7, JBAS	7	31 650	106 160	1 708	25 507
Spring	STS, BATCH INT, ROO	4	13 444	42 793	633	9 481
Total		**30**	**159 816**	**692 578**	**14 302**	**275 954**

Tabla 5.1: Servidores, proyectos, incidencias y comentarios considerados en el estudio.

Entre todos los proyectos se obtiene un total de 159,816 incidencias y 692,578 comentarios, además, también se muestra el total de incidencias (14,302) y comentarios (275,954) después de aplicar un filtro para obtener incidencias con 10 comentarios o más, los cuales son lo que se utilizan en los análisis posteriores (Esto con el fin de permitir que fluya una conversación y evitar que el sentimiento general por incidencia se vea influenciado por unos pocos comentarios).

5.1. Sentimientos en repositorios de software

En este caso se realizó un análisis general de los sentimientos en comentarios de incidencias. En la tabla 5.2 se muestran algunos ejemplos de comentarios escritos junto con su etiqueta de sentimiento asignada por el clasificador mencionado en el capítulo anterior. Se observó que en los comentarios positivos generalmente se observa gratitud y amabilidad por parte de los desarrolladores, en donde se utilizan constantemente palabras como *Thanks, Good, Great* y emoticones como " :) o :D ". En comentarios negativos se transmite generalmente lamento, preocupación y enojo, utilizando palabras como *Sorry, Hell, Horrible, Shit* y emoticones como " :(o :/ ". Los comentarios neutros, generalmente, contienen un lenguaje más técnico en donde se utilizan palabras más relacionas al contexto como *Patch, Update, Version* y donde es muy común encontrar dentro del texto fragmentos de códigos, logs, enlaces (http), detalles de errores, versiones de software, etc.

Comentarios de Incidencias	Sentimiento
Thanks for the support, Suresh. Thanks also to Arpit for writing some of the code, and to all the reviewers.	**Positivo**
Andrey, your experience is very useful to me. The improvement rate is very significant, so we should do this work :) I have assigned you this task, so please continue your work. Thanks for your help!	
Wait, what? This is a horrible idea. I can't believe such a major change was hidden in another Jira :(	**Negativo**
How is this considered ""minor""? I've been breaking my head on failed tests that shouldn't fail. This is a major problem. What's even worse is that there is no clear and easy solution to this. Running the update many times is horrible and immensely expensive.	
This patch updates the code to use libuv 0.12 instead of libuv 0.11. APIs changed a bit.	**Neutral**
Can you provide the following versions? Python, Node, Mongoengine, Gevent. Also, what operating system are you using?	

Tabla 5.2: Ejemplos de comentarios de incidencias junto con su sentimiento asociado

Por otro lado en la figura 5.1 (tabla y gráfica) se muestra la distribución de sentimientos para cada servidor, así como el total considerando todos (para incidencias con al menos 10 comentarios). Un 10.76 % lo componen comentarios positivos (representan 29,693 del total), un 1.77 % de los comentarios son negativos (representan 4,884) y un 87.47 % de los comentarios son neutros (representan 241,377). Se puede observar que la gran mayoría de los sentimientos son neutrales, es decir, existe una ausencia de sentimiento. Esto puede deberse principalmente a que, en proyectos de esta naturaleza, la comunicación entre los profesionales del software tiende a ser muy técnica, que como ya se mencionó, en la mayoría de los comentarios se comparten códigos y terminologías específicas.

Más allá que el sentimiento neutral es el que más se expresa en los comentarios, también una cantidad considerable tienden a expresar un sentimiento positivo o negativo, y es en estos en donde se centró la mayor parte de nuestro análisis, estudiando cómo estos se expresan y relacionan con otras variables, como el estado de resolución de la incidencia, el tiempo qué tardó en resolverse , etc. Estos análisis se presentan en las secciones siguientes.

Servidor	% Sent Positivo	% Sent Negativo	% Sent Neutral
MONGO	9.89	1.77	88.33
APACHE	12.8	1.89	85.32
JBOSS	6.65	1.31	92.04
SPRING	10.3	2.01	87.69
TOTAL	**10.76**	**1.77**	**87.47**

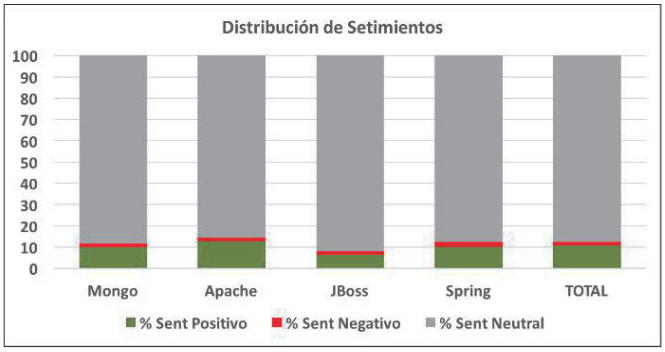

Figura 5.1: Distribución de sentimientos en comentarios de incidencia: positivos, negativos y neutral

5.2. Sentimientos y resolución de incidencias

Se estudió la relación entre los sentimientos en comentarios de incidencias y el estado de resolución de las incidencias, esto es, de qué forma se expresan y se distribuyen los sentimientos cuando las incidencias fueron resueltas y cuando no lo fueron. Para llevar a cabo este análisis, el sistema de Jira permite asignar varios tipos de etiqueta de resolución a las incidencias de acuerdo a su estado de finalización (*Done, Won't Do, etc*), por lo que, con el fin de facilitar el análisis, cada incidencia se agrupó en una de las dos categorías principales: Resuelta y No Resuelta. En la tabla 5.3 se observa esta agrupación, se muestran las etiquetas de resolución más comúnmente utilizadas en Jira (lado derecho) junto con su categoría de resolución (lado izquierdo).

Resolución de Incidencia	**Etiqueta**
Resuelta	Done, Fixed, Resolved, Answered, Complete, Resolved Locally, Deployed, Works as Designed, Community Answered, Implemented,
No Resuelta	Won't Do, Won't Fix, Invalid, Rejected, Cannot Reproduce, Duplicate, Incomplete, Duplicate, Out of Date, Gone away, Abandoned

Tabla 5.3: Etiquetas del sistema Jira para incidencias resueltas y no resueltas.

En la tabla 5.4 se observa un ejemplo de una incidencia resuelta (tipo *Bug*) para el proyecto *STORM* de *APACHE*. Se muestra su secuencia de 12 comentarios junto con sus sentimientos asociados, lo que corresponde a que la incidencia contiene un total de 41.67 % comentarios positivos, 8.33 % negativos y 50 % neutrales. Al leer la secuencia de comentarios se puede notar, en general, una actitud positiva, donde se expresa gratitud y acuerdo entre los desarrolladores[1].

Por otro lado, en la tabla 5.5 se observa un ejemplo de una incidencia no resuelta (tipo *Bug*) para el proyecto *PHP* de *MONGO*; se muestra la secuencia de 17 comentarios junto con su sentimiento asociado (se omiten 5 neutros). Donde el sentimiento total de la incidencia es de 11.76 % comentarios positivos, 29.41 % negativos y 58.82 % neutrales. Contrario a la incidencia resuelta, al leer la secuencia de comentarios se distingue, en general, un sentimiento más negativo, donde se expresa menos gratitud, más desacuerdo y molestia entre los desarrolladores.

[1]En cada comentario se remarcan las palabras más relevantes según el modelo de clasificación (coeficientes); el color verde indica una correlación positiva y el color rojo una negativa según la clase.

#	Comentario	Sentimiento	#	Comentario	Sentimiento
1	thanks for reporting the issue. Processing-time based windows internally uses a ScheduledExecutor.	**Positivo**	**7**	I made some minor tweak to process the tuples with some time offset, please pull the current changes and test.	**Neutral**
2	Sorry for late response, call on getStartTimestamp() and getEndTimestamp()	**Negativo**	**8**	I pull and test again, yes, no tuple lost now. Thanks.	**Positivo**
3	You might want to try your tests with the patch	**Neutral**	**9**	when you get some time can you pull the updated changes and test again ?	**Neutral**
4	I tested several times with your patch, now it seems all right, no tuples overlap. Thank you	**Positivo**	**10**	I test again, as the same arguments as before, it seems all right. But when I make the spout emit more frequently, the ahead tuples are expired	**Neutral**
5	Eh, it still lose data, following is one test result, and you can see that the 15073 is lost.	**Neutral**	**11**	The initial tuples expired because the trigger was not fired exactly after the window interval but after a delay	**Neutral**
6	thanks for the update. I made some minor changes to the patch today. If you could pull the latest changes and run your tests a few times will be great.	**Positivo**	**12**	So that's it, I got, thanks !	**Positivo**

Tabla 5.4: Secuencia de comentarios para una Incidencia Resuelta del proyecto *STORM-APACHE*. Sentimientos: 41.67 % positivos, 8.33 % negativos y 50 % neutrales.

#	Comentario	Sentimiento	#	Comentario	Sentimiento
1	Sorry, but anyone had a chance to look at this? The error gets quite frequent after primary/secondary switch, but after restart it becomes fine.	**Negativo**	**7**	If you see the log I sent you there's some lines logging setting this option. Sorry, but I think that's not the root cause of the problem.	**Negativo**
2	Sorry - not yet. I'll have a look tomorrow unless somebody beats me to it.	**Negativo**	**8**	Hi guys, am I right about above comment?	**Neutral**
3	Hi Derick, thanks for looking into this. Any luck about this issue?	**Positivo**	**9**	Sorry but we've both been on vacations so haven't had the time to look into it again. Last I tried I couldn't reproduce it though.. I'll see if I can look into it tomorrow	**Negativo**
4	I got as far as setting up an SSL replicated environment, but this is causing me some issues that I haven't been able to nail down. I have not forgotten about this yet!	**Neutral**	**10**	Thanks for your reply. We found that the issue only happens when accessing the test script via apache, Let me know if you need any more information.	**Positivo**
5	Sorry to ask, but this issue is really blocking us from moving forward for hot-standby with PHP . Is there any way we can do or any resource we can reach to speed it up? Thanks	**Negativo**	**11**	You are not hitting anything similar to PHP-329, your issues are SSL specific.	**Neutral**
6	I cannot reproduce this at all. Can you create a short reproduce script that reproduces this issue for you?	**Neutral**	**12**	Closing this due to inactivity.	**Neutral**

Tabla 5.5: Secuencia de comentarios para una Incidencia No Resuelta del proyecto *PHP-MONGO*. Sentimientos: 11.76 % positivos, 29.41 % negativos y 58.82 % neutrales.

Posteriormente se realizó el mismo análisis descrito con los ejemplos anteriores pero sobre todas las incidencias disponibles. Esto es, tomando el sentimiento total de cada incidencia se compararon sus cantidades de sentimiento positivo y negativo, registrando el sentimiento que prevaleció en cada una (tanto para incidencias resueltas y no resueltas). Del total de incidencias analizadas (14,302) un 87 % fueron resueltas, mientras que un 13 % fueron no resueltas. En la figura 5.2 se muestran los resultados obtenidos.

Figura 5.2: Sentimientos expresados en incidencias A) Resueltas y B) No Resueltas.

Para las Incidencias Resueltas (A) se observa que un 86 % de estas presentó mayor cantidad de sentimiento positivo que negativo, un 6.1 % obtuvo mayor cantidad de sentimiento negativo que positivo y en 7.9 % el sentimiento fue el mismo; por otro lado, para las Incidencias No Resueltas (B) se observa que el 75.8 % presentó mayor cantidad de sentimiento positivo que negativo, el 16.4 % obtuvo mayor cantidad de sentimiento negativo que positivo y en un 7.9 % el sentimiento fue el mismo. En ambos tipos, el sentimiento positivo es el que prevaleció en la mayoría de las incidencias, pero se puede observar un aumento de sentimiento negativo y un decremento del positivo en incidencias no resueltas respecto a las incidencias resueltas.

5.3. Sentimientos y hora/día de la semana

Como ya se mencionó anteriormente, además de su etiqueta de sentimiento, cada comentario cuenta con el registro de cuándo fue publicado o escrito, esto nos permite conocer la fecha exacta en las que se trabajaron las incidencias. Son con estos datos que se estudia la relación entre los sentimientos y la hora del día y el día de la semana, esto es, cómo se expresan o varían los sentimientos a lo largo del día o semana. Por otro lado, de los 30 proyectos considerados, se cuenta con un total de 844 semanas de actividad que abarcan desde el año 2002 al 2020, siendo este rango de fechas las que se tomaron en cuenta para el análisis.

Comenzando con los sentimientos a lo largo del día, para facilitar su análisis y visualización, se agruparon los comentarios escritos durante las 24 hrs del día en partes o periodos del día (calculando el sentimientos promedio por periodo). Esto es, comentarios de incidencias escritos durante la Madrugada (desde 00:00 hrs a las 5:59 hrs), Mañana (desde 06:00 hrs a las 11:59 hrs), Tarde (desde 12:00 hrs a las 17:59 hrs) y Noche (desde 18:00 hrs a las 23:59 hrs).

De los 275,954 comentarios analizados, la mayoría se publicó por la Tarde con 32.7 %, seguido de la Noche con 31 %, después por la Mañana con 19.1 % y por último por la Madrugada con 17.2 %. En la figura 5.3 se muestran los resultados de este análisis, presentando la forma en que se expresan/varían los sentimientos a lo largo del día.

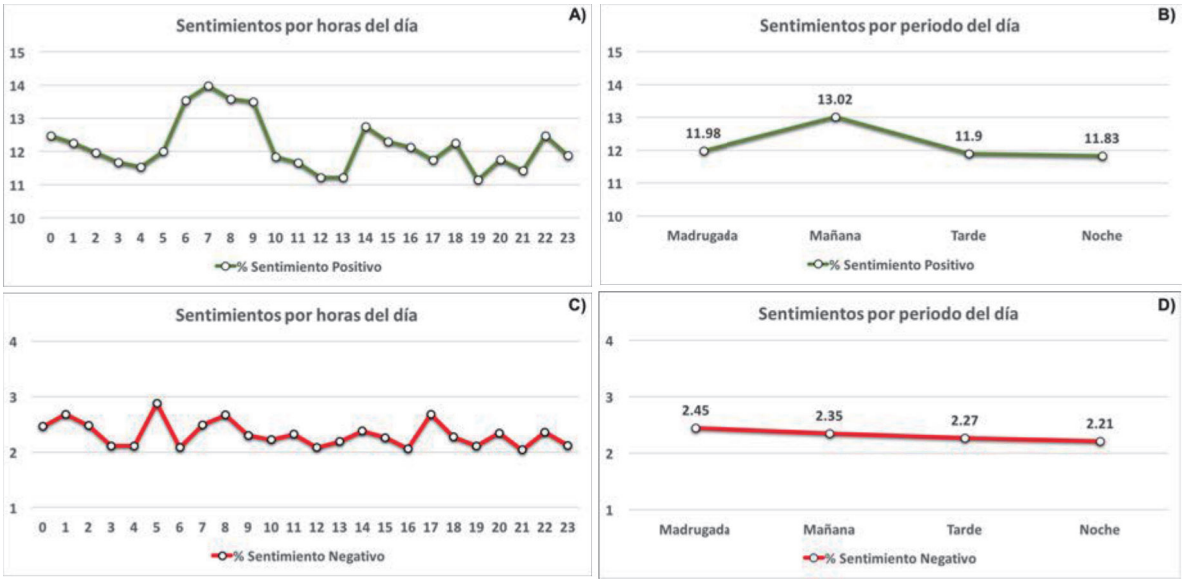

Figura 5.3: Distribución de sentimientos a lo largo del día (24 hrs). A) % Sentimiento positivo por hora. B) Sentimiento positivo por periodo del día. C) Sentimiento negativo por hora. D) Sentimiento negativo por periodo del día.

Se exponen los sentimientos positivos por hora y periodo del día (A y B respectivamente) y también los sentimientos negativos por hora y periodo del día (C y D respectivamente). En las gráficas se observa que por la mañana es cuando más se expresan los sentimientos positivos llegando a un máximo entre las 6-8 hrs, mientras avanza el día este sentimiento va disminuyendo hasta llegar a un mínimo en la noche entre las 19-21 hrs. Mientras que los sentimientos negativos suelen expresarse más por la madrugada y menos por la noche.

Posteriormente se estudió la relación entre los sentimientos y los días de la semana. El 75 % de los comentarios de incidencias se escribieron entre semana (Lunes a Jueves), mientras que en el fin de semana (Viernes a Domingo) se escribieron el 25 %. En la figura 5.4 se muestra las gráficas del % de sentimiento positivo y negativo contra el día de la semana (A y B respectivamente), de Lunes a Domingo.

En la gráfica 5.4 A) se observa que, a partir del Lunes, el sentimiento positivo va decreciendo ligeramente conforme avanza la semana hasta llegar al día Viernes, que es donde más se expresan respecto del resto; mientras que los días Sábados y Domingo hay una caída de sentimiento positivo, siendo estos días cuando menos se expresan. Por otro lado, en cuanto a la expresión de sentimientos negativos, en la gráfica 5.4 B) se observa que, desde el día Jueves a Sábado el sentimiento va disminuyendo, siendo los días Sábados y Lunes cuando menos se expresa la negatividad, mientras que el día Domingo es cuando mayor es la negatividad.

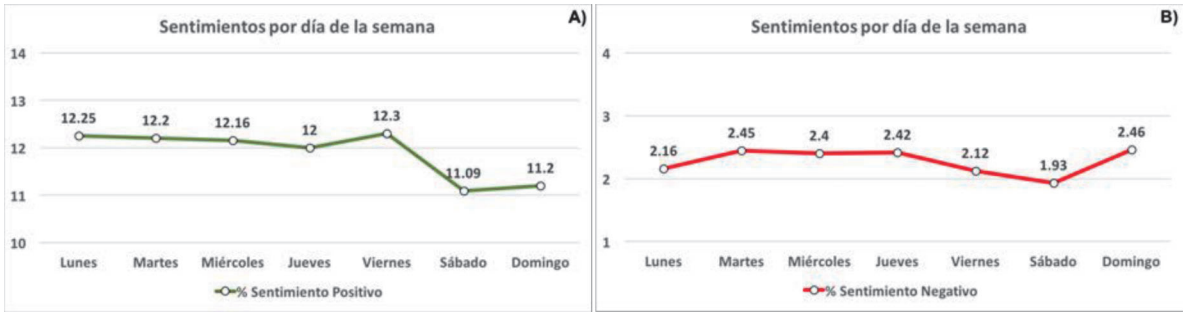

Figura 5.4: Sentimientos a lo largo de la semana. A) Sentimiento Positivo. B) Sentimiento Negativo.

5.4. Sentimientos y tiempo de resolución

En esta sección se muestran los resultados en el estudio de la relación entre los sentimientos y el tiempo de resolución de las incidencias (considerando las Resueltas y No Resueltas). Para esto, se mide el grado de dependencia lineal entre estas dos variables utilizando el coeficiente de correlación de *Pearson* [48]. Para obtener el tiempo de resolución, para cada incidencia se toma su fecha de creación y de finalización (ver figura 4.3: *issueCreated y resolutionDate*) y se aplica una resta de fechas para tener el tiempo total implementado en días.

Por otro lado, se aplicaron métodos de diagramas de caja para obtener una distribución de los datos (sentimientos y los tiempos de resolución) en sus rangos y cuartiles, principalmente para observar los valores atípicos que se presentan en cada caso y con ellos eliminar aquellas incidencias que estén fuera de rango. En la figura 5.5 se muestran los diagramas después de la depuración en incidencias resueltas y no resueltas para A) Tiempos de Resolución, para B) Sentimientos Neutrales, para C) Sentimientos Positivos y para D) Sentimientos Negativos.

Una observación interesante en 5.5 A) es que el tiempo promedio de resolución en incidencias no resueltas (40 días) es mayor que en las resueltas (25 días), también se observa esta tendencia para las incidencias del primer cuartil (14 días en resueltas y 9 días en no resueltas) y el tercer

cuartil (78 días en resueltas y 54 días en no resueltas). Otra observación en 5.5 D), es que la cantidad promedio de sentimiento negativo es mayor en incidencias no resueltas (7.14 %) que en las resueltas (6.25 %), cumpliéndose esta tendencia para el primer y tercer cuartil.

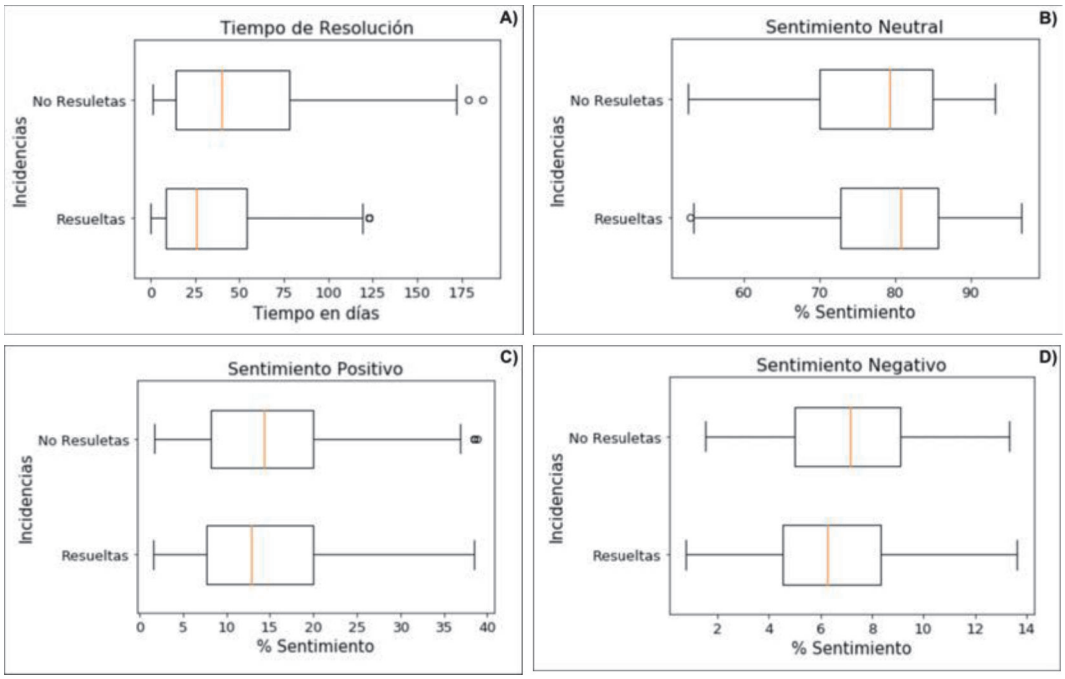

Figura 5.5: Diagramas de caja de incidencias Resueltas y No Resueltas para su: A) Tiempo de Resolución, y sus Sentimientos: B) Neutrales, C) Positivos y D) Negativos.

En la figura 5.6 se muestra la distribución de las incidencias no resueltas, en donde se gráfica su sentimiento contra su tiempo de resolución en días; presentando en A) Sentimientos Positivos, B) Sentimientos negativos y C) Sentimientos neutros. Se puede observar que, en general, tanto en A) como B), conforme aumenta el tiempo de resolución el sentimiento disminuye, mientras que para C) el sentimiento aumenta. Calculando los coeficientes de correlación de *Pearson* (r) [2] para cada caso se obtienen los siguientes resultados:

- En los sentimientos positivos vs el tiempo de resolución existe una correlación negativa de $r = -0.28$, con una significación (p-valor) de 0.0015 y con una fuerza de asociación pequeña entre las variables [3].
- En los sentimientos negativos vs el tiempo de resolución también existe una correlación negativa de $r = -0.30$, con una significación de 0.0006 y con una fuerza de asociación mediana.
- En los sentimientos neutros vs el tiempo de resolución existe una correlación positiva de $r = 0.32$, con una significación de 0.0003 y con una fuerza de asociación mediana.

[2]Este valor de correlación varía en un intervalo de -1 a 1, indicando el signo el sentido de la relación. Esto es, si $0 < r < 1$, existe una correlación positiva; si $-1 < r < 0$, existe una correlación negativa; si $r = 0$, no existe relación.

[3]La fuerza de asociación se cataloga como pequeña si $\pm 0.1 < r < \pm 0.3$, como mediana si $\pm 0.3 < r < \pm 0.5$, y como grande si $\pm 0.5 < r < \pm 1$

Figura 5.6: Distribución de los sentimientos A) Positivos, B) Negativos y C) Neutrales con respecto al tiempo de resolución para Incidencias No Resueltas

Ahora se presenta el mismo análisis anterior pero solo considerando las incidencias resueltas. En la figura 5.7 se muestra la distribución de las incidencias, en donde se grafica su sentimiento contra su tiempo de resolución en días; presentando en A) Sentimientos Positivos, B) Sentimientos negativos y C) Sentimientos neutros. Se puede observar que, al igual que en el caso de las incidencias no resueltas, tanto en A) como en B) conforme aumenta el tiempo de resolución el sentimiento disminuye, mientras que para C) el sentimiento aumenta. Calculando los coeficientes de correlación de *Pearson* (r) para cada caso se obtienen los siguientes resultados:

- En los sentimientos positivos vs el tiempo de resolución existe una correlación negativa de $r = -0.11$, con una significación (p-valor) de 0.0006 y con una fuerza de asociación pequeña.
- En los sentimientos negativos vs el tiempo de resolución también existe una correlación negativa de $r = -0.17$, con una significación de $1.07x10^{-07}$ y con una fuerza de asociación pequeña.
- En los sentimientos neutrales vs el tiempo de resolución existe una correlación positiva con valor de $r = 0.16$, con una significación de $4.26x10^{-06}$ y con una fuerza de asociación pequeña.

Figura 5.7: Distribución de los sentimientos A) Positivos, B) Negativos y C) Neutrales con respecto al tiempo de resolución para Incidencias Resueltas

Capítulo 6

Conclusiones y trabajo futuro

> "No hay nada imposible para aquel que lo intenta"
>
> *Alejandro Magno*

En este capítulo se presentan las conclusiones finales del presente trabajo de tesis. En el cual se realizó un análisis de sentimientos (expresados en comentarios de incidencias) en los repositorios de software de Jira con el fin de estudiar la expresión de sentimientos de los profesionales de software en sus tareas del día a día, considerando factores como el día y hora de la semana en la que se trabajó, como el estado de resolución de sus tareas (cuando eran resueltas y cuando no) y como el tiempo (en días) que fue requerido para dar por terminada una tarea (resuelta o no resuelta).

Para llevar acabo estos análisis, se implementó una metodología basada principalmente en el proceso de Minería a de Textos para la tarea de clasificación (descrita en el capítulo 4). La cual consistió en cuatro etapas principales:

1. Etapa de desarrollo y entrenamiento del modelo de clasificación de sentimientos basado en aprendizaje supervisado.

2. Etapa de extracción y almacenamiento de los datos alojados en los repositorios de software Jira.

3. Etapa de evaluación del rendimiento del modelo de clasificación y predicción de sentimientos sobre los nuevos datos extraídos de los repositorios.

4. Etapa de exploración y análisis de los sentimientos encontrados.

Siguiendo esta metodología es que se da respuesta a las preguntas de investigación planteadas en esta tesis, las cuales se describen y responden en la siguiente **sección 6.1**. Posteriormente, en la **sección 6.2** se exponen las principales contribuciones de este trabajo. Por último, en la **sección 6.3** se menciona el trabajo a futuro en relación con esta tesis y dentro del área.

6.1. Conclusiones finales

A partir de los resultados obtenidos, podemos contestar las preguntas de investigación que se plantearon al inicio de este trabajo de tesis. A continuación respondemos brevemente dichas preguntas:

1. **¿De qué forma se expresan y cómo se distribuyen los sentimientos en los repositorios de software Jira?**

   De acuerdo a los resultados de la sección 5.1 del capítulo anterior, para los proyectos analizados, los sentimientos expresados por los profesionales del software en comentarios escritos, se distribuyen en un 10.76 % como positivos, un 1.77 % como negativos y un 87.47 % como neutrales; siendo estos últimos los que prevalecen en la gran mayoría de los comentarios, esto se debe a que, por un lado, es muy común que en proyectos de esta naturaleza se haga uso de un lenguaje muy técnico en las conversaciones escritas, y por otro lado, el corpus/dataset utilizado para entrenar el modelo muestra poco balance en su distribución de clases, siendo la clase neutral la de mayor peso (66 % del total), por lo que es posible que el modelo tienda a clasificar más comentarios como neutrales. Por otro lado, analizando el contenido de los comentarios para cada clase de sentimiento, se observa que en los comentarios positivos se expresa en su mayoría gratitud y amabilidad por parte de los desarrolladores; en los comentarios negativos se expresa generalmente lamento, preocupación y enojo; y en los comentarios neutros, generalmente se expresa un lenguaje más técnico, donde es común encontrar fragmentos de códigos, enlaces (http), descripción de errores, etc.

2. **¿Cómo es la relación entre los sentimientos y el tipo/estado de resolución de las incidencias?**

   Con base en los resultados de la sección 5.2 del capítulo anterior, se observó que cuando las incidencias no terminaban resolviéndose el sentimiento negativo aumenta (en un 10 %) y el sentimiento positivo disminuye (en un 10 %) respecto a los sentimientos de las incidencias resueltas. Esto considerando incidencias con 10 comentarios o más, aunque, también se hizo el estudio considerando cantidades mayores (con al menos 20, 30, 40, 50 comentarios por incidencias) mostrando la misma tendencia anterior y, además, la diferencia de porcentaje de sentimiento negativo aumenta entre incidencias resueltas y no resueltas a medida que se consideran más comentarios, siendo estas últimas la de mayor porcentaje negativo. Con esto, se puede decir que cuando las incidencias no terminan resolviéndose, está relacionado con el hecho de que los profesionales del software expresen una mayor negatividad y menor positivismo en sus comentarios (sucediendo lo contrario en las incidencias resueltas).

3. **¿Cómo se expresan los sentimientos de acuerdo a la hora del día y el día de la semana en la que se trabajaron las incidencias?**

   De los resultados obtenidos en la sección 5.3 del capítulo anterior, se observa que para el análisis de sentimientos a lo largo de un día[1] (24 hrs), se observó que es por las mañanas cuando los desarrolladores expresan mayor cantidad sentimiento positivo (entre las 6-8 hrs), y mientras avanza el día este sentimiento va disminuyendo hasta la noche, que es cuando menor es la expresión de sentimiento del día (entre las 19-21 hrs). En cuanto al

[1] La mayor actividad (mayor cantidad de comentarios publicados) se presentó por las tardes, seguido de las noches, después por las mañanas y por último por la madrugada.

sentimiento negativo, existe menos variación de este sentimiento durante el día, pero se observa estos suelen expresarse más por la madrugada y menos por las noches

Para el análisis de sentimientos a lo largo de la semana[2], se observó que, desde el día lunes a jueves la expresión de sentimiento positivo disminuye, después el día viernes aumenta, siendo este el día más positivo, por último, para los sábados y domingos hay una caída de sentimientos, siendo estos los días menos positivos de la semana. En cuanto al sentimiento negativo, son los sábados y lunes los días cuando menos se expresa negatividad, mientras que, son los domingos el día más negativo de la semana.

4. **¿Cómo es la relación entre los sentimientos y el tiempo de resolución de las incidencias?**

   En este caso se estudió cómo se correlacionan los sentimientos (positivos, negativos y neutral) con el tiempo de resolución de incidencias (resueltas y no resueltas). Se mide el grado de dependencia lineal entre estas dos variables (porcentaje de sentimiento y días de resolución) utilizando el coeficiente de correlación de *Pearson* (r), además, se lleva a cabo un proceso de depuración de datos atípicos utilizando el método de diagrama de caja[3] (eliminando incidencias fuera del rango máximo-mínimo). Los resultados de este análisis se muestran en la sección 5.4 del capítulo anterior, en los cuales se observó que, para las incidencias no resueltas, entre los sentimientos positivos y tiempo de resolución se obtiene una correlación negativa ($r = -0.28$), para los sentimientos negativos se obtiene una correlación negativa ($r = -0.30$) y para los sentimientos neutrales se obtiene una correlación positiva ($r = 0.32$); esto nos indica que existe una fuerza de asociación mediana entre ambas variables, y nos dice que, conforme aumenta el tiempo de resolución para incidencias no resueltas, el sentimiento positivo y negativos disminuye, mientras que el sentimiento neutral aumenta.

   Para este mismo análisis considerando solo incidencias resueltas, se observa que en este caso existe una fuerza de asociación pequeña entre sentimientos y tiempo de resolución, en donde, para los sentimientos positivos y tiempo de resolución se obtiene una correlación negativa ($r = -0.11$), para los sentimientos negativos se obtiene una correlación negativa ($r = -0.17$) y para los sentimientos neutrales se obtiene una correlación positiva ($r = 0.16$); lo que indica que, al igual que para las incidencias resueltas (pero en una menor medida), conforme aumenta el tiempo de resolución para incidencias no resueltas, el sentimiento positivo y negativos disminuye, mientras que el sentimiento neutral aumenta.

[2]En este caso el 75 % de los comentarios se escribieron entre semana (Lunes a Jueves), mientras que en el fin de semana el 25 % (Viernes a Domingo).

[3]Una observación interesante al aplicar este método (y que se muestra en la figura 5.5 A) es que el tiempo promedio de resolución en incidencias no resueltas fue de 40 días mientras que en las resueltas fue de 25 días, lo que indica que se requirió menos tiempo de trabajo en las incidencias resueltas que para las no resueltas; lo cual puede deberse a que, cuando las tareas se resuelven, existe mayor acuerdo, comunicación y coordinación de trabajo entre los desarrolladores que en aquellas tareas no resueltas (habiendo en estas más conflicto y menos comunicación), lo que permite terminar en menos tiempo sus tareas

6.2. Contribuciones del trabajo

A continuación se describen las tres principales contribuciones que tiene este trabajo de tesis:

1. **Análisis de sentimientos en repositorios de software.**

   La contribución principal de este trabajo de tesis es la aportar nuevos conocimientos a la Ingeniería de Software con el fin de ayudar a mejorar sus procesos. Al aplicar un análisis de sentimientos sobre los rastros de comunicación (textos) en los repositorios Jira, permite estudiar la relación y el impacto que estos tienen sobre los profesionales del software, principalmente en el desarrollo de sus tareas/actividades diarias. Eventualmente, estos resultados pueden contribuir en el desarrollo de tecnologías que permitan un monitoreo de los sentimientos que se experimentan durante el desarrollo de software y así ayudar a los jefes o líderes de proyectos para la toma de decisiones. Por ejemplo, una detección temprana de sentimientos negativos puede permitir aplicar acciones correctivas con el fin de aumentar las posibilidades de éxito de las tareas/proyectos de software, o bien, ayudar a tomar ciertas medidas para garantizar el bienestar en los equipos. Además, los resultados obtenidos pueden servir como punto de comparación para otros trabajos de investigación, por ejemplo, al analizar sentimientos en otros repositorios, validación de teorías y métodos, entre otros.

2. **Modelo de clasificación de sentimientos.**

   En este trabajo se lleva a cabo el diseño y desarrollo de un clasificador de sentimientos optimizado para repositorios de software Jira. Para su construcción se aplicaron técnicas de procesamiento de lenguaje natural, minería de textos y aprendizaje de máquina. Con esto, se va a contribuye con una herramienta más para el análisis de sentimientos en el contexto de la Ingeniería de Software, que hoy en día son pocas las disponibles al público para la investigación.

3. **Datos extraídos de repositorios de software.**

   La extracción de datos del ITS implica el desarrollo de un módulo que permita el consumo de una API REST que Atlassian-Jira expone para los desarrolladores. Para lograrlo, primero es necesario documentarse con su API y conocer los distintas rutas y métodos, posteriormente, hacer una búsqueda de los repositorios de código abierto que estén disponible al público, y por último, almacenar esta información para etapas posteriores. Al contribuir con el módulo (código) que permita la extracción de datos, así como con los datos extraídos de los repositorios utilizados en este trabajo, va a permitir un gran ahorro de tiempo para futuros trabajos de investigación enfocados a la MSR.

6.3. Trabajo futuro

Como trabajo a futuro sería interesante mejorar el desempeño de los modelos de clasificación de sentimientos, o bien, entrenar el modelo con otro conjunto de datos más equilibrado y representativo, y con ello replicar nuestro estudio para verificar si se obtienen resultados similares. Además, considerar otros servidores y/o proyectos disponibles (público o privados) para aplicar análisis de sentimientos y comparar resultados; también se podrían considerar otros factores para estudiar su relación con los sentimientos, por ejemplo: ¿Cómo se relacionan los sentimientos con el tipo de incidencia trabajada (*Bug, New Features, Improvement, etc*)?

o bien ¿Cómo es la relación entre sentimientos y el número de desarrolladores involucrados en una tarea?, entre otros.

Por otro lado, replicar los estudios implementados en esta tesis en otros sistemas de seguimiento de incidencias (como *Backlog, Asana o Trac*), o bien, probar con otros tipos de repositorios de otra naturaleza, como de pregunta-respuesta (como *StackOverflow*) o repositorio de control de versiones (como *GitHub o Bitbucket*) y comparar con nuestros resultados obtenidos.

Por último, también sería interesante que en el análisis se consideraran otros canales de comunicación colaborativos en donde se pudieran extraer más datos textuales, como por ejemplo el correo electrónico, plataformas de mensajería o incluso redes sociales; esto permitiría capturar en mejor y mayor medida el estado afectivo de los profesionales del software y conocer de forma más precisa cómo estos afectan o impactan en el rendimiento de sus tareas.

Bibliografía

[1] Ruchika Malhotra. *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. Chapman & Hall/CRC, 2015.

[2] Bing Liu. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*, pages 1–14. Cambridge University Press, 2015.

[3] Jira | software de seguimiento de proyectos e incidencias. `https://www.atlassian.com/es/software/jira`. (Accessed on 08/13/2020).

[4] N. Novielli and A. Serebrenik. Sentiment and emotion in software engineering. *IEEE Software*, 36(5):6–23, 2019.

[5] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli. Are bullies more productive? empirical study of affectiveness vs. issue fixing time. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 303–313, 2015.

[6] Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 352–355, New York, NY, USA, 2014. Association for Computing Machinery.

[7] Akshay Kulkarni and Adarsha Shivananda. *Natural Language Processing Recipes: Unlocking Text Data with Machine Learning and Deep Learning Using Python*. Apress, USA, 1st edition, 2019.

[8] Mohsen Farhadloo and Erik Rolland. *Fundamentals of Sentiment Analysis and Its Applications*, pages 1–24. Springer International Publishing, Cham, 2016.

[9] A. B. Pawar, M. A. Jawale, and D. N. Kyatanavar. *Fundamentals of Sentiment Analysis: Concepts and Methodology*, pages 25–48. Springer International Publishing, Cham, 2016.

[10] Brendan O'Connor, Ramnath Balasubramanyan, Bryan Routledge, and Noah Smith. From tweets to polls: Linking text sentiment to public opinion time series. volume 11, 01 2010.

[11] Xue Zhang, Hauke Fuehres, and Peter Gloor. Predicting stock market indicators through twitter – "i hope it is not as bad as i fear. *Procedia - Social and Behavioral Sciences*, 26:55–62, 12 2011.

[12] Bing Liu. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*, pages 47–135. Cambridge University Press, 2015.

[13] vinay kumar. Sentiment analysis techniques for social media data: A review. 09 2019.

[14] G. Ignatow and R. Mihalcea. *An Introduction to Text Mining: Research Design, Data Collection, and Analysis*. SAGE Publications, 2017.

[15] J. Zizka, F. Dařena, and A. Svoboda. *Text Mining with Machine Learning: Principles and Techniques*. CRC Press, 2019.

[16] Taeho Jo. *Text Mining: Concepts, Implementation, and Big Data Challenge*. Springer International Publishing, 2019.

[17] A. Géron and an O'Reilly Media Company Safari. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, Incorporated, 2019.

[18] R. Bilbro, T. Ojeda, and B. Bengfort. *Applied Text Analysis with Python*. O'Reilly Media, Incorporated, 2018.

[19] Fernando Doglio. *REST API Development with Node.Js: Manage and Understand the Full Capabilities of Successful REST Development*. Apress, USA, 2nd edition, 2018.

[20] Ryan Mitchell. *Web Scraping with Python: Collecting Data from the Modern Web*. O'Reilly Media, Inc., 1st edition, 2015.

[21] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[22] A.C. Müller and S. Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016.

[23] A gentle introduction to the gradient boosting algorithm for machine learning. `https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/`. (Accessed on 08/13/2020).

[24] Bagging and random forest ensemble algorithms for machine learning. `https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/`. (Accessed on 08/13/2020).

[25] Understanding auc - roc curve. in machine learning, performance... | by sarang narkhede | towards data science. `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`. (Accessed on 08/13/2020).

[26] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, Steve Counsell, Michele Marchesi, and Roberto Tonelli. Would you mind fixing this issue? an empirical analysis of politeness and attractiveness in software developed using agile boards. volume 212, 05 2015.

[27] V. Sinha, A. Lazar, and B. Sharif. Analyzing developer sentiment in commit logs. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 520–523, 2016.

[28] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 262–271, New York, NY, USA, 2014. Association for Computing Machinery.

[29] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. Anger and its direction in collaborative software development. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, ICSE-NIER '17, page 11–14. IEEE Press, 2017.

[30] W. N. Robinson, T. Deng, and Z. Qi. Developer behavior and sentiment from data mining open source repositories. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 3729–3738, 2016.

[31] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. What happens when software developers are (un)happy. *Journal of Systems and Software*, 140, 07 2017.

[32] R. Jongeling, S. Datta, and A. Serebrenik. Choosing your weapons: On sentiment analysis tools for software engineering research. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 531–535, 2015.

[33] M. R. Islam and M. F. Zibran. Leveraging automated sentiment analysis in software engineering. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 203–214, 2017.

[34] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[35] Steven Bird. Nltk: The natural language toolkit. COLING-ACL '06, page 69–72, USA, 2006. Association for Computational Linguistics.

[36] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 94–104, 2018.

[37] N. Novielli, D. Girardi, and F. Lanubile. A benchmark study on sentiment analysis for software engineering research. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 364–375, 2018.

[38] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams. The emotional side of software developers in jira. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 480–483, 2016.

[39] P. Shaver, J. Schwartz, D. Kirson, and C. O'connor. Emotion knowledge: further exploration of a prototype approach. *Journal of personality and social psychology*, 52 6:1061–86, 1987.

[40] J. Cohen. Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological bulletin*, 70 4:213–20, 1968.

[41] N. Novielli, F. Calefato, and F. Lanubile. A gold standard for emotion annotation in stack overflow. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 14–17, 2018.

[42] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi. Senticr: A customized sentiment analysis tool for code review interactions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 106–111, 2017.

[43] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. [journal first] sentiment polarity detection for software development. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 128–128, 2018.

[44] M. Thelwall, Kevan Buckley, and G. Paltoglou. Sentiment strength detection for the social web 1. 2012.

[45] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Edouard Duchesnay, and Gilles Louppe. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12, 01 2012.

[46] The jira software cloud rest api. `https://developer.atlassian.com/cloud/jira/software/rest/intro/`. (Accessed on 04/25/2020).

[47] jira · pypi. `https://pypi.org/project/jira/`. (Accessed on 04/25/2020).

[48] Pearson product-moment correlation. `https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php`. (Accessed on 10/07/2020).