Finding Defective Software Modules by Means of Data Mining Techniques

J. C. Riquelme, R. Ruiz, D. Rodríguez y J. S. Aguilar-Ruiz

***Abstract*— The characterization of defective modules in software engineering remains a challenge. In this work, we use data mining techniques to search for rules that indicate modules with a high probability of being defective. Using datasets from the PROMISE repository1, we first applied feature selection to work only with those attributes from the datasets capable of predicting defective modules. Then, a genetic algorithm search for rules characterising subgroups with a high probability of being defective. This algorithm overcomes the problem of unbalanced datasets where the number of non-defective samples in the dataset highly outnumbers the defective ones.**

***Keywords*— Defect detection and defect prediction in software modules, data mining.**

I. Introducción

En los últimos años se ha incrementado el número de trabajos de investigación que tratan de aplicar a la Ingeniería del Software técnicas de análisis de datos basadas en la minería de datos. Usualmente la toma de decisiones en el desarrollo de software basadas en informaciones y datos anteriores se confía a los expertos, que normalmente realizan esta tarea de manera manual, fruto de la experiencia acumulada de la que extraen un conocimiento útil. El principal inconveniente para automatizar esta tarea viene dado porque las empresas no tienen un sistema que permita obtener datos sobre el proceso de producción (ni a priori ni a posteriori) de forma que pueda ser usado en procesos de extracción automática de conocimiento.

El conjunto de técnicas que tradicionalmente proporciona la Minería de Datos tienen dos objetivos distintos en función del problema que queramos resolver: desde el punto de vista predictivo, en el que se intenta obtener conocimiento para clasificación o predicción; o desde el punto de vista descriptivo, donde se intenta descubrir conocimiento de interés dentro de los datos, intentando obtener información que describa de una manera comprensible para el experto el modelo que hay tras los datos. En el primer caso puede ser suficiente un "modelo caja negra" donde lo importante es que la tasa de acierto ante un nuevo elemento sea lo más alta posible. En el segundo caso, el aspecto cuantitativo pierde importancia ante la explicación cualitativa que el modelo sea capaz de aportar.

La existencia de repositorios con información obtenida de proyectos reales permite a los expertos disponer de datos de los que poder sacar conclusiones mediante la aplicación de técnicas de minería de datos. En estos repositorios se pueden encontrar tablas de datos, en las que cada fila representa un módulo software al que se le han medido un conjunto de métricas (líneas de código, puntos de función, número de operaciones, etc.). De esta forma los valores concretos para cada módulo representan una tupla en la que cada dimensión es una columna de la tabla (en este caso cada columna se denomina atributo o característica). Una de las columnas clasifica estos módulos en función de un objetivo concreto: por su esfuerzo de desarrollo, por su coste de mantenimiento, por el número de errores, etc., y se denomina etiqueta de clase.

En este trabajo hemos seleccionado el problema de la identificación de fallos en módulos software. Esto quiere decir que queremos determinar cuáles son las características que más influyen para que un determinado código presente errores, cuestión de vital importancia para mejorar la calidad del software. El principal inconveniente que presentan los datos para este caso es que normalmente el conjunto de tuplas a analizar no está balanceado, es decir, existen un pequeño número de módulos con defectos en comparación con los que no lo tienen. Esta descompensación afecta a las técnicas de aprendizaje que normalmente se guían en la fase de entrenamiento por intentar ajustar el modelo para acertar lo máximo posible. Sin embargo, si de partida un modelo que asigna todos los registros a la clase mayoritaria acierta el 90% o más de los ejemplos, aumentar la complejidad del mismo para ganar algún punto de exactitud es normalmente descartado. De esta forma la mayoría de los algoritmos de aprendizaje ante datos desbalanceados dan lugar a modelos degenerados en los cuales no se considere la clase minoritaria.

En este trabajo, dada la especial naturaleza de los conjuntos de datos abordados, además del interés en la predicción de fallos, nos centraremos en los procesos de inducción descriptiva, más concretamente en el descubrimiento de subgrupos, cuyo objetivo es el de identificar subconjuntos con características específicas dentro de los datos para obtener información que los describa.

Por otra parte, no debemos de olvidar que con frecuencia, la existencia de muchos atributos o características dificulta la aplicación de técnicas de extracción de conocimiento. Teóricamente, el tener más atributos daría más poder discriminatorio. Sin embargo, la experiencia con algoritmos

Este trabajo ha sido elaborado en el marco del proyecto de investigación oficial TIN2007-68084-C02-00, financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT).

J. C. Riquelme imparte docencia en el Departamento de Lenguajes y Sistemas de la Universidad de Sevilla, Sevilla, riquelme@lsi.us.es.

R. Ruiz y J. S. Aguilar-Ruiz imparten docencia en el Área de Lenguajes y Sistemas Informáticos de la Universidad Pablo de Olavide de Sevilla, Sevilla, robertoruiz@upo.es; aguilar@upo.es.

D. Rodríguez imparte docencia en el Departamento de Ciencias de la Computación de la Universidad de Alcalá, Madrid, daniel.rodriguezg@uah.es.

de aprendizaje ha demostrado que no es siempre así, detectándose algunos problemas: tiempos de ejecución muy elevados, aparición de muchos atributos redundantes y/o irrelevante, y la degradación en el error de clasificación, etc.

En este trabajo, vamos a caracterizar mediante reglas las condiciones que cumplen unas métricas para que los módulos software presenten errores, basándonos en la aplicación de técnicas de selección de atributos y de descubrimiento de subgrupos aplicadas a datos extraídos de repositorios públicos. La estructura del trabajo es la siguiente: en la sección II. se presentan los trabajos relacionados más conocidos; en la sección III. se describen los conceptos más importantes utilizados en el proceso; a continuación, en la sección IV. se muestran los experimentos realizados; y finalmente, en la sección V. se recogen las conclusiones.

II. Trabajos Relacionados

Un amplio rango de modelos estadísticos y de aprendizaje automático han sido desarrollados y aplicados a predecir defectos en módulos software. En [2] Basili et al investigaron el impacto de las métricas de diseño orientado a objetos presentadas en [4] para la predicción de *fault-prone classes* usando regresión logística. Guo et al [7] presentaron una técnica de *random forest* para predecir *fault-proneness* en sistemas software de la NASA. La metodología propuesta se comparó con otras técnicas, encontrando que la tasa de aciertos de su propuesta era la más alta. Khoshgoftaar et al [11] investigaron el uso de una red neuronal como modelo para predecir la calidad del software. Su banco de pruebas fue un sistema de telecomunicaciones comparando el modelo neuronal con un modelo discriminante no paramétrico.

En Khoshgoftaar et al [12] se aplicó un árbol de regresión con reglas de clasificación para el mismo problema. Fenton et al [5] propusieron una red bayesiana para la predicción de fallos. Otras técnicas usadas para la predicción en la calidad del software incluyen: análisis discriminante, técnicas de reducción, algoritmos genéticos, árboles de decisión, razonamiento basado en casos y redes de Dempster-Shafer.

Hasta ahora pocos de estos autores han investigado la aplicación previa de técnicas de reducción de atributos a datos provenientes de la ingeniería del software. Entre otros cabe citar Chen et al [3] o Kirsopp y Shepperd [13], en cuyos respectivos trabajos se analizó la aplicación de una selección de atributos usando una técnica wrapper a la estimación de costes, concluyendo que el conjunto reducido mejoraba la estimación.

En cuanto a referencias para tratar con datos no balanceados las técnicas de muestreo son las más usadas. Por ejemplo, Seiffert et al [16] redujeron el efecto del ruido en los datos mediante diversas técnicas de muestreo con diferentes técnicas de aprendizaje. Las técnicas de muestreo usadas incluyen Random undersampling (RUS), random oversampling (ROS), one-sided selection (OSS), cluster-based oversampling (CBOS), Wilson's editing (WE), SMOTE (SM) y borderline-SMOTE (BSM). Los autores concluyen que hay técnicas más robustas (RUS, WE y BSM) que otras (ROS y SM) y que el nivel de desbalanceado afecta a la mejor solución. Los autores usaron un solo conjunto de datos público con diferentes niveles de ruido añadido artificialmente.

Finalmente, Vandecruys et al [17] han generado reglas usando una técnica de optimización basada en Colonias de Hormigas. Nuestro objetivo en este trabajo es similar ya que nosotros también tratamos de generar reglas. Sin embargo, su aproximación adolece de no manejar atributos continuos ni clases desbalanceadas como es capaz de manejar la aproximación aquí presentada.

III. Detección y Predicción de Fallos en Módulos Software utilizando técnicas de Minería de Datos

Un proceso de minería de datos tiene como objetivo el descubrimiento de conocimiento, obteniendo modelos de forma autónoma sobre los datos. Se distingue entre dos tipos de tareas:

- Predicción: donde el sistema encuentra patrones para predecir el comportamiento futuro utilizando el conjunto de datos para pronosticar valores futuros desconocidos de alguna variable de interés, como es caso de las tareas de clasificación o regresión.
- Descripción: donde sistema encuentra patrones para presentarlos a un experto en una forma comprensible para él, y que describen y aportan información de interés sobre el problema y el modelo que subyace bajo los datos. Entre las tareas principales con un objetivo descriptivo se encuentra el agrupamiento (*clustering*), la asociación y el descubrimiento de subgrupos.

La aplicación de un algoritmo de aprendizaje tiene como objetivo extraer conocimiento de un conjunto de datos y modelar dicho conocimiento para su posterior aplicación en la toma de decisiones. Existen distintas formas de representar el modelo generado, representación proposicional, árboles de decisión, reglas de decisión, listas de decisión, reglas con excepciones, reglas jerárquicas de decisión, reglas difusas y probabilidades, redes neuronales, etc. Sin embargo, cuando los resultados proporcionados por el algoritmo de aprendizaje han de ser interpretados directamente por el usuario o experto, la complejidad y la legibilidad de la representación del conocimiento adquieren especial importancia. En este sentido, la representación mediante reglas es a menudo, más sencilla de comprender que el resto de representaciones.

El objetivo central de este trabajo es el generar un modelo útil en la predicción de fallos futuros, y además que el modelo sea fácilmente interpretable para estudiar las características especiales que inducen al fallo. Para ello, utilizaremos reglas de decisión, pero antes de exponer los patrones obtenidos realizaremos unas observaciones sobre los conceptos más mportantes utilizados en el proceso.

A. Estudios Preliminares

1) Reglas de Decisión

En general, una regla de decisión es del tipo "si X es verdad e

Y es falso entonces C", donde X e Y son las condiciones que pueden ser evaluadas de forma directa para atributos discretos, mientras que para parámetros continuos es necesario dividir en intervalos su dominio y usar operadores relacionales del tipo "mayor que" o "menor que"; y la evaluación de estas condiciones como ciertas implicaría la clasificación con etiqueta C.

El objetivo de un sistema de este tipo es encontrar un conjunto de reglas que cubra los casos de una clase particular sin cubrir los casos de otras clases, o al menos, minimizando el número de estos casos o errores cometidos.

Geométricamente un sistema que aprende y que encuentra reglas como la anteriormente definida, describe regiones de decisión cuyas fronteras son líneas paralelas a los ejes, formando zonas rectangulares. Por ejemplo con dos variables continuas X e Y y las condiciones X>2 AND Y>1 AND X<4 AND Y<4, la región sería un rectángulo, que ajustándolo adecuadamente, se puede aproximar cualquier superficie y cubrir cualquier clase. Por tanto, la efectividad y eficiencia con la que un sistema de aprendizaje pueda cubrir los datos mediante regiones de forma rectangular determinará la bondad de estos sistemas. En este trabajo, se va a usar como técnica para la obtención de reglas un algoritmo evolutivo [1] con alguna modificación para adaptarlo a la naturaleza no balanceada de los datos.

2) Selección de Atributos

Como se ha indicado anteriormente, al seleccionar los atributos más relevantes se pretende ayudar al proceso siguiente de aprendizaje.

Se debe tener en cuenta que los atributos irrelevantes y redundantes existentes en una base de datos pueden tener un efecto negativo en los algoritmos de clasificación: (1) El tener más atributos implica normalmente la necesidad de disponer de más instancias para garantizar la fiabilidad de los patrones obtenidos (variabilidad estadística entre patrones de diferente clase). Por consiguiente, el algoritmo de clasificación entrenado con todos los datos tardará más tiempo. (2) El exceso inútil de atributos puede confundir a los algoritmos de aprendizaje, por lo que en general, el clasificador obtenido será menos exacto que otro que aprenda sobre datos relevantes. (3) Con la presencia de atributos redundantes y/o irrelevantes, el clasificador obtenido será más complejo, dificultando el entendimiento de los resultados. (4) Además, la reducción de características se podría tener en cuenta en futuras capturas de datos, reduciendo el coste de obtención. Por tanto, podemos colegir que la selección es efectiva en eliminar atributos no necesarios, incrementando la eficiencia en las tareas de minería, mejorando el rendimiento y la comprensión de los resultados.

3) Bases de Datos No Balanceadas.

Los conjuntos de datos con los que trabajamos están claramente desbalanceados, existiendo una gran diferencia entre el número de módulos con defectos y los que no lo tienen. Esto puede llegar a causar una gran degradación en lo modelos obtenidos en función de los algoritmos de aprendizaje utilizados. Se pueden destacar dos problemas principalmente: ocultación del impacto de algunos factores, así como tener en cuenta falsas variables representativas.

Cuando se aplican algoritmos de minería de datos, los modelos generados pueden verse afectados por el hecho de que estos algoritmos supongan que los datos están balanceados, y aparentemente ofrezcan una buena exactitud predictiva. En estos casos disponemos de dos opciones, mejorar los algoritmos haciéndolos más robustos a este tipo de circunstancias, o bien, utilizar técnicas de muestreo. Este muestreo se puede realizar de dos formas: generando nuevos datos de la clase minoritaria (over-sampling) o decrementando los de la clase mayoritaria (under-sampling), es decir, los datos de módulos sin fallos. En Yasutaka et al [10] analizaron la aplicación de cuatro técnicas de muestreo (ROS, SMOTE, RUS, and ONESS) para clasificar módulos erróneos con cuatro tipos de modelos (Análisis Discriminante Lineal (LDA), Regresión Logística (LR), Redes Neuronales (NN) y árboles de clasificación (CT) en un sistema software. Los autores concluyen que el muestreo no mejora el funcionamiento de los modelos NN y CT y sí en el caso de los otros dos. Li y Reformat [14] usan un aprendizaje basado en lógica fuzzy para generar modelos robustos a este problema.

B. Descubrimiento de Subgrupos

En un algoritmo de descubrimiento de subgrupos se extraen reglas o patrones de interés para un subconjunto de los datos especificado previamente. El concepto se basa en el descubrimiento de propiedades características de subgrupos cuyo comportamiento se diferencia respecto al de la totalidad de los datos.

El descubrimiento de subgrupos es un tipo de inducción descriptiva que ha recibido últimamente mucha atención por parte de los investigadores. Con él se pretenden generar modelos basados en reglas cuya finalidad es descriptiva, empleando una perspectiva predictiva para obtenerlos. Se trata por tanto de una tarea con objetivos básicamente descriptivos que incluye algunas características de la inducción predictiva. Las reglas utilizadas en la tarea de descubrimiento de subgrupos tienen la forma "*Condición→Clase*" anteriormente comentada, donde la propiedad de interés para el descubrimiento de subgrupos es el valor de la Clase que aparece en el consecuente de la regla y el antecedente (*Condición*) es una conjunción de variables (parejas atributo-valor) seleccionadas entre las variables del conjunto de datos. Como se ha señalado anteriormente se ha usado para hallar el modelo un algoritmo evolutivo que originalmente no es específico para el descubrimiento de subgrupos. La modificación que se ha implementado es que en la generación inicial sólo se introducen reglas para cubrir registros que representen módulos con errores. De esta manera, se consigue que las reglas sólo se enfoquen hacia la clase minoritaria, sin necesidad de realizar ningún tipo de muestreo para resolver el desbalanceado de los datos.

IV. EXPERIMENTACIÓN

En este trabajo se pretende analizar el comportamiento de nuestra propuesta con cuatro bases de datos (CM1, KC1, KC2, y PC1) disponibles en el repositorio PROMISE [18] para predecir módulos defectuosos. Estos conjuntos de datos se

originaron a partir de proyectos llevados a cabo por la NASA (http://mdp.ivv.nasa.gov/) y suelen aparecer en estudios recientes. La tabla I muestra el número de instancias, cuantas son no defectuosas y cuantas sí, porcentaje de defectos y además el lenguaje de programación en el que fueron escritos los módulos.

TABLA I
CONJUNTO DE DATOS

Datos	#instancias	Sin-defectos	Defectuosos	% defectuosos	Lenguaje
CM1	498	449	49	9.83	C
KC1	2109	1783	326	15.45	C++
KC2	522	415	107	20.49	C++
PC1	1109	1032	77	6.94	C

Todas las bases de datos contienen los mismos 22 atributos: 5 diferentes medidas de líneas de código, 3 métricas McCabe [15], 4 basadas en medidas Halstead [9], 8 derivadas de las anteriores, 1 contador de bifurcaciones y la clase que indica si el módulo tiene defecto (tabla II).

TABLA II
DEFINIÓN DE ATRIBUTOS

	#	Símbolo	Métrica
McCabe	1	loc	Líneas de c'odigo
	2	v(g)	Complejidad ciclomática
	3	ev(g)	Complejidad esencial
	4	iv(g)	Complejidad del diseño
Halstead Derived	5	n	Total operadores + operandos
	6	v	Volumen
	7	l	Longitud
	8	d	Dificultad
	9	I	Inteligencia
	10	e	Esfuerzo
	11	b	Esfuerzo estimado
	12	t	Tiempo
	13	lOCode	Líneas de Código: blancos+comen.+cód.
	14	lOComment	Líneas de Comentarios
	15	lOBlank	Líneas vacías
	16	lOCodeAndComment	
Halstead base	17	uniq_Op	Operadores únicos
	18	uniq_Opnd	Operandos únicos
	19	total_Op	Total operadores
	20	total_Opnd	Total operandos
Branch	21	branchCount	branchCount
Clase		{false, true}	Si el modulo defectuoso

Las métricas de McCabe fueron introducidas en 1976 y se basan en el recuento del número de caminos lógicos individuales contenidos en un programa utilizando la teoría de grafos. Para hallar la complejidad ciclomática, el programa se representa como un grafo, donde las instrucciones son nodos y los posibles caminos las aristas del grafo. La complejidad ciclomática, i.e., número de caminos, se puede calcular matemáticamente como: $v(g)=e-n+2$ donde e representa el número de aristas y n el número de nodos. En psicología es conocido que el número de manipulaciones mentales simultaneas de los humanos se limita a 7±2. Por tanto, si tenemos una serie de módulos donde todos los factores son iguales salvo la complejidad, aquellos más complejos son los que pueden generar más problemas. También puede utilizarse para medir la complejidad de la integración de módulos. La complejidad ciclomática mide la complejidad en sentido de cantidad, pero para medir la calidad del código (evitando lo que se conoce como "código spaghetti"), McCabe definió una métrica similar, complejidad esencial, *ev(g)* (essential complexity). En la programación estructurada las únicas estructuras necesarias son la *secuencia*, la *selección* e *iteración* y la complejidad esencial se calcula como la complejidad ciclomática, pero quitando dichas estructuras del grafo. También, la *complejidad del diseño* del módulo *iv(g)* utiliza un grafo simplificado para calcular la complejidad, pero en este caso para hacer la reducción se tiene en cuenta las llamadas a otros módulos.

El otro conjunto de métricas utilizadas en este estudio y también desarrolladas hacia finales de 1970 por Halstead [9] en lo que denominó Ciencia del software de Halstead (*Halstead Software Science*), se basan en palabras reservadas del lenguaje (testigos o *tokens* cuando usamos la jerga de compiladores). Las métricas se basan en simples cuentas de: (i) los operadores son las palabras reservadas del lenguaje, tales como IF-THEN, READ, FOR; los operadores aritméticos +, -, *, etc; los de asignación y los operadores lógicos AND, EQUAL, etc.; (ii) los operandos son las variables, literales y las constantes del programa. Halstead distingue entre el número de operadores y operandos únicos y el número total de operadores y operando. En concreto, se utiliza la siguiente notación:

- n_1, número de operadores únicos que aparecen en un programa
- N_1, número total de ocurrencias de operadores
- n_2, número de operandos únicos que aparecen en un programa
- N_2, número total de ocurrencias de operandos

A partir de estas simples cuentas han sido elaboradas diferentes medidas para diversas propiedades de los programas (independientemente del lenguaje de programación). Estas incluyen:

- Longitud, $l = N_1+N_2$
- Vocabulario, $n = n_1+ n_2$
- Volumen, $v = N \log_2(n)$. Halstead con esta métrica intenta calcular el número de comparaciones mentales necesarias para escribir un programa de longitud N.
- Esfuerzo, $e = v / l$. El esfuerzo que cuesta escribir un programa se mide por el número de discriminaciones mentales.
- Inteligencia, $i = (2 n_2 / n_1 N_2) (N_1+N_2)\log_2(n_1+n n_2)$, viene a medir cuanto "dice" un programa, es decir, un programa en diferentes lenguajes debería de dar un valor similar.

Estas métricas también parece que sirven para predecir módulos defectuosos aunque recomendamos acudir al trabajo de Fenton y Pfleeger [6] para un estudio más profundo de estas métricas.

Como se señaló en el apartado III.A.2, la selección de los

atributos relevantes puede ayudar en la fase posterior de extracción de conocimiento. En nuestro caso el método utilizado es CFS [8] (Correlation-based Filter Selection), selección de tipo filtro independiente de cualquier algoritmo de aprendizaje automático. Selecciona un conjunto de atributos altamente correlados con la clase y con bajo grado de redundancia entre ellos. Diremos que este método es supervisado, secuencial, filtro y que genera un subconjunto de atributos representativo.

De este modo, las bases de datos originales CM1, KC1, KC2, y PC1, de 22 atributos pasan a tener 8, 9, 4 y 7 respectivamente, incluyendo el atributo clase que indica si existe o no fallo en el módulo. En la tabla III se muestra la relación de atributos finales, donde se puede resaltar el atributo *uniq_Opnd* que se escoge en las cuatro ocasiones, y los atributos *intelligence*, *lOComment* y *lOBlank* en tres.

TABLA III
MÉTRICAS SELECCIONADAS MEDIANTE EL ALGORITMO CFS

CM1	KC1	KC2	PC1
loc	v	ev(g)	v(g)
iv(g)	d	b	i
i	i	uniq_Opnd	lOComment
lOComment	lOCode		locCodeAndComment
lOBlank	lOComment		lOBlank
uniq_Op	lOBlank		uniq_Opnd
uniq_Opnd	uniq_Opnd		v(g)
loc	branchCount		i
	v		

En este trabajo las técnicas de muestreo no ayudan a obtener mejores reglas de clasificación, dado que si utilizamos técnicas que incrementan el número de registros de módulos con fallos lo hacen duplicando los ya existentes, y esto no influye para nada en las reglas; y las técnicas de *under-sampling* que eliminan datos de módulos sin fallos, confunden si eliminan datos que están en la frontera de decisión o no afectan en el resto de casos.

En otras publicaciones [17] el algoritmo de modelado utilizado para la obtención de las reglas requiere de la discretización de las variables de los conjuntos de datos, sin embargo el método utilizado en este artículo trabaja con los datos originales para no introducir un elemento de distorsión que pueda acarrear una mejora ficticia de la exactitud en la clasificación.

El mecanismo de selección de atributos realizado nos permite disminuir el número de condiciones en las reglas, simplificando su legibilidad. Por ejemplo, la selección de atributos muestra que el atributo *uniq_opnd* ha sido seleccionado en tres de las cuatro reglas más significativas, lo cual indica la relevancia de esta métrica para determinar módulos defectuosos.

Las reglas presentes en la Tabla IV indican que es más probable encontrar un fallo en un módulo si las métricas están dentro del rango indicado. En todos los casos estos rangos de valores se aproximan a los umbrales indicados en el repositorio Web donde se han obtenido los datos. Así por ejemplo para los datos KC1, la métrica *d* no debería exceder de 30 y la regla proporciona un límite de 24, el umbral previsto de *uniq_opnd* es 20 y la regla encontrada propone 24 y finalmente para la métrica *branchCount* el umbral previsto es 19 y la regla nos estima 12. Las diferencias son pequeñas, teniendo en cuenta que tanto los valores umbrales como los obtenidos por las reglas son aproximados, y que distintos entornos de desarrollo pueden variar estos valores. Por ultimo, las reglas confirman que los umbrales previstos se cumplen y establecen una combinación de valores más significativa que los umbrales independientes para cada métrica.

TABLA IV
REGLAS DE DECISIÓN OBTENIDAS PARA CADA CONJUNTO DE DATOS

Datos	Regla
CM1	37.27 <= i AND 27 <= lOBlank
KC1	24.33 <= d AND 24 <= uniqOpnd AND 12 <= branchCount
KC2	5 <= ev(g) AND 37 <= uniqOpnd
PC1	15 <= lOComment AND 85 <= uniqOpnd

V. CONCLUSIONES

En este trabajo se han aplicado diversas técnicas provenientes de la minería de datos para identificar patrones que caracterizan el software con fallos. Para ello se han usado cuatro conjuntos de datos del repositorio público PROMISE que nos proporcionan un conjunto de valores de métricas aplicadas a módulos software y añaden una etiqueta de si existe fallo o no en él. Estos conjuntos de datos se caracterizan por tener muchos más ejemplos sin fallos, lo que condiciona la selección de los métodos de aprendizaje automático. A partir de estos datos, aplicamos sucesivamente una técnica de selección de atributos y un algoritmo evolutivo que sea capaz de caracterizar el subgrupo de registros que representan a los módulos con fallos. El resultado es un conjunto de reglas que identifican intervalos de valores para las métricas que caracterizan los módulos con software.

Como trabajo futuro nos proponemos intentar simplificar el conjunto de reglas hallado aumentando su soporte (número de datos que cubre cada regla) sin disminuir la confianza.

REFERENCIAS

[1] J. S. Aguilar-Ruiz, R. Giráldez and J. C. Riquelme, "Natural encoding for evolutionary supervised learning", *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 4, pp. 466-479, 2007.

[2] V. Basili, L. Briand and W. Melo, "A validation of object-oriented design metrics as quality indicators", *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, 1996.

[3] Z. Chen, T. Menzies, D. Port and B. Boehm, "Finding the right data for software cost modeling", *IEEE Software*, vol. 22, pp. 38-46, 2005.

[4] S. Chidamber and C. Kemerer, "A metrics suite for object-oriented design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.
[5] N. Fenton, M. Neil and P. Krause, "Software measurement: uncertainty and causal modeling", *IEEE Software*, vol. 19, pp. 116-122, 2002.
[6] N. E. Fenton and S. L. Pfleeger, *Software Metrics: a Rigorous \& Practical Approach*, International Thompson Press, 1997.
[7] L. Guo, Y. Ma, B. Cukic and H. Singh, "Robust prediction of fault-proneness by random forests", in *Proc. 2004 15th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 417-428.
[8] M. A. Hall, "Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning", in *Proc. 2000 17th Int. Conf. on Machine Learning*, pp. 359-366.
[9] M. Halstead, *Software Metrics: a Rigorous \& Practical Approach*, Elsevier, 1977.
[10] Y. Kamei, A. Monden, S. Matsumoto and T. K. K. ichi Matsumoto, "The effects of over and under sampling on fault-prone module detection", in *Proc. 2007 Empirical Software Engineering and Measurement (ESEM)*.
[11] T. Khoshgoftaar, E. Allen, J. Hudepohl and S. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system", *IEEE Transactions on Neural Networks*, vol. 8, no. 4, pp. 902-909, 1997.
[12] T. Khoshgoftaar, E. Allen and J. Deng, "Using regression trees to classify fault-prone software modules", *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 455-462, 2002.
[13] C. Kirsopp and M. Shepperd, "Case and feature subset selection in case-based software project effort prediction", in *Proc. 22nd SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence*.
[14] Z. Li and M. Reformat, "A practical method for the software fault-prediction", in *Proc. 2007 IEEE International Conference Information Reuse and Integration (IRI)*, pp. 659-666.
[15] T. J. McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, 1976.
[16] C. Seiffert, T. M. K. J. Van Hulse and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data", in *Proc. 2007 IEEE International Conference Information Reuse and Integration (IRI)*, pp. 651-658.
[17] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer and R.Haesen, "Mining software repositories for comprehensible software fault prediction models", *J. Syst. Softw.*, vol. 81, no. 5, pp. 823-839, 2008.
[18] G. Boetticher, T. Menzies, T. Ostrand, PROMISE Repository of empirical software engineering data. http://promisedata.org/, West Virginia University, Department of Computer Science (2007)

José C. Riquelme es Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Autor de más de un centenar de artículos en revistas y congresos internacionales, ha dirigido ocho tesis doctorales y es miembro de numerosos Comités de Programa nacionales e internacionales. Como Presidente del Comité de Programa de JISBD 2006, ha sido editor invitado de un número especial de IEEE América Latina.

Roberto Ruiz es Licenciado en informática por la Universidad de Sevilla en 1994 y doctorado en informática por la Universidad de Sevilla en 2006, donde ha sido profesor desde octubre del 2001 hasta septiembre de 2006, y desde entonces es profesor en la Escuela Politécnica Superior de la Universidad Pablo de Olavide de Sevilla. Además, Roberto tiene 7 años de experiencia en la empresa privada. Sus intereses en la investigación se centran en la minería de datos y la aplicación de distintas técnicas de la minería de datos y computación a la ingeniería del software y a bases de datos genómicas.

Daniel Rodríguez es Licenciado en informática por la Universidad del País Vasco en 1995 y doctorado en informática por la Universidad de Reading, Reino Unido, en 2003, donde ha sido profesor desde octubre del 2001 hasta septiembre 2006. Daniel obtuvo el certificado en educación universitaria por la Universidad de Reading en julio del 2005. Desde octubre 2006, es profesor en el departamento de Ciencias de la Computación de la Universidad de Alcalá de Henares, Madrid. Además, Daniel tiene 2 años de experiencia en la empresa privada como ingeniero de software y consultor. Sus intereses en la investigación se centran en la ingeniería del software, y la aplicación de distintas técnicas de la minería de datos y computación a la ingeniería del software. Daniel es miembro de las asociaciones profesionales ACM e IEEE.

Jesus S. Aguilar Ruiz es Doctor Ingeniero en Informática por la Universidad de Sevilla. Actualmente es Profesor Titular de Universidad y Director de la Escuela Politécnica Superior de la Universidad Pablo de Olavide, de Sevilla. Ha sido miembro de numerosos Comités de Programa de conferencias internacionales relevantes al área de Minería de Datos, así como revisor de revistas afines. Es autor de más de un centenar de artículos. Sus principales áreas de interés son la Computación Evolutiva, la Minería de Datos y la Bioinformática. Desde 2008 es Editor Jefe de la revista BioData Mining, de la editorial BioMed Central. http://www.upo.es/eps/aguilar.