Mundo Fesc 12 (23) pp. 8-15, Enero-Junio 2022 ,ISSN 2216-0353, 2216-0388

Un Enfoque de Machine Learning para Apoyar la Identificación de la Deuda Técnica en Arquitectura

A Machine Learning Approach to Support the Identification of Architectural Technical Debt

[a].Boris RainieroPérez-Gutiérrez, [b].Dario Ernesto-Correal, [c]. Fredy Humberto Vera-riviera

a. Phd En Ingeniería, borisperezg@Ufps.edu.co, Universidad Francisco De Paula Santander, Cúcuta, Colombia.

b.Phd En Ingeniería, dcorreal@Uniandes.edu.co,Universidad De Los Andes, Bogotá Colombia

c.Phd En Ingeniería , fredyhumbertovera@Ufps.edu.co,Universidad Francisco De Paula Santander, Cúcuta Colombia.

Recibido: Julio 1 de 2021 Aceptado: Noviembre 8 de 2021

Forma de citar: B.R Pérez-Gutiérrez, D.E Correal, F.H Vera-Rivera "Un Enfoque de Machine Learning para Apoyar la Identificación de la Deuda Técnica en Arquitectura", *Mundo Fesc, vol. 12, no. S2, pp. 8-24, 2022*

Resumen

Antecedentes: Las decisiones tomadas por los arquitectos para favoreces objetivos a corto plazo y en posible detrimento de la calidad del software a largo plazo, se conoce como Deuda Técnica en Arquitectura. Este tipo de deuda técnica es difícil de identificar porque está relacionada con los atributos de calidad no visibles para el cliente, como la mantenibilidad y capacidad de evolucionar del sistema. Objetivo: Es por esto que, en este artículo, se un modelo supervisado de machine learning para apoyar la identificación de la deuda técnica en arquitectura que se ubique en la etapa de diseño de la arquitectura. Métodos: Esta propuesta se apoya en la información recogida de los artefactos producidos durante el diseño de la arquitectura para construir un dataset que permita evaluar diferentes algoritmos de aprendizaje supervisado y así establecer el que ofrezca mejor exactitud. La identificación de la deuda técnica, dentro del marco de esta propuesta y a diferencia de las propuestas en la literatura, no considera el código fuente. Resultados: El desempeño del modelo fue evaluado a través de un caso real de la industria y permitió descubrir que tanto la exactitud como el recall presentan niveles aceptables. Conclusiones: Los datos utilizados para entrenar el modelo, si bien son apropiados, son susceptibles de seguir mejorando. Este enfoque permitirá a los arquitectos apoyar la identificación de ATD conscientes e inconscientes inyectados en sus arquitecturas.

Palabras clave: arquitectura de solución, artefactos heterogéneos, deuda técnica en arquitectura, machine learning

Autor para correspondencia:

*Correo electrónico: **borisperez@ufpso.edu.co**

Abstract

Background: Decisions made by architects to favor short-term objectives and to the possible detriment of long-term software quality are known as Architectural Technical Debt. This type of technical debt is difficult to identify because it is related to quality attributes not visible to the customer, such as the maintainability and evolvability of the system. Objective: For this reason, in this paper, a supervised machine learning model is proposed to support the identification of technical debt in architecture that is located in the design stage of the architecture. Methods: This proposal relies on the information collected from the artifacts produced during the architecture design to build a dataset to evaluate different supervised learning algorithms and thus establish the one that offers the best accuracy. The identification of the technical debt, within the framework of this proposal and unlike those proposed in the literature, does not consider the source code. Results: The performance of the model was evaluated through a real industry case and allowed discovering that both accuracy and recall present acceptable levels. Conclusions: The data used to train the model, while appropriate, has room for further improvement. This approach will allow architects to support the identification of conscious and unconscious ATD injected into their architectures.

Keywords: heterogeneous artifacts, machine learning, technical debt in architecture, solution architecture

Introducción

Las empresas relacionadas con el desarrollo de software tienen una presión creciente para mejorar su eficacia en cada nuevo despliegue, reduciendo el tiempo o los recursos, y al mismo tiempo, entregar una solución de alta calidad capaz de mantenerse funcional a largo plazo [1]. Esto lleva a los equipos de software a tomar decisiones para lograr objetivos a corto plazo pero que posiblemente afecten negativamente a la mantenibilidad del sistema. Este tipo de decisiones de diseño se denomina Deuda Técnica (TD) [1].

La inyección intencionada de TD es común en los equipos de software porque puede ayudar a alcanzar los objetivos del proyecto antes o de forma más barata. Sin embargo, esta TD podría suponer riesgos para los proyectos si no se gestiona [2], tales como problemas financieros y técnicos relacionados con los costos de mantenimiento y evolución del software [3].

Según [4], las decisiones arquitectónicas son la fuente más importante de TD. Por lo tanto, es importante entender la TD a nivel arquitectónico. La deuda técnica arquitectónica (ATD) se produce principalmente por las decisiones de arquitectura que comprometen los atributos de calidad de todo el sistema, en particular la mantenibilidad y la capacidad de evolución. Algunos casos de ATD son violaciones de las mejores prácticas, la coherencia y las restricciones de integridad de las arquitecturas de software, como la presencia de violaciones estructurales y la introducción de smells de arquitectura [5], [1]. La ATD se considera un tipo de riesgo importante para un proyecto de software y, por lo tanto, es importante hacerlo visible para que las empresas de software reconozcan su impacto negativo en los proyectos y productos del ciclo de vida del software.

A pesar del impacto de las decisiones arquitecturales en el sistema desarrollado o por desarrollar, la identificación de los ATD sigue siendo

difícil de llevar a cabo. Algunas razones incluyen la falta de tiempo, la falta de iniciativas para manejarlo, la falta de herramientas eficaces para apoyar esta tarea, la falta de conocimiento de cómo realizar esta tarea, o la falta de estrategias para llegar a saber qué tipo de información debe ser recogida [6]. Se requiere establecer qué, dónde y cuándo se inyectaron elementos ATD [7].

Algunos enfoques para lidiar con la ATD se centran en actividades individuales dentro de un proceso global de gestión de los ATD [5], [1], o cubren todas las actividades de gestión de los ATD [5]. Sin embargo, estos enfoques se centran en el código fuente, lo que puede llevar a una importante reelaboración con el fin de pagar el ATD [8], o se basan en gran medida en las entrevistas con el equipo de arquitectura, lo que los vuelve subjetivos y propensos a errores.

En este artículo presentamos un modelo supervisado para apoyar a los arquitectos de solución a identificar la ATD inyectada en sus arquitecturas. La construcción del modelo implica la selección de los datos requeridos para el entrenamiento, así como la evaluación de múltiples algoritmos. El modelo hace uso de la información registrada dentro de un conjunto de artefactos producidos durante la etapa de diseño de la arquitectura de la solución para lograr su objetivo. Esta información se utiliza para establecer el tipo de ATD, la ubicación dentro de la arquitectura y el momento en que se inyectó.

La ATD puede encontrarse desde la arquitectura empresarial hasta la arquitectura de software. Esta investigación se centra en la arquitectura de soluciones porque es la responsable de representar las soluciones tecnológicas a problemas empresariales específicos. La arquitectura de soluciones se encuentra entre la arquitectura empresarial de alto nivel, orientada al negocio, y la arquitectura de software, más técnica y orientada al detalle. Por lo tanto, como parte del proceso de diseño del software, las primeras decisiones técnicas se toman en la arquitectura de soluciones.

Las técnicas de aprendizaje automático son eficaces para llevar a cabo tareas específicas sin depender de instrucciones o reglas explícitas. Por ejemplo, las técnicas de aprendizaje automático supervisado se han utilizado para construir modelos que pueden predecir el número de smells de arquitectura en futuras versiones del software [9], para construir un modelo de vocabulario contextualizado basado en los comentarios en el código [10], para modelar y predecir la evolución de la TD [11], y para identificar qué palabras están más relacionadas con la TD en los issue trackers [12]. Según [13], el estilo de aprendizaje dominante es el de los algoritmos de aprendizaje supervisado (89%), seguido del no supervisado (6%) y del semisupervisado (5%). Las redes bayesianas fueron el algoritmo más utilizado, seguido de los algoritmos de árboles de decisión (ID3, C4.5, CART). Basándonos en la mencionada aplicabilidad de las tecnologías de ML, creemos que existe la oportunidad de aplicar el ML en la identificación de la deuda técnica arquitectónica.

El modelo se evalúa a través de un análisis de casos de identificación de ATD de un proyecto de software real en una empresa de software en Cúcuta (Colombia). Los resultados obtenidos hasta el momento muestran que el modelo

es capaz de identificar ciertos tipos de ATD, además, funciona bien como punto de partida para la identificación de ATD y puede ser utilizado en otros proyectos de software.

El resto del artículo está organizado de la siguiente manera: La Sección 2 presenta la estrategia específica seguida para la selección de los datos que permitirán el entrenamiento del modelo, así como los pasos requeridos para realizar todo el preprocesamiento de estos. La Sección 3 presenta el caso práctico de evaluación utilizado y los resultados obtenidos de esta experimentación. De igual manera se presenta una discusión alrededor de los mismos en la que se incluyen los comentarios de la empresa participante. Finalmente, la Sección 4 cierra el artículo con las conclusiones alcanzadas luego de este trabajo, así como futuras líneas de investigación.

Materiales y métodos

El modelo presentado en este trabajo, así como cualquier modelo de predicción, requiere de un dataset que pueda ser utilizado como entrenamiento. Considerando el enfoque que se presenta para identificar la ATD, no fue posible encontrar datasets disponibles que sirvan para este propósito. A raíz de esto se decidió que el modelo utilizara información de casos de ATD que los mismos arquitectos conozcan y que puedan identificar y justificar. Esta información, unida a la información de los artefactos heterogéneos es la que se utilizará para realizar la construcción de un modelo preliminar de identificación de ATD [14].

Los artefactos heterogéneos considerados en este trabajo son: los modelos de arquitectura (junto con los cambios entre sus versiones), las decisiones de arquitectura, los correos, chats y commit logs. Estos artefactos fueron considerados partiendo del hecho de que son utilizados como medios de intercambio de información durante la etapa de diseño de la arquitectura.

El modelo construido con este dataset inicial servirá de punto de partida para identificar ATD en nuevas arquitecturas. A su vez, los casos de ATD identificados por el modelo pueden ser revisados y ajustados, y así mismo, utilizarlos para mejorar el modelo al incluir más información para su entrenamiento. Los casos de ATD identificados por el modelo se les nombra como “candidatos”.

La información utilizada para entrenar el modelo debe ser conforme con un meta-modelo [15] para poder realizar análisis y recorridos sobre la misma. Así pues, el artefacto debe estar construido en un formato particular, y la información extraída se modela conforme a un meta-modelo. Representar la información en meta-modelos es útil porque permite tener todo al mismo nivel de abstracción, y simplificar el cruce y búsqueda de información.

Los modelos de arquitectura requieren que estén diseñados en el lenguaje de modelado ArchiMate [16]. Estos modelos son analizados y comparados entre versiones para poder obtener los cambios realizados. Estos cambios son los que los arquitectos deben revisar para marcar como casos de ATD, cuando correspondan. Los chats deberán ser exportados desde Whatsapp. Los correos deberán respetar el formato EML. Los commit

logs deberán seguir el export hecho desde Git. Finalmente, las decisiones de arquitectura deberán estar registradas usando plantillas de Architectural Decision Record (ADR) [17]. La idea con los ADR es proveer un formato común para toda la información propia de una decisión de arquitectura, como lo es la motivación, las consecuencias, la decisión de arquitectura tomada, entre otros.

Toda la información extraída y modelada de los artefactos es luego requerida que se unifique en un meta-modelo que haga de pivote al contener toda la información. De esta manera, los datos requeridos para la construcción del dataset se pueden extraer en una consulta. La Figura 1 presenta una versión simplificada del meta-modelo pivote. En este meta-modelo se evidencia la representación de los artefactos heterogéneos, así como el e-class Fact, el cual corresponde a cada uno de los cambios sucedidos, para un elemento de arquitectura, de entre todas las versiones del modelo de arquitectura.

La generación del dataset de entrenamiento implica recorrer todo el meta-modelo (Figura 1) y recoger toda la información sobre los cambios, relaciones, información encontrada en los artefactos heterogéneos, etc. Esta información se utiliza para construir un conjunto de datos y luego se escribe como un archivo CSV. Entre la generación del dataset y su uso para entrenar el modelo, se realizan algunas operaciones de preprocesamiento que se detallarán más adelante.

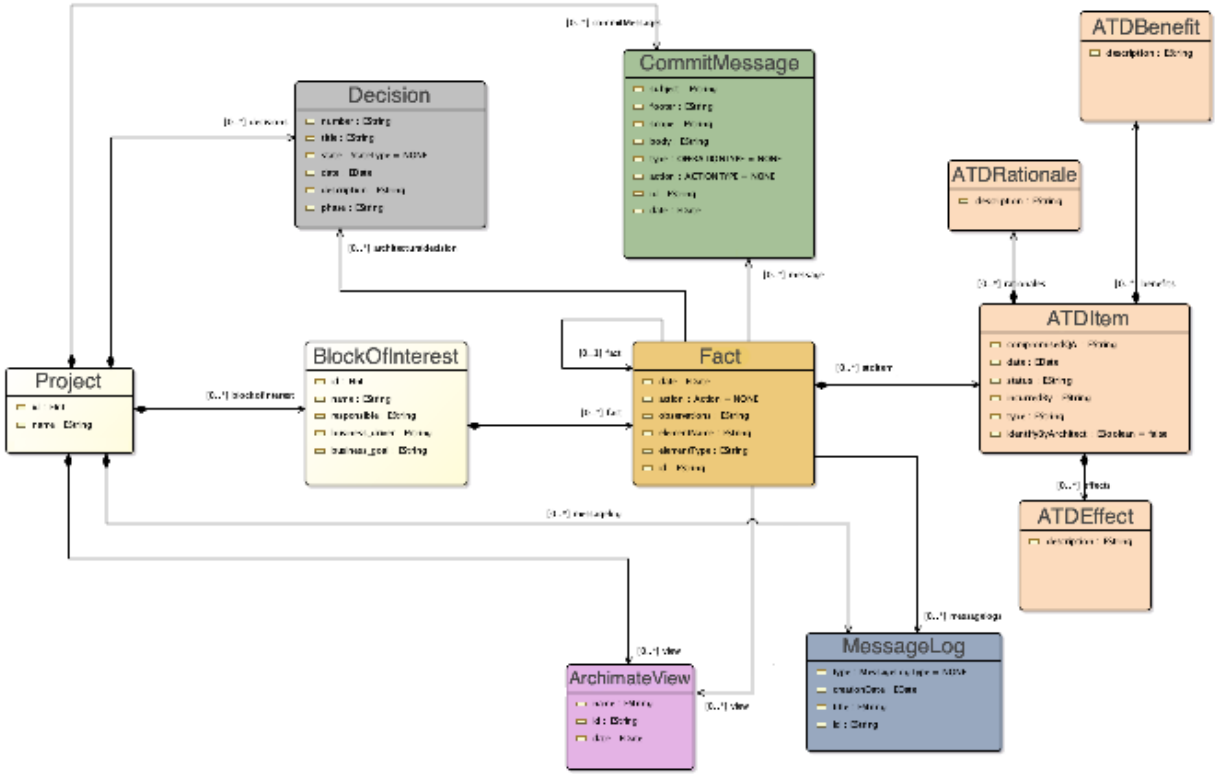


Figura 1. Meta-modelo pivote de representación de cambios arquitecturales.

El dataset de entrenamiento utilizado consta de 25 propiedades. A continuación, se describirán algunas de las más importantes:

- **sourceelementname.** Nombre del elemento de origen de una conexión.

- **sourceelementtype.** Tipo del elemento fuente de una conexión. Ejemplos de tipos son Componente de Aplicación, Servicio de Negocio, entre otros.

- **layersource.** Es la capa del elemento fuente. Algunos ejemplos son la capa de Aplicación y la capa de Negocio.

- **isnewelement.** Valor booleano para identificar si el elemento de destino es un elemento nuevo en el modelo revisado o si el elemento proviene de un modelo anterior. Para calcular este valor, lo primero es comprobar la fecha del modelo al que pertenece el elemento de origen. A continuación, se comprueba si el elemento de destino también fue creado en esta versión del modelo, o tal vez en un modelo más antiguo. Esto es importante para entender si el elemento de destino fue creado para conectarse con el elemento de origen.
- **iscyclic.** Valor booleano para identificar si existe una conexión inversa entre el elemento origen y el elemento destino. Si el Hecho es de tipo Relación entre el origen y el destino, entonces se comprueba si existe una relación entre el destino y el origen.
- **incoming.** Está relacionado con el número de conexiones entrantes del elemento fuente. Este campo busca comprender la relevancia del elemento fuente dentro del espacio completo de las conexiones.

- outgoing. Está relacionado con el número de conexiones salientes del elemento fuente.
- **commitlogs.** Este campo representa el cuerpo del texto de todos los registros de mensajes de commit en los que se ha encontrado una similitud con el nombre del elemento arquitectural. Todos los textos están concatenados.

- **adrlogs.** Este campo representa la descripción de una decisión arquitectónica.
- **property.** Nombre de una dimensión o propiedad de los conectores, de acuerdo con [18].

El dataset finaliza con las variables target u objetivo, que son:

- **atdcause.** Este campo representa al tipo de ATD identificado. Esta información la establece el arquitecto (usuario), sin embargo, en la literatura [8], [19], [20] se pueden encontrar algunos tipos como son: No uniformidad de políticas y patrones, architecture smells, dependencias complejas, tendencia al cambio, entre otros.
- **affectedqa.** Este campo representa el atributo de calidad afectado por la ATD inyectada en la arquitectura. Generalmente, los atributos de calidad afectados son la mantenibilidad y la capacidad de evolucionar del sistema.

El resultado del entrenamiento es un modelo de clasificación multiclase. La clasificación multiclase es una tarea de clasificación en la que cada muestra se asigna a una y sólo una etiqueta, pero

con una o más etiquetas para asignar. Por ejemplo, una imagen de fruta podría etiquetarse como naranja, manzana o plátano, pero la fruta puede ser una manzana o un plátano, pero no ambas cosas a la vez.

Como se mencionó anteriormente, el preprocesamiento es una etapa clave para que el entrenamiento del modelo sea adecuado. En la Figura 2 se presentan los pasos realizados como parte del preprocesamiento del dataset.

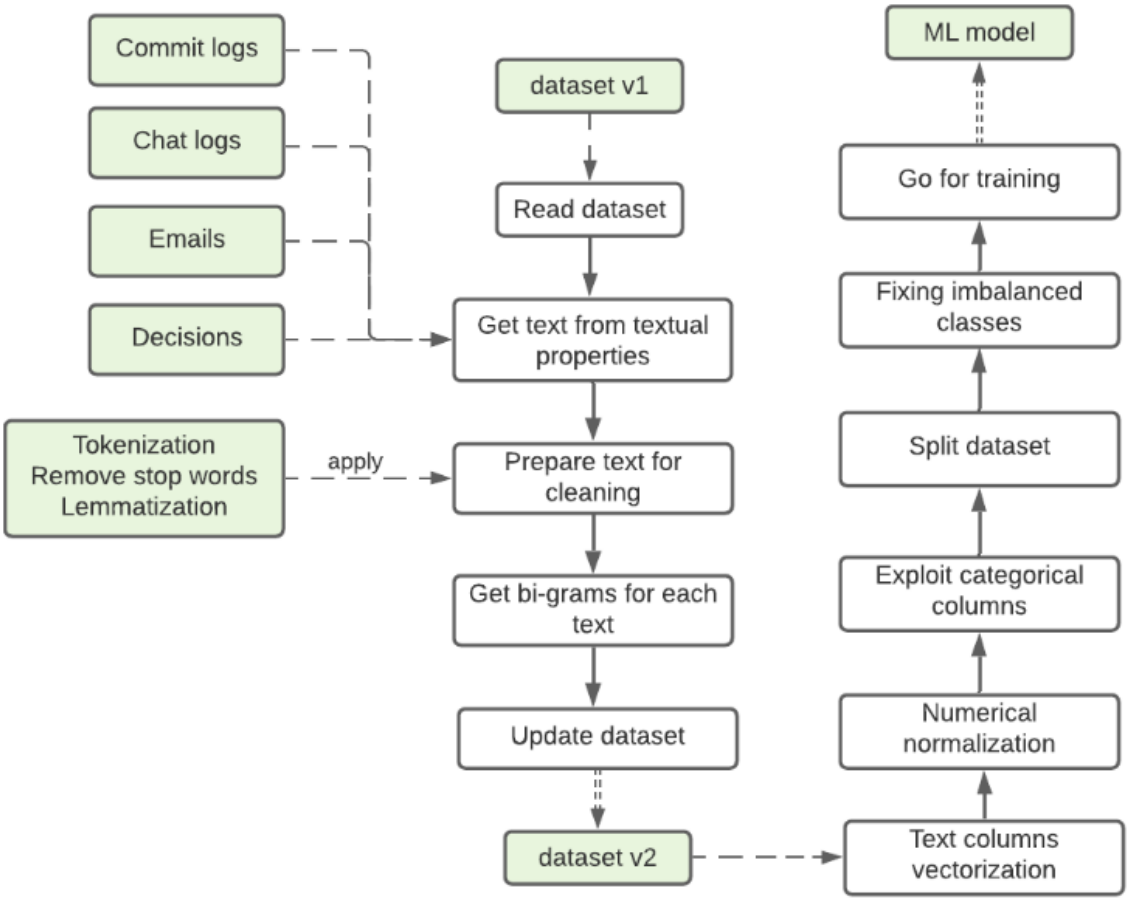
Como se puede apreciar, luego de obtener el dataset y de extraer el texto correspondiente de los artefactos heterogéneos, el siguiente paso consiste en limpiar el texto (Prepare text for cleaning). En este paso se realizan unas operaciones de tokenización (tokenization), que consiste en separar en tokens todo el texto. Un token puede ser una sílaba o puede ser una palabra. Para el caso de este trabajo, un token corresponde a una palabra. Luego de esto, la otra operación de limpieza consiste en remover las palabras que no aportan significado (remove stop words) como por ejemplo: el, para, con, contra, etc. Finalmente, la tercera operación de limpieza es la lematización (lemmatization) que consiste en llegar cada palabra a su lema, o palabra origen. El lema de una palabra es la palabra que podría encontrarse en un diccionario tradicional. Por ejemplo, decir es el lema de dije, pero también de *diré* o *dijéramos*; guapo es el lema de *guapas*. Paso seguido, se calculan lo bi-gramas como mecanismo de identificación del tópico o tema que corresponda al texto analizado.

El resultado de la ejecución de las operaciones de limpieza es un segundo dataset. Luego de esto, se llevan a cabo las siguientes operaciones de preprocesamiento:

- Los datos textuales se procesan aplicando la vectorización de las características del texto y se vectorizan utilizando TF-IDF. El texto se convierte en una matriz de características TF-IDF. Tf-idf significa frecuencia de términos-frecuencia de documentos inversa, y el peso tf-idf es un peso que se utiliza a menudo en la recuperación de información y la minería de textos. Este peso es una medida estadística utilizada para evaluar la importancia de una palabra en un documento de una colección o corpus [21]. Para ello se utiliza la clase TfidfVectorizer de sklearn.

- Los datos numéricos se procesan aplicando la normalización para escalar los datos numéricos. La normalización reescala los valores en un rango de (0,1). El objetivo de la normalización es cambiar los valores de las columnas numéricas del conjunto de datos a una escala común, sin distorsionar las diferencias en los rangos de valores.

- Los datos categóricos se procesan aplicando One Hot Encoding (OHE) para vectorizarlos. Esto es necesario porque la mayoría de los algoritmos esperan valores numéricos para lograr resultados óptimos. En OHE cada valor de categoría se convierte en una nueva columna y se le asigna un valor de 1 o 0 (notación para verdadero/falso). Esta práctica elimina los problemas de jerarquía/orden relacionados con la codificación de etiquetas.



Además de este procesamiento, hay otra cuestión que era necesario manejar: la desproporción de las muestras en cada clase. Esto se llama clases desequilibradas y es un problema común en la clasificación de aprendizaje automático, ya que la mayoría de los algoritmos de aprendizaje automático funcionan mejor teniendo el mismo o similar número de muestras en cada clase. De este modo, los algoritmos pueden maximizar la precisión y reducir el error. Hay al menos tres estrategias para tratar los datos desequilibrados: Sobremuestreo de la clase minoritaria, submuestreo de la clase mayoritaria y, generación de muestras sintéticas.

El sobremuestreo funciona añadiendo más copias de la clase o clases minoritarias. La biblioteca Scikit-Learn incluye un módulo de remuestreo para replicar aleatoriamente las muestras de la clase minoritaria. El submuestreo funciona eliminando algunas observaciones de la clase mayoritaria. Una desventaja del submuestreo es que se elimina información que puede ser valiosa. Por último, la generación de muestras sintéticas consiste en crear muestras sintéticas. Estas muestras se generan utilizando el algoritmo de vecinos más cercanos. La técnica SMOTE o Synthetic Minority Oversampling Technique se utiliza para generar datos nuevos y sintéticos.

En el contexto de esta modelo, se utilizan dos estrategias: El sobremuestreo y la generación de muestras sintéticas. El sobremuestreo se utiliza para aumentar el número de muestras de las clases

minoritarias, y luego se utiliza SMOTE para equilibrar el número de muestras de la clase dominante. Después de todo este procesamiento, el conjunto de datos está listo para entrenar el modelo. El paso siguiente consiste en dividir el dataset en conjunto de datos de entrenamiento y conjunto de datos de prueba. El conjunto de datos de prueba es el 20% del conjunto de datos original y el conjunto de datos de entrenamiento es el 80% del conjunto de datos original. Para dividir los datos y probar el modelo se utilizó StratifiedKFold, la cual es la versión mejorada de KFold, y garantiza que cada pliegue o fold del conjunto de datos tenga la misma proporción de observaciones con una etiqueta determinada.

Junto con el dataset, se utilizaron siete algoritmos ML para construir y probar la exactitud del modelo, y este valor fue el utilizado para construir el modelo que se utilizará para la identificación de ATD. Los siete algoritmos son: Logistic Regression, Linear Discriminant Analysis, K-nearest Neighbors classifier, Decision Tree classifier, Gaussian Naive Bayes, Naive Bayes classifier for multinomial models y C-Support Vector Classification.

Una vez se tiene el modelo construido, se inició su validación con un caso real de la industria. El caso y los resultados son presentados en la Sección 3.

Resultados y discusión

GN (anonimizado por razones de privacidad) es una empresa que apoya a las instituciones de Educación Superior para generar una transformación digital dentro de las instituciones. Esta empresa enfocada en la Transformación Digital que asesora al sector público y privado en Colombia en la introducción efectiva de las Tecnologías de la Información y la Comunicación. Esta empresa está ubicada en Cúcuta, Colombia.

Como parte de la experimentación, se utilizó un proceso de negocio focalizado en la generación de un reporte de calificaciones. Este proceso es crítico porque tiene alta demanda, siendo la disponibilidad y el desempeño, de los principales atributos de calidad a enfrentar.

Para la experimentación, se utilizaron tres versiones de una arquitectura de solución propuesta. Además, para la experimentación se utilizó un modelo previamente entrenado con información de casos de ATD definidos en proyectos anteriores. Algunos de los casos de ATD utilizados para el entrenamiento consistieron en casos de no uniformidad de patrones y políticas, architecture smells y dependencias complejas.

Puede darse el caso de que hayas casos donde el caso marcado como ATD en realidad no corresponda a ATD. Por ello, siempre es requerido que el arquitecto revise los resultados y pueda ofrecer una retroalimentación. Esta acción es importante porque permitirá que la información corregida pueda ser luego utilizado para entrenar futuros modelos. Entre más datos se usen para el entrenamiento, mejor será la exactitud.

A continuación, se presentarán los resultados de la utilización del modelo ML supervisado para identificar los candidatos de ATD en el proyecto de la empresa GN. El modelo ML supervisado identificó 14 elementos candidatos de ATD, pero no todos estos candidatos fueron identificados correctamente. Los

resultados obtenidos por el modelo se presentan a continuación:

- Verdaderos positivos (TP - true positives): 8
- Falsos negativos (FN - false negatives): 1
- Falsos positivos (FP - false positives): 6
- Verdaderos negativos (TN - true negatives): 44

Estos cuatro parámetros (TP, NF, FP y TN) se utilizan para evaluar el rendimiento del modelo ML supervisado las siguientes cuatro métricas de evaluación: Exactitud (accuracy), Precisión (precision), Exhaustividad (recall) y Puntuación F1 (F1 score).

La Exactitud es la proporción de observaciones predichas correctamente sobre el número total de observaciones. La Precisión es la relación entre las observaciones positivas predichas correctamente y la cantidad de predicciones correctas e incorrectas. La Exhaustividad es la relación entre las observaciones positivas predichas correctamente y la cantidad de observaciones positivas. Finalmente, la puntuación F1 es la media ponderada de la precisión y la recuperación.

Los resultados de estas métricas se presentan a continuación:

- Exactitud: 0,881
- Precisión: 0,571
- Exhaustividad: 0,889
- Calificación F1: 0,696

Discusión de los resultados

Como se pudo evidenciar, se calcularon las cuatro métricas, sin embargo, como se presenta a continuación, no todas ellas son necesarias para la evaluación del modelo ML supervisado de este trabajo. En primer lugar, es importante mencionar que los falsos negativos son el parámetro más importante, y por lo tanto, para este trabajo, el costo asociado a los falsos negativos va a ser alto. Por lo tanto, la Exhaustividad es más útil que la Precisión.

En esta propuesta, un falso negativo es cuando hay una situación que está inyectando deuda, pero el modelo ML supervisado no lo reconoce o identifica como deuda. En este caso, el costo asociado al falso negativo será alto porque estará relacionado con futuros problemas de mantenibilidad. Por lo tanto, la métrica más relevante a considerar es la Exhaustividad. Para nuestro modelo, la medida de esta métrica es de 0,889, lo cual es bueno considerando que está por encima de 0,5.

En segundo lugar, la distribución de clases del dataset utilizado para el entrenamiento estuvo equilibrado. Por lo tanto, la Exactitud podría ser más útil que la F1 Score. Sin embargo, la F1 Score se utiliza cuando los falsos negativos y los falsos positivos son cruciales.

Para nuestro modelo, obtuvimos una exactitud de 0,881, lo que significa que nuestro modelo es exacto en aproximadamente un 88%. Para calcular la F1 es necesario calcular primero la Precisión. En nuestro modelo, la medida de la Precisión es de 0,571, que es algo baja, pero en esta propuesta, el coste asociado a los Falsos Positivos no es necesariamente alto en relación con el ATD. Un caso incorrectamente predicho como ATD podría implicar una pérdida de

tiempo al analizarlo, pero no implicaría futuros problemas de mantenibilidad.

La medida de la puntuación F1 es de 0,696, lo que significa que el modelo tiene pocos falsos positivos y pocos falsos negativos, por lo que está identificando correctamente las amenazas reales y no se ve perturbado por las falsas alarmas.

Conclusiones

La ATD es un aspecto importante que debe ser considerado en el diseño de la arquitectura, pero que actualmente, rara vez se aborda. El modelo presentado en este trabajo propone una estrategia para identificar ATD a partir del conocimiento previo de casos conocidos de ATD por parte de los arquitectos, y de la información recopilada en los diagramas de arquitectura y resto de artefactos heterogéneos.

Este modelo se centra en la información generada durante la etapa de diseño de la arquitectura y proporciona la capacidad de apoyar la identificación automática de los tipos de ATD que los enfoques de identificación de ATD existentes, principalmente basados en el análisis del código fuente, no pueden soportar. Este enfoque se basa en el conocimiento arquitectónico que pueda documentarse, la cual podrá requerir cierto esfuerzo. La calidad de los datos extraídos y la descripción de la arquitectura y los fundamentos dependen en gran medida de la calidad de los artefactos disponibles.

Es importante mantener la ATD explícita y visible, de esta manera, podrá formar parte del proceso de toma de decisiones de la arquitectura considerando su impacto actual y su impacto sobre otras decisiones de la arquitectura. Por otro lado, los elementos de ATD no identificados seguirán acumulando interés, lo que supondrá un coste prohibitivo en el mantenimiento y la evolución del sistema.

El análisis de la ATD nos lleva a algunas conclusiones importantes: i) es necesario tener en cuenta más artefactos para aumentar la comprensión de la inyección de ATD, como los requisitos funcionales, los escenarios de atributos de calidad o incluso un catálogo de olores de arquitectura; ii) la identificación de la ATD tiene una estrecha relación con el conocimiento de la arquitectura. En la medida en que el conocimiento puede ser estandarizado, la identificación también puede ser estandarizada; y iii) un enfoque totalmente automatizado para la identificación de los ATD es todavía difícil de conseguir. Es necesario incluir más datos, tanto generales para todas las arquitecturas (independientes del dominio) como específicos del dominio.

Además, este modelo aún requiere algunas mejoras para poder ser utilizado plenamente por la industria del software. En el futuro, tenemos previsto mejorar nuestro trabajo actual ampliándolo en varias direcciones: i) incluir más estudios empíricos para validar nuestro modelo de identificación de ATD. Se necesitan más proyectos industriales de diferentes tamaños y de varios dominios; ii) incluir un mapeo entre elementos arquitectónicos y elementos de código. Esta mejora sería útil para proyectos ágiles en los que la arquitectura podría cambiar en cada sprint, y se entrega algo de código; y iii) incluir modelos de arquitectura representados en UML. Esta mejora proporcionará una mejor relación entre los objetivos empresariales y las arquitecturas de solución específicas

utilizadas para lograrlos. Esta inclusión podría comenzar con los diagramas de Componentes y Conectores y de Despliegue. Esta mejora apoyará a los arquitectos de solución no familiarizados con el lenguaje de modelado ArchiMate.

Referencias

[1] A. Martini, E. Sikander y N. Madlani, "A semi-automated framework for the identification and estimation of Architectural Technical Debt", Information and Software Technology, vol. 93, pp. 264–279, January 2018. https://doi.org/10.1016/j.infsof.2017.08.005.

[2] N. Rios, R. Spínola, M. Mendonça y C. Seaman, "The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil", Empirical Software Engineering, vol. 25, ed. 5, pp. 3216-3287, September 2020. https://doi.org/10.1007/s10664-020-09832-9.

[3] A. Martini, J. Bosch y M. Chaudron, "Architecture Technical Debt: Understanding Causes and a Qualitative Model", in Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 85-92, 2014. https://doi.org/10.1109/SEAA.2014.65.

[4] N. Ernst, S. Bellomo, I. Ozkaya, R. Nord y I. Gorton, "Measure it? Manage it? Ignore it? software practitioners and technical debt", in Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015), pp. 50-60, 2015. https://doi.org/10.1145/2786805.2786848.

[5] Z. Li, P. Liang y P. Avgeriou, "Architectural Technical Debt Identification Based on Architecture Decisions and Change Scenarios", in Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture, pp. 65-74, 2015. http://doi.org/10.1109/WICSA.2015.19.

[6] A. Martini, T. Besker y J. Bosch, "The Introduction of Technical Debt Tracking in Large Companies", in Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC), pp. 161-168, 2016. http://doi.org/10.1109/APSEC.2016.032.

[7] R. L. Nord, I. Ozkaya, P. Kruchten and M. Gonzalez-Rojas, "In Search of a Metric for Managing Architectural Technical Debt," in Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, pp. 91-100, 2012 http://doi.org/10.1109/WICSA-ECSA.212.17.

[8] Zengyang Li, Paris Avgeriou y Peng Liang, "A systematic mapping study on technical debt and its management", Journal of Systems and Software, vol. 101, pp. 193-220, 2015. https://doi.org/10.1016/j.jss.2014.12.027.

[9] J. A. Díaz-Pace, A. Tommasel y D. Godoy, "Towards Anticipation of Architectural Smells Using Link Prediction Techniques", in Proceedings of the IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 62-71, 2018. http://goi.org/10.1109/SCAM.2018.00015.

[10] M. A. de Freitas Farias, M. G. de

Mendonça Neto, A. B. d. Silva y R. O. Spínola, "A Contextualized Vocabulary Model for identifying technical debt on code comments", in Proceedings of the IEEE 7th International Workshop on Managing Technical Debt (MTD), pp. 25-32, 2015. http://doi.org/10.1109/MTD.2015.7332621.

[11] D. Tsoukalas, D. Kehagias, M. Siavvas y A. Chatzigeorgiou, "Technical debt forecasting: An empirical study on open-source repositories", Journal of Systems and Software, vol. 170, pp. 1-35, 2020. https://doi.org/10.1016/j.jss.2020.110777

[12] K. Dai y P. Kruchten, "Detecting Technical Debt through Issue Trackers", in Proceedings of the 5th International Workshop on Quantitative Approaches to Software Quality, pp. 59-65, 2017. https://dx.doi.org/10.14288/1.0374920.

[13] A. Tsintzira, E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, "Applying Machine Learning in Technical Debt Management: Future Opportunities and Challenges", Communications in Computer and Information Science, vol. 1266, pp. 53-67, 2020. https://doi.org/10.1007/978-3-030-58793-2_5

[14] J. Musil et al., "Continuous Architectural Knowledge Integration: Making Heterogeneous Architectural Knowledge Available in Large-Scale Organizations", in Proceedings of the IEEE International Conference on Software Architecture (ICSA), pp. 189-192, 2017. https://doi.org/10.1109/ICSA.2017.28

[15] C. Atkinson y T. Kuhne, "Model-driven development: a metamodeling foundation", IEEE Software, vol. 20, no. 5, pp. 36-41, September 2003. https://doi.org/10.1109/MS.2003.1231149.

[16] H. Jonkers, E. Proper, M. M. Lankhorst, D. A. C. Quartel y M. Iacob, "ArchiMate(R) for Integrated Modelling Throughout the Architecture Development and Implementation Cycle", in Proceedings of the IEEE 13th Conference on Commerce and Enterprise Computing, pp. 294-301, 2011. https://doi.org/10.1109/CEC.2011.52.

[17] M. Nygard, "Documenting Architecture Decisions", November 2011. [Online]. Disponible en: https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions.html

[18] R. N. Taylor, N. Medvidovic y E. M. Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley India Pvt. Limited. 2010.

[19] R. Verdecchia, I. Malavolta y P. Lago, "Architectural Technical Debt Identification: The Research Landscape", in Proceedings of the IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 11-20, 2018. https://doi.org/10.1145/3194164.3194176

[20] T. Besker, A. Martini y J. Bosch, "Managing architectural technical debt: A unified model and systematic literature review", Journal of Systems and Software, vol. 135, pp. 1-16, 2018. https://doi.org/10.1016/j.jss.2017.09.025

[21] H. C. Wu, R. W. Pong Luk, K. F. Wong

y K. L. Kwok, “Interpreting TF-IDF term weights as making relevance decisions”, ACM Transactions on Information Systems, vol. 26, pp. 1-37, June 2008. https://doi.org/10.1145/1361684.1361686