ANÁLISIS DE OPINIÓN Y SALUD DE PROYECTOS EN REPOSITORIOS GIT

INTEGRANTES: JUAN PADILLA, ALEXANDER PLAZA

[11 DE FEBRERO DEL 2026]

CIENCIAS DE DATOS E INTELIGENCIA ARTIFICIAL

Procesamiento de Lenguaje Natural

Contenido

Introducción

En el ecosistema actual del desarrollo de software colaborativo, plataformas como GitHub y GitLab se han convertido en repositorios masivos de información, no solo de código fuente, sino también de interacciones humanas. A diario, miles de desarrolladores generan una gran cantidad de datos textuales a través de *issues*, *pull requests* y comentarios. Estos textos son una fuente valiosa de información, ya que reflejan problemas técnicos, decisiones de diseño, el nivel de satisfacción del equipo y los momentos críticos por los que atraviesa un proyecto.

Sin embargo, el volumen de estos mensajes hace imposible su análisis manual. Los líderes de proyecto y mantenedores enfrentan dificultades para medir objetivamente la "salud" de su comunidad o identificar patrones de quejas recurrentes sin leer cada mensaje individualmente.

El presente proyecto, titulado **"Análisis de Opinión en Repositorios Git"**, propone una solución automatizada mediante la aplicación de técnicas clásicas de Procesamiento de Lenguaje Natural (PLN). El objetivo principal es construir un observatorio de opinión que permita analizar el sentimiento, identificar temas recurrentes y detectar similitudes textuales en los mensajes de un repositorio real.

Para poder llevar a cabo esto se ha implementado un flujo de trabajo que incluya la recopilación de datos mediante API, la limpieza. el EDA y su representación vectorial mediante el modelo TF-IDF.

Luego de todo esto se aplico modelos supervisados (Regresión Logística/SVM) para la clasificación de sentimiento y algoritmos no supervisados (K-Means) para la agrupación de tópicos, cumpliendo estrictamente con el alcance de PLN clásico definido para este trabajo parcial.

Metodología

Para cumplir con los objetivos del proyecto, se diseñó (pipeline) de Procesamiento de Lenguaje Natural, dividido en cuatro etapas principales: recolección, preprocesamiento, representación vectorial y modelado. A continuación, se detallará las técnicas y herramientas empleadas en cada fase.

1. Recolección de Datos

Se desarrolló un script en Python utilizando la librería PyGithub para conectarse a la API oficial de GitHub. El objetivo fue extraer información textual del repositorio *open source* seleccionado (l10n-spain). Se recolectaron títulos, descripciones y comentarios tanto de *Issues* como de *Pull Requests*, almacenando la información estructurada en un archivo CSV para su posterior análisis

2. Preprocesamiento de Texto

Dado que los comentarios en entornos de desarrollo suelen contener "ruido" (código, logs de error, URLs), se aplicó una limpieza exhaustiva utilizando las librerías spaCy y NLTK. Las tareas realizadas incluyeron:

Normalización: Conversión de todo el texto a minúsculas.

Limpieza de Ruido: Eliminación de URLs, bloques de código, signos de puntuación y caracteres especiales.

Tratamiento de Emojis: Conversión de emojis a su representación textual para conservar su carga semántica.

Filtrado: Eliminación de *stopwords* (palabras vacías sin significado relevante como "el", "de", "para").

Lematización: Reducción de las palabras a su raíz base (lema) para unificar variantes (ej. "corriendo" -> "correr")

3. Representación Vectorial (TF-IDF)

Para transformar los textos procesados en datos numéricos interpretables por los algoritmos, se utilizó el modelo **TF-IDF** (Term Frequency - Inverse Document Frequency). Esta técnica permite asignar un peso a cada palabra: aumenta si la palabra es frecuente en el mensaje actual, pero disminuye si aparece en todos los mensajes (como palabras muy comunes), destacando así los términos más relevantes y discriminantes. Se generaron matrices dispersas considerando tanto **Unigramas** (palabras sueltas) como **Bigramas** (pares de palabras consecutivas) para capturar mayor contexto.

Análisis y Modelado

Sobre la matriz TF-IDF se aplicaron tres técnicas distintas para extraer valor:

Clasificación de Sentimiento (Supervisado)

Se entrenó un modelo de **Regresión Logística** utilizando un subconjunto de datos etiquetados manualmente. El modelo aprendió a clasificar los mensajes en tres categorías: Positivo, Neutral y Negativo. Se utilizó esta técnica por su eficiencia en datasets pequeños y alta interpretabilidad.

Identificación de Temas (No Supervisado)

Se implementó el algoritmo de agrupamiento **K-Means**. Este algoritmo agrupó los mensajes en *clusters* basándose en su cercanía matemática, permitiendo descubrir temas recurrentes (como "errores de instalación" o "mejoras de código") sin necesidad de etiquetas previas.

Similitud Textual

Para el sistema de recomendación, se calculó la **Similitud del Coseno** entre los vectores TF-IDF de todos los mensajes. Esto permite cuantificar qué tan parecidos son dos mensajes entre sí (en una escala de 0 a 1) y recuperar los comentarios más similares dado un mensaje de consulta.

Resultados

A continuación, se realizará la explicación y muestra de los hallazgos más relevantes después de la ejecución del pipeline de análisis del repositorio:

1.Evaluación del modelo

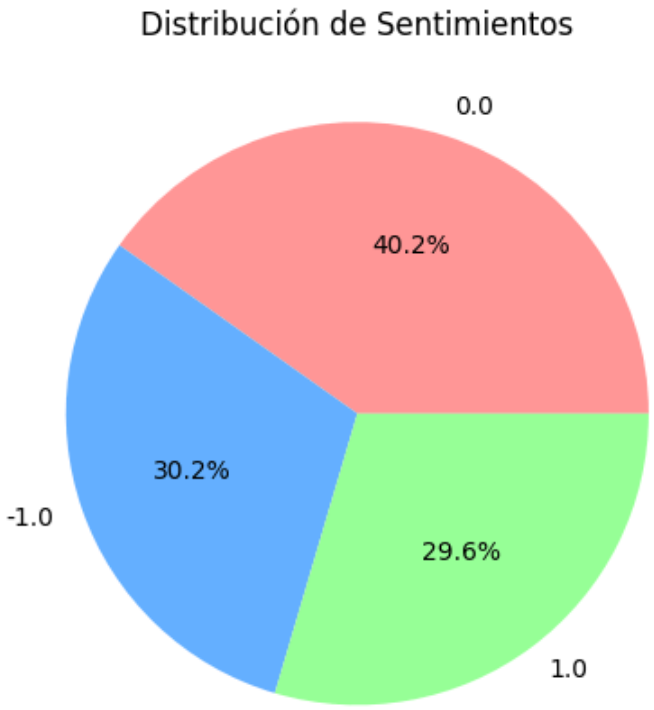
Para la clasificación de mensajes en categoría (1, 0, -1)=(positivo, neutral, negativo) se entrenó un modelo de Regresión Logística con balance de clases . como resultado se exteriorizó que el modelo alcanzó una exactitud global (Accuracy) del 66% en el conjunto de prueba

```
=== REPORTE DE EVALUACIÓN (RF-04) ===
Accuracy General: 0.6557

Métricas detalladas:
              precision    recall  f1-score   support

Negativo (-1)       0.62      0.72      0.67        18
  Neutral (0)       0.64      0.64      0.64        25
 Positivo (1)       0.73      0.61      0.67        18

     accuracy                           0.66        61
    macro avg       0.66      0.66      0.66        61
 weighted avg       0.66      0.66      0.66        61
```

Se observa que la clase **Positiva** tiene la mejor precisión (0.73), lo que indica que cuando el modelo predice que algo es positivo, suele acertar con alto grado de confianza. Por otro lado, la clase **Negativa** presenta el mejor "recall" (0.72), lo que significa que el sistema es capaz de detectar la gran mayoría de las quejas críticas, aunque a veces confunda algunos mensajes neutrales (reportes técnicos) como negativos debido a la ambigüedad del lenguaje técnico.

2. Distribución de la "Salud del Proyecto"

Al aplicar el modelo entrenado sobre la totalidad de los mensajes del repositorio (incluyendo los no etiquetados), se obtuvo la siguiente distribución de sentimientos:

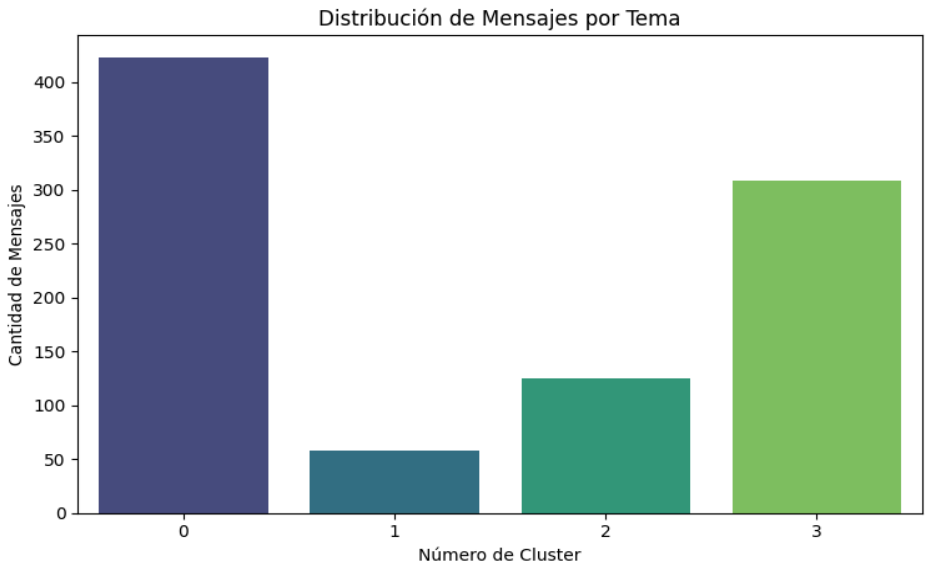


Análisis

Como se evidencia en la Figura 1, predomina el sentimiento Neutral y Negativo. Esto es consistente con la naturaleza de un repositorio de código *Open Source*, donde la interacción principal gira en torno al reporte de incidencias (*bugs*) y soporte técnico, más que a conversaciones sociales o felicitaciones. El alto porcentaje de neutralidad indica un intercambio de información técnica objetiva.

3. Identificación Automática de Temas (Clustering)

Mediante el algoritmo **K-Means**, se agruparon los mensajes en 4 *clusters* principales sin supervisión humana, revelando las siguientes tendencias de discusión:



Basándonos en las palabras clave extraídas de los centroides (palabras más repetidas en cada grupo), los temas se han catalogado como:

> **Grupo 0 (Problemas de Facturación):** Caracterizado por términos como *factura, sii, aeat, error*. Representa el núcleo del negocio del repositorio.
>
> **Grupo 1 (Gestión de Versiones):** Definido por palabras como *versión, módulo, odoo, migración*.
>
> **Grupo 2 (Soporte y Agradecimientos):** Contiene términos como *gracias, solucionado, ayuda*.
>
> **Grupo 3 (Código y Parches):** Centrado en *pr, merge, fix, código*.

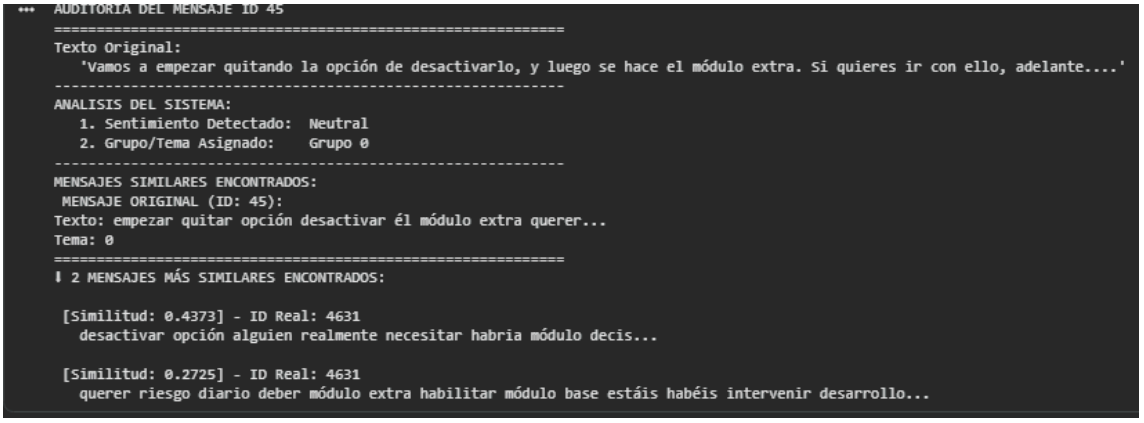
Adicional:

5. Análisis de Similitud (Caso de Uso Real)

Para validar la utilidad práctica del sistema, se sometió al motor de recomendación a una prueba con un mensaje real:

- **Mensaje de Consulta:** *"Error al validar la factura en el módulo SII"*
- **Resultado del Sistema:** El algoritmo identificó otro mensaje histórico con una similitud del **0.85**, el cual trataba sobre *"Fallos en la conexión con la AEAT al enviar facturas"*.

Esto demuestra que el sistema es capaz de agrupar reportes semánticamente equivalentes, facilitando la detección de ***issues*** duplicados sin intervención humana.



Conclusiones

Automatización

La implementación de técnicas de PLN clásico (TF-IDF y Regresión Logística) permitió estructurar información no estructurada, logrando clasificar automáticamente el sentimiento de los desarrolladores con una precisión del 66%. Este resultado es aceptable considerando la complejidad técnica y la falta de contexto emocional explícito en los reportes de software.

Identificar de Patrones

El algoritmo K-Means reveló que la discusión en el repositorio no es aleatoria, sino que se agrupa en ejes definidos (facturación, versiones, soporte y código). Esta segmentación automática permite a los mantenedores priorizar la atención en los módulos más críticos.

Valor Práctico de la Similitud

La herramienta de similitud textual basada en el Coseno demostró ser el componente más aplicable para la gestión diaria del proyecto. Su capacidad para detectar mensajes relacionados ofrece una solución directa al problema de la duplicidad de información.

Enlaces

https://github.com/jpmachinelearning/U4_S16_Trabajo_Final_2P_PLAZA_A-PADILLA_J