IV Jornadas de Calidad de Software y Agilidad

JCSA 2021

12 y 13 de noviembre de 2021

IV JORNADAS DE CALIDAD DE SOFTWARE Y AGILIDAD

IV

LIBRO DE ACTAS

UNIVERSIDAD NACIONAL DE MISIONES

UTN

Libro de Actas de las Cuartas Jornadas de Calidad de Software y Agilidad / Gladys Noemí Dapozo, Emanuel Irrazábal, María de los Ángeles Ferraro, Horacio D. Kuna, Eduardo Zamudio, Alice Rambo, César Acuña, Verónica Bollati y Noelia Pinto ; compilación de Gladys Noemí Dapozo ; Emanuel Agustín Irrazábal. - 1a ed compendiada. - Corrientes : Universidad Nacional del Nordeste. Facultad de Ciencias Exactas, 2021. Libro digital, PDF

Editorial de la Universidad Nacional del Nordeste (EUDENE)

ISBN 978-987-3619-72-4

1. Software. 2. Jornadas. 3. Argentina. I. Dapozo, Gladys Noemí, comp. II. Irrazábal, Emanuel Agustín, comp.

CDD 004.0711

Fecha de catalogación: 03/01/2022

Autoridades

Universidad Nacional del Nordeste

Rectora: Prof. María Delfina Veiravé
Vicerrector: Dr. Mario H. Urbani

Facultad de Ciencias Exactas y Naturales y Agrimensura

Decana: Mgter. María Viviana Godoy Guglielmone
Vicedecano: Dr. Enrique Laffont

Universidad Nacional de Misiones

Rectora: Mgter. Alicia Violeta Bohren
Vicerrector: Ing. Fernando Luis Kramer

Facultad de Ciencias Exactas, Químicas y Naturales

Decano: Dr. Luis Brumobsky
Vicedecano: Dr. Marcelo Marinelli

Universidad Tecnológica Nacional

Rector: Ing. Héctor Aiassa
Vicerrector: Ing. Haroldo Avetta

Facultad Regional Resistencia

Decano: Jorge A. De Pedro
Vicedecano: Dr. Ing. Walter Gustavo Morales

IV Jornadas de Calidad de Software y Agilidad

Este evento es organizado en forma conjunta por las universidades de la región que tienen unidades académicas que ofrecen carreras de Informática, UNNE, UTN y UNaM, a través de un acuerdo específico institucional, con el objetivo de difundir avances significativos en el campo de conocimiento de la Ingeniería de Software, Calidad y Agilidad; y propiciar el encuentro entre las universidades, las empresas y los organismos del Estado para contribuir al desarrollo de la industria del software en la región de influencia de las universidades participantes. Las Jornadas de Calidad de Software y Agilidad (JCSA) se inician en el año 2017, bajo el nombre original de Jornadas de Calidad de Software, en esta edición, las jornadas se consolidaron como un foro regional de referencia, ampliando su espectro para enfatizar también temas relacionados al uso de la agilidad. Además, se propuso la publicación de los trabajos académicos presentados, previa evaluación de pares, que se incorporan en este libro de actas. Durante el evento se realizaron tres talleres vinculados con los temas de la Jornada, se expusieron los artículos y posters académicos aceptados y se presentaron experiencias de la industria del software, con la participación de empresas y organismos del Estado. La conferencia inaugural denominada "Estrategias pruebas de aceptación para Entrega Continua", estuvo a cargo de Diego Fontdevilla (UNTREF). Las actividades propuestas estuvieron destinadas a ingenieros y licenciados en sistemas, estudiantes y docentes de estas especialidades, profesionales y empresarios del sector del software y servicios informáticos, así como también público interesado en la temática. En total participaron más de 200 personas en las actividades programadas, lo que evidencia el interés que estos temas suscitan en los destinatarios.

Comité Organizador

Mgter. Gladys Noemí Dapozo
(FaCENA - UNNE)

Dr. Emanuel Irrazábal
(FaCENA - UNNE)

Lic. María de los Ángeles Ferraro
(FaCENA - UNNE)

Dr. Horacio D. Kuna
(FQCEyN - UNaM)

Dr. Eduardo Zamudio
(FQCEyN - UNaM)

Ing. Alice Rambo
(FQCEyN - UNaM)

Dr. César Acuña
(FRRe - UTN)

Dra. Verónica Bollati
(FRRe - UTN)

Dra. Noelia Pinto
(FRRe - UTN)

Comité de Programa

Mgter. Cristina Greiner
(GICS - FaCENA - UNNE)

Mgter. Gladys Noemí Dapozo
(GICS - FaCENA - UNNE)

Dr. Emanuel Irrazábal
(GICS - FaCENA - UNNE)

Dr. Rubén Bernal
(GICS - FaCENA - UNNE)

Dr. David la Red Martínez
(FaCENA - UNNE)

Dra. Sonia Mariño
(FaCENA - UNNE)

Mgter. M. Viviana Godoy Guglielmone
(FaCENA - UNNE)

Mgter. Mónica Tugnarelli
(FCAD - UNER)

Dra. Gabriela Arévalo
(DCyT - UNQ)

Dra. María Fernanda Golobisky
(UTN - FRStaFe)

Master Ariel Pasini
(LIDI - UNLP)

Mgter. Pablo Thomas
(LIDI - UNLP)

Dr. Fernando Emmanuel Frati
(DCByT - UNdeC)

Dra. Marcela Genero Bocco
(Grupo Alarcos - UCLM)

Dr. Jorge Andrés Diaz Pace
(ISISTAN - CONICET)

Dr. Nazareno Aguirre
(FCEFQyN - UNRC / CONICET)

Dra. Nancy Ganz
(IIDII-FCEQyN - UNaM)

Esp.Ing.Alice Rambo
(IIDII-FCEQyN - UNaM)

Ing. Selva Nieves Ivaniszyn
(FCEQyN - UNaM)

Lic. Sergio Caballero
(FCEQyN - UNaM)

Lic. Martín Rey
(IIDII-FCEQyN - UNaM)

Dr.Eduardo Zamudio
(IIDII - FCEQyN - UNaM)

Dr.Horacio Kuna
(IIDII - FCEQyN - UNaM)

Dr. Diego Godoy
(UGD)

Ing. Edgardo Belloni
(UGD)

Mag. Liliana Cuenca Pletsch
(CINApTIC - UTN - FRRe)

Dr. César J. Acuña
(CINApTIC - UTN - FRRe)

Dra. Verónica Bollati
(CINApTIC - UTN - FRRe / CONICET)

Dra. Noelia Pinto
(CINApTIC - UTN - FRRe)

Esp. Gabriela Tomaselli
(CINApTIC - UTN - FRRe)

Ing. Nicolas Tortosa
(CINApTIC - UTN - FRRe)

Ing. Valeria Sandobal Verón
(GIESIN- UTN - FRRe)

Ing. Germán Gaona
(CINApTIC - UTN - FRRe)

Dr. Horacio Leone
(INGAR - UTN - FRSF)

Dr. Silvio Gonnet
(INGAR - UTN - FRSF)

Dr. Gustavo Rossi
(LIFIA - UNLP)

Dra. Alejandra Garrido
(LIFIA - UNLP)

Dr. Andrés Rodriguez
(LIFIA - UNLP)

Dr. Marcelo Estayno
(UNSAM)

Dr. Luis Olsina
(GIDIS - UNLPam)

Dra. Luciana Ballejos
(CIDISI - UTN - FRSF)

Dra. Luciana Roldan
(INGAR - UTN - FRSF)

Dra. Milagros Gutierrez
(UTN - FRSF)

Dra. Mariel Alejandra Ale
(CIDISI - UTN - FRSF)

Dra. Elsa Estevez
(UNSur / CONICET)

Dra. Alicia Mon
(UNLaM)

Evaluación de conjuntos de datos utilizados en la construcción de modelos para la predicción de defectos en clases de proyectos software

Juan Andrés Carruthers, Celeste Ojeda Rodríguez

Grupo de Investigación en Calidad de Software, Facultad de Ciencias Exactas y Naturales y Agrimensura, Universidad Nacional del Nordeste
jacarruthers@exa.unne.edu.ar
malu221098@gmail.com

Resumen. La predicción de defectos consiste principalmente en la identificación de componentes de software con mayor probabilidad de errores para una asignación de recursos efectiva. Esto se realiza por medio de modelos predictivos que son entrenados con datos de los proyectos, como información del proceso de desarrollo o el producto software. En este estudio, se presentan tres clasificadores entrenados con tres conjuntos de datos conformados por valores de métricas orientadas a objetos obtenidas del análisis estático del código de cinco sistemas Java. Para el entrenamiento de cada modelo se utiliza el método de ensamble validación y votación configurado con los algoritmos Regresión Logística, Función de Base Radial, Árbol de Decisión, Perceptrón Multicapa, Naïve Bayes y Máquinas de Vector Soporte. Los modelos predictivos registran resultados favorables, con valores mayores a 0.8 de exactitud y mayores a 0.87 de especificidad.

Palabras Clave: Predicción de defectos, Métricas orientadas a objetos, Mantenimiento del software

1 Introducción

El mantenimiento del software dedica una importante cantidad de recursos dentro del proceso de desarrollo, siendo en algunos casos una de las etapas que más presupuesto requiere [1]. La reparación de defectos es una actividad central en el mantenimiento del software, pero es necesario identificarlos previamente. De esta manera surgen los modelos de predicción de defectos, donde el escenario principal es la asignación de recursos a los componentes de software que requieren ser revisados [2]. El tiempo y la mano de obra son recursos finitos, por lo que tiene sentido asignar personal y recursos a áreas de un sistema de software con una mayor cantidad probable de errores.

La predicción de defectos ha generado un interés generalizado durante un período de tiempo considerable y numerosos artículos han reportado diferentes enfoques implementados. Desde la utilización de métricas de cambio en la historia del proyecto en [2], la información de defectos previos en [3] o el contexto del proceso de desarrollo en [4].

Otro enfoque para la predicción de defectos asume que el diseño y el comportamiento actuales del programa influyen en la presencia de defectos futuros. Estos enfoques no requieren la historia del sistema, sino que analizan su estado actual con más detalle, utilizando una variedad de métricas como el caso de [3], [5] y [6]. De esta manera, se busca responder a la siguiente pregunta de investigación:

RQ: ¿Qué conjunto de métricas orientadas a objetos obtenidas de una versión del sistema tienen mejor rendimiento para predecir defectos en las clases de un proyecto software?

Para ello se proponen tres modelos clasificadores con el último enfoque mencionado. Se emplean métricas obtenidas del análisis estático del código fuente de Chidamber y Kemerer (CK) [7] y un conjunto de métricas orientadas a objetos (OO) para entrenar los clasificadores e información defectos para la creación de las etiquetas. Un clasificador es entrenado solamente con las métricas de CK, el segundo con las métricas OO y el tercero con una combinación de ambos conjuntos de datos (CK + OO). La intención es determinar los predictores más efectivos para la detección de clases propensas a tener defectos.

El resto del artículo se encuentra organizado de la siguiente manera. En la sección 2, están los trabajos relacionados. En la sección 3 y 4 se presentan la descripción detallada del diseño del experimento y los resultados obtenidos respectivamente. Finalmente, en la sección 5 se encuentran las discusiones y conclusiones del trabajo desarrollado.

2 Trabajos relacionados

La predicción de defectos en artefactos software ha sido estudiado por una gran cantidad de autores a lo largo del tiempo. En la literatura referente se pueden encontrar diferentes enfoques y fuentes de datos empleados para estos fines. Por ejemplo, Zimmermann et al. [3] proponen un modelo de regresión logística con la información de defectos, métricas de complejidad y cantidad de estructuras específicas halladas en el código de tres versiones de lanzamiento de Eclipse a nivel de archivos y paquetes.

D'Ambros y Robbes [2] evalúan seis enfoques en cinco sistemas de fuente libre, para ello reunieron medidas de cambios de la historia del proyecto; la entropía de estos cambios; información de defectos previos; métricas del código fuente de una versión; múltiples versiones; y la entropía entre estas versiones. A diferencia del trabajo de Zimmermann et al., D'Ambros y Robbes analizan los sistemas a nivel de clases en vez de paquetes dado que estas son unidades más pequeñas y por ende su revisión requiere menor trabajo. Además, en el caso que fuera estrictamente necesario generar los valores a nivel de paquete es posible recrearlos desde las clases que integran cada paquete. Jureczko y Madeyski [8] también usan valores de 19 métricas de múltiples versiones, reportadas como indicadores de buena calidad en otros estudios.

Por su parte Zhang et al. [4] consideran el contexto de desarrollo del sistema para la predicción de defectos teniendo en cuenta factores tales como el lenguaje de programación, el uso de un sistema de rastreo de defectos, cantidad totales de líneas de código,

de archivos, de commits y de desarrolladores. Ghotra et al. [5] entrenan modelos clasificadores con el conjunto de datos NASA [9] utilizando distintos algoritmos tales como Naïve Bayes, Árboles de Decisión, Función de Base Radial, Máquinas de Vector Soporte, entre otros. El conjunto de datos NASA contiene valores de 40 métricas de código fuente de una versión.

Antecedentes más actuales, como el de Pecorelli y Nucci [6], también emplean el conjunto de datos publicado por Jureczko y Madeyski [8] para evaluar el desempeño de clasificadores entrenados con seis técnicas de ensamble: bagging, boosting, validación y votación, selección adaptativa de clasificadores, COmbined DEfect Predictor (CODEP) [10] y bosque aleatorio. Qiao et al. [11] implementan y evalúan un enfoque de aprendizaje profundo para la predicción de defectos con un conjunto de datos compuesto por métricas de complejidad y tamaño del código fuente.

3 Experimento

A continuación, se describen el conjunto de datos, los métodos, las técnicas, las herramientas y las métricas de desempeño para desarrollar y evaluar el experimento propuesto.

3.1 Conjunto de datos

El conjunto de datos utilizado fue obtenido de [12], publicado inicialmente en [2]. Es un repositorio público usado con fines científicos en varios estudios [13 - 15]. El mismo contiene información evolutiva, de una versión y de reportes de defectos de las clases de cinco sistemas Java de fuente abierta. Eclipse JDT Core es el conjunto de herramientas núcleo del entorno de desarrollo integrado Eclipse para la construcción de software. Eclipse PDE UI, también es un conjunto de herramientas de desarrollo, pero para la creación de extensiones de Eclipse. Equinox Framework es un marco de trabajo que implementa el estándar OSGi. Lucene es una biblioteca específica para la indexación y búsqueda de cadenas de texto. Mylyn es un marco de trabajo para la gestión del ciclo de vida del sistema para Eclipse.

Se emplearon los valores de métricas estructurales de CK, OO y cantidad de defectos de una sola versión a nivel de clases de los sistemas anteriormente mencionados para entrenar los modelos. Los defectos provienen de los registros de los sistemas de rastreo de defectos Bugzilla y Jira de los sistemas anteriormente mencionados. La Tabla 1 muestra con un mayor nivel de detalle estas métricas.

4

Tabla 1. Descripción de las métricas. (Fuente [2]).

Tipo	Sigla de la métrica en inglés	Descripción
CK	CBO	Acoplamiento entre objetos
	DIT	Profundidad de árbol de herencia
	LCOM	Falta de cohesión en métodos
	NOC	Cantidad de hijos
	RFC	Respuesta por clase
	WMC	Peso de métodos por clase
OO	FanIn	Cantidad de clases diferentes que hacen referencia a la clase
	FanOut	Cantidad de clases diferentes referenciadas por la clase
	NOA	Cantidad de atributos
	NOAI	Cantidad de atributos heredados
	LOC	Cantidad de líneas de código
	NOM	Cantidad de métodos
	NOMI	Cantidad de métodos heredados
	NOPRA	Cantidad de atributos privados
	NOPRM	Cantidad de métodos privados
	NOPA	Cantidad de atributos públicos
	NOPM	Cantidad de métodos públicos

El conjunto de datos fue dividido en dos partes. Por un lado, las métricas de CK y por otro las demás métricas OO. Con ellos se entrenaron tres clasificadores. El primero utilizó los datos de CK, el segundo los demás datos de métricas OO y el tercero la combinación de estos dos (CK + OO). En base a la cantidad de defectos por clase se crearon las etiquetas con los valores 0 y 1 para los clasificadores binarios. Si el número de defectos en una clase era mayor a 0 se asignó un 1, y en el caso contrario un 0.

3.2 Datos desbalanceados

Generalmente la predicción de defectos es un problema desbalanceado, porque normalmente el número de clases defectuosas en un sistema es mucho menor que las no defectuosas. Este conjunto de datos no es la excepción, siendo que las clases defectuosas representa menos del 16% del total. Dado que esto puede sesgar el rendimiento del clasificador del modelo predictor [16] se aplicó la técnica de muestreo Synthetic Minority Over-sampling TEchnique (SMOTE) [17] para asegurar que los conjuntos de datos de entrenamiento tengan una proporción similar de clases defectuosas que no defectuosas. Se utilizó la implementación de SMOTE de la biblioteca de Python imbalanced-learn [18].

3.3 Algoritmos y herramientas

Siguiendo las recomendaciones de [6], se entrenaron los modelos con el algoritmo de ensamble validación y votación. Validación y votación [19] (también denominada votación) combina las puntuaciones de confianza obtenidas por los clasificadores subyacentes. Para cada instancia a predecir, cada clasificador devuelve una puntuación de confianza que varía entre 0 y 1. Un operador combina las puntuaciones (por ejemplo, promedio o máximo). Una clase se marca con defectos si la combinación de las puntuaciones de confianza es superior a 0.5, mientras que, en caso contrario, se predice como sin defectos. El método de votación fue configurado usando los siguientes clasificadores: Regresión Logística, Función de Base Radial, Árbol de Decisión, Perceptrón Multicapa, Naïve Bayes y Máquinas de Vector Soporte.

Tanto el método de votación como los demás clasificadores fueron implementados con la biblioteca de Python Scikit-learn [20].

3.4 Métricas de evaluación

Para evaluar el rendimiento de los modelos de predicción, se calcularon las matrices de confusión como se muestra en la Tabla 2. En la matriz de confusión, el verdadero positivo (VP) es el número de clases defectuosas que se predicen correctamente como defectuosas; falso negativo (FN) cuenta el número de clases defectuosas que se predice incorrectamente como no defectuosas; falso positivo (FP) mide la cantidad de clases que están libre de defectos, pero se predice incorrectamente como defectuosas; y verdadero negativo (VN) representa el número de clases sin defectos que se predicen correctamente como no defectuosas.

Tabla 2. Matriz de confusión.

Real \ Predicho	**Defectuosa**	**No defectuosa**
Defectuosa	Verdadero positivo (VP)	Falso negativo (FN)
No defectuosa	Falso positivo (FP)	Verdadero negativo (VN)

Se calcularon las métricas precisión, exhaustividad y exactitud al igual que [3] y [8], para evaluar el desempeño de los clasificadores. También se incluyeron los valores obtenidos de la métrica de especificidad como en [21] y coeficiente de correlación de Matthew como en [6].

Precisión. Relaciona el número de verdaderos positivos (predichos y observados como propensos a defectos) con el número de clases predichas como propensas a defectos.

$$Precisión = \frac{VP}{VP+FP} \quad (1)$$

Un valor cercano a uno es deseable y significaría que todas las clases que se predijeron que tenían defectos en realidad tenían defectos.

6

Exhaustividad. Relaciona el número de verdaderos positivos (predichos y observados como propensos a defectos) con el número de clases que realmente tenían defectos.

$$Exhaustividad = \frac{VP}{VP+FN} \quad (2)$$

Un valor cercano a uno es el mejor y significaría que se predijo que todas las clases que tenían defectos observados tenían defectos.

Especificidad. Relaciona el número de verdaderos negativos (predichos y observados como no propensos a defectos) con el número de clases que no tenían defectos.

$$Especificidad = \frac{VN}{VN+FP} \quad (3)$$

Un valor cercano a uno es el mejor y significaría que se predijo que todas las clases que no tenían defectos observados efectivamente no los tenían.

Exactitud. Relaciona el número de clasificaciones correctas (verdaderos positivos y verdaderos negativos) con el número total de clases.

$$Exactitud = \frac{VP+VN}{VP+VN+FP+FN} \quad (4)$$

Un valor de uno es lo mejor y significaría que el modelo se clasificó perfectamente, es decir, no cometió un solo error.

Coeficiente de correlación de Matthew (CCM). Indica hasta qué punto las variables independientes y dependientes están relacionadas entre sí.

$$CCM = \frac{VN \times VP - FP \times FN}{\sqrt{(VN+FN)(FP+VP)(VN+FP)(FN+VP)}} \quad (5)$$

Los valores van de -1 a 1, donde los valores cercanos a 1 indican un mayor rendimiento.

4 Resultados

Tal y como se mencionó en la sección 3, se entrenaron tres clasificadores con conjuntos de datos diferentes CK, OO y CK + OO. En cada caso se aplicó la técnica de sobre-muestreo SMOTE para la generación de nuevos casos en los datos de entrenamiento para evitar sesgar los resultados. Los modelos fueron construidos con el método de votación con seis algoritmos clasificadores. Finalmente se computaron las métricas exactitud, precisión, exhaustividad, especificidad y CCM para evaluar el rendimiento de los modelos.

En la Tabla 3 se encuentran los resultados obtenidos de las métricas de desempeño de los tres clasificadores. En cada caso se puede observar una exactitud mayor a 0.8 y

ninguno exhibe una diferencia significativa; por otra parte CCM presenta índices similares para cada clasificador, siendo CK y CK + OO los que tienen valores más altos con 0.331 y 0.324 respectivamente.

Con respecto a la precisión, el clasificador CK + OO presenta el mejor desempeño de los tres con 0.508. Continuando con exhaustividad, OO registró el mejor valor con 0.476, diferenciándose en más de 0.14 comparado con CK. Finalmente, la especificidad no presenta valores dispares entre los clasificadores, siendo CK + OO el que obtuvo mayor puntaje con 0.93.

Tabla 3. Resultados de métricas de desempeño de los clasificadores.

Clasificador	Precisión	Exhaustividad	Especificidad	Exactitud	CCM
CK	0.455	0.329	0.927	0.834	0.331
OO	0.404	0.476	0.87	0.808	0.294
CK + OO	0.508	0.35	0.933	0.837	0.324

5 Amenazas a la validez

En esta sección, discutimos las amenazas que podrían afectar la validez del estudio empírico realizado en este artículo.

5.1 Validez de constructo

Las amenazas en esta categoría se refieren a la relación entre la teoría y la observación, es decir, las variables medidas pueden no medir realmente la variable conceptual. La amenaza es el ruido que afecta a los repositorios de Bugzilla. En [22] Antoniol et al. mostró que una fracción considerable de los informes de problemas marcados como errores en Bugzilla son de hecho "no errores", es decir, problemas no relacionados con el mantenimiento correctivo. De igual manera los autores del conjunto de datos inspeccionaron y verificaron manualmente una muestra estadísticamente significativa (107) de los errores de Eclipse JDT Core que más del 97% de ellos eran errores reales. Por tanto, el impacto de esta amenaza en los experimentos es limitado.

5.2 Validez de la conclusión

Está vinculado con la relación entre el tratamiento y el resultado. Para asegurar que los resultados no hubieran sido sesgados por efectos de confusión debidos al desequilibrio de datos se adoptó el procedimiento SMOTE [17].

En cuanto a la evaluación del desempeño de los modelos experimentados, se consideró CCM, el cual ha sido muy recomendado [23, 24] para interpretar correctamente los resultados.

5.3 Validez externa

Relacionada con la generalización de los hallazgos. Se han aplicado técnicas de predicción solo a sistemas de software de código abierto y ciertamente existen diferencias entre el desarrollo industrial y el de código abierto. Los autores minimizaron esta amenaza mediante el uso de partes de Eclipse, un sistema que, si bien es de código abierto, tiene una sólida formación industrial. Una segunda amenaza se refiere al lenguaje: todos los sistemas de software están escritos en Java. Incluir sistemas que no sean Java aumentaría su valor, pero introduciría problemas ya que los sistemas necesitarían ser procesados por diferentes analizadores, produciendo resultados variables.

Para disminuir el impacto de una tecnología o herramienta específica, los autores del conjunto de datos incluyeron sistemas desarrollados con diferentes sistemas de control de versiones y diferentes sistemas de rastreo de defectos (Bugzilla y Jira). Además, los sistemas de software del conjunto de datos son desarrollados por equipos de desarrollo independientes y surgieron del contexto de dos comunidades no relacionadas (Eclipse y Apache).

6 Discusiones y conclusiones

La asignación de recursos humanos y económicos a los componentes de software defectuosos es una tarea esencial para el mantenimiento del sistema. Por lo tanto, también resulta esencial definir cuáles son los atributos que predicen donde están estos componentes. Este es objetivo principal de este artículo, por medio de la siguiente pregunta de investigación: ¿qué conjunto de métricas orientadas a objetos obtenidas de una versión del sistema tienen mejor rendimiento para predecir defectos en las clases de un proyecto software? Para ello, se construyeron y evaluaron tres modelos predictivos, utilizando en cada caso un conjunto de datos diferente.

En la Tabla 3 del presente trabajo, los tres modelos tienen una exactitud mayor a 0.8. Comparando con experiencias previas como [3], donde el máximo valor registrado fue de 0.789, todos los modelos hicieron un mejor trabajo de clasificación. Pero, ciertamente esta métrica no es suficiente para describir en profundidad el desempeño de los clasificadores, como se puede observar en los valores de precisión y exhaustividad en 5 de los 6 casos se registra un valor menor a 0.5.

La precisión y exhaustividad representan qué tan bien se identificaron las clases defectuosas. Dicho esto, ningún clasificador se destacó significativamente con respecto al resto en el proceso de identificación de clases con defectos. Sin embargo, los tres modelos han detectado exitosamente las clases sin defectos, esto es confirmado tanto por los valores de exactitud y especificidad en todos los modelos, con resultados mayores a 0.8. Esto no es atribuible a la sobre proporción de clases sin defectos de las muestras originales, porque, como se mencionó en la sección 3, se aplicó el método de muestreo SMOTE para evitar este sesgo.

Finalmente, se puede mencionar que en la mayoría de las métricas de desempeño CK + OO ha obtenido mejores resultados exceptuando CCM y exhaustividad. Por lo tanto, se puede afirmar que utilizar solamente las métricas OO para predecir defectos

podría implicar en un peor desempeño del clasificador, y sería preferible emplearla complementariamente con CK, o inclusive reemplazarla por completo.

Referencias

1. Erlikh, L. Leveraging legacy system dollars for e-business. IT Prof. 2, 17–23 (2000).
2. D'Ambros, M., Lanza, M. & Robbes, R. An extensive comparison of bug prediction approaches. in Proceedings - International Conference on Software Engineering 31–41 (2010). doi:10.1109/MSR.2010.5463279.
3. Zimmermann, T., Premraj, R. & Zeller, A. Predicting defects for eclipse. in Proceedings - ICSE 2007 Workshops: Third International Workshop on Predictor Models in Software Engineering, PROMISE'07 9–9 (IEEE, 2007). doi:10.1109/PROMISE.2007.10.
4. Zhang, F., Mockus, A., Keivanloo, I. & Zou, Y. Towards building a universal defect prediction model with rank transformed predictors. Empir. Softw. Eng. 21, 2107–2145 (2016).
5. Ghotra, B., McIntosh, S. & Hassan, A. E. Revisiting the impact of classification techniques on the performance of defect prediction models. in Proceedings - International Conference on Software Engineering vol. 1 789–800 (IEEE Computer Society, 2015).
6. Pecorelli, F. & Di Nucci, D. Adaptive selection of classifiers for bug prediction: A large-scale empirical analysis of its performances and a benchmark study. Sci. Comput. Program. 205, 102611 (2021).
7. Chidamber, S. R. & Kemerer, C. F. Towards a metrics suite for object oriented design. ACM SIGPLAN Not. 26, 197–211 (1991).
8. Jureczko, M. & Madeyski, L. Towards identifying software project clusters with regard to defect prediction. in ACM International Conference Proceeding Series 1 (ACM Press, 2010). doi:10.1145/1868328.1868342.
9. Shepperd, M., Song, Q., Sun, Z. & Mair, C. Data quality: Some comments on the NASA software defect datasets. IEEE Trans. Softw. Eng. 39, 1208–1215 (2013).
10. Panichella, A., Oliveto, R. & De Lucia, A. Cross-project defect prediction models: L'Union fait la force. in 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014 - Proceedings 164–173 (IEEE Computer Society, 2014). doi:10.1109/CSMR-WCRE.2014.6747166.
11. Qiao, L., Li, X., Umer, Q. & Guo, P. Deep learning based software defect prediction. Neurocomputing 385, 100–110 (2020).
12. Bug prediction dataset, https://bug.inf.usi.ch, último acceso: 16/09/2021
13. D'Ambros, M., Lanza, M. & Robbes, R. Evaluating defect prediction approaches: A benchmark and an extensive comparison. in Empirical Software Engineering vol. 17 531–577 (Springer, 2012).
14. Jing, X., Wu, F., Dong, X., Qi, F. & Xu, B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. in 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings 496–507 (Association for Computing Machinery, Inc, 2015). doi:10.1145/2786805.2786813.
15. Xu, Z., Liu, J., Yang, Z., An, G. & Jia, X. The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison. in Proceedings - International Symposium on Software Reliability Engineering, ISSRE 309–320 (IEEE Computer Society, 2016). doi:10.1109/ISSRE.2016.13.

10

16. Bennin, K. E., Keung, J., Phannachitta, P., Monden, A. & Mensah, S. MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction. IEEE Trans. Softw. Eng. 44, 534–550 (2018).
17. Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. SMOTE: Synthetic Minority Over-sampling Technique. J. Artif. Intell. Res. 16, 321–357 (2011).
18. Imbalanced-learn Documentation, https://imbalanced-learn.org/stable/, último acceso 16/09/2021.
19. Kittler, J., Hatef, M., Duin, R. P. W. & Matas, J. On combining classifiers. IEEE Trans. Pattern Anal. Mach. Intell. 20, 226–239 (1998).
20. Scikit-learn, https://scikit-learn.org/stable/, último acceso 16/09/2021.
21. Rathore, S. S. & Kumar, S. Software fault prediction based on the dynamic selection of learning technique: findings from the eclipse project study. Appl. Intell. (2021) doi:10.1007/s10489-021-02346-x.
22. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F. & Guéhéneuc, Y. G. Is it a bug or an enhancement? A text-based approach to classify change requests. Proc. 2008 Conf. Cent. Adv. Stud. CASCON'08 (2008) doi:10.1145/1463788.1463819.
23. Hall, T., Beecham, S., Bowes, D., Gray, D. & Counsell, S. Developing fault-prediction models: What the research can show industry. IEEE Softw. 28, 96–99 (2011).
24. Yao, J. & Shepperd, M. Assessing software defection prediction performance: Why using the Matthews correlation coefficient matters. ACM Int. Conf. Proceeding Ser. 120–129 (2020) doi:10.1145/3383219.3383232.