Sistemas expertos con aplicaciones 42 (2015) 1872–1879

Listas de contenidos disponibles en ScienceDirect

Sistemas Expertos con Aplicaciones

página de inicio de la revista: www.elsevier.com/locate/eswa

Una comparación de algunos métodos de computación blanda para la predicción de fallas de software

Ezgi Ertürk [a], Ebru Akcapinar Sezer b,

[a] El Consejo de Investigación Científica y Tecnológica de Turquía (TUBITAK), Instituto de Investigación de Tecnologías de Software, Ankara, Turquía
[b] Universidad Hacettepe, Departamento de Ingeniería Informática, 06800 Ankara, Turquía

información del artículo

Historial del artículo:
Disponible en línea el 23 de octubre de 2014

Palabras clave:
Predicción de fallos de software
Métricas McCabe
Sistemas difusos neuro adaptativos
Redes neuronales artificiales
Máquinas de vectores de soporte

abstracto

La principal expectativa del software confiable es la minimización de la cantidad de fallas que ocurren cuando se ejecuta el programa. Es importante determinar si los módulos de software son propensos a fallar porque hacerlo ayuda a identificar los módulos que requieren refactorización o pruebas detalladas. La predicción de fallas de software es una disciplina que predice la propensión a fallas de los módulos futuros mediante el uso de métricas de predicción esenciales y datos históricos de fallas. Este estudio presenta la primera aplicación del Adaptive Neuro Fuzzy Inference System (ANFIS) para el problema de predicción de fallas de software. Además, los métodos de la Red Neural Artificial (ANN) y la Máquina de Vectores de Soporte (SVM), que se experimentaron anteriormente, están diseñados para analizar el rendimiento de ANFIS. Los datos utilizados en este estudio se recopilan del repositorio de ingeniería de software de PROMISE y se seleccionan las métricas de McCabe porque abordan de manera integral el esfuerzo de programación. ROC-AUC se utiliza como medida de rendimiento. Los resultados alcanzados fueron 0,7795, 0,8685 y 0,8573 para los métodos SVM, ANN y ANFIS, respectivamente.

1. Introducción

La calidad del software ha experimentado un aumento de la demanda en los últimos diez años debido al importante papel que desempeñan los sistemas de software en la vida humana diaria. La tolerancia al comportamiento impredecible del software durante el tiempo de ejecución está disminuyendo debido al alto costo de este tipo de comportamiento y porque puede causar situaciones irrecuperables. Como resultado, los estudios sobre la calidad del software intentan aumentar la calidad del ciclo de vida del desarrollo del software y producir sistemas de software de alta calidad. La mayor expectativa del software de alta calidad es su confiabilidad; por lo tanto, la cantidad de fallas que ocurren cuando el programa se está ejecutando debe minimizarse para garantizar un software confiable. Las principales causas de las fallas son fallas generadas por errores en la acción humana. Los errores, fallas, fallas y defectos se definen de la siguiente manera. Un error es una acción humana faltante o una acción humana que incrusta errores particulares en el producto. Por ejemplo, los errores cometidos durante la implementación del código se aceptan como errores. La falla ocurre cuando el comportamiento del software no cumple con las expectativas del usuario. Una falla es un paso, código, proceso, definición de datos o defecto físico incorrecto que ocurre en el hardware o el software. Una falla es la causa principal de las fallas en un programa de software. Por ejemplo, el software puede entrar en un estado inesperado debido a un error del programador y, como resultado, este tipo de falla puede causar una falla. El término defecto se utiliza como expresión general de error, falla y culpa. En resumen, los errores pueden causar fallas y las fallas pueden causar fallas cuando se ejecuta el programa de software, y el término defecto incluye todos estos términos.

La determinación de módulos de software propensos a fallas es un proceso importante porque ayuda a identificar módulos que requieren refactorización o pruebas detalladas. De esta manera, se pueden desarrollar productos de software más calificados. La predicción de fallas de software es una disciplina que predice la propensión a fallas de los módulos futuros utilizando métricas de predicción esenciales y datos históricos de fallas. Al considerar los sistemas de predicción de fallas de software, se puede planificar un cronograma de proyecto de manera más eficiente, especialmente para las fases de prueba y mantenimiento.
Los beneficios de la predicción de fallas de software se enumeran a continuación (Catal, 2011).

El proceso de prueba se puede refinar, aumentando así la calidad del sistema.
Es posible especificar módulos que requieran refactorización durante la fase de mantenimiento.
La aplicación de la predicción de fallas de software utilizando métricas de nivel de clase durante la fase de diseño permite seleccionar la mejor de las alternativas de diseño.
La predicción de fallas de software proporciona estabilidad y alta seguridad al sistema de software.

Autor de correspondencia. Tel.: +90 3122977500; fax: +90 3122977502.
Direcciones de correo electrónico: erturkezgi@gmail.com (E. Erturk), ebruakcapinarsezer@gmail. com (EA Sezer).

http://dx.doi.org/10.1016/j.eswa.2014.10.025 0957-4174/ 2014

La predicción de fallas de software puede reducir el tiempo y el esfuerzo invertidos en el proceso de revisión del código.

Debido a estos beneficios, la predicción de fallas de software se convierte en un importante problema de investigación. De hecho, el objetivo de la predicción es válido. y disponibles en muchas áreas diferentes, y existen muchos métodos de predicción o clasificación propuestos: árboles de decisión, redes neuronales, máquinas de vectores, regresión logística, etc. Además, algunos de estos métodos se aplican al problema de predicción de fallas de software enumerado en la Sección 2. Este estudio selecciona tres métodos de computación suave diferentes: Sistema de inferencia neurodifuso adaptativo (ANFIS), Máquina de vectores de soporte (SVM) y red neuronal artificial (ANN): para crear modelos de predicción de fallas de software y comparaciones sus resultados de rendimiento. De hecho, estudios previos han aplicado ANN y SVM a este problema (por ejemplo, Thwin y Quah (2003), Gondra (2008), Mahaweerawat, Sophatsathit, Lursinsap y Musílek (2004), Malhotra (2014), Xing, Guo y Lyu (2005)). Sin embargo, ANFIS no se ha aplicado a este problema utilizando un enfoque comparativo.

ANFIS es un poderoso método de predicción que combina el aprendizaje habilidad y conocimiento experto y logra muchos resultados exitosos en la predicción de problemas en diferentes áreas (por ejemplo, Daoming y Jie (2006), Lo (2003), Najah, El-Shafie, Karim y Jaafar (2012), Pérez, González y Dopico (2010), Pradhan, Sezer, Gokceoglu, y Buchroithner (2010), Sezer, Pradhan y Gokceoglu (2011)). Como se enfatiza en Catal y Diri (2009), el algoritmo seleccionado para la identificación de los módulos de software defectuosos es importante tanto como los parámetros seleccionados. En este punto aportamos la solución métodos de este problema proponiendo ANFIS. La diferencia entre ANFIS y los otros métodos basados en datos (ANN, SVM, y árbol de decisión (DT)) es que ANFIS utiliza el conocimiento experto para construye el modelo y usa datos para optimizarlo. En realidad, modelador tiene que decidir qué parámetros se utilizan independientemente de la método empleado cuando él / ella utiliza métodos basados en datos puros. Sin embargo, el modelador también tiene que especificar la función de pertenencia. tipo y número, o conjuntos borrosos para cada parámetro durante el modelado con ANFIS y esta información especifica directamente la forma de manejo de la borrosidad o vaguedad en el modelo predictivo. Por ejemplo, si El parámetro "línea de código (loc)" se usa mientras se modela con ANN, su formato de datos y su relación entre el parámetro de salida son consideró. Sin embargo, si el método predictivo es ANFIS, además a estas consideraciones, el experto debe especificar el número de borrosos conjuntos que se utilizarán para el parámetro "loc". Si se especifica como 2 (es decir, bajo, alto) con función de pertenencia triangular, significa que vaguedad del modelo especificado en el nivel máximo. Si 3 borrosos conjuntos (es decir, bajo, moderado, alto) se utilizan en forma de triangular función de pertenencia, la vaguedad del modelo disminuye según al caso anterior. Las decisiones del experto se basan en la distribución natural de los valores de los parámetros, los números de clase naturales del parámetro, el grado de borrosidad al pasar una clase a otros y la relación entre las clases de parámetros de entrada y salida. En consecuencia, se puede decir que el modelado con ANFIS requiere una mayor conciencia sobre las condiciones y resultados incrustados en los datos. Por lo tanto, el uso de cada parámetro es decidido uno por uno, no como un conjunto de parámetros completo. Como resultado, El uso de ANFIS se vuelve importante por tres razones (i) ANFIS es un poderoso método predictivo pero no ha sido experimentado antes para el problema de predicción de fallas de software (ii) modelado con ANFIS desencadena una reconsideración detallada de cada parámetro de entrada, incluso si se ha utilizado comúnmente en estudios anteriores (iii) propiedades del modelo ANFIS proporciona información valiosa (mejor número de clase para cada parámetro, grado de vaguedad, distribución de los datos, reglas más efectivas) al modelador de expertos sistemas basados En otras palabras, ANFIS es una forma más fácil de experimentar para un experto que otros métodos de aprendizaje automático para optimizar su conocimiento.

Con base en estas razones, el ANFIS se emplea para predecir la falla del software por primera vez en este estudio y también se usan dos de los métodos de aprendizaje automático más utilizados (ANN y SVM). empleados para hacer una comparación entre los ANFIS y sus rendimientos. Este estudio utiliza las métricas de McCabe (Thomas, 1976) como funciones de software y el conjunto de datos de los experimentos es creado por componer proyectos utilizando PROMISE Software Engineering Repositorio (Sayyad & Menzies (2005); Promise Software Conjuntos de datos públicos del repositorio de ingeniería (2013)). En el paso de experimentación, se utilizan todas las métricas de McCabe (loc, v(g), ev(g) y iv(g)). en los modelos y el rendimiento de los mismos se evalúan en un primer momento. Después examen de los casos defectuosos, experimentos que utilizan 3 McCabe las métricas (loc, v(g) y iv(g)) se organizan en segundo lugar. Los modelos' los rendimientos se comparan utilizando el área bajo la curva ROC (AUC). Los resultados de desempeño muestran que ANFIS es una empresa competitiva y fuerte método para resolver este problema y nos anima a llevar a cabo estudios adicionales. Adicionalmente, se propone el uso de 3 métricas de McCabe en lugar de todos ellos. Los resultados experimentales muestran SVM y ANFIS conseguir mejores resultados con 3 parámetros que con 4.

2. Trabajos relacionados

Los estudios relacionados con el problema de predicción de fallos de software. se resumen aquí en dos partes, como los estudios más antiguos publicados antes del año 2013 y los estudios presentados en la última dos años. Catal (2011) analizó 90 artículos sobre predicción de fallas de software que se publicaron entre 1990 y 2009. La mayoría importante contribución del estudio fue que proporciona una guía para investigadores sobre métricas de software, métodos utilizados para software predicción de fallas, conjuntos de datos y criterios de evaluación del rendimiento. Catal, Sevim y Diri (2011) desarrollaron un software basado en Eclipse herramienta de predicción de fallas utilizando aprendizaje automático y técnicas estadísticas. Los objetivos del estudio eran tanto de manera práctica como Predecir teóricamente fallas en los programas de software. Ellos prefirieron el algoritmo Naive Bayes por su alto rendimiento. catal y Diri (2009) investigó el efecto del tamaño del conjunto de datos, conjuntos de métricas y técnicas de selección de características en problemas de predicción de fallas de software, que no se investigaron previamente en esta área de investigación. Utilizaron el algoritmo Random Forest y Artificial Immune Systems como métodos de aprendizaje automático y el repositorio PROMISE como método. un conjunto de datos Como resultado, determinaron que el algoritmo seleccionado es más importante que las métricas seleccionadas. Este hallazgo fuertemente admite el empleo del método ANFIS para este problema. Mahaweerawat, Sophatsathit, Lursinsap y Musílek (2006) desarrolló un modelo mejorado llamado MASP para predecir e identificar fallas en los sistemas de software orientados a objetos. El modelo MASP puede filtre las métricas apropiadas para tipos de fallas particulares. El objetivo de los Vandecruys et al. (2008) estudio fue predecir de manera eficiente fallas módulos de software y apoyar el desarrollo de software mediante la investigación de repositorios de software utilizando técnicas de minería de datos (AntMiner+ basado en la optimización de colonias de hormigas). Alsmadi y Najadat (2011) tuvo como objetivo predecir módulos propensos a fallas y determinar las relaciones entre ellos. Consideraron que los atributos con alta correlación entre ellos podrían afectarse negativamente entre sí. Hu, Xie, Ng y Levitin (2007) propusieron corregir fallas además de predecir partes defectuosas del software. Para ello, aplicaron iterativamente redes neuronales y un algoritmo genético. El algoritmo genético aumentó el rendimiento de la predicción modelo. Gondra (2008) intentó mostrar qué métrica de software era más importante para la predicción de fallas y usaba ANN y SVM métodos para este fin. Además, comparó los resultados de los modelos construidos usando ANN y SVM. Zimmermann, Premraj, y Zeller (2007) pretendieron desarrollar un modelo de predicción de fallas que se ejecutó en las versiones 2.0, 2.1 y 3.0 de Eclipse mediante la construcción de una falla base de datos. Zhou y Leung (2006) estudiaron la predicción de fallas que tenían alta y baja prioridad utilizando métricas de nivel de clase y logística

regresión, Naive Bayes, Random Forest y el vecino más cercano utilizando técnicas de generalización en un conjunto de datos KC1 de la NASA. Descubrieron que la profundidad del árbol de herencia y la cantidad de niños no son métricas importantes y que el conjunto de métricas de Chidamber-Kemerer es útil para la predicción de fallas de software. Menzies, DiStefano, Orrego y Chapman (2004) estudiaron predictores de fallas usando métricas a nivel de método y la técnica Naive Bayes en los conjuntos de datos del depósito de datos de PROMISE. Utilizaron la probabilidad de detección, la probabilidad de falsa alarma y la precisión como criterios de evaluación.

En los últimos dos años, los estudios sobre la predicción de fallas del software han abordado el problema desde dos puntos: proponiendo nuevos métodos o combinaciones de métodos para aumentar el rendimiento de la predicción; utilizando nuevos parámetros o enfoques de selección de características para proponer métricas más efectivas para la predicción. Scanniello, Gravino, Adrian y Menzies (2013) utilizaron la regresión lineal multivariante para ayudar a seleccionar las variables independientes que tienen una correlación estadísticamente significativa con la falla. El proceso de aprendizaje fue realizado por grupos de clases relacionadas. En Rodríguez, Ruiz, Riquelme y Harrison (2013), se propuso un enfoque descriptivo basado en el descubrimiento de subgrupos para la predicción de fallas de software.

Las reglas inducidas para la predicción de fallas se extrajeron de los conjuntos de datos con los algoritmos correspondientes y luego, las reglas se aplicaron a nuevas instancias para su clasificación. Dejaeger, Verbraken y Baesens (2013) demostraron que se pueden construir redes comprensibles con menos nodos utilizando clasificadores de redes bayesianas (BN) para predecir fallas de software. Para lograr este objetivo, se utilizaron 15 clasificadores BN diferentes y los resultados de los modelos se compararon con los resultados de los modelos que empleaban otras técnicas populares de aprendizaje automático. Además, se investigó el efecto del principio de la manta de Markov en el rendimiento del modelo de predicción y los experimentos muestran que no existe un efecto significativo de la técnica de selección de características en el rendimiento del modelo. Oyetoyan, Cruzes y Conradi (2013) utilizaron métricas orientadas a objetos extendidas (dependencias cíclicas) para identificar el perfil de defectos de los componentes de software. Para probar la efectividad de las métricas de dependencia cíclica, las métricas cíclicas y no cíclicas se usaron por separado y se realizaron análisis a nivel de clase y nivel de paquete en modelos de predicción. Cotroneo, Natella y Pietrantuono (2013) intentaron predecir la ubicación de errores relacionados con el envejecimiento (ARB) en sistemas de software complejos mediante el uso de métricas de complejidad de software como variables predictoras y algoritmos de aprendizaje automático (NB, BN, DT y LR con transformación logarítmica).

Después de analizar los informes de errores de los proyectos que se utilizaron como conjunto de datos en el estudio, se calcularon las métricas de complejidad y se utilizaron para predecir módulos propensos a ARB.
Malhotra (2014) analizó y comparó diferentes métodos estadísticos (LR) y de aprendizaje automático (LR, DT, ANN, Cascade Correlation Network, SVM, Group Method of Data Handling Method y Gene Expression Programming) para la predicción de fallas. Fueron validados empíricamente para encontrar la relación entre las métricas del código estático y la propensión a fallas de un módulo. En Couto, Pires, Valente, Bigonha y Anquetil (2014), se utilizó la prueba de causalidad de Granger para evaluar si las variaciones pasadas en los valores de las métricas del código fuente se pueden usar para pronosticar cambios en las series temporales de defectos.

En el primer paso del estudio, se calcularon los valores de umbral para las métricas del código fuente. En el segundo paso, los valores de umbral y las clases cambiadas se tomaron como entradas para el modelo de predicción de defectos y el modelo determinó el estado de falla de la clase.
Czibula, Marian y Czibula (2014) presentaron un modelo de clasificación novedoso basado en la minería de reglas de asociación relacional para la predicción de defectos de software. Para disminuir la multidimensionalidad de las entradas, las relaciones entre las características se analizaron utilizando el coeficiente de correlación de rangos de Spearman en la fase de preprocesamiento de datos y luego se definió un conjunto de relaciones entre los valores de las características y el conjunto de reglas. Khosgoftaar, Xiao y Gao (2014) utilizaron modelos basados en reglas (RB) y métodos de aprendizaje basado en casos (CBL).

se recopilaron para clasificar los módulos de software como propensos a fallas o no propensos a fallas. El modelo generado (RB2CBL) utilizó el algoritmo genético para optimizar los parámetros del modelo. Laradji, Alshayeb y Ghouti (2014) combinaron métodos de aprendizaje por conjuntos con modelos de selección de características para identificar las fallas del software. La selección codiciosa de sala se utilizó como método de selección de características. El modelo de aprendizaje de conjuntos utilizado en el experimento incluía siete clasificadores diferentes y cada clasificador predecía la probabilidad de errores. La salida fue el promedio de estas probabilidades.

En realidad, se consideraron las selecciones de conjuntos de datos, criterios de evaluación y métodos predictivos de estudios previos, y se aplicaron preferencias similares para obtener resultados comparables con la literatura.
En detalle, hay 2 criterios de evaluación populares para obtener resultados experimentales como ROC-AUC (p. ej., Czibula et al. (2014), Dejaeger et al. (2013), Gondra (2008), Laradji et al. (2014), Mahaweerawat et al. (2006), Malhotra (2014)) y memoria de precisión (por ejemplo, Cotroneo et al. (2013), Couto et al. (2014), Oyetoyan et al.

(2013), Rodríguez et al. (2013)). En consecuencia, empleamos el ROC-AUC para la evaluación del desempeño para presentar resultados comparables con estudios previos. Al mismo tiempo, el repositorio PROMISE es muy popular para la predicción de fallas de software (p. ej., Czibula et al.
(2014), Dejaeger et al. (2013), Laradji et al. (2014), Malhotra (2014), Rodríguez et al. (2013)) y proyectos de la NASA con métricas McCabe en el repositorio PROMISE se utilizaron aquí. Además, se utilizaron métricas a nivel de método y no métricas orientadas a objetos debido a las propiedades del conjunto de datos seleccionado. La selección de ANN y SVM se basa en su uso común para este problema (p. ej., Gondra (2008), Kanmani, Uthariaraj, Sankaranarayanan y Thambidurai (2004), Khoshgoftaar, Allen, Hudepohl y Aud (1997), Laradji et al. (2014), Mahaweerawat y otros (2004), Malhotra (2014), Thwin y Quah (2003), Wang, Yu y Zhu (2004), Xing y otros (2005)).

Siguiendo estos estudios, este estudio presenta los resultados complementarios para el problema de predicción de fallas de software empleando ANFIS como método de predicción. Además, aquí se propone el uso de 3 parámetros en el conjunto métrico McCabe como resultado del proceso de modelado ANFIS.

3. Métricas y datos

Las métricas utilizadas en la predicción de fallos de software se pueden separar en seis grupos: métricas de nivel de método, métricas de nivel de clase, métricas de nivel de archivo, métricas de nivel de componente, métricas de nivel de proceso y diferentes métricas cuantitativas de nivel de valor (Catal, 2011) .). Las métricas de McCabe, que emplea este estudio, corresponden al nivel de método.
Las métricas a nivel de método se enfocan en conceptos de programación, y recopilarlos de los códigos fuente desarrollados utilizando enfoques estructurales o orientados a objetos es fácil. Catal y Diri (2009) determinaron que los módulos propensos a fallas previstos son los métodos que probablemente tengan fallas durante las pruebas del sistema o las pruebas de campo cuando se utilizan métricas a nivel de método. Como resultado, se dice que las métricas a nivel de método abordan integralmente el esfuerzo de programación. Además, Catal (2011) encontró que las métricas a nivel de método se usaron comúnmente entre 1990 y 2007. Todas estas ventajas permiten que este estudio sea comparable con los otros estudios presentados en la literatura.

Hay cuatro tipos de métricas de McCabe (http://promise.site.uotta wa.ca/SERepository/datasets/cm1.arff):

Número McCabe de línea de código (loc)
Complejidad ciclomática (v(g)): Cantidad de caminos independientes lineales en un diagrama de flujo dibujado para un módulo. La complejidad ciclomática cambia según los nodos de decisión de la estructura del módulo.
La capacidad de prueba disminuye a medida que aumenta la complejidad ciclomática.
Complejidad esencial (ev(g)): El valor de la complejidad del gráfico que no incluye gráficos D-prime (gráficos de operaciones de secuencia, selección e iteración) en el diagrama de flujo de un módulo. La mantenibilidad y la modularidad disminuyen a medida que aumenta la complejidad esencial.

Complejidad de diseño (iv(g)): Complejidad ciclomática del grafo de flujo reducido de un módulo según decisiones y bucles que no tienen llamadas a submódulos. La mantenibilidad y la reutilización disminuyen a medida que aumenta la complejidad del diseño.

En el estudio, los proyectos (CM1, JM1, KC1, KC2 y PC1) que tienen métricas de McCabe en el repositorio de PROMISE se seleccionan y utilizan para producir resultados más confiables. De esta forma, se genera un conjunto de datos con 15.123 instancias y se implementan modelos predictivos utilizando 4 y 3 entradas (métricas) y una salida (valor de falla medido).

Para validar los modelos predictivos se emplea la validación cruzada N-fold, que es la técnica más conocida. La validación cruzada N-fold divide los datos en N número de particiones, y cada una de ellas tiene el mismo número de muestras. Posteriormente, se realiza el entrenamiento con (N 1) número de particiones, y se realiza la prueba con las particiones restantes. En este estudio, el conjunto de datos preparado se divide en cinco partes (N = 5), lo que indica que el 80 % de los datos se usa para entrenamiento y el 20 % para pruebas. Además, cada partición del conjunto de datos tiene la misma tasa de instancias defectuosas.

4. Método

Aunque se utilizaron algunos métodos empíricos para predecir fallas de software, ANFIS, una de las técnicas de computación blanda, no se ha empleado previamente para tal propósito. En general, ANFIS es un método supervisado que combina las ventajas de un sistema de inferencia difusa (FIS) y métodos de redes neuronales artificiales (ANN).

En otras palabras, ANFIS puede construir un mapeo de entrada-salida basado tanto en el conocimiento humano como en los pares de datos de entrada-salida estipulados (Jang, 1993). La estructura de las redes neuronales difusas se asemeja a la estructura de las redes neuronales normales con tres o cuatro capas ocultas. Cada capa en una red neuronal difusa es diferente y representa una tarea de un sistema de inferencia difuso. Las tareas son fuzzificación, ejecución de reglas, normalización y defuzzificación, en orden. El conocimiento experto proporcionado a los modelos ANFIS presentados por Jang (1993) es el número de conjuntos borrosos y los tipos de funciones de pertenencia.

Las reglas se construyen automáticamente dentro del tipo Sugeno de primer orden (si x es A e y es B entonces z = mx + ny + k), y los pesos de las reglas, los parámetros de las funciones de pertenencia y los rangos de los conjuntos difusos se ajustan mediante entrenamiento y aprendiendo. En la Fig. 1 se presenta una arquitectura típica de ANFIS que consta de un total de seis capas , y los pasos de cálculo con los detalles de implementación se dan a continuación:

– Capa 0: permite tomar entradas nítidas del entorno; en este estudio se emplean 3 y 4 parámetros de entrada basados en métricas de McCabe.

– Capa 1: difumina las entradas utilizando números de conjuntos difusos y funciones de pertenencia especificados. Se emplea principalmente una función de pertenencia de tipo campana, pero es opcional. En este estudio, se utiliza una función en forma de campana con 3 conjuntos difusos para cada entrada en los experimentos. La membresía tipo campana tiene 3 parámetros: centro, ancho y pendiente. Se puede suponer que son los parámetros antecedentes no lineales que se aprenden de los datos.
– Capa 2: incluye un nodo específico para cada regla, y cada nodo calcula la fuerza de disparo de la regla asociada utilizando el operador de producto. En este estudio se generan y disparan un total de 81 y 27 reglas para 4 y 3 entradas respectivamente. Los conjuntos de datos de tren utilizados en los experimentos constan de aproximadamente 12.123 casos.
– Capa 3: normaliza las fuerzas de disparo de cada regla proveniente de la capa 2.

– Capa 4: activa las reglas usando valores de fuerza normalizados provenientes de la capa 3 y calcula la contribución de cada regla al resultado general.

– Capa 5: contiene solo un nodo para producir resultados nítidos al sumar todos los resultados provenientes de la capa 4.

Un modelo ANFIS utiliza un algoritmo de aprendizaje híbrido que combina el estimador de mínimos cuadrados y el método de descenso de gradiente (Jang, 1993). En cada época, se emplea un estimador de mínimos cuadrados en el paso hacia adelante y el método de descenso de gradiente se emplea en el paso hacia atrás. En el paso hacia atrás, se ajustan los rangos y parámetros de las funciones de pertenencia y, en el paso hacia adelante, se ajustan los coeficientes en la expresión del polinomio (Sen, Sezer, Gokceoglu y Yagiz, 2012). El entrenamiento del modelo continúa hasta que se obtiene el cambio mínimo de errores entre la iteración actual y la anterior o se alcanza la época deseada numerada.

En este estudio, se utilizan un total de 100 iteraciones como criterio de parada.

ANN se deriva de inspiraciones de la capacidad de aprendizaje del cerebro humano. Ofrece un método mejorado, genérico y práctico para aprender funciones reales, discretas o de valores vectoriales. ANN es tolerante con datos imperfectos en un conjunto de datos de entrenamiento y consta de una gran cantidad de unidades, como neuronas y relaciones ponderadas dirigidas entre estas unidades (Lee & Park, 1992). A medida que crece el número de unidades en una red, aumenta la complejidad de la estructura. En otras palabras, el número de unidades especifica la complejidad del sistema. Con estas propiedades, ANN resuelve problemas que no pueden resolverse utilizando métodos clásicos como lo hace el cerebro humano (Lee & Park, 1992).

Los modelos ANN construidos en este estudio consisten en 4 y 3 entradas (métricas de McCabe) y una salida (falla). El número de neuronas en la capa oculta debe seleccionarse cuidadosamente de acuerdo con las heurísticas sugeridas (Hecht-Nielsen, 1987; Kanellopoulas y Wilkinson, 1997; Yagiz, Sezer y Gokceoglu, 2012). La arquitectura ANN sugerida por Hecht-Nielsen (1987) es [62Ni + 1] donde Ni se refiere al número de entradas. En este estudio, se utilizan un total de 8 y 6 neuronas en las capas ocultas de los modelos que se construyen para 4 y 3 entradas. En otras palabras, el número de neuronas utilizado aquí satisface la heurística propuesta por Hecht-Nielsen (1987). El algoritmo de entrenamiento del modelo es el gradiente conjugado escalado. El método de gradiente conjugado escalado es una función de entrenamiento de red que actualiza los valores de ponderación y sesgo. El modelo ANN funciona como una red de reconocimiento de patrones. En otras palabras, es una red de retroalimentación que se puede entrenar para clasificar las entradas de acuerdo con las clases objetivo.

Los datos de destino para nuestra red de reconocimiento de patrones consisten en información de fallas en cada componente de software. Los detalles de la ANN se encuentran en Negnevitsky (2005). El modelo realizó la tarea de clasificación como resultado de aproximadamente 50 iteraciones con 0,134 errores cuadráticos medios. El proceso de entrenamiento se detiene cuando el gradiente de rendimiento cae por debajo del gradiente de rendimiento mínimo. El modelo ANN construido para 4 entradas en este estudio se muestra en la Fig. 2.

SVM es un método de clasificación que se puede utilizar tanto para datos lineales como no lineales. Utiliza mapeo no lineal para transformar los datos de entrenamiento en una dimensión superior. Busca el mejor hiperplano separador lineal con la nueva dimensión. SVM encuentra este hiperplano usando vectores de soporte y márgenes (Vapnik, 1998).

Los datos de dos clases se pueden separar mediante un hiperplano con un mapeo no lineal adecuado a una dimensión alta (Fig. 3). El método de optimización mínima secuencial (SMO) se utiliza para encontrar el hiperplano de separación para el modelo SVM. El consumo de memoria está controlado por el valor que especifica el tamaño de la memoria caché de la matriz del núcleo (5000). El algoritmo SMO almacena solo una submatriz de la matriz del núcleo, limitada por el tamaño especificado por el tamaño de la memoria caché de la matriz del núcleo. El modelo SVM utiliza un método de optimización para identificar los vectores de soporte si, los pesos ai y el sesgo b, que se utilizan para clasificar los vectores x según la siguiente ecuación (http://www.mathworks.com/help/stats/svmtrain. HTML):

$$c = \sum a_i k(s_i, x) + b$$

donde k es una función kernel y un producto escalar. Si $c \geq 0$, entonces x se clasifica como miembro del primer grupo; de lo contrario, se clasifica como miembro del segundo grupo.

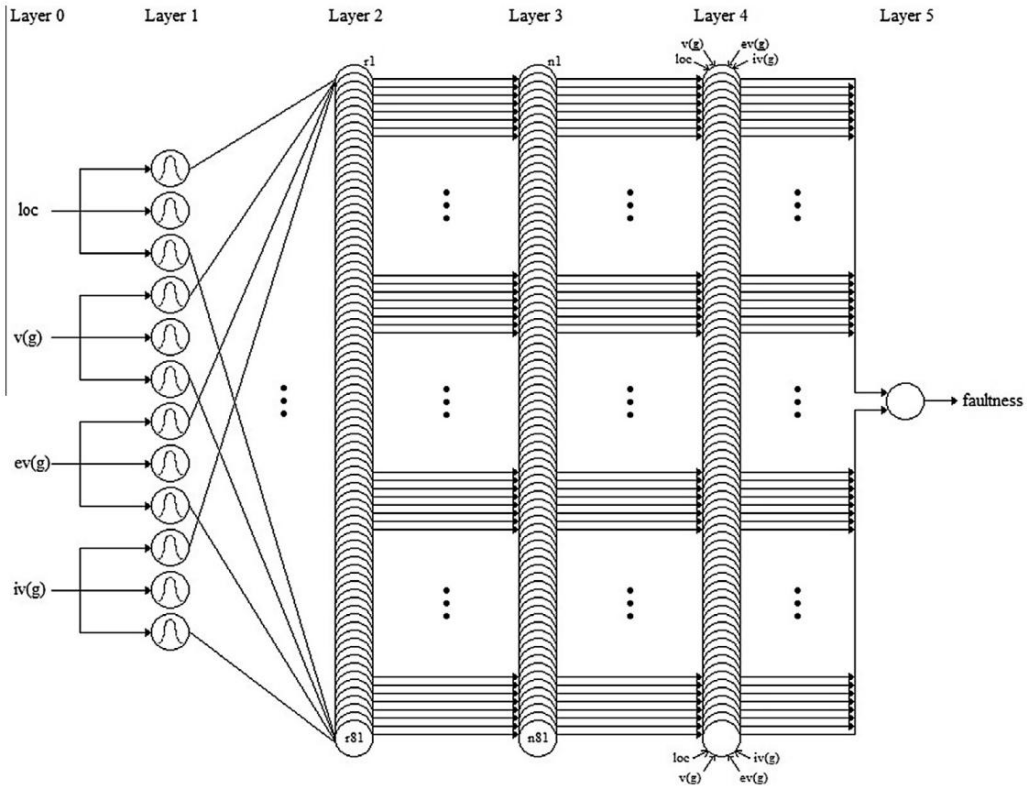

Fig. 1. Estructura del modelo ANFIS.

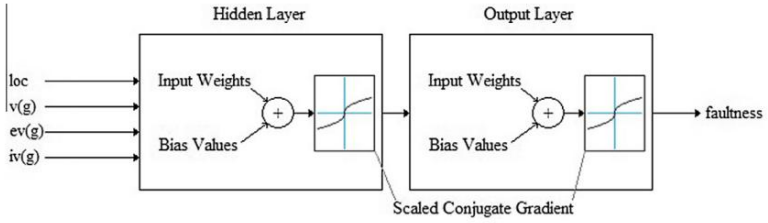

Figura 2. Estructura del modelo ANN.

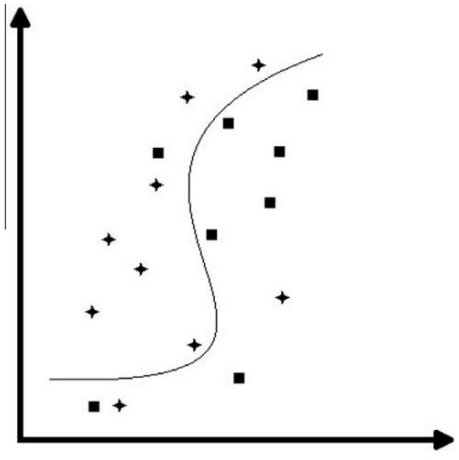

Fig. 3. Dos clases separadas por un hiperplano.

El proceso de formación puede ser lento; sin embargo, la precisión suele ser alta debido a la capacidad de modelar límites de decisión no lineales. SVM es adecuado tanto para la clasificación como para la predicción. El número máximo de iteraciones se especifica como 22 000 y no hay conversión. gencia se logra en un menor número de iteraciones. Se pueden encontrar más detalles sobre SVM en (http://www.mathworks.com/help/stats/svmtrain.html).

Las curvas de características operativas del receptor (ROC) con el enfoque del área bajo la curva (AUC) son criterios ampliamente utilizados para evaluar el rendimiento de los modelos de predicción. Son criterios sin umbral y muestran cambios en las tasas de predicción de verdaderos positivos frente a las tasas de predicción de falsos positivos. En otras palabras, una curva ROC traza las tasas de verdaderos positivos (TPR) en el eje y y las tasas de falsos positivos (FPR) en el eje x. Como resultado, una curva ROC comienza en los puntos (0, 0) y llega a (1, 1). La curva ROC ideal pasa por el punto de (0, 1) con AUC = 1, lo que indica que no hay error de predicció Un valor AUC aceptable para una curva ROC debe ser superior a 0,5. En la figura 4 se muestra una curva ROC típica (Menzies, Greenwald y Frank, 2007; Menzies et al., 2004). Según Catal (2011, 2012), el área bajo una curva ROC representa un criterio ampliamente utilizado en conjuntos de datos separados para evaluar el rendimiento de los algoritmos de aprendizaje automático en el problema de predicción de fallas de software.

5. Experimenta

El objetivo de los experimentos es predecir fallas de software utilizando ANFIS, ANN y SVM y comparar los resultados de

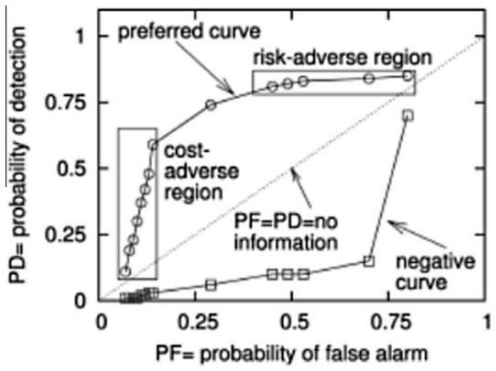

Fig. 4. Curva ROC típica [20, 21].

tabla 1
Valores ROC-AUC para todos los conjuntos de datos y métodos con 4 métricas McCabe.

		AUC para SVM	AUC para ANN	AUC para ANFIS
1er Grupo	Tren	0,6726	0,7513	0.7551
	Prueba	0,5939	0,6543	0.6525
2do Grupo	Tren	0,6644	0,7621	0.7509
	Prueba	0.596	0.6487	0.6433
3er grupo	Tren	0,6582	0,7318	0.739
	Prueba	0,6223	0,6812	0.6856
4to Grupo	Tren	0,6165	0,6921	0.6947
	Prueba	0,7788	0,8727	0.8457
5to Grupo	Tren	0,6392	0,7279	0.7216
	Prueba	0,6434	0,7603	0.7564

los modelos. Para lograr estos objetivos, tres modelos diferentes fueron construidos utilizando MATLAB 7.13.0 (R2011b) (MATLAB, 2009) ambiente. Luego, el proceso de predicción de fallas del software se realizó en el conjunto de datos de prueba y entrenamiento preparado derivado del repositorio PROMESA. Finalmente, los valores ROC-AUC de entrenamiento y prueba Se calculan y comparan conjuntos de cada modelo.

En los experimentos, se producen un total de 5 conjuntos de datos de entrenamiento y prueba. Los conjuntos de datos de trenes consisten en 12,123 casos y 2115 de ellos son defectuosos Significa que el 17,45% del conjunto de datos del tren son casos defectuosos. En los conjuntos de datos de prueba, hay un total de 3000 casos y 550 de ellos son defectuoso. En otras palabras, cada conjunto de datos de prueba tiene un 18,33 % de casos defectuosos. Como resultado, la cantidad y el porcentaje de casos defectuosos son similares entre diferentes conjuntos (prueba o tren).

En los primeros experimentos, todas las métricas de McCabe (loc, v(g), ev(g) y iv (g)) se utilizan para ser compatibles con los experimentos anteriores presentados en la literatura (Tabla 1). Además, el parámetro "ev(g)" se extrae del conjunto métrico de McCabe, y los segundos experimentos se organizan con los parámetros restantes (loc, v(g) y iv(g)) (Cuadro 2). La razón de esta extracción se basa en las observaciones en los casos de culpa. Cuando los casos defectuosos se consideran juntos, el efecto de Los parámetros "ev(g)" no son sensatos. En otras palabras, hay casi no hay cajas defectuosas que se puedan separar de las cajas sin fallas debido únicamente a su valor "ev(g)". El parámetro de "loc" puede ser dado como ejemplo opuesto al parámetro de "ev(g)". Allá Hay muchos casos defectuosos que tienen diferentes valores "loc" de los casos libres de fallas incluso si tienen valores "v(g), ev(g) y iv(g)" similares. Esta observación requiere la discusión sobre el uso de "ev(g)" en el modelado, porque puede no ser esencial mientras que el otro Se utilizan métricas de McCabe. Como se sabe, el uso de insumos no esenciales Los parámetros pueden afectar negativamente el rendimiento de la predicción. y provocar el aumento de la complejidad de los modelos. tabla 1 enumera todos los resultados de rendimiento logrados en los primeros experimentos que utilizan 4 parámetros de entrada en los modelos ANN, ANFIS y SVM. Además, los resultados de rendimiento obtenidos de los segundos experimentos se dan en la Tabla 2.

Tabla 2
Valores ROC-AUC para todos los conjuntos de datos y métodos con 3 métricas McCabe.

		AUC para SVM	AUC para ANN	AUC para ANFIS
1er Grupo	Tren	0,6730	0,7533	0.7541
	Prueba	0,5931	0,6475	0.6515
2do grupo	Tren	0,6644	0,7402	0.7508
	Prueba	0,5964	0,6457	0.6437
3er grupo	Tren	0,6586	0,7262	0.7382
	Prueba	0,6219	0,6741	0.6904
4to Grupo	Tren	0.6160	0.6944	0.6944
	Prueba	0,7795	0,8685	0.8573
5to Grupo	Tren	0,6396	0,7450	0.7212
	Prueba	0,6449	0,7350	0.7579

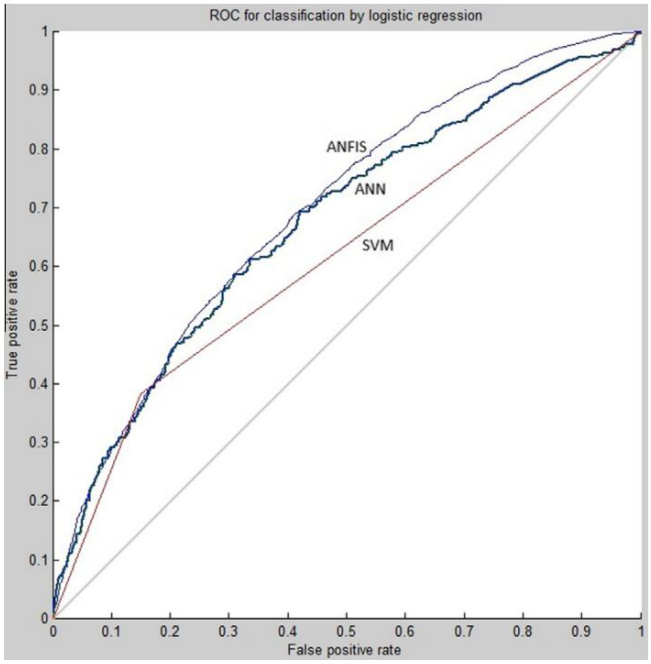

Fig. 5. Curvas ROC obtenidas del conjunto de trenes-4 (4 entradas).

Se dice que los modelos construidos en ambos experimentos no se ajustan demasiado al conjuntos de datos de entrenamiento porque los valores AUC de los conjuntos de entrenamiento no son mucho mayor que los valores AUC de los conjuntos de prueba (Tablas 1 y 2). En en otras palabras, los modelos tienen la capacidad de generalización para compararlos entre sí.

Todos los valores de AUC (Tablas 1 y 2) son aceptables en esta investigación área, y los resultados obtenidos de los métodos SVM y ANN están de acuerdo con los resultados presentados en la literatura. En De hecho, se utilizan para aclarar la idoneidad del ANFIS. método para el problema de predicción de fallos de software. En otras palabras, el rendimiento del ANFIS debe compararse con los demás metodos experimentales. El modelo con SVM tiene el peor desempeño, mientras que ANN y ANFIS predicen fallas con mayor éxito (Tablas 1 y 2). De hecho, los mejores resultados de la primera los experimentos se obtienen del set-4 (SVM: 0.7788, ANN: 0.8724, ANFIS: 0,8457) y el conjunto 4 es el conjunto más exitoso también para los segundos experimentos (SVM: 0,7795 ANN: 0,8685 ANFIS: 0,8573). Esto significa, set-4 tiene la mayor capacidad de representación, y el más adecuado set produce los modelos predictivos que tienen la mayor capacidad de generalización. Cuando se comparan los resultados de ANN y ANFIS, la cercanía entre todos los resultados se observa (Cuadros 1 y 2).

Las curvas ROC obtenidas en los primeros experimentos con 4 entradas se muestran en las Figs. 5 y 6 para una comparación visual. Estos resultados indican que ANN y ANFIS tienen un gran éxito y que sus

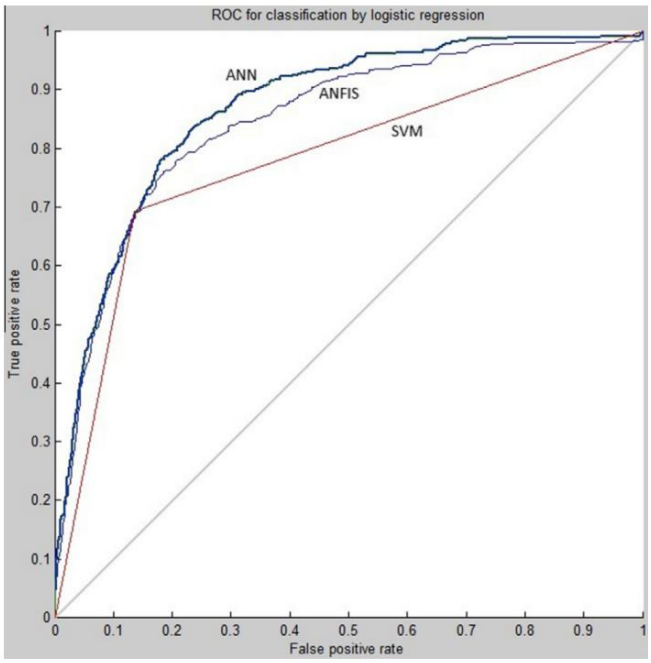

Fig. 6. Curvas ROC obtenidas del conjunto de prueba-4 (4 entradas).

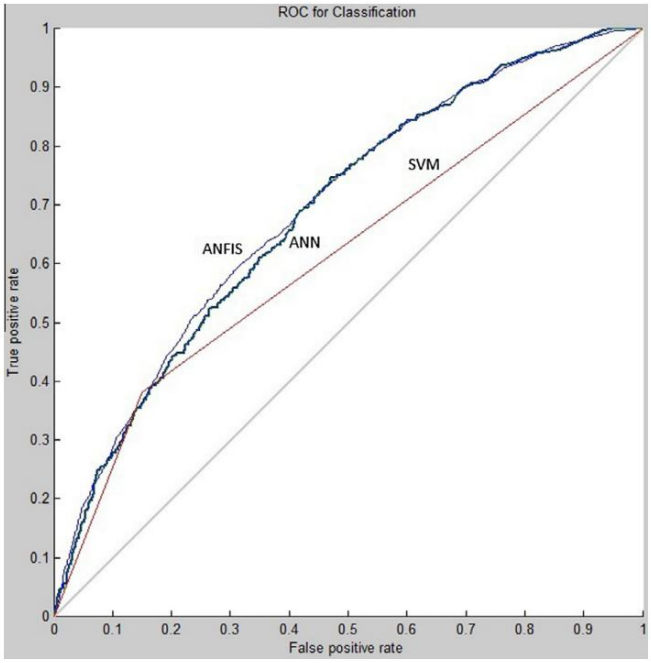

Fig. 7. Curvas ROC obtenidas del conjunto de trenes-4 (3 entradas).

Las actuaciones están cerca unas de otras. De acuerdo con estos resultados, el mejor desempeño pertenece al modelo construido usando ANN. Sin embargo, se debe considerar que ANFIS sigue de cerca los resultados de ANN. El modelo que utiliza SVM muestra el peor rendimiento, pero todos los valores de AUC son aceptables en esta área de investigación.

Las curvas ROC obtenidas en los segundos experimentos con 3 entradas se muestran en las Figs. 7 y 8. Los valores de rendimiento de los métodos ANN y ANFIS están más cerca en estos experimentos que en los primeros. En otras palabras, la complejidad de los modelos ANFIS se reduce mediante la extracción del parámetro "ev(g)" y este cambio proporciona un aumento en el rendimiento de los modelos ANFIS y también de SVM.

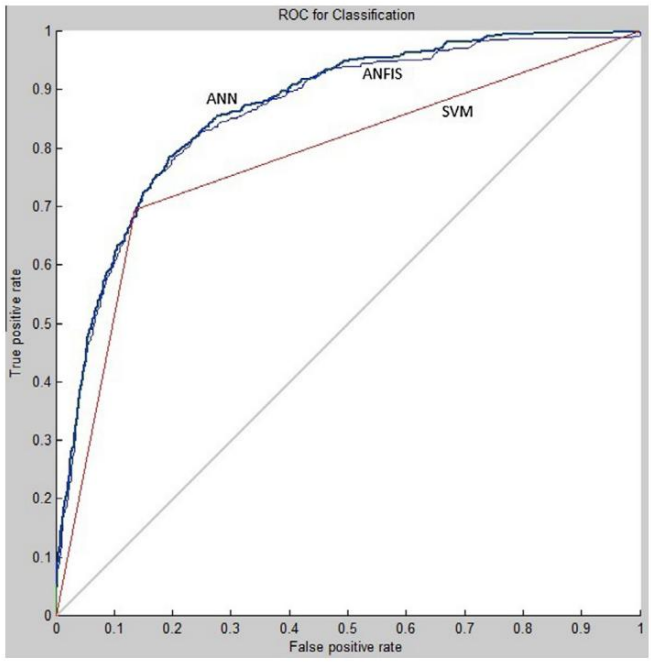

Fig. 8. Curvas ROC obtenidas del conjunto de trenes-4 (3 entradas).

6. Conclusión

La predicción de fallas de software se considera una habilidad muy importante cuando se planifica un proyecto de software y se necesita un esfuerzo mucho mayor para resolver este problema complejo utilizando un enfoque no lineal Los métodos de aprendizaje automático, que se han empleado para predecir las fallas del software, tienen la capacidad de aprender relaciones ocultas incrustadas en los datos. De hecho, dependen de las características de los datos y, cuando los datos cambian, el modelo puede perder su rendimiento drásticamente. En la predicción de fallas de software, el cambio de datos significa el cambio en el tamaño del proyecto, el dominio o la arquitectura del software y es indispensable. Para construir el modelo que es menos sensible al cambio en los datos, el uso del conocimiento experto con los datos en la etapa de aprendizaje se convierte en una sugerencia plausible. En este punto, el enfoque híbrido de ANFIS se vuelve notable. La principal motivación de este estudio es triple: (1) proponer ANFIS como uno de los métodos de solución para el problema de predicción de fallas de software (2) comparar y evaluar el rendimiento predictivo de ANFIS con ANN y SVM, que son los métodos más empleados en este problema (3) para hacer una discusión sobre el conjunto de parámetros de McCabe y reducir la cantidad de parámetros que se pueden usar durante el modelado.

El rendimiento alcanzado por ANFIS con 3 parámetros es 0.8573 y este resultado puede convencernos de que ANFIS es un método adecuado para la predicción de fallas de software. A diferencia de los métodos de aprendizaje automático, ANFIS permite que los expertos manejen la vaguedad de los datos de manera más directa y se puede decir que los expertos en el dominio de la ingeniería de software pueden ser tan útiles como los datos en el proceso de aprendizaje. Para generalizar esta declaración, se deben realizar más experimentos con diferentes conjuntos de datos y se deben extraer las propiedades arquitectónicas comunes de los modelos ANFIS.

En la predicción de fallas de software, las métricas de software se proponen como un conjunto (p. ej., McCabe, Healstead, OOMetrics) y el conjunto completo se usa en el modelado con algunos métodos predictivos. En este estudio utilizamos las métricas de McCabe y propusimos el uso de 3 parámetros (loc, v(g) e iv(g)) como resultado del proceso de modelado de ANFIS.
Se deben considerar los parámetros de los otros conjuntos de métricas, pero el punto más desafiante es que las métricas para la predicción de fallas de software se proponen con la perspectiva de códigos fuente completos de

se va a desarrollar el proyecto. Es decir, se ignora el uso de bibliotecas, paquetes existentes y capacidades de marco en el proceso de desarrollo de software. Sin embargo, el mal uso de estos códigos externos puede causar fallas y las fallas pueden provenir de dependencias incorrectas entre los módulos de software internos y externos. Este punto estará enfocado a acercar las métricas del software a los entornos de desarrollo.

Es importante hacer que los sistemas de predicción de fallas de software sean más prácticos, porque el costo de corregir fallas en las últimas fases del ciclo de vida del desarrollo de software aumenta cada vez más. Como resultado, existe la necesidad de un clasificador que pueda usarse en las primeras etapas del tiempo de desarrollo del proyecto para clasificar el módulo, ya sea que esté defectuoso o no. En realidad, los investigadores y los profesionales del software pueden utilizar el modelo propuesto aquí en las primeras fases del desarrollo. El momento más temprano del ciclo de desarrollo de software en el que el modelo puede usarse como clasificador depende de la capacidad de generalización del modelo. En otras palabras, los modelos más generalizables pueden ser útiles en las primeras fases del desarrollo y ANFIS tiene la mayor ventaja de ser general debido al conocimiento experto que se le proporciona. Sin embargo, se deben realizar investigaciones similares con proyectos de software de gran tamaño y diferentes tipos para obtener resultados generalizados.

Referencias

Almsadi, I. y Najadat, H. (2011). Evaluar el cambio de comportamiento de fallas de software con atributos de conjuntos de datos basados en la correlación categórica. Avances en software de ingeniería, 42(8), 535–546.

Catal, C. (2011). Predicción de fallas de software: una revisión de la literatura y tendencias actuales. Sistemas expertos con aplicaciones, 38(4), 4626–4636.

Catal, C. (2012). Métricas de evaluación de desempeño para estudios de predicción de fallas de software. Acta Polytechnica Hungarica, 9(4), 193–206.

Catal, C. y Diri, B. (2009). Investigar el efecto del tamaño del conjunto de datos, los conjuntos de métricas y las técnicas de selección de características en el problema de predicción de fallas de software. Ciencias de la información , 179(8), 1040–1058.

Catal, C., Sevim, U. y Diri, B. (2011). Desarrollo práctico de una herramienta de predicción de fallas de software basada en eclipse utilizando el algoritmo Naive Bayes. Sistemas expertos con aplicaciones, 38(3), 2347–2353.

Cotroneo, D., Natella, R. y Pietrantuono, R. (2013). Predecir errores relacionados con el envejecimiento utilizando métricas de complejidad de software. Evaluación del desempeño, 70(3), 163–178.

Couto, C., Pires, P., Valente, MT, Bigonha, RS y Anquetil, N. (2014). Predicción de defectos de software con pruebas de causalidad. Revista de Sistemas y Software, 93, 24–41.

Czibula, G., Marian, Z. y Czibula, IG (2014). Predicción de defectos de software mediante minería de reglas de asociación relacional. Ciencias de la Información, 264, 260–278.

Daoming, G. y Jie, C. (2006). ANFIS para predicción de limpieza por chorro de agua a alta presión. Tecnología de superficies y revestimientos, 201(3), 1629–1634.

Dejaeger, K., Verbraken, T. y Baesens, B. (2013). Hacia modelos comprensibles de predicción de fallas de software utilizando clasificadores de red bayesianos. IEEE Transactions on Software Engineering, 39(2), 237–257.

Gondra, I. (2008). Aplicación del aprendizaje automático a la predicción de la propensión a fallas del software . Revista de Sistemas y Software, 81(5), 186–195.

Hecht-Nielsen, R. (1987). Teorema de existencia de redes neuronales de mapeo de Kolmogorov. En Actas de la primera conferencia internacional IEEE sobre redes neuronales (págs. 11–14). San Diego, CA, Estados Unidos.

Hu, QP, Xie, M., Ng, SH y Levitin, G. (2007). Modelado robusto de redes neuronales recurrentes para detección de fallas de software y predicción de corrección. Ingeniería de confiabilidad y seguridad del sistema, 92(3), 332–340.

Jang, JSR (1993). ANFIS: Sistema de inferencia difusa basado en redes adaptativas. IEEE Transactions on Systems, Man & Cybernetics, 23(3), 665–685.

Kanellopoulas, I. y Wilkinson, GG (1997). Estrategias y mejores prácticas para la clasificación de imágenes de redes neuronales. Revista Internacional de Percepción Remota, 18(4), 711–725.

Kanmani, S., Uthariaraj, VR, Sankaranarayanan, V. y Thambidurai, P. (2004). Predicción de calidad de software orientada a objetos utilizando redes neuronales de regresión general . Notas de ingeniería de software SIGSOFT, 29(5), 1–6.

Khoshgoftaar, TM, Allen, EB, Hudepohl, JP y Aud, SJ (1997). Aplicación de redes neuronales al modelado de calidad de software de un sistema de telecomunicaciones muy grande. Transacciones IEEE en redes neuronales, 8(4), 902–909.

Khosgoftaar, TM, Xiao, Y. y Gao, K. (2014). Evaluación de la calidad del software mediante un clasificador multiestrategia. Ciencias de la Información, 259, 555–570.

Laradji, IH, Alshayeb, M. y Ghouti, L. (2014). Predicción de defectos de software utilizando el aprendizaje conjunto en características seleccionadas. Tecnologías de la información y el software. http://dx.doi.org/10.1016/j.infsof.2014.07.005.

Lee, KY y Park, JH (1992). Pronóstico de carga a corto plazo usando una neural artificial red. IEEE Transactions on Power Systems, 7(1), 124–132.

Lo, SP (2003). Un sistema de inferencia difusa basado en red adaptativa para la predicción de la rugosidad de la superficie de la pieza de trabajo en el fresado final. Revista de Tecnología de Procesamiento de Materiales, 142(3), 665–675.

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C. y Musílek, P. (2004). Predicción de fallas en software orientado a objetos utilizando técnicas de redes neuronales. En Actas de la conferencia InTech (págs. 27–34). Houston, TX, Estados Unidos.

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C. y Musílek, P. (2006). MASP: un modelo mejorado de identificación de tipos de fallas en la ingeniería de software orientada a objetos. Revista de inteligencia computacional avanzada e informática inteligente, 10(3), 312–322.

Malhotra, R. (2014). Análisis comparativo de métodos estadísticos y de aprendizaje automático para predecir módulos defectuosos. Computación blanda aplicada, 21, 286–297.

MATLAB (2009). Guía del usuario Versión 7.8, R2009a. MathWorks Co., EE . UU. <http://www.mathworks.com/help/stats/svmtrain.html >, MathWorks Co., EE. UU., fecha de visita 12.03.2013.

Menzies, T., DiStefano, J., Orrego, A. y Chapman, R. (2004). Evaluación de predictores de defectos de software. En Actas del taller de modelos predictivos de software. chicago

Menzies, T., Greenwald, J. y Frank, A. (2007). Atributos de código estático de minería de datos para aprender predictores de defectos. IEEE Transactions on Software Engineering, 32(1), 2–13.

Najah, AA, El-Shafie, A., Karim, OA y Jaafar, O. (2012). Modelo de predicción de la calidad del agua que utiliza el modelo integrado wavelet-ANFIS con validación cruzada. Informática neuronal y aplicaciones, 21(5), 833–841.

Negnevitsky, M. (2005). Inteligencia artificial una guía para sistemas inteligentes (2ª ed.). Harlow, Inglaterra: Addison-Wesley.

Oyetoyan, TD, Cruzes, DS y Conradi, R. (2013). Un estudio de las dependencias cíclicas en el perfil de defectos de los componentes de software. Revista de Sistemas y Software, 86(12), 3162–3182.

Pérez, JA, González, M. y Dopico, D. (2010). Modelado adaptativo neurofuzzy ANFIS de tratamientos superficiales con láser. Informática neuronal y aplicaciones, 19(1), 85–90.

Pradhan, B., Sezer, EA, Gokceoglu, C. y Buchroithner, MF (2010). Mapeo de susceptibilidad a deslizamientos de tierra mediante un enfoque neurodifuso en un área propensa a deslizamientos de tierra (Cameron Highland, Malasia). IEEE Transactions on Geoscience and Remote Sensing, 48(12), 4164–4177.

Conjuntos de datos públicos del repositorio de ingeniería de software Promise (2013). <http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff >.

Rodríguez, D., Ruiz, R., Riquelme, JC y Harrison, R. (2013). Un estudio de enfoques de descubrimiento de subgrupos para la predicción de defectos. Tecnología de la información y el software , 55(10), 1810–1822.

Sayyad, S. y Menzies, T. (2005). El repositorio PROMISE de bases de datos de ingeniería de software. Canadá: Universidad de Ottawa, <http://promise.site.uottawa.ca/ SERepository>.

Scanniello, G., Gravino, C., Adrian, M. y Menzies, T. (2013). Predicción de fallas a nivel de clase usando agrupamiento de software. En Actas de la 28.ª conferencia internacional sobre ingeniería de software automatizada (págs. 640–645), Silicon Valley, CA.

Sen, S., Sezer, EA, Gokceoglu, C. y Yagiz, S. (2012). Sobre estrategias de muestreo para datos pequeños y continuos con el modelado de programación genética y sistema de inferencia neuro-borroso adaptativo. Journal of Intelligent & Fuzzy System, 23(6), 297–304.

Sezer, EA, Pradhan, B. y Gokceoglu, C. (2011). Manifestación de un modelo difuso neuro adaptativo en el mapeo de susceptibilidad a deslizamientos: valle de Klang, Malasia. Sistemas expertos con aplicaciones, 38(7), 8208–8219.

Tomás, J. (1976). McCabe, una medida de complejidad. Transacciones IEEE en software Ingeniería, 2(4), 308–320.

Thwin, MMm y Quah, T. (2003). Aplicación de redes neuronales para la predicción de la calidad del software utilizando métricas orientadas a objetos. En Actas de la 19.ª conferencia internacional sobre mantenimiento de software (113–122). Ámsterdam, Países Bajos.

Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M. y Haesen, R. (2008). Repositorios de software de minería para modelos comprensibles de predicción de fallas de software. Revista de Sistemas y Software, 81(5), 823–839.

Vapnik, V. (1998). El método del vector de soporte para la estimación de funciones. En JAK Suykens & J. Vandewalle (Eds.), Modelado no lineal: Técnicas avanzadas de caja negra (págs. 55–85). Boston: Editores académicos de Kluwer.

Wang, Q., Yu, B. y Zhu, J. (2004). Extraiga reglas del modelo de predicción de la calidad del software basado en la red neuronal. En Actas de la 16.ª conferencia internacional IEEE sobre herramientas con inteligencia artificial (págs. 191–195). Boca Raton, FL, EE. UU.: IEEE Computer Society.

Xing, F., Guo, P. y Lyu, MR (2005). Un método novedoso para la predicción temprana de la calidad del software basado en una máquina de vectores de soporte. En Actas de la 16.ª conferencia internacional IEEE sobre ingeniería de confiabilidad del software (págs. 213–222). Chicago, IL.

Yagiz, S., Sezer, EA y Gokceoglu, C. (2012). Redes neuronales artificiales y técnicas de regresión no lineal para evaluar la influencia de los ciclos de durabilidad de slake en la predicción de la resistencia a la compresión uniaxial y el módulo de elasticidad de las rocas carbonatadas. Revista internacional de métodos numéricos y analíticos en geomecánica, 36 (14), 1636–1650.

Zhou, Y. y Leung, H. (2006). Análisis empírico de métricas de diseño orientadas a objetos para predecir fallas de alta y baja severidad. IEEE Transactions on Software Engineering, 32(10), 771–789.

Zimmermann, T., Premraj, R. y Zeller, A. (2007). Predicción de defectos para Eclipse. En Actas del 3er taller internacional sobre modelos predictores en ingeniería de software, 20 al 26 de mayo (pág. 9). Minneapolis, MN, Estados Unidos.