Universidad de Granada

Escuela Técnica Superior de Ingenierías Informática y de Comunicaciones
Departamento de Ciencias de la Computación e Inteligencia Artificial
Programa Oficial de Doctorado en
"Tecnologías de la Información y la Comunicación"

Modelo para la estimación del esfuerzo de desarrollo en tareas de ingeniería de proyectos de software empleando aprendizaje automático

---

Tesis Doctoral

Héctor Raúl Velarde Bedregal

Granada, enero de 2017

Editor: Universidad de Granada. Tesis Doctorales
Autor: Héctor Raúl Velarde Bedregal
ISBN: 978-84-9163-136-1
URI: http://hdl.handle.net/10481/45264

Universidad de Granada

Escuela Técnica Superior de Ingenierías Informática y de Comunicaciones
Departamento de Ciencias de la Computación e Inteligencia Artificial
Programa Oficial de Doctorado en
"Tecnologías de la Información y la Comunicación"

Modelo para la estimación del esfuerzo de desarrollo en tareas de ingeniería de proyectos de software empleando aprendizaje automático

---

MEMORIA QUE PRESENTA

Héctor Raúl Velarde Bedregal

PARA OPTAR AL GRADO DE DOCTOR

DIRECTOR DE TESIS

Dr. Jorge Casillas Barranquero

Granada, enero de 2017

El doctorando Héctor Raúl Velarde Bedregal y el director de la tesis Jorge Casillas Barranquero. Garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección del director de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, enero de 2017

Director de la Tesis	Doctorando
	
Fdo. Jorge Casillas Barranquero	Fdo. Héctor Raúl Velarde Bedregal

A la memoria de Rosa y Moisés, mis queridos padres, por su confianza, apoyo y enseñanzas durante todo el tiempo que me acompañaron en esta vida y por protegerme y orientarme desde el cielo en todos los proyectos que emprendo.

A Doris, mi esposa y compañera, por su aliento, soporte, sacrificio y por impulsarme en todo el tiempo que duro este trabajo.

A Héctor Rafael, mi hijo, por sus sacrificios y porque ha sabido comprender, desde su perspectiva, el esfuerzo de un padre doctorando.

Agradecimientos

Este proyecto, si bien ha necesitado de mucha voluntad y entrega del autor, no hubiese sido posible sin la colaboración desprendida de varias personas que han sido un soporte en momentos de desazón y desmoralización.

Mi primer agradecimiento va para mi Director de Tesis, el Dr. Jorge Casillas Barranquero, al que considero un amigo, el cual siempre me alentó, brindándome su apoyo constante, paciencia, conocimiento y compromiso incondicional. A mi cotutora la Dra. Ana María García Pérez por su ayuda y compromiso. A mi compañero, el Dr. Cosme Santisteban Toca, quien no vaciló en ayudarme en los momentos más difíciles, y que sin su aporte la culminación de esta tesis no hubiese sido posible. Al claustro docente de las Universidades Andaluzas y Cubanas que impartieron los cursos del Doctorado. A la Junta de Andalucía y a la Asociación Universitaria Iberoamericana de Postgrado, por darme la oportunidad de ser parte de este programa. A mi centro de trabajo, la Universidad Católica de Santa María de Arequipa, Perú, por todas las facilidades concedidas.

Finalmente, quisiera agradecer a todas las personas que han compartido conmigo el desarrollo de esta tesis doctoral, proporcionándome en todo momento su colaboración y amistad.

Resumen

Dentro del contexto de las inexactitudes de las primeras etapas de un proyecto de software se encuentra que uno de los aspectos más críticos es la estimación del esfuerzo. En la actualidad, el desarrollo ágil de software ha ganado en popularidad, sobre las metodologías tradicionales. A pesar que existen múltiples estudios donde se aplican técnicas para el desarrollo ágil, éstas no resultan efectivas en este entorno de desarrollo, estas técnicas como promedio, tienden a subestimar el esfuerzo de desarrollo ágil en un 55% y a sobre estimarlo en un 25,5%, razón por la cual, la estimación del esfuerzo basada en el desarrollo ágil se sigue considerando un reto. Es por ello que el objetivo de la presente investigación, es el de definir un modelo para la estimación del esfuerzo de desarrollo de software empleando técnicas de aprendizaje automático y a partir del análisis e interpretación del razonamiento que sigue el modelo, entrenar al responsable de proyectos para que este optimice el tiempo, rediseñando las características de software de sus tareas de ingeniería. Como resultado, se propuso un nuevo método para la estimación del esfuerzo de desarrollo de software, especialmente diseñado para metodologías ágiles basado en tareas de ingeniería, para ello, se introdujo una nueva base y se sugirieron nuevas métricas de evaluación del desempeño del equipo de trabajo y de evaluación del proyecto. Además, se propusieron dos nuevos algoritmos de estimación del esfuerzo de desarrollo de proyectos de software: EEpred, basado en la combinación de clasificadores en serie; y ETTpred, basado en tareas de ingeniería y el árbol de regresión M5P, diseñado especialmente para la estimación en el desarrollo ágil. El algoritmo EEpred es un multiclasificador que funciona en serie y se basa en el conteo de líneas de código fuente, primero realizando una clasificación granular basada en un árbol de decisión, donde determina si el proyecto será orgánico, semi-libre o empotrado; posteriormente, se realiza una predicción fina basada en árboles de regresión y

devolviendo el valor del esfuerzo estimado. EEpred, fue sometido a un proceso de validación interna que permitió analizar su robustez y capacidad de generalización, así como el error en la predicción, lo cual demostró que es capaz de predecir el esfuerzo de proyectos orgánicos y semi-libre, con una precisión de hasta el 78%. El algoritmo ETTpred, fue sometido a un proceso de validación interna que permitió analizar su robustez y capacidad de generalización, así como el error en la predicción, demostrandose que el método propuesto es capaz de predecir el esfuerzo con una precisión de hasta el 85% y un error relativo de 47,24%. De igual forma, se demostró la superioridad de la estimación del esfuerzo de desarrollo de software basado en tareas de ingeniería y empleado en metodologías ágiles, sobre los basados en conteo de líneas de código y puntos de función. La principal ventaja de EEpred y ETTpred, ante el resto de los modelos estudiados, es que ambos presentan un mecanismo de interpretación de los resultados sobre la base de un conjunto de reglas semánticas que son de fácil entendimiento, lo cual los convierte en modelos entendibles para los expertos en el área de aplicación.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

A. Motivación

Dentro de los errores de las etapas iniciales de un proyecto de software se encuentra que uno de los aspectos más críticos es la estimación del esfuerzo, la cual proporciona valores aproximados, por lo que se acepta un grado de incertidumbre, contar con dicha información en etapas tempranas de desarrollo, permite tomar decisiones importantes antes de iniciar cualquier proyecto. Estimar es una tarea complicada debido a la gran cantidad de factores que influyen en el valor de la variable objeto de la predicción. Para poder realizar predicciones sobre características del proyecto actual es necesario disponer de información obtenida en proyectos pasados sobre las variables a estimar y los factores que las afectan, el porcentaje de error en la estimación es lo que determina el éxito o fracaso del proyecto (Dejaeger, Verbeke, Martens, y Baesens, 2012a).

Hasta la actualidad han sido desarrollados múltiples modelos de predicción del esfuerzo. Los modelos de predicción basados en técnicas de aprendizaje automático han demostrado ser superiores a los modelos tradicionales (Wen, Li, Lin, Hu, y Huang, 2012). Dentro de los métodos de aprendizaje más empleados se encuentran las redes neuronales, algoritmos genéticos, árboles de decisión y regresión, algoritmos de agrupamiento, combinación de clasificadores, entre otras técnicas de minería de datos (Algabri, Saeed, Mathkour y Tagoug, 2015; Almache, Raura, Ruiz y Fonseca, 2015; Borade y Khalkar, 2013a; Dejaeger et al., 2012a; Kad y Chopra, 2012; Minku y Yao, 2013; Najadat, Alsmadi y Shboul, 2012a; Waghmode y Kolhe, 2014; Wen et al., 2012). En comparación con otros escenarios en los que han sido aplicadas exitosamente las técnicas de aprendizaje automático, la estimación del esfuerzo de desarrollo de software posee muchos otros retos como: bases de datos pequeñas y desactualizadas, información

imprecisa, incompleta y altamente subjetiva, datos cualitativos, entre otros, además la mayoría de éstas bases de datos se basan en información obtenida empíricamente y no como parte de un proceso sistemático (Nguyen-Cong y De Tran-Cao, 2013; Dejaeger et al., 2012; Saroha y Sahu, 2015).

Sin embargo, todos estos modelos se basan en proyectos de software desarrollados sobre la base de metodologías predictivas o tradicionales y no están enfocados a metodologías ágiles (Nguyen-Cong y De Tran-Cao, 2013; Popli y Chauhan, 2014a). Tampoco logran modelos capaces de predecir el esfuerzo con un precisión superior al 25% (Popli y Chauhan, 2014), ni cuentan con una base de conocimiento entendibles para los expertos en el área de aplicación (Nguyen-Cong y De Tran-Cao, 2013; Dejaeger et al., 2012; Saroha y Sahu, 2015).

Según *The Standish Group* en su publicación *Chaos Report 2015* (http://www.infoq.com/articles/standish-chaos-2015), en el año 2015, las metodologías ágiles han dado lugar a más proyectos que concluyeron a tiempo y dentro del presupuesto (39% *vs* 11%), con menos cancelaciones (9% *vs* 29%) y discutidos (52% *vs* 60%) que las metodologías de cascada. Ahora bien, cuando subimos en tamaño hacia proyectos grandes la diferencia se vuelve mucho más grande.

B. Planteamiento

Un gran error en la estimación del esfuerzo puede ser lo que marque la diferencia entre beneficios y pérdidas. Se precisa que, como promedio, la estimación difiere en un 30% del esfuerzo requerido (Basten y Sunyaev, 2011), este, es un proceso que depende de múltiples factores, donde el más importante es la experiencia del estimador.

Cada vez es mayor el número de aportes en la creación de modelos para la estimación del esfuerzo de desarrollo de software, lo cual demuestra un interés creciente de la comunidad científica en estos temas, la mayoría de estos modelos se basan en proyectos de software desarrollados sobre la base de metodologías predictivas o tradicionales y no están enfocadas a metodologías ágiles, sin embargo, aún no existen

modelos estandarizados, capaces de predecir el esfuerzo con una precisión superior al 25% (Popli y Chauhan, 2014), ni cuentan con una base de conocimiento entendibles para los expertos en el área de aplicación (Dejaeger, Verbeke, Martens, y Baesens, 2012).

En la actualidad, el desarrollo ágil de software ha ganado en popularidad, sobre las metodologías tradicionales. A pesar que existen múltiples estudios donde se aplican técnicas para el desarrollo ágil, éstas no resultan efectivas en este entorno de desarrollo. Como promedio, estas técnicas tienden a subestimar el esfuerzo de desarrollo ágil en un 55% y a sobre estimarlo en un 25,5% (Britto, Mendes, y Borstler, 2015), razón por la cual, la estimación del esfuerzo basada en el desarrollo ágil se sigue considerando un reto.

La propuesta de esta investigación se sustenta en las siguientes razones:

- Imprecisiones en los modelos de estimación del esfuerzo de desarrollo y las consecuencias de estas imprecisiones.

- No contar con un modelo predictivo para nuevas aplicaciones y que sea generado a partir de datos propios provenientes de proyectos de software actuales (de contexto cambiante y desarrollados con metodologías ágiles).

- No existir un modelo estandarizado de estimación del esfuerzo, que satisfaga todas las metodologías de desarrollo de forma efectiva.

- Inconveniente de que, pese a que los modelos predictivos de caja negra proporcionan muy buenos resultados de estimación del esfuerzo de desarrollo, no se puede interpretar su razonamiento.

Es por ello que, el **problema** que afronta esta investigación es la de no contar en la actualidad con un modelo estandarizado de estimación del esfuerzo, que satisfaga todas las metodologías de desarrollo de forma efectiva. Por otra parte, los “dataset” utilizados para la estimación son obsoletos y el número de proyectos en estos conjuntos de datos son pequeños, además los trabajos presentados solo se refieren a la estimación del esfuerzo y no a la interpretación del modelo con la finalidad de que el responsable de

proyectos pueda entrenarse y de esta manera optimizar el tiempo, rediseñando las características de las variables intervinientes en este.

Esta investigación se **enmarca** en el proceso de estimación del esfuerzo de desarrollo de software. Específicamente en el empleo de tareas de ingeniería y aprendizaje automático que permitan esclarecer dicho proceso.

C. Objetivos

El **objetivo general** de esta investigación es el de definir un modelo para la estimación del esfuerzo de desarrollo de software empleando técnicas de aprendizaje automático y a partir del análisis e interpretación del razonamiento que sigue el modelo, entrenar al responsable de proyectos para que este optimice el tiempo, rediseñando las características de software de sus tareas de ingeniería.

Este objetivo se puede dividir en otros más **específicos** como se plantea a continuación:

1. Caracterizar el estado del arte de los trabajos sobre aplicaciones de técnicas de minería de datos supervisadas que se centren en la creación de modelos de estimación de esfuerzo de desarrollo, tamaño del software, esfuerzo de mantenimiento, etc.

2. Seleccionar bases de datos públicas, extraídas de repositorios gratuitos.

3. Caracterizar las variables necesarias para la estimación a partir de elementos de COCOMO (Boehm et al., 2000) y agregando otras a partir de la experiencia de los equipos de desarrollo.

4. Crear una base de datos a partir de las tareas de ingeniería, obtenidas de aplicaciones de software de base, implementadas por los centros de desarrollo de software de la Universidad de Ciencias Informáticas de las provincias de Las Villas y Ciego de Ávila en Cuba.

5. Determinar la técnica más apropiada para nuestro objetivo general, utilizando algoritmos de aprendizaje, validando los resultados obtenidos a través de una evaluación cruzada.

6. Realizar un análisis de resultados para entender el razonamiento que sigue el modelo predictivo. con la finalidad de optimizar el tiempo de desarrollo y el diseño de las características de software de sus tareas de ingeniería.

D. Fundamentación de la investigación

Estimar es una tarea complicada debido a la gran cantidad de factores que influyen en el valor de la variable objeto de la predicción. Para poder realizar predicciones sobre características del proyecto actual es necesario disponer de información obtenida en proyectos pasados sobre las variables a estimar y los factores que les afectan, estos datos permiten establecer un modelo de relaciones entre dichas variables que constituye la base del proceso de estimación.

Con el fin de mejorar la precisión de la estimación, muchos investigadores han propuesto el uso de técnicas de aprendizaje desde 1990. Sin embargo, no se encuentran trabajos centrados en casos estudio o encuestas de la industria, y los “dataset” usados para sus experimentos son obsoletos y el número de proyectos en estos conjuntos de datos son pequeños (Nguyen-Cong y De Tran-Cao, 2013; Dejaeger et al., 2012b; Saroha y Sahu, 2015), además los trabajos presentados solo se refieren a la estimación del esfuerzo y no a la interpretación del modelo con la finalidad de optimizar el esfuerzo a partir del rediseño de las características de las variables intervinientes en este.

Lo dicho anteriormente justifica plenamente la realización de nuestro estudio, para el cual se contaría además con el apoyo del departamento Ciencias de la Computación e Inteligencia Artificial de la universidad de Granada y del Centro de Desarrollo de la Universidad de Ciencias Informáticas – Filial Santa Clara, Cuba; quien proporcionaría la información para generar el “dataset” necesario para nuestros experimentos, a partir de

órdenes de trabajo o peticiones (tareas de ingeniería), teniendo en cuenta que esta es la forma en la que dicho Centro planifica su labor de desarrollo.

La **importancia** de esta investigación se debe a que se proponen modelos de estimación del esfuerzo de desarrollo de software con capacidad explicativa, los cuales convierten el proceso de estimación del esfuerzo de desarrollo de software en un conjunto de reglas semánticas que facilitan la tarea de estimación del esfuerzo.

La **novedad** de la investigación está en que se propone un nuevo modelo de estimación del esfuerzo para proyectos de software utilizando tareas de ingeniería y árboles de regresión con capacidad explicativa, el cual presenta como ventaja fundamental, la robustez y capacidad de generalización, mediante un proceso de validación interna. Además, el mecanismo de interpretación, convierte el proceso de estimación del esfuerzo de desarrollo de software en un conjunto de reglas que faciliten esta tarea.

Las diversas aportaciones de investigadores en la aplicación de la Minería de Datos en la Ingeniería de software y más específicamente en el uso de técnicas supervisadas de aprendizaje automático en la estimación del esfuerzo, coste y tamaño de proyectos de desarrollo de software son muy representativas obteniéndose buenos resultados, razón por la cual se pretende con este trabajo obtener estimaciones confiables del esfuerzo en el desarrollo de Tareas de Ingeniería de Proyectos de Software, utilizando una técnica de aprendizaje automático, como son los algoritmos de árboles.

E. Principales contribuciones

Artículos en revistas (JCR):

- **H. Velarde**, C. Santiesteban, A. García, J. Casillas, “Software Development Effort Estimation based-on multiple classifier system and Lines of Code”, **Latin America Transactions August 2016.** IEEE, vol. 14, no. 8, pp. 3906-3912, 2016. ISSN: 1548-0992 (2016). Impacto: 0.326.

Publicación de los enlaces:

IEEE Xplore:

http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9907

Revista IEEE Latin America Transactions:

http://www.ewh.ieee.org/reg/9/etrans/esp/

- **H. Velarde**, C. Santiesteban, A. García, J. Casillas. "Analyzing the Effect of Variables in the Software Development Effort Estimation". **Latin America Transactions August 2016.** IEEE, vol. 14, no. 8, pp. 3798-3804, 2016. ISSN: 1548-0992 (2016). Impacto: 0.326.

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Estimación del esfuerzo de desarrollo de software basado en tareas de ingeniería: una aproximación para metodologías ágiles". XIV Congreso Nacional de Reconocimiento de Patrones - RECPAT. UCLV, Las Villas – Cuba. (2016). ISBN 978-959-312-205-4 (2016).

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Empleo de múltiples clasificadores para la Estimación del esfuerzo de desarrollo de software". XIV Congreso Nacional de Reconocimiento de Patrones - RECPAT. UCLV, Las Villas – Cuba. (2016). ISBN 978-959-312-205-4 (2016).

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Análisis de las métricas para la estimación del esfuerzo de desarrollo software". XIV Congreso Nacional de Reconocimiento de Patrones - RECPAT. UCLV, Las Villas – Cuba. (2016). ISBN 978-959-312-205-4 (2016).

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Estimación del esfuerzo de desarrollo de software basado en múltiples clasificadores y líneas de código". II Conferencia Científica Internacional UCIENCIA. UCI, La Habana – Cuba. (2016). ISBN 978-959-286-054-4 (2016).

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Estimación del esfuerzo en el desarrollo ágil: una aproximación basada en tareas de ingeniería y árboles de regresión". II Conferencia Científica Internacional UCIENCIA. UCI, La Habana – Cuba. (2016). ISBN 978-959-286-054-4 (2016).

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Análisis del efecto de las variables en la estimación del esfuerzo de desarrollo software". II Conferencia Científica Internacional UCIENCIA. UCI, La Habana – Cuba. (2016). ISBN 978-959-286-054-4 (2016).

- **Hector R. Velarde Bedregal**, Cosme E. Santiesteban Toca, Ana María García Pérez, Jorge Casillas Barranquero. "Software effort estimation based on engineering tasks: an appropriate methodology for agile development". 35th International Conference of the Chilean Computer Science Society. SCCC, Valparaíso – Chile. IEEE Conference Record #39569, ISBN 978-1-5090-3339-3 (2016).

- **Hector R. Velarde Bedregal**, Ana María García Pérez, Jorge Casillas Barranquero. "Estimación del esfuerzo de desarrollo de software". Seminario Internacional sobre Softcomputing. UCLV, Cuba. ISBN 978-959-250-525-4 (2009).

- **Hector R. Velarde Bedregal**, Ana María García Pérez, Jorge Casillas Barranquero. "Estimación del esfuerzo utilizando técnicas de minería de datos". IX Jornada Internacional de Ingeniería de Sistemas. UCSM, Arequipa - Perú. (2009).

F. Resumen

La memoria está organizada en cinco capítulos y dos apéndices. A continuación introducimos los contenidos de cada uno de ellos:

Capítulo 1: se abordan los conceptos básicos sobre la gestión y las metodologías de desarrollo de proyectos de software, temas referentes a la metodología de estimación del esfuerzo basado en tareas de ingeniería, los fundamentos de la estimación en el desarrollo de proyectos de software, se hace una valoración crítica de las diferentes

modelos de estimación. Se hace un estudio taxonómico de dichas técnicas y se describen los métodos de estimación del esfuerzo de desarrollo de software

Capítulo 2: se exponen las bases teóricas necesarias para el presente estudio, en primer lugar, se muestra una breve introducción al proceso de descubrimiento de conocimiento en bases de datos o KDD, las técnicas de minería de datos y sus herramientas. Después describiremos los diferentes métodos para la estimación de esfuerzo en el desarrollo de software, incluyendo los Árboles de clasificación y regresión

Capítulo 3: se formalizan los algoritmos propuestos, ETTpred (basado en tareas de ingeniería y árboles de regresión) y EEpred (basado en conteo de líneas de código y la combinación de clasificadores), proponiéndose una nueva metodología que puede catalogarse como de analogía, heurística, algorítmica y no paramétrica a la vez, debido a que se basa en tareas de ingeniería de proyectos ya resueltos y no calcula el esfuerzo a partir de los parámetros de entrada, sino que emplea algoritmos heurísticos.

Capítulo 4: en este se presenta la metodología empleada para evaluar los modelos propuestos, se muestran los estadígrafos, pruebas estadísticas y resultados experimentales para el análisis de la robustez y capacidad de generalización, así como del comportamiento del error. Adicionalmente, se expone el mecanismo de interpretación del modelo.

Capítulo 5: se muestran las conclusiones y recomendaciones, enfatizándose en los resultados alcanzados y proponiendo líneas de continuación del trabajo aquí presentado.

Finalizaremos la memoria con una recopilación bibliográfica que recoge aquellas referencias que hemos considerado relevantes e incluiremos dos anexos. El primero estará destinado a presentar los resultados de las pruebas estadísticas en orden de los rangos promedios de los modelos usados en la experimentación (TI, PF y LOC). El siguiente anexo muestra los resultados de las pruebas estadísticas en orden de los rangos promedios de los algoritmos utilizados en la experimentación (M5P, M5Rules, REPTree, Random Tree)

Capítulo 1
Gestión de Proyectos de Software y estado actual de las técnicas de estimación del esfuerzo de desarrollo de software

La gestión de proyectos busca las técnicas necesarias para planificar, organizar, supervisar y controlar proyectos de software. El objetivo de gestionar proyectos es tener un producto de alta calidad.

La gestión de un proyecto de software se centra en tres partes como son:

- Personal:

El factor humano es importante en la ingeniería de software. Es importante tener la capacidad de gestión del personal con el fin de aumentar la preparación en la organización del software; ayudando a atraer, motivar y retener el talento necesario para mejorar su capacidad de desarrollo de software. En toda organización que alcanza la madurez en el área de gestión de personal tiene una mayor probabilidad de implementar unas eficaces prácticas de ingeniería de software, esto guía a que las organizaciones tengan un proceso de software maduro.

- Problema:

Se establecen los objetivos y se deben considerar soluciones alternativas e identificar las dificultades técnicas y de gestión. Con esta información es posible definir unas estimaciones razonables del costo, una valoración efectiva del riesgo, una subdivisión realista de las tareas del proyecto o una planificación del proyecto asequible que proporcione una indicación fiable del progreso.

El desarrollador de software y el cliente deben reunirse para definir los objetivos del proyecto, los cuales identifican las metas generales del proyecto sin considerar cómo se conseguirán.

Se identifican los datos primarios, funciones y comportamientos que caracterizan el problema, y se abordan estas características de una manera cuantitativa, también se consideran las soluciones alternativas, las que permiten a los gestores y a los profesionales seleccionar el mejor enfoque.

Analizar detalladamente los requisitos del software proporcionaría la información necesaria para las estimaciones, claro está que puede llevar semanas o meses, se debe examinar el problema al inicio del proyecto, con la finalidad de definirlo y delimitarlo. La primera gestión de un proyecto de software es determinar su ámbito, definir su contexto, que limitaciones se imponen; el objetivo de la información, que datos se obtienen del software; y la función, que realiza el software para transformar la información.

Para la descomposición del problema tiende a aplicarse la estrategia de divide y vencerás, cuando se enfrenta a problemas complejos, es decir el problema se parte en problemas más pequeños y con esto se busca que sean manejables.

- Proceso:

El proceso de software proporciona la estructura desde la que se puede establecer un detallado plan para el desarrollo del software, las actividades estructurales se pueden

aplicar a todos los proyectos de software, sin tener en cuenta su tamaño o complejidad, además permiten a las actividades estructurales adaptarse a las características del proyecto de software y a los requisitos del equipo del proyecto.

Entre las fases que caracterizan el proceso de software, se encuentran la definición, el desarrollo y el mantenimiento. El gestor del proyecto decide qué modelo es el más adecuado para desarrollar el proyecto, luego debe definir un plan preliminar, para luego continuar con la descomposición del proceso, se debe tener en cuenta la maduración del problema y la descomposición del proceso. Es importante invertir tiempo al principio del proyecto para tener un plan con buenas bases que se evidencian durante su desarrollo, permitiendo llevar el control de calidad de dicho proyecto.

1.1. Metodologías de desarrollo de proyectos Software

Existen numerosas propuestas metodológicas para proceso del desarrollo de software, las cuales podrían agruparse en metodologías predictivas o tradicionales y en metodologías adaptivas o ágiles.

1.1.1. Metodologías predictivas o tradicionales

Las “metodologías tradicionales” se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos.

Las metodologías tradicionales, están guiadas por una fuerte planificación durante todo el proceso de desarrollo, dónde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema. Algunas metodologías orientadas a objetos que utilizan la notación UML son:

- Rational Unified Process (RUP) (Britto, Usman, y Mendes, 2014).
- OPEN. (Britto et al., 2014).

- MÉTRICA (que también soporta la notación estructurada) (Britto et al., 2014).

Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales, aunque en el caso particular de RUP, por el especial énfasis que presenta en cuanto a su adaptación a las condiciones del proyecto (mediante su configuración previa a aplicarse) y realizando una configuración adecuada, podría convertirse Ágil (AUP) (Ramakrishnan, 2009).

1.1.2. Metodologías adaptivas o ágiles

Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las "metodologías ágiles" (MAS), las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas (Jodpimai, Sophatsathit, y Lursinsap, 2010). Éstas, han sido especialmente pensadas para un amplio rango de proyectos industriales de desarrollo de software, en especial para aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías (Garcia, Gonzalez, Colomo-Palacios, Lopez, y Ruiz, 2011).

Aunque los creadores e impulsores de las metodologías ágiles más populares han suscrito el manifiesto ágil y coinciden con los principios enunciados anteriormente, cada metodología tiene características propias y hacen hincapié en algunos aspectos más específicos (Balbín, Ocrospoma, Soto, y Pow-sang, 2009). Entre las principales metodologías de este tipo se encuentran:

- Crystal Methodologies de Alistarir Cockburn. (Cockburn, 2004)
- SCRUM, de Ken Schwaber. (Schwaber y Beedle, 2001)
- DSDM (Dynamic Systems Development Method). (Stapleton, 1997)
- Lean Programming de Tom y Mary Poppendieck. (Poppendieck y Poppendieck, 2003)

- FDD (Feature-Driven Development) de Peter Coad y Jeff De Luca. (Coad, Lefebvre, De Luca, y Luca, 1999)
- ASD (Adaptive Software Development) de Jim Highsmith. (Highsmith, 2000)
- eXtreme Programming (XP) de Kent Beck. (Beck, 1999)

A continuación, se exponen brevemente las principales características de las MAS. (Letelier, Canós, Sánchez, y Penadés, 2003):

- Los métodos ágiles son adaptables en lugar de predictivos. Los métodos tradicionales planean detalladamente gran parte del proceso de elaboración del producto para un plazo largo de tiempo, esto cuenta con la limitación de que, si surge un cambio, en muchas ocasiones, se tenga que planear nuevamente el proceso. Las metodologías ágiles se adaptan a las nuevas condiciones sin afectar el tiempo planificado inicialmente para la elaboración del proyecto.

- Los métodos ágiles son orientados a las personas y no a los procesos. El objetivo que persiguen estos métodos es definir un proceso que funcionará correctamente con cualquiera que lo use. Los métodos ágiles afirman que las habilidades del equipo de desarrollo están por encima de cualquier proceso y que el papel de este proceso es apoyar el trabajo del equipo de desarrollo. Puntualizan la necesidad de trabajar en función de las personas y nunca en su contra, enfatizando que el desarrollo de software es una labor agradable.

- Las MAS satisfacen al cliente por medio de entregas frecuentes y tempranas de software con valor, así el cliente participa también en la elaboración del producto, expresando su opinión para que el sistema resultante sea de su agrado.

- Existe una estrecha vinculación entre los responsables del negocio, los desarrolladores y el cliente, donde la participación de este último permite una constante actualización del producto, la realización de sugerencias y proposiciones para que el producto final tenga la calidad requerida.

- Otro de los objetivos de las MAS es relacionar a las personas como elemento fundamental del desarrollo del producto, nunca tratándolos como componentes que realizan roles, trabajando de esta forma con profesionales motivados.

- Las métricas de calidad son poco usadas para medir el éxito del proyecto, esto se mide por el funcionamiento de éste y por la satisfacción del cliente.

1.2. Estado actual de las técnicas de estimación de esfuerzo de desarrollo de software

1.2.1. Fundamentos de la estimación en el desarrollo de proyectos de software

La realidad de la industria de software, demuestra que un alto porcentaje de las estimaciones realizadas para proyectos son lejanas a los resultados del esfuerzo neto invertido en éstos. Según el Chaos Report, realizado por The Standish Group en el 2015, se reflejó que el 52% de los proyectos de software fueron concluidos después de la fecha estimada, 19% fueron cancelados y sólo el 29% terminaron a tiempo y dentro del presupuesto (Tabla 1.), lo cual refleja que la problemática generada por los errores en las estimaciones de proyectos de software es un tema crítico, y sumado a esto la falta de información (variables a estimar y los factores que las afectan) sobre proyectos de software actuales, desarrollados con metodologías ágiles, cuyos datos, son necesarios para poder realizar predicciones sobre características del nuevo proyecto.

Tabla 1. Datos de proyectos de software del Chaos Report.

Proyectos	2009	2010	2011	2012	2013	2014	2015
Exitosos	32%	37%	29%	27%	31%	29%	29%
Deficientes o Discutidos	44%	42%	49%	56%	50%	55%	52%
Cancelados o Fallidos o Fracasados	24%	21%	22%	17%	19%	17%	19%

En la literatura se han publicado y aún se sigue publicando un gran número de trabajos que hacen referencia a la estimación del tamaño del software, esfuerzo de desarrollo, esfuerzo de mantenimiento, defectos, facilidad de reutilización, facilidad de prueba, etc.

Técnicas supervisadas de Aprendizaje Automático (Modelos cuya superioridad frente a los tradicionales ha sido demostrada en numerosos estudios (Mendonca y Sunderhaft, 1999) como los algoritmos de inducción de árboles y tablas de decisión se han utilizado en la construcción de modelos de estimación del tamaño del software (Moreno García y García-Peñalvo, 2005). Árboles de decisión y redes bayesianas se han usado también para la predicción de defectos del software (Fenton y Neil, 1999). Los árboles de decisión han sido también los modelos en los que se han basado trabajos para la mejora de la fiabilidad (Tian y Palma, 1998) y estudios sobre la repercusión de algunas propiedades internas del software orientado a objetos (herencia, acoplamiento y complejidad) en su facilidad de reutilización (Sahraoui y Lounis, 1998), etc. La mayoría de estos trabajos están referidos a predicciones realizadas sobre integración de datos heterogéneos (diferentes organizaciones y tipos de proyecto) de gran envergadura y desarrollados bajo esquemas tradicionales y no han sido aplicados a datos de proyectos homogéneos y actuales (pequeños y medianos) donde el contexto es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad (uso de metodologías ágiles). De forma general, ningún modelo ha sido probado como exitoso en la predicción efectiva y coherente del esfuerzo del desarrollo de software.

Hablar de estimación del esfuerzo en el desarrollo de software viene a ser lo mismo que hablar de estimación de coste, pues este último está íntimamente relacionado con el esfuerzo necesario para realizar el proyecto. El esfuerzo es la magnitud de coste de elaboración de un proyecto, y se expresa mediante unidades tales como personas-mes, personas-año, horas-tarea, etc.

Una estimación es siempre un cálculo aproximado, realizado con algún tipo de método, por lo tanto, tiene implícito un margen de error posible frente al resultado real sobre lo que se estimó.

Los métodos tradicionales de estimación son ampliamente conocidos y utilizados desde hace muchos años, sin embargo, actualmente están siendo reemplazados por modelos de predicción basados en técnicas de Minería de datos cuya superioridad respecto a los primeros ha sido demostrada en numerosos estudios. En esta sección se presenta una visión general de estos métodos y en particular de los algoritmos de aprendizaje automático.

1.2.1.1. Clasificación taxonómica de los métodos de estimación

La predicción del coste y del esfuerzo del proyecto son los objetivos de la mayoría de los métodos descritos en la bibliografía. No existe el método perfecto para realizar estimaciones de proyectos de software. La regla de oro para el uso de técnicas es tener en cuenta los resultados que se puedan obtener por varias técnicas y contrastarlos.

De acuerdo a (Fenton y Pfleeger, 1997) pueden establecerse cuatro categorías para clasificar estas técnicas:

- Métodos basados en la opinión de expertos: Un desarrollador o gestor describe los parámetros del proyecto y los expertos realizan estimaciones basadas en su experiencia.
- Métodos basados en analogía: Los expertos comparan el proyecto propuesto con uno o más proyectos anteriores intentando encontrar similitudes y diferencias particulares.
- Métodos basados en descomposición: Análisis minucioso de las características que afectan al coste del proyecto mediante la descomposición del producto o del proceso software.
- Métodos basados en modelos.

Modelos

Técnicas que identifican los factores clave que contribuyen al esfuerzo y generan un modelo matemático que relaciona dichos factores con el esfuerzo. Los modelos se basan normalmente en información obtenida de experiencias pasadas.

En general, las ventajas de estos modelos están en que son objetivos, repetibles, calibrados y fáciles de usar, representando el enfoque más formal y proporcionando los resultados más fiables. Inicialmente se denominaron métodos algorítmicos o paramétricos debido a que realizan sus estimaciones de esfuerzo en función de un conjunto de variables y coeficientes fijos (parámetros). Estas variables están consideradas como los principales factores de coste. Los diferentes métodos de estimación algorítmicos se diferencian tanto por los factores que emplean en su modelo, como por la forma de la función que utilizan. Dependiendo del tipo de fórmula matemática que utiliza un modelo, se puede clasificar en modelos lineales, multiplicativos o con función de rendimiento (Power Function Models).

Según (Leung y Fan, 2002), los métodos algorítmicos se dividen en dos grupos: a) métodos empíricos, los que se basan en la experiencia a la hora de realizar una formulación matemática que modele el esfuerzo de desarrollo software, siendo el modelo COCOMO el más representativo de este grupo y b) métodos analíticos, que se basan en una comprensión del problema, descomponiéndolo para así comprender mejor su comportamiento, siendo ejemplos, los modelos SOFTCOST y SLIM.

Actualmente, los métodos de estimación basados en modelos empíricos incorporan técnicas heurísticas consideradas no algorítmicas en el sentido de "no deterministas" (Dolado, 2000b), es decir, no sólo se considera la solución, sino la probabilidad de que esa solución sea cierta. La inducción de árboles de decisión, redes neuronales, algoritmos genéticos y otras técnicas de aprendizaje automático se están utilizando desde hace algunos años para construir modelos de predicción de diferentes atributos del software.

1.2.2. Valoración crítica de los modelos de estimación

Uno de los principales problemas que afectan el desarrollo de los proyectos de software es la inadecuada estimación del esfuerzo de desarrollo (Basten y Sunyaev, 2011). Se estima que, como promedio, la estimación del esfuerzo de desarrollo de software difiere en un 30% del esfuerzo real requerido, razón por la cual, los expertos no esperan que la estimación sea totalmente efectiva. Éste, es un proceso que depende de múltiples factores, donde el más importante es la experiencia del estimador (Basten y Sunyaev, 2011).

Por otra parte, a pesar de que han sido desarrollado múltiples métodos para la estimación del esfuerzo de desarrollo de software, aún no existe un método estandarizado que satisfaga todas las metodologías de desarrollo de forma efectiva (Nguyen-Cong y De Tran-Cao, 2013; Dejaeger et al., 2012; Saroha y Sahu, 2015). Dentro de estos métodos de estimación, los más empleados son los basados en: líneas de código fuente, puntos de función, criterio de expertos, póker, Delphi y puntos de casos de uso (Britto et al., 2014).

La estimación basada en líneas de código fuente (LOC), es una de las técnicas más simple y, a la vez, más ampliamente empleada. Ésta, mide el esfuerzo de desarrollo de software en función del número de LOC, debido a la alta correlación que existe entre ambos. La desventaja de esta técnica está en que no existe una correlación entre las funcionalidades del software y LOC. Además, el número de LOC es dependiente de la plataforma y el lenguaje de programación y no es capaz de brindar una medida del rendimiento de un desarrollador (Albrecht y Gaffney, 1983; Basha y Ponnurangam, 2010; Britto, Mendes, y Borstler, 2015).

La estimación basada en puntos de función (PF), es una técnica que resulta más adecuada que LOC. Esto se debe a que estima el esfuerzo de desarrollo de software tomando en cuenta las funcionalidades del software desarrollado. Esta técnica es independiente del lenguaje y la plataforma de desarrollo. La principal desventaja de la

técnica es que el cálculo requiere de un gran nivel de detalle del proceso para lograr la estimación (Albrecht y Gaffney, 1983; Borade y Khalkar, 2013; Britto et al., 2015; Mohanty y Bisoi, 2012; Wen et al., 2012).

Tanto la estimación basada en LOC como en PF, no están diseñadas para el desarrollo orientado a objetos. Por lo que estas técnicas fueron extendidas empleando la estimación basada en puntos de casos de uso (PCU). Su principal desventaja está en que la estimación no se puede realizar en etapas tempranas del proyecto, sino hasta que no se realice el análisis y diseño del proyecto y se conozcan los casos de uso (Borade y Khalkar, 2013; Mohanty y Bisoi, 2012; Saroha y Sahu, 2015).

No obstante, la estimación del esfuerzo de desarrollo de software sigue recayendo sobre los expertos. Siendo las técnicas basadas en expertos las más empleadas en la actualidad (Basten y Sunyaev, 2011; Manoj, 2012)

La estimación basada en el criterio de expertos, tiene como limitante principal que depende de la experiencia, el conocimiento y las habilidades del experto. Por lo que varía en dependencia del experto y de su experiencia en el tipo de proyecto (Britto et al., 2014; Wen et al., 2012).

El póker de planeamiento es una estimación basada en consenso. Está inspirada en el juego de póker, donde cada carta representa un grupo de características, por lo general una lista de historias de usuarios que describen un software. Es una variación del método Delphi (Britto et al., 2015, 2014; Nguyen-Cong y De Tran-Cao, 2013).

El método Delphi, es una técnica de estimación estructurada en grupo. Se basa en las estimaciones individuales varios expertos que luego se reúnen para ponerse de acuerdo en una estimación. Tiene las mismas desventajas de los métodos anteriores, pero, además requiere de suficiente información de los requerimientos (Borade y Khalkar, 2013; Britto et al., 2015; Mohanty y Bisoi, 2012).

Sin embargo, en la actualidad, el desarrollo ágil de software ha ganado en popularidad, sobre las metodologías tradicionales. A pesar que existen múltiples

estudios donde se aplican las técnicas antes descritas en el desarrollo ágil, éstas no resultan efectivas en este entorno de desarrollo (Britto et al., 2015, 2014; Popli y Chauhan, 2014).

Recientemente, se propuso el empleo de la estimación basada en puntos de historias. Donde, el valor de los puntos de historias depende de la complejidad de desarrollo, el esfuerzo necesario y el riesgo que implica el desarrollo de una funcionalidad. A diferencia de la estimación basada en expertos, en este caso se puede realizar en base a historias similares (Britto et al., 2015, 2014; Jørgensen, 2013; Popli y Chauhan, 2014).

Como promedio, estas técnicas tienden a subestimar el esfuerzo de desarrollo ágil en un 55% y a sobre estimarlo en un 25,5% (Britto et al., 2015), razón por la cual, la estimación del esfuerzo basada en el desarrollo ágil se sigue considerando un reto.

1.2.3. Métodos de estimación del esfuerzo de desarrollo de software

Los métodos de estimación del esfuerzo de desarrollo de software pueden organizarse de forma jerárquica, tomando en cuenta múltiples criterios. En la Figura 1, se muestra una taxonomía que agrupa a estos métodos, tomando en consideración el modo en que se realiza la estimación.

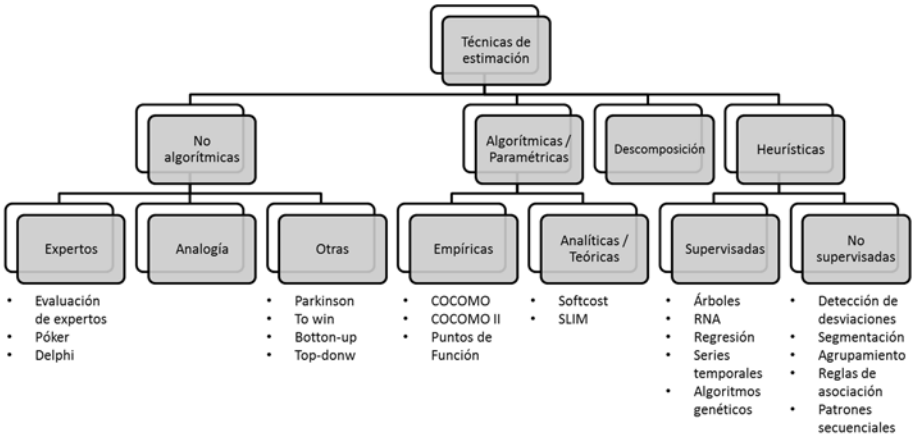


Figura 1 Taxonomía de las técnicas de estimación del esfuerzo de desarrollo de software basada en el método que emplean

La Figura 1, muestra una de las formas de agrupar las técnicas de estimación, más empleadas en la actualidad. Sin embargo, estas técnicas podrían agruparse teniendo en cuenta el tipo de metodología de desarrollo de software para cuales fueron diseñadas (Figura 2).

Figura 2 Taxonomía de las técnicas de estimación del esfuerzo de desarrollo de software basada en la metodología desarrollo de software.

Las técnicas de estimación del esfuerzo de desarrollo de software han sido diseñadas tomando en cuenta una metodología de desarrollo, luego, éstas se han aplicado o, en algunos casos, adaptado a otras metodologías de desarrollo; de igual forma, han sido creadas nuevas técnicas ajustadas a estas metodologías. Por la razón anterior, se decidió estudiar las técnicas de estimación del esfuerzo agrupadas en función de las metodologías de desarrollo para las cuales se crearon o donde fueron empleadas (Basha y Ponnurangam, 2010; Basten y Sunyaev, 2011b; Borade y Khalkar, 2013).

1.2.3.1. Métodos de estimación para metodologías predictivas

Múltiples *surveys* y *reviews* sobre la estimación del esfuerzo de desarrollo de software han sido publicados, en ellos se realizan diferentes análisis de los métodos de estimación desde la década de los 70's hasta la actualidad (Alsmadi y Nuser, 2013; Suri y Ranjan, 2012). Donde, algunos concluyen que en hasta el 2012 existe una tendencia a combinar diferentes métodos de estimación y utilizar a la par técnicas de inteligencia artificial, entre las que destacan: la lógica difusa, los sistemas basados en el conocimiento y las redes neuronales artificiales (Barcel, n.d.; Isasi-Viñuela y Galván-León, 2004).

1.2.3.1.1. Modelos en el paradigma de programación estructurada

Dentro de los modelos existentes para la estimación del esfuerzo necesario para el desarrollo de proyectos de software, uno de los más reconocidos es el Modelo Constructivo de Costos (o COCOMO, por su acrónimo del inglés **CO**nstructive **CO**st **MO**del) (Boehm et al., 2000; University of Southern California., 2000), este se puede considerar como un modelo matemático de base, en el cual se incluyen tres submodelos, cada uno ofrece un nivel de detalle y aproximación cada vez mayor, a medida que avanza el proceso de desarrollo del software: básico, intermedio y detallado.

El modelo COCOMO original se ha convertido en uno de los modelos de estimación de coste del software más utilizados y estudiados en la industria, entre sus principales características, podemos mencionar:

- Es una herramienta basada en las líneas de código la cual la hace muy poderosa para la estimación de costos y no como otros que solamente miden el esfuerzo en base al tamaño.
- Representa el más extenso modelo empírico para la estimación de software.
- Existen herramientas automáticas que estiman costos basados en COCOMO como ser: Costar, COCOMO 81.

Debido a las limitaciones propias que presupone un modelo matemático para la estimación del esfuerzo, el modelo COCOMO ha sido objeto de múltiples modificaciones, ajustes, mejoras y versiones (Figura 3.).

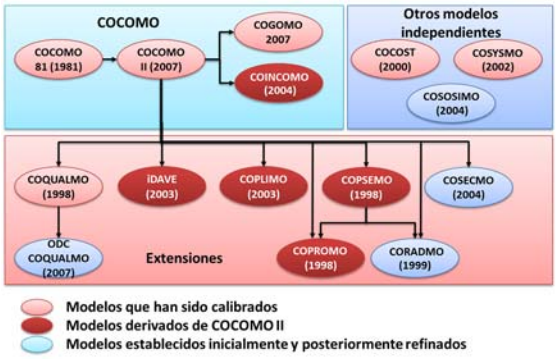


Figura 3 Familia de modelos derivados de COCOMO.

A pesar de que COCOMO II se caracteriza por estimar aceptablemente los proyectos, en base al número de líneas de código Fuente, éste no logra cubrir las diferentes metodologías y paradigmas de desarrollo de software. Por tal motivo, a partir de COCOMO II se desarrollaron toda una serie de modelos y extensiones que se muestran en la Figura 3.

A partir de aquí, múltiples técnicas han sido empleadas para mejorar el proceso de estimación del esfuerzo de desarrollo se software basada en líneas de código. En múltiples casos, empleando técnicas de Soft Computing y aprendizaje automático, para crear modelos con mayor eficiencia. Dentro de estas técnicas, las más empleas han sido las redes neuronales, los algoritmos genéticos, la programación genética, las técnicas de regresión, la lógica difusa, los algoritmos basados en inteligencia de enjambre, entre otros (Aljahdali y Sheta, 2010)(Prasad-Reddy, P., Sudha, K.R., Rama, P., y Ramesh, 2010).

Modelos de árboles de decisión han sido usados para la mejora de la fiabilidad (Tian y Palma, 1998) y estudios sobre la repercusión de algunas propiedades internas del software orientado a objetos (herencia, acoplamiento y complejidad) en su facilidad de reutilización (Sahraoui y Lounis, 1998).

Los modelos de predicción de la calidad del software se basan fundamentalmente en la detección de propiedades que conducen a defectos del software en diferentes etapas del desarrollo. En (Evett, Raton, Allen, Khoshgoftar, y Chien, 1998), se hace uso de técnicas de programación genética para desarrollar modelos de calidad que

relacionan atributos medidos en fases iniciales del proyecto con defectos descubiertos en fases posteriores o durante el periodo de operación del sistema. La predicción del número de defectos de diseño es el objeto de otro estudio en el que se demuestra la superioridad de los clasificadores CBR (Case Based Reasoning) frente a las técnicas de regresión (Ganesan, Khoshgoftar y Allen, 2000).

En el trabajo de (Mendonca y Sunderhaft, 1999), podemos apreciar referencias de muchos trabajos realizados sobre las aplicaciones de las técnicas de minería de datos en el área de la Ingeniería del software. Se han utilizado tanto técnicas supervisadas como no supervisadas para resolver problemas de características variadas, aunque los algoritmos más usados han sido los de aprendizaje automático, de ellos cabe destacar, por su influencia en el coste y calidad de los proyectos, los que se centran en la creación de modelos de estimación del tamaño del software, esfuerzo de desarrollo, esfuerzo de mantenimiento, defectos, facilidad de reutilización, facilidad de prueba, etc. En esta sección ofreceremos una revisión breve de varios de ellos.

Algoritmos de árboles de decisión y redes bayesianas han sido utilizados por (Fenton y Neil, 1999) para la predicción de defectos del software, proponiendo derivar las distribuciones de probabilidad de defectos introducidos, detectados y densidad de defectos a partir de variables relativas a los procesos del ciclo de vida (especificación, diseño, implementación y prueba).

Las redes neuronales han sido usadas en varios trabajos, como por ejemplo tenemos a (Khoshgoftaar y Lanning, 1995) en la predicción de riesgos de mantenimiento en módulos de programa o a (Thwin y Quah, 2005) que las utilizan en sistemas orientados a objetos para predecir el número de defectos en cada clase y el número de líneas modificadas por clase, usando como variables medidas relativas a la herencia, complejidad, acoplamiento, cohesión o asignación de memoria.

Se ha comprobado que la combinación de clasificadores mediante técnicas de bagging, boosting o logitBoost proporciona modelos de calidad del software más exactos

y robustos que los producidos por los clasificadores individuales (Khoshgoftaar, Geleyn, y Nguyen, 2003).

En el trabajo de (Moreno y García, 2005) se emplean con buenos resultados técnicas de aprendizaje automático supervisadas como son los algoritmos de inducción de árboles y tablas de decisión, en la construcción de modelos de estimación del tamaño del software.

También han sido utilizadas técnicas combinadas como en (Huang, Ho, Ren y Capretz, 2007), donde se propone un nuevo Modelo Constructivo de Costo (COCOMO) neurodifuso para la estimación de costo de software, este modelo lleva consigo algunas de las propiedades deseables del enfoque neuro-difuso tales como la habilidad de aprendizaje y la buena interpretabilidad, mientras mantiene los méritos del modelo COCOMO. A diferencia del enfoque estándar de redes neuronales, el modelo propuesto puede ser interpretado y validado por expertos y tiene una buena posibilidad de generalización, este trata con efectividad las entradas imprecisas y con incertidumbre y mejora la confiabilidad de las estimaciones de costo. Además, permite que las entradas tengan valores continuos y lingüísticos, evitando así el problema de que proyectos similares tengan costos de estimación muy diferentes. Se presenta en este trabajo un algoritmo detallado de aprendizaje. La validación usando datos de proyectos industriales muestra que el modelo mejora en gran medida la precisión de la estimación en comparación con el modelo bien conocido COCOMO o en (Dolado, 2000a) donde se realiza la validación de dos métodos de estimación del esfuerzo basados en técnicas de regresión. Los resultados de dichos métodos fueron mejorados mediante la aplicación de redes neuronales y programación genética.

(Engel y Last, 2007) proponen un Modelado de pruebas de software, costos y riesgos usando paradigmas de lógica difusa, en este trabajo se amplía la investigación sobre el problema de imprecisión por naturaleza de los datos de entrada, ciertos parámetros se capturan mejor usando las estructuras de tupla. Otros parámetros se pueden encapsular

mejor usando términos lingüísticos tales como “alto” o “bajo”. Se estima la calidad del coste que ocurre durante el desarrollo del software para una cabina en un avión de combate y se demuestra que los resultados de las producciones de la metodología de lógica difusa son comparables a las valoraciones basadas en modelos usando paradigmas Probabilísticos.

En el artículo de (Büyüközkan y Ruan, 2008) se presenta un modelo de evaluación basado en multi-criterios difusos de toma de decisiones (MCDM), método de medición del desempeño de los proyectos de desarrollo de software. En un problema de MCDM, una persona que toma las decisiones (DM) tiene que escoger la mejor alternativa que satisface los criterios de evaluación entre un conjunto de soluciones candidatas. Por lo general, es difícil encontrar una alternativa que cumpla con todos los criterios simultáneamente, así que una solución buena de compromiso es preferida. Este problema puede llegar a ser más complejo cuando múltiples DMs están implicados, no teniendo una percepción común sobre cada alternativa. Recientemente, un método de la clasificación del compromiso (conocido como el método VIKOR) ha sido propuesto para identificar tales soluciones de compromiso, proporcionando una utilidad máxima para el grupo mayoritario y un mínimo de pesar individual para el adversario. En su actual configuración, el método trata los valores exactos para la evaluación de las alternativas, que puede ser bastante restrictivo con criterios no cuantificables. Esto será verdad especialmente si la evaluación es hecha por medio de términos lingüísticos. Por esta razón se extiende el método VIKOR para procesar tales datos y para proporcionar una evaluación más completa en un ambiente difuso. Para demostrar el potencial de la metodología, la extensión propuesta es utilizada para medir el desempeño de la planificación de recursos empresariales (ERP).

Las técnicas de descubrimiento del conocimiento han sido utilizadas en menor proporción, pero de igual manera han favorecido a mejora de la calidad del software. Estos algoritmos suelen utilizarse con fines descriptivos, aunque es posible tomarlos

como base para la construcción de clasificadores que serán usados para realizar predicciones.

Así es posible realizar estimaciones del tamaño del software haciendo uso de un clasificador basado en reglas de asociación (García, Quintales, Peñalvo, y Martín, 2004). El problema que presentan estos algoritmos es el gran número de reglas que generan. En (M. García, Peñalvo, y Martín, 2004) se ha propuesto un algoritmo de refinamiento de reglas de asociación que está particularmente indicado para el uso de las mismas con fines de clasificación.

De la misma manera, las técnicas de clustering se han utilizado en: La estimación de coste en el desarrollo de software, como sucede en el trabajo de (Garre, Cuadrado, Sicilia, Rodríguez, y Rejas, 2007), en el que se plantea, como mejora del proceso de estimación, segmentar la base de datos ISBSG en diferentes grupos de proyectos mediante la utilización de tres algoritmos de agrupamiento diferentes: COBWEB, EM, y k-means, de manera que para cada uno de estos grupos (formados por proyectos homogéneos entre sí) se obtenga una relación matemática diferente. La segmentación llevada a cabo por estos algoritmos mejora la estimación con respecto al modelo que utiliza la base de datos sin segmentar. Por otra parte, si se comparan entre sí los resultados obtenidos al aplicar cada uno de ellos, se observa que el algoritmo que presenta un mejor comportamiento es EM, debido a su naturaleza probabilista (Krohn y Boldyreff, 1999) en la planificación del mantenimiento y (Podgurski, Masri, McCleese, Wolff, y Yang, 1999) en la estimación de la fiabilidad del software.

En Dick, Meeks, Last, Bunke, y Kandel, 2004, se realiza un estudio del uso de técnicas de Minería de Datos en el análisis de métricas del software. Valiéndose de un algoritmo de fuzzy clustering clasifican módulos con mayor o menor propensión a errores.

Sheta en el 2006, propone un modelo que explora las ventajas de la lógica difusa, empleando la técnica de Takagi-Sugeno (TS-Model), donde, se construye con conjunto

de modelos de regresión lineal sobre cada dominio posible de software, para estimar la cantidad de líneas de código fuente.

Chiu y Huang en el 2007 y Huang, Chiu, y Chen en el 2008, proponen mejorar la estimación del esfuerzo basados en analogía, y el valor final se ajusta mediante el empleo de algoritmos genéticos. Estos modelos calculan la distancia entre el proyecto a estimar y los proyectos conocidos, el valor de la estimación asignado sería el del proyecto más similar, luego un algoritmo genético, ajusta el esfuerzo reutilizado en función de las distancias de similitud entre pares de los proyectos. Los algoritmos genéticos, en general, requieren muchos recursos de cómputo para procesar las posibles soluciones.

En el 2008, Sheta, Rine, y Ayesh, presentan un modelo basado en Optimización de Enjambre de Partículas (PSO), para mejorar los parámetros del modelo COCOMO II.

El agrupamiento (clusterización), ha sido empleado con el objetivo de salvar el inconveniente de los modelos paramétricos que utilizan una única ecuación que representa a toda una base de datos de proyectos (Rubio, 2006). Con este método, los proyectos se agrupan homogéneamente, de forma que, un nuevo proyecto será asociado al grupo con el que tenga mayor similitud, este modelo requiere gran disponibilidad de proyectos conocidos.

En el 2008, Kumar, Ravi, Carr, y Kiran, sugieren el empleo de redes neuronales con funciones de transferencia de Morlet y Gaussiana, estas funciones logran un buen desempeño cuando se emplean más de dos capas ocultas. En este caso, fueron utilizadas dos configuraciones de las redes neuronales: WNN (*Wavelet Neural Network*) y TAWNN (*Threshold Accepting Trained Wavelet Neural Network*). La experimentación mostró que WNN presenta mejor desempeño que TAWNN.

Jodpimai y otros, en el 2010, proponen un red neuronal en cascada, basada en la reducción de características del modelo COCOMO. La propuesta consta de tres pasos:

a) Preparación de datos tomados de dominios públicos.

b) Reducción del número de características, considerando sólo aquellas relevantes.

c) Transformación del problema de estimación en un problema de clasificación y aproximación funcional, empleando una red neuronal en cascada.

Aljahdali y Sheta en el 2010, proponen el modelo COCOMO-DE, el cual emplea un algoritmo evolutivo diferencial[1] con el objetivo de mejorar la estimación del esfuerzo. Este es un algoritmo muy similar a un genético, pero que emplea mucho menos parámetros de configuración y es totalmente dependiente del operador de mutación empleado, así como del tamaño de la población inicial. Donde, los individuos o soluciones candidatas, son seleccionadas para pasar a la siguiente generación, si y solo si logran alcanzar mejor valor de calidad que el umbral de calidad establecido por la función de comparación.

Para analizar el desempeño de los modelos anteriores, se empleó la métrica de error MMRE en la base de datos NASA 93 (Tabla 2.).

Tabla 2. Comparativa entre los algoritmos basados en líneas de código fuente en cuanto a MMRE.

Model	COCOMO-DE	COCOMO-PSO	TS-Model	Halstead Model	Walston-Felix Model	Bailey-Basili Model	Doty Model
MMRE	0,0074	0,0074	0,0046	0,1479	0,0822	0,0095	0,1848

En la Tabla 2, se aprecia la comparación el modelo COCOMO-DE con otras técnicas de estimación del esfuerzo de desarrollo de software basadas en líneas de código fuente y técnicas de Soft Computing, en cuanto al error cometido. Como resultado, COCOMO-DE se desempeña similar al método COCOMO-PSO y supera a los algoritmos Halstead Model, Walston-Felix Model, Bailey-Basili Model y Doty Model, siendo superado únicamente por un modelo difuso TS-Model.

[1] La principal diferencia entre un algoritmo genético y un evolutivo diferencial está en que los algoritmos genéticos tienen el cruzamiento como operador principal y mutan con una baja probabilidad, mientras que los evolutivos diferenciales se basan en el operador de mutación.

Algabri y otros, proponen en el 2015 un modelo para la optimización del proceso de estimación del modelo COCOMO II, empleando algoritmos genéticos. En este modelo, los individuos se generan aleatoriamente, cada uno representa una solución apropiada, se calcula la calidad de cada individuo (i) para un proyecto (j) y de todos los individuos cada proyecto. La calidad de cada individuo de calcula como el promedio de todos los valores de calidad del individuo (i) en todos los proyectos (j), la selección de los individuos se realiza mediante el método de la ruleta. Para la generación de nuevos individuos se emplea el cruce sobre un punto y la mutación es uniforme, los mejores individuos entre los padres e hijos, se seleccionan para la próxima generación; este proceso se repite hasta alcanzar un máximo número de generaciones.

Este modelo fue evaluado empleando la base de datos NASA 93. Donde, los proyectos fueron clasificados en:

- **Orgánicos**: aquellos que tienen menos de 50 mil líneas de código fuente.
- **Semi-Libres**: aquellos que tienen entre 50 y 300 mil líneas de código fuente.
- **Empotrados**: aquellos que tienen más de 300 mil líneas de código fuente.

En todos los casos, el modelo propuesto supera al modelo COCOMO II en la estimación, excepto cuando los proyectos tienen un número muy elevado de líneas de código fuente, donde tiende a sobre estimar los valores.

1.2.3.1.2. Modelos en el paradigma de programación orientada a objetos

Los métodos de estimación del esfuerzo de desarrollo de software, diseñados para el paradigma estructurado, presentan varios inconvenientes ante el paradigma orientado a objetos, los cuales se mencionan a continuación:

- Se basan fundamentalmente en el número de líneas de código fuente. Lo cual lo hace fuertemente dependiente del lenguaje de programación empleado.
- No toman en cuenta la facilidad de reutilización de código. A partir del empleo de clases previamente diseñadas.

Por lo dicho anteriormente, Albrecht y Gaffney en 1983, proponen un nuevo método para la medición de la productividad en el desarrollo de software. Este método es capaz de estimar la cantidad de trabajo o esfuerzo necesario para diseñar y desarrollar una aplicación de software, a partir de listar y cuantificar el número de entradas externas, consultas, salidas y ficheros principales a ser entregados en el desarrollo del proyecto, donde, cada una de estas categorías son contabilizadas y ponderadas (se le asignan pesos), en pos de reflejar un valor relatico de la función final a entregar al usuario. La suma ponderada de las entradas y salidas es conocida como "Puntos de Función", en este método, la estimación de los pesos, asignados a cada entrada o salida, depende de la experiencia de los expertos, por lo que se considera como un proceso determinado por "debate y prueba".

La estimación del esfuerzo de desarrollo de software empleando puntos de función, se basa en la hipótesis de que el monto cantidad de la función que se proveerá, puede ser estimada por la relación de los principales componentes de datos que serán empleados o provistos por ésta. Consecuentemente, la estimación de la función debe correlacionarse con el número de líneas de código fuente necesarias para su implementación y, por tanto, con el esfuerzo de desarrollo.

Además, de estar pensada para su empleo en la estimación del esfuerzo de desarrollo con independencia del lenguaje de programación, existen varias razones por las cuales emplear Punto de Función:

- La medición de los Puntos de Función puede realizarse relativamente fácil, en discusión con el cliente, en las etapas temprana del proceso de desarrollo de software. Esto se debe a que está relacionada directamente a los requerimientos del usuario y a que es un mecanismo más fácilmente entendible para los usuarios que las líneas de código fuente.

- La disponibilidad de la información necesaria, si se toma en cuenta que este método emplea el cómputo de las entradas y salidas de las funciones, la estimación puede ser fácilmente validada en las etapas tempranas del proceso de desarrollo.

- Los Puntos de Función, pueden ser empleados para realizar una medición general de la productividad de los desarrolladores. Si se analizan los Puntos de Función por meses, se tendría una estimación de la tendencia en la productividad.

- Los Puntos de Función son independientes a los efectos que puedan tener la tecnología empleada, el lenguaje de programación para la codificación, la expansión que pueda producirse por el empleo de macros y llamadas, o la reutilización de código.

En el 2011, Garcia y otros, hacen una propuesta basada en redes neuronales para estimar el tiempo de duración en proyectos de software, esta, recurre a un subconjunto de datos obtenido de la organización International Software Benchmarking Standards Group (ISBSG[2]). En esta propuesta se filtra el conjunto de características disponibles y se excluyen aquellas referentes al cálculo de puntos de función.

En el 2015, Almache y colaboradores, proponen un modelo basado en redes neuronales denominado MONEPS, el cual considera 42 atributos extraídas del estándar ISO/IEC 25000[3].

Balbín y otros en el 2009, proponen TUPUX, una herramienta para la estimación, basada en puntos de función. El mérito fundamental de esta propuesta está en que realiza la estimación de forma incremental, asume que el resultado del cálculo de los puntos de función sin ajustar (UPF), para el proyecto completo, sin considerar los incrementos, debe ser igual a la suma de los UPF para cada incremento; para ello asume dos tipos de funciones: las funciones de transacciones y las funciones de dato, además, determina los UPF para los ficheros lógicos internos y externos, para cada caso de uso.

[2] El sitio web de ISBSG está disponible en: http://www.isbsg.org/
[3] El sitio web de esta norma está disponible en: http://iso25000.com/

Otra metodología empleada para la estimación en el paradigma orientado a objetos es la basada en puntos de casos de uso (PCU) (Remón, 2010). Ésta, es una derivación de la metodología de puntos de función propuesta, pero, se basa en la utilización de casos de uso como dato de entrada para calcular el esfuerzo en horas- hombre (hh) que son necesarias para el desarrollo de un proyecto de software, el método de estimación del esfuerzo utiliza cuatro variables principales:

1. **Clasificación de los Actores Involucrados**: los cuales se clasifican de acuerdo a su característica intrínseca y la forma en que interactúan con el sistema. Asignándoles su peso en dependencia de si interactúa como interfaz, a través de un protocolo o si es un actor complejo.

2. **Clasificación de Caso de Uso** (CU): los pesos de los casos de uso se asignan según la cantidad de transacciones que poseen, incluyendo las transacciones de escenarios alternativos y excluyendo las extensiones o inclusiones de otros casos de uso. De igual forma, a cada caso de uso le corresponde un peso.

3. **Factor de Complejidad Técnica del Proyecto de Software** (T): Los factores técnicos están definidos por las influencias técnicas que puedan afectar el proceso de desarrollo del sistema a construir, cada factor técnico posee un grado de complejidad, que oscila entre 0 y 5, donde 0 significa un valor irrelevante o nulo y 5 determina un valor con alto grado de influencia.

4. **Factores de Entorno del Proyecto** (E): estos indican la influencia del grupo humano involucrado en el proyecto sobre el sistema a desarrollar, de manera similar a los factores técnicos, los factores de entorno poseen un grado de influencia que oscila entre 0 y 5, donde 0 significa un valor irrelevante o nulo y 5 determina un valor con alto grado de influencia.

En el 2008, Frohnhoff y Engels, realizaron pruebas de estimación empleando PCU sobre 15 proyectos de software de distintos sectores industriales, compararon el esfuerzo estimado aplicando el método PCU de Karner contra el esfuerzo real del

proyecto, obteniendo como resultado una desviación estándar de un 42% (Remón, 2010).

1.2.3.2. Métodos de estimación para metodologías ágiles

A pesar del auge alcanzado por las metodologías ágiles y de la importancia que reviste la estimación del esfuerzo de desarrollo de proyectos de software, se encuentran solo unos pocos métodos de estimación específicamente desarrollados para este paradigma (Britto et al., 2015, 2014; Torkar, Awan, Alvi, y Afzal, 2006), existiendo la tendencia de adaptar las metodologías de estimación del esfuerzo de desarrollo de software, diseñadas para otros paradigmas, al paradigma ágil.

Estudios realizados en 364 organizaciones demostraron que solo 51 emplean modelos para estimar el esfuerzo de desarrollo, además, las organizaciones que emplean los modelos no mejoran su estimación sobre aquellas que no los emplean, esto significa que, para el desarrollo ágil, hasta el momento, el empleo de modelos existentes no mejora la estimación de un jurado de expertos (Popli y Chauhan, 2014).

Abrahamsson y Koskela en el 2004, propusieron un método para medir la productividad, calidad y estimar el esfuerzo para el desarrollo ágil. Mientras que Layman, Williams, y Cunningham en el 2004, proponen el empleo de puntos de casos de uso en XP.

Por otra parte, Finnie y Wittig en el 1996, aplicaron las redes neuronales y el razonamiento basado en casos para la estimación del esfuerzo, empleando un conjunto de datos de la "*Australian Software Metrics Association*", las RNA fueron capaces de estimar el esfuerzo de desarrollo con un 25% de error en más del 75% de los proyectos.

1.2.3.3. Resumen de los métodos de estimación del esfuerzo de desarrollo de software

En la Tabla 3., se muestra una comparación entre los modelos revisados, en base a la(s) técnica(s) que emplean, a sus principales ventajas y desventajas, así como las bases de datos que utilizan. No se incluyen los valores de efectividad o las mediciones de error,

debido a que, en la mayoría de los artículos, estas no aparecen, en ninguno de los casos, por lo que estos valores fueron calculados para esta investigación, los cuales fueron reportados, con independencia de la base de datos empleada.

Como se puede apreciar, el uso de las técnicas de aprendizaje automático en la estimación del esfuerzo necesario para el desarrollo de proyectos de software ha experimentado un crecimiento acelerado en los últimos tiempos, alcanzando el 86% del total de las técnicas usadas hasta el momento. Esto se debe, fundamentalmente, al alto nivel de adaptabilidad de estos algoritmos, dentro de las técnicas de aprendizaje automático las más empleadas son las basadas en regresión acaparando el 34% aproximadamente, los algoritmos basados en analogía y agrupamiento que representan el 32% y las redes neuronales artificiales que son el 24%, por otra parte, los algoritmos genéticos y evolutivos, así como los algoritmos basados en árboles solo representan solo un 5% cada uno.

Otra conclusión, evidente, que puede extraerse de este análisis es que el 14% de los métodos están diseñados para trabajar con el paradigma orientado a objetos, mientras que prácticamente el 85% de los métodos se basan en los métodos COCOMO y COCOMO II. Donde, aproximadamente el 36% de los métodos estudiados, son modificaciones o mejoras a estos métodos originales. Dejando, a penas, el 1 % a los métodos orientados al paradigma ágil.

Tabla 3. Tabla comparativa de modelos de estimación del esfuerzo de desarrollo de software.

Método	Basado en	Ventajas	Desventajas	Experimentación
1979 Albrecht (Albrecht y Gaffney, 1983)	• Puntos de Función.	• Fácil de realizar la estimación. • La estimación ocurre en etapas tempranas del ciclo de desarrollo del software. • Independientes de la tecnología empleada, el lenguaje de programación para la codificación, la expansión que pueda producirse por el empleo de macros y llamadas, o la reutilización de código.	• No está pensada para el desarrollo ágil de software.	
Vicinanza et al., 1991	• Regresión • COCOMO • Jurado de expertos	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Mukhopadhyay et al., 1992	• Regresión • COCOMO • Analogía • Expertos	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Briand et al. 1992	• Regresión • COCOMO • OSR	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Subramanian, Breslawski 1993	• Regresión • COCOMO	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Jorgensen, 1995	• Regresión • RNA	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Srinivasan, Fischer, 1995	• Regresión • COCOMO • SLIM • CART • RNA	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	

Método	Basado en	Ventajas	Desventajas	Experimentación
Bisio, Malabocchia, 1995	• COCOMO • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Finnie et al., 1997	• Regresión • Analogía • RNA	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Shepperd, Schofield, 1997	• Regresión • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Kitchenham, 1998	• CART • Stepwise ANOVA	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Myrtveit, Stensrud, 1999	• Regresión • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Walkerden, Jeffery, 1999	• Regresión • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Randy K. Smith, 2001	• COCOMO	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Tim Menzies, 2005	• COCOMO	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Bente Anda, 2005	• COCOMO • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Parag C, 2005	• CART • RNA	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
TS-Model, 2006 Sheta (A. Sheta, 2006)	• Técnica de Takagi-Sugeno de lógica difusa.	• Construye un conjunto de modelos de regresión lineal par cada dominio de software.	• Solo se ajusta al paradigma estructurado.	NASA 93
2006 Rybio (Rubio, 2006)	• Clusterización.	• El valor asignado puede diferir en la misma media en que difiera el grado de similaridad.	• Requiere de gran cantidad de proyectos conocidos. • Es altamente consumidor de recursos computacionales.	

Método	Basado en	Ventajas	Desventajas	Experimentación
Simon, 2006	• COCOMO		• Solo se ajusta al paradigma estructurado.	
Petrônio L. Braga, 2007	• Regresión		• Solo se ajusta al paradigma estructurado.	
Jingzhou Li, 2007	• COCOMO • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
Fabiana 2007	• RNA • COCOMO • Regresión • Analogía	• Hibridación de métodos	• Solo se ajusta al paradigma estructurado.	
2007 Chiu (Chiu y Huang, 2007)	• Clusterización • Algoritmo genético	• Realiza un ajuste más fino del esfuerzo.	• Requiere de gran cantidad de proyectos conocidos. • Es altamente consumidor de recursos computacionales.	
Chao-Jung Hsu, 2007			• Solo se ajusta al paradigma estructurado.	
Kristian M Furulund, 2007			• Solo se ajusta al paradigma estructurado.	
Bilge Başkeleş, 2007	• COCOMO		• Solo se ajusta al paradigma estructurado.	
Da Deng, 2007	• COCOMO		• Solo se ajusta al paradigma estructurado.	
2008 Huang (S. J. Huang et al., 2008)	• Clusterización • Algoritmo genético	• Realiza un ajuste más fino del esfuerzo.	• Requiere de gran cantidad de proyectos conocidos. • Es altamente consumidor de recursos computacionales.	
WNN y TAWNN, 2008 Kumar (Kumar et al., 2008)	• RNA con funciones de transferencia de Morlet y Gaussiana.	• Las funciones de transferencia logran un buen desempeño cuando se emplean más de dos capas ocultas.	• Solo se ajusta al paradigma estructurado.	

Método	Basado en	Ventajas	Desventajas	Experimentación
COCOMO-PSO, 2008 Sheta (A. Sheta et al., 2008)	• Optimización de enjambre de partículas (PSO).	• Incrementa la efectividad del modelo COCOMO II.	• Solo se ajusta al paradigma estructurado.	NASA 93
TUPUX, 2009 Balbín (Balbín et al., 2009)	• Puntos de Función. • Asume dos tipos de funciones: las funciones de transacciones y las funciones de dato.	• Realiza la estimación de forma incremental.	• Solo se ajusta al paradigma estructurado.	
Luciana Q, 2009	• Regresión		• Solo se ajusta al paradigma estructurado.	
Yeong-Seok, 2009	• Regresión			
Jianfeng Wen, 2009	• COCOMO	•	•	
COCOMO-DE, 2010 Aljahdali (Aljahdali y Sheta, 2010)	• Evolución diferencial (algoritmo basado en programación genética, centrado en la mutación)	• Incrementa la efectividad del modelo COCOMO II. • Converge rápidamente. • Es de simple implementación.	• Solo se ajusta al paradigma estructurado.	NASA 93
2010 Jodpimai (Jodpimai et al., 2010)	• RNA en cascada.	• Reduce las características del modelo COCOMO.	• El modelo COCOMO no ha podido ser acoplado satisfactoriamente a la dinámica del software.	
2011 García (Garcia et al., 2011)	• RNA • Puntos de función	• Se filtra el conjunto de características disponibles y se excluyen aquellas referentes al cálculo de puntos de función		

Método	Basado en	Ventajas	Desventajas	Experimentación
MONEPS, 2015 (Almache, Raura, R, C, et al., 2015)	• RNA	• Considera 42 atributos extraídas del estándar ISO/IEC 25000.		9 proyectos académicos desarrollados por estudiantes pertenecientes a los últimos niveles de la carrera de Ingeniería en Sistemas e Informática de la Universidad de las Fuerzas Armadas ESPE de Ecuador.
2015 Algabri (Algabri et al., 2015)	• Modelo COCOMO • Algoritmo genético	• Incrementa la efectividad del modelo COCOMO II.	• Solo se ajusta al paradigma estructurado. • No se garantiza la convergencia del modelo. • EL método de selección es elitista. • Las iteraciones están condicionadas a un número de generaciones y no a un valor de la función de calidad.	NASA 93
Karner (Frohnhoff y Engels, 2008)	• Puntos de Casos de Uso	• Se basa en: actores, casos de uso, proyecto y entorno del proyecto.	• No pensada para el desarrollo ágil.	15 proyectos de software de distintos sectores industriales.

1.2.3.4. Software para la estimación del esfuerzo de desarrollo de software

La estimación del esfuerzo de desarrollo de software es una tarea altamente subjetiva e imprecisa (Nguyen-Cong y De Tran-Cao, 2013; Dejaeger et al., 2012; Saroha y Sahu, 2015). En el intento de disminuir cada vez más las diferencias entre valor estimado y valor real, las pequeñas y medianas empresas de software, enfrentan el reto de seleccionar, no solo el método de estimación más adecuado sino el de adoptar mejores prácticas de estimación. Lo cual ha propiciado el desarrollo de múltiples softwares para la estimación.

Entre los softwares más conocidos y empleados, se encuentran(Almache, Raura, R, C, et al., 2015):

- **COCOMO-II** (Boehm et al., 2000; University of Southern California, 2000): Herramienta basada en el modelo matemático de base empírica "Modelo Constructivo de Costos". Incluye tres submodelos, cada uno ofrece un nivel de detalle y aproximación, cada vez mayor, a medida que avanza el proceso de desarrollo del software: básico, intermedio y detallado.

- **CoStar** (B. W. Boehm y Valerdi, 2008): herramienta basada en COCOMO-II. Produce estimaciones de la duración de un proyecto, cantidad de personal, esfuerzo y costo. Costar le permite realizar compensaciones y experimentar con "qué pasaría si" análisis para llegar al plan óptimo proyecto.

- **CostModeler** (Axelrad, Granik, Boksha, y Rollins, 1994): herramienta dirigida por base de datos. Permite almacenar, acceder y recuperar toda la información relacionada con la estimación de costos en una sola herramienta, realizar búsquedas, etc. La centralización de los datos permite que todos a los estimadores para acceder de forma rápida y utilizar la misma información, lo que aumenta drásticamente la velocidad y la consistencia del proceso de estimación de costos.

- **CostXpert** (Madachy, 2007): herramienta de estimación de costos de que integra múltiples modelos de estimación para proporcionar estimaciones más exactas y

completas. CostXpert es la única herramienta para ofrecer modelos específicos para proyectos de servicios sociales, proyectos de telecomunicaciones, proyectos de Lotus Domino, y desarrollador de proyectos de Oracle / 2000.

- **KnowledgePlan®** (Jones, 1996): herramienta que propone plantillas de proyectos para garantizar la eficacia de las estimaciones de la herramienta. Las estimadores individuales permiten personalizar aún más los componentes de la plantilla de proyecto para hacer frente a los requisitos particulares de un proyecto individual. El uso de plantillas predefinidas del proyecto, sin embargo, ayuda a asegurar la consistencia de las estimaciones y reduce sustancialmente el tiempo y el esfuerzo requeridos por los líderes del equipo del proyecto para la estimación de cada nuevo proyecto.

- **SEER** (Madachy y Boehm, 2008): herramienta de gestión de proyectos que permite describir un proyecto en unos pocos pasos simples y basados en conocimientos científicos sólidos y las historias de proyectos aplicables. Proporciona una estimación más probable de los costos del proyecto, el esfuerzo y la duración. Permite asignar probabilidades a los objetivos del proyecto específicos, lo cual permite ser mejoradas por pequeños cambios en diversas limitaciones subyacentes y asunciones.

- **SoftCost**-R (Southwell, 1996): herramienta que integra un poderoso editor, un modelo de estimación paramétrico híbrido y un generador de reportes flexible.

Por otra parte, existen algunas herramientas de estimación de costos que están tendiendo a la obsolescencia, debido principalmente a su temprana aparición; así tenemos (Almache, Raura, R, C, et al., 2015):

- CheckPoint (Ferens, 1998).
- ESTIMACS (Boehm, Abts, y Chulani, 2000).
- REVIC (Webber, 1995).
- SPQR/20 (Cordero, 2013).

Debido a que no se adaptan a los modelos disponibles y a la falta de mantenimiento de los softwares existentes, la mayoría de estos softwares tienden a perder su utilidad con gran rapidez. Dando lugar a la aparición de nuevos softwares y herramientas, sin embargo, esto no significa que la estimación del esfuerzo de desarrollo de proyectos de software se logre mejorar favorablemente (Mizell y Malone, 2007).

1.3. Sumario

Hemos dedicado este capítulo a la Gestión de Proyectos y al estado actual de las técnicas de estimación del esfuerzo del software, obteniendo las siguientes conclusiones parciales:

- Existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo, y que las podemos dividir en: a) Tradicionales (se centran especialmente en el control del proceso, estableciendo actividades involucradas, artefactos que se deben producir, y las herramientas y notaciones que se usarán) y b) Agiles (efectivas en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad)
- Las metodologías ágiles han revolucionado la manera de producir software, y a la vez han generado un amplio espacio para la investigación y el desarrollo de nuevos modelos de estimación del esfuerzo de desarrollo de los proyectos de software.
- El análisis del estado del arte, evidenció, que desafortunadamente los estimadores más efectivos suelen ser no compatibles o no presentan un mecanismo adecuado a las tendencias actuales de desarrollo ágil de software. Lo cual representa una debilidad para la estimación del esfuerzo en la actualidad.

Capítulo 2
Fundamentos sobre aprendizaje automático

Como hemos visto en el capítulo anterior, la gestión de proyectos y el desarrollo de software no es una tarea fácil, prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo, por una parte, están las más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Por otra parte, están las metodologías ágiles, enfoque que está demostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad. Las metodologías ágiles han revolucionado la manera de producir software, y a la vez han generado un amplio espacio para la investigación y el desarrollo de nuevos modelos de estimación del esfuerzo de desarrollo de los proyectos de software.

El análisis del estado del arte permitió realizar un estudio extensivo de los métodos y técnicas de estimación del esfuerzo necesario para el desarrollo de software, proponiéndose una nueva taxonomía. Se evidenció, además, que desafortunadamente los estimadores más efectivos suelen ser no compatibles o no presentan un mecanismo adecuado a las tendencias actuales de desarrollo ágil de software. Lo cual representa una debilidad para la estimación del esfuerzo en la actualidad.

En este apartado se detallan las bases teóricas necesarias para el presente estudio, en primer lugar, se muestra una breve introducción al proceso de descubrimiento de conocimiento en bases de datos o KDD, minería de datos y sus herramientas. Después describiremos los diferentes métodos para la estimación de esfuerzo en el desarrollo de software, incluyendo los Árboles de clasificación y regresión.

2.1. El proceso de descubrimiento de conocimiento en bases de datos o KDD y minería de datos

Hoy en día, la cantidad de datos que ha sido almacenada en las bases de datos excede nuestra habilidad para reducir y analizar los mismos sin el uso de técnicas de análisis automatizadas. Muchas bases de datos comerciales transaccionales y científicas crecen a una proporción fenomenal. El Descubrimiento de Conocimiento en Bases de Datos (en inglés Knowledge Discovery in Databases, con el acrónimo KDD) es un área de la computación que intenta explotar la ingente cantidad de información mediante el descubrimiento de patrones representativos y útiles, extrayendo conocimiento que pueda asistir a un humano para llevar a cabo tareas de forma más eficiente y satisfactoria. (Fayyad, Piatetsky-Shapiro, y Smyth, 1996) definen el proceso de KDD como el proceso no trivial de identificación de patrones válidos, originales, potencialmente útiles y comprensibles en los datos. El KDD está compuesto por una serie de etapas, siendo la Minería de Datos (MDD) la encargada de extraer modelos a partir de la información recogida.

Debido al tamaño de las bases de datos, a la presencia de ruido, datos inconsistentes, redundantes, etc., se hace necesaria la aplicación de técnicas de preprocesamiento sobre los conjuntos de datos (Zhang, Zhang, y Yang, 2010). El objetivo perseguido por el preprocesamiento es obtener conjuntos de datos tales que al aplicar técnicas de MDD sobre ellos se generen modelos representativos con mayores prestaciones.

2.1.1. Introducción a la extracción del conocimiento

El aumento del volumen y variedad de información ha crecido de forma espectacular en los últimos años. La cantidad de información es tal que es necesario tratarla para poder utilizarla adecuadamente. Los datos tal cual se almacenan suelen ser procesados previamente para proporcionar beneficios directos.

Su valor real reside en la información que podamos extraer de ellos: información que nos ayude a tomar decisiones o a mejorar nuestra comprensión de los fenómenos que nos rodean.

El método tradicional de convertir los datos en conocimiento consiste en un análisis e interpretación realizado de forma manual. Sin embargo, cuando la cantidad de datos de los que disponemos aumenta, el uso de herramientas de extracción de conocimiento y MDD es esencial (Hernández, Ramírez, y Ramírez, 2004).

En la aplicación de técnicas de KDD (Figura 4.), un gran esfuerzo se dedica a la preparación de los datos. Los métodos de MDD son muy potentes en la búsqueda de información de interés y pueden serlo aún más conforme se adecuan los datos mediante el preprocesamiento.

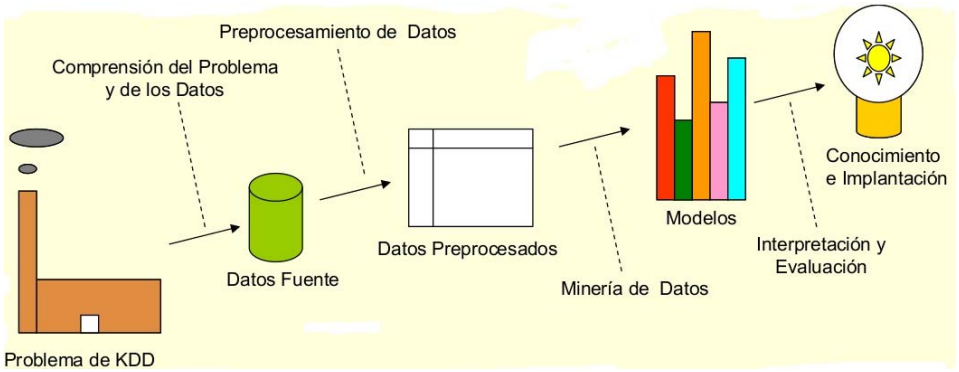


Figura 4 Etapas del procesamiento de la información.

Como razones para la preparación de los datos podemos citar las siguientes (Pyle, y Cerra, 1999):

- Ajuste del tamaño del conjunto de datos para poder ser evaluado por técnicas de MDD.

- Ajuste del formato de los datos para poder ser evaluado por técnicas concretas de MDD.
- Gestión de datos perdidos.
- Tratamiento de ruido en la información.
- Eliminación de información redundante.
- Eliminación de datos inconsistentes.

El proceso de KDD se divide en una serie de etapas, como aparece reflejado en la Figura 5. Tanto el número de etapas como la función que se desempeña en cada una de ellas varían de una descripción de KDD a otra. A continuación, analizaremos el proceso de KDD siguiendo el modelo descrito en (Chapman et al., 2000).

2.1.2. El proceso de descubrimiento de conocimiento en bases de datos o KDD.

La metodología CRISP-DM (Cross-Industry Standard Process for Data Mining (Chapman et al., 2000)) estructura el ciclo de vida de un proyecto de MDD en seis fases, que interactúan entre ellas de forma iterativa durante el desarrollo del proyecto, según podemos ver en la Figura 5.

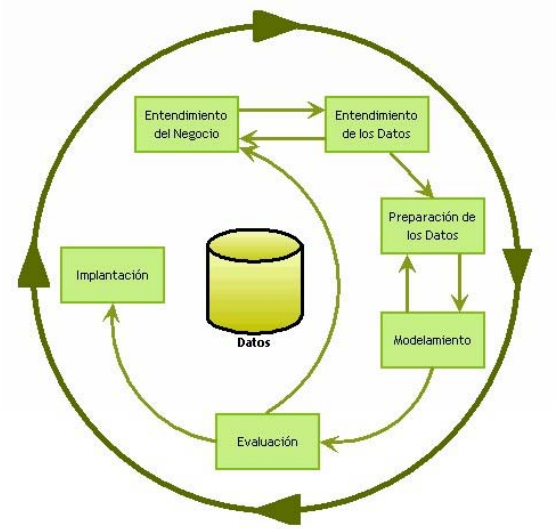


Figura 5 Fases de KDD según el modelo iterativo CRISP-DM.

Normalmente los proyectos de MDD no terminan en la implantación del modelo, sino que se deben documentar y presentar los resultados de manera comprensible para lograr así un aumento del conocimiento. Además, en la fase de explotación se debe de asegurar el mantenimiento de la aplicación y la posible difusión de los resultados (Fayyad et al., 1996).

A continuación, describiremos cada una de esas etapas.

- Comprensión del problema: Esta primera etapa se centra en la comprensión del problema y en concretar los objetivos perseguidos, para así convertir este conocimiento en la definición de un problema de KDD. Para delimitar los objetivos, al finalizar esta fase será necesario:
    - Obtener un conocimiento adecuado del problema.
    - Tener descritos claramente los objetivos.
    - Establecer el criterio de éxito o utilidad que se desea alcanzar.

  En esta fase se diseñará el plan para alcanzar dichos objetivos. Esta información será el punto de partida para la siguiente etapa dentro del KDD.

- Comprensión de los datos: En esta fase se comienza a trabajar con el conjunto inicial de datos. Se llevan a cabo diversas actividades tales como:
    - Familiarizarse con la información.
    - Identificar problemas en la calidad de los datos.
    - Detectar subconjuntos interesantes para formular hipótesis sobre información oculta.

  Tras estudiar el conjunto inicial de datos, se efectúa una descripción de dichos datos, desarrollando una exploración general sobre los mismos para buscar información oculta. Finalmente, se verifica la calidad de los datos.

- Preparación de los datos: Tomando como partida la información recogida en la etapa anterior en ésta fase se desarrollan actividades destinadas a confeccionar el conjunto de datos final (conjunto que servirá de entrada al algoritmo de MDD) a partir del conjunto inicial. Las tareas dedicadas a la preparación de los datos se pueden aplicar repetidas veces, sin tener por qué utilizarse en un orden concreto. Al final de esta etapa tendremos el conjunto de datos preparado para ser utilizado en MDD.

- Minería de datos: En esta etapa se efectúa la extracción de modelos que aportan conocimiento sobre los datos procesados. Normalmente se pueden emplear diferentes tipos de técnicas sobre el mismo problema.

- Evaluación: Llegados a esta fase, ya tenemos una técnica desarrollada de la que se ha evaluado la calidad de sus soluciones. Antes de proceder en la última etapa a la implantación de la solución para su uso habitual, es importante revisar concienzudamente tanto la técnica como los resultados proporcionados. Habría que revisar los pasos llevados a cabo en la aplicación del algoritmo para asegurarnos que se adecúan, tanto el método como las soluciones, a los objetivos perseguidos. Al final de esta fase se habría decidido la utilización de los resultados obtenidos con la técnica de MDD.

Al evaluar los resultados habría que considerar tanto la exactitud como la capacidad de generalización de la técnica.

- Implantación: La utilización del algoritmo no es generalmente el punto final del proceso de KDD. Incluso si el propósito del desarrollo es obtener conocimiento sobre los datos, el conocimiento ganado deberá ser organizado y preparado para poder ser utilizado. Dependiendo de los requerimientos, el proceso de implantación puede ser tan simple como la generación de un informe o tan complejo como construir un producto software comercial.

2.1.3. Minería de datos

Minería de Datos es un término genérico que engloba resultados de investigación, técnicas y herramientas usadas para extraer información útil de grandes bases de datos. Si bien Minería de Datos es una parte del proceso completo de KDD, en buena parte de la literatura los términos Minería de Datos y KDD se identifican como si fueran lo mismo. Concretamente, el término Minería de Datos es usado comúnmente por los estadísticos, analistas de datos, y por la comunidad de administradores de sistemas informáticos como todo el proceso del descubrimiento, mientras que el término KDD es utilizado más por los especialistas en Inteligencia Artificial.

El análisis de la información recopilada (por ejemplo, en un experimento científico) es habitual que sea un proceso completamente manual (basado por lo general en técnicas estadísticas). Sin embargo, cuando la cantidad de datos de los que disponemos aumenta la resolución manual del problema se hace intratable. Aquí es donde entra en juego el conjunto de técnicas de análisis automático al que nos referimos al hablar de Minería de Datos o KDD.

Hasta ahora, los mayores éxitos en Minería de Datos se pueden atribuir directa o indirectamente a avances en bases de datos (un campo en el que los ordenadores superan a los humanos). No obstante, muchos problemas de representación del conocimiento y de reducción de la complejidad de la búsqueda necesaria (usando conocimiento a priori) están aún por resolver. Ahí reside el interés que ha despertado el tema entre investigadores de todo el mundo.

2.1.4. Herramientas de minería de datos.

La mayoría de las herramientas que aplican técnicas de Minería de datos para el análisis y la búsqueda de tendencias y asociaciones entre los datos de las bases disponibles, emplean ciertos procedimientos estadísticos tales como árboles de decisión, redes neuronales, series temporales, criterios bayesianos, etc. El empleo de estos

métodos es indispensable para la clasificación, reducción, simulación, asociación y predicción en los datos.

Ello no implica, sin embargo, que los usuarios de las herramientas de minería de datos existentes necesiten disponer de los conocimientos teóricos y estadísticos asociados a dichos procedimientos. Los entornos en que se desarrollan estos programas suelen presentar una visualización intuitiva y enlaces a través de iconos y ediciones que permiten su uso de forma rápida y sencilla. Existen muchas soluciones con diferentes enfoques; casi todas ellas coinciden en que surgen como proyectos en el contexto universitario y siguen teniendo profundos lazos.

Entre las herramientas más usadas en la minería de datos, tenemos: WEKA (*Waikato Environment for Knowledge Analysis*), KEEL (*Knowledge Extraction Evolutionary Learning*), RapidMiner, antiguamente llamado YALE (*Yet Another Learning Environment*), Knime (Konstanz Information Miner).

También existen programas que ofrecen la extracción de conocimientos con la aplicación de algoritmos. Pero estos no son distribuciones libres, se cita a los siguientes: SPSS Clementine, Oracle Data Mining, KnowledgeSTUDIO.

2.2. Técnicas de minería de datos

La minería de datos se utiliza en la extracción no superficial de la información que reside de manera sobreentendida en los datos. Dicha información era previamente desconocida y podrá resultar útil para algún proceso.

La minería de datos está sustituyendo el enfoque del análisis de datos orientado a la verificación por un análisis de datos dirigido al descubrimiento del conocimiento, el cual descubre información sin necesidad de formular previamente una hipótesis. La aplicación automatizada de algoritmos de minería de datos permite detectar fácilmente patrones en los datos, por lo que esta técnica es mucho más eficiente que el análisis orientado a la verificación cuando se trata de repositorios complejos y de gran tamaño.

Estas técnicas emergentes de minería de datos, se encuentran en continua evolución como resultado de la colaboración entre campos de investigación tales como bases de datos, estadística, inteligencia artificial, reconocimiento de patrones, sistemas expertos, recuperación de información, visualización, y computación de altas prestaciones.

2.2.1. Clasificación de las técnicas de minería de datos

Los algoritmos de aprendizaje automático se pueden englobar dentro de un grupo más general de técnicas denominadas de Minería de Datos, cuyo propósito es extraer conocimiento no explícito a partir de grandes volúmenes de datos. De acuerdo a (Weiss y Indurkhya, 1998), las técnicas de Minería de Datos se pueden clasificar en dos grandes categorías:

- Algoritmos supervisados o predictivos: predicen uno o varios datos desconocidos a priori, a partir de otros conocidos.

- Algoritmos no supervisados o de descubrimiento del conocimiento: se utilizan cuando una aplicación no es lo suficientemente madura y no tiene el potencial necesario para una solución predictiva.

2.2.1.1. Algoritmos de aprendizaje automático supervisados o predictivos

Los algoritmos supervisados o predictivos predicen un dato (o un conjunto de ellos) desconocido a priori (etiqueta), a partir de otros conocidos (atributos descriptivos). A partir de datos cuya etiqueta se conoce se induce un modelo que relaciona dicha etiqueta y los atributos descriptivos. Y esas relaciones sirven para realizar la predicción en datos cuya etiqueta es desconocida. Esta forma de trabajar se conoce como aprendizaje supervisado.

En este grupo se encuentran, por una parte, algoritmos que resuelven problemas de clasificación debido a que trabajan con etiquetas discretas (tablas de decisión, árboles de decisión, inducción de reglas, bayesiana, basado en ejemplares, redes neuronales, lógica borrosa, técnicas genéticas) y, por otra, algoritmos que se utilizan en la predicción de valores continuos (regresión, árboles de predicción, estimador de núcleos).

Dado que en el presente trabajo vamos a utilizar algoritmos de árboles de decisión y regresión, haremos una descripción de los mismos en la sección 2.3.2.

2.2.1.2. Algoritmos de aprendizaje automático no supervisados o de descubrimiento del conocimiento o descriptivos

Los algoritmos no supervisados o de descubrimiento del conocimiento realizan tareas descriptivas como el descubrimiento de patrones y tendencias en los datos actuales (no utilizan datos históricos).

El descubrimiento de esa información sirve para llevar a cabo acciones y obtener un beneficio científico o de negocio de ellas. A esta categoría pertenecen los algoritmos de: detección de desviaciones, segmentación, agrupamiento ("clustering"), reglas de asociación, patrones secuenciales.

La aplicación de los algoritmos de minería de datos requiere la realización de una serie de actividades previas encaminadas a preparar los datos de entrada debido a que, en muchas ocasiones dichos datos proceden de fuentes heterogéneas, no tienen el formato adecuado o contienen ruido. Por otra parte, es necesario interpretar y evaluar los resultados obtenidos.

El proceso completo consta de las siguientes etapas (Hirji, 1999):

1. Determinación de objetivos
2. Preparación de datos:
    - Selección: Identificación de las fuentes de información externas e internas y selección del subconjunto de datos necesario.
    - Preprocesamiento: estudio de la calidad de los datos y determinación de las operaciones de minería que se pueden realizar.
3. Transformación de datos: conversión de datos en un modelo analítico.
4. Minería de datos: tratamiento automatizado de los datos seleccionados con una combinación apropiada de algoritmos.

5. Análisis de resultados: interpretación de los resultados obtenidos en la etapa anterior, generalmente con la ayuda de una técnica de visualización.

6. Asimilación de conocimiento: aplicación del conocimiento descubierto.

Aunque los pasos anteriores se realizan en el orden en que aparecen, el proceso es altamente iterativo, estableciéndose retroalimentación entre los mismos. Además, no todos los pasos requieren el mismo esfuerzo, generalmente la etapa de preprocesamiento es la más costosa ya que representa aproximadamente el 60 % del esfuerzo total, mientras que la etapa de minería sólo representa el 10%.

2.2.1.3. El proceso de aprendizaje Supervisado

Para llevar a cabo el aprendizaje supervisado se divide el conjunto de datos con etiqueta conocida en dos subconjuntos disjuntos, generalmente uno de mayor tamaño que el otro. El proceso se desarrolla en dos fases: (Tuya, Román, y Dolado Cosín, 2007)

- Entrenamiento: construcción de un modelo de predicción usando el mayor los subconjuntos.

- Prueba: se comprueba la validez del modelo inducido probándolo sobre el resto de los datos. El error se calcula comparando las predicciones realizadas por el modelo con el valor real del atributo etiqueta.

Cuando la etiqueta toma valores discretos, cada uno de estos valores representa una clase, por lo que el modelo predictivo resultante del proceso de aprendizaje se denomina clasificador.

El proceso de aprendizaje consiste en crear automáticamente un clasificador a partir de un conjunto de entrenamiento y de un inductor (Figura 6.):

- Conjunto de entrenamiento: datos utilizados para realizar el entrenamiento. Son datos cuya etiqueta se conoce, es decir, son ejemplos ya clasificados.

- Inductor: algoritmo que construye automáticamente un clasificador a partir de un conjunto de entrenamiento.

El modelo inducido es el clasificador. Éste consiste en una serie de patrones que son útiles para distinguir las clases.

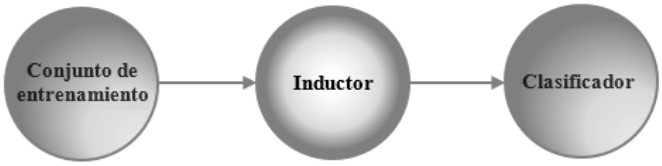


Figura 6 Inducción de un clasificador

Una vez que se ha inducido el modelo se puede utilizar para predecir automáticamente la clase de otros registros no clasificados (Figura 7.).

Figura 7 Uso de un clasificador para predecir las clases de datos sin clasificar.

2.2.2.- Árboles de clasificación y regresión

Los árboles de clasificación, también llamados "árboles de decisión", es uno de los paradigmas de clasificación más utilizados en el mundo del aprendizaje automático.

Los árboles de clasificación forman parte de los métodos de clasificación supervisada, es decir, tendremos un conjunto de entrenamiento clasificado con el que se construirá un modelo que trate de clasificar nuevos casos no presentes en el conjunto inicial (casos de test). La construcción del árbol de clasificación se realiza mediante un proceso de inducción (*Top-Down-Induction-Decision-Trees*).

El árbol de clasificación puede ser definido como una función d(x) que relaciona cada patrón de entrada x del espacio de clasificación con una clase del conjunto de posibles valores de clase.

Otra definición más próxima al funcionamiento de los árboles de clasificación es considerar al clasificador como una partición del espacio de clasificación X en M

subconjuntos disjuntos A1, A2 ,..., AM siendo X la unión de todos ellos. El particionado tiene como criterio de agrupación la clase a la que pertenecen las instancias.

Un árbol de clasificación está compuesto por un nodo raíz, nodos intermedios y nodos hoja. El nodo raíz es el primer nodo del árbol de donde cuelgan los demás nodos. Los nodos intermedios o no terminales se encuentran entre el nodo raíz y las hojas. Corresponden con subdivisiones del árbol. Los nodos hoja son aquellos nodos terminales que no se van a dividir más. Cada nodo hoja se corresponderá con una categoría concreta de la variable de clase. De esta manera, los nodos hoja representan las diferentes particiones en las que se ha dividido el espacio de clasificación.

La Figura 8., muestra el aspecto de un árbol de clasificación simple. Tanto los nodos intermedios como el nodo raíz corresponden con una pregunta que se le hace a una de las variables del vector de propiedades, también denominada tupla. Cuando la pregunta se responde de manera binaria (Si/No) el árbol es binario y cada nodo padre solo se dividirá en dos nodos hijos correspondientes a cada una de las respuestas posibles. Así, dado un objeto a clasificar, éste va "cayendo" por un lado u otro de los nodos según responda la pregunta correspondiente. Finalmente, la tupla caerá en un nodo hoja que no tiene asignada una pregunta, sino una clase en la que dicha tupla quedará clasificada.

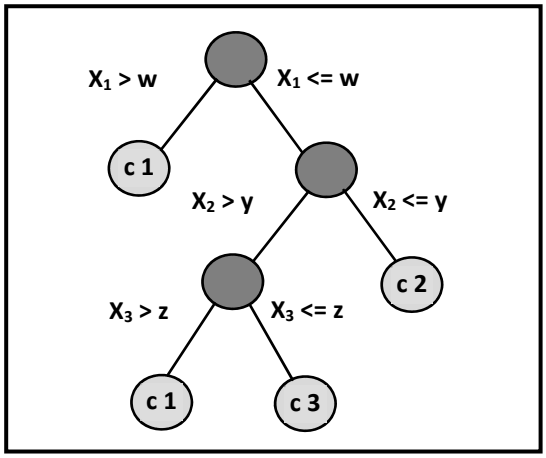


Figura 8 Ejemplo de árbol de clasificación binario.

Como se observa en la Figura 8., hay 3 clases (c1, c2 y c3) y 4 nodos hoja. Pueden existir diferentes nodos hoja etiquetados con la misma clase. Al número de nodos hoja que tiene un árbol se le denomina complejidad del árbol y para el caso de la figura la complejidad del árbol sería 4, porque es número de partes en los que ha dividido el espacio de clasificación. Esto es distinto al número de clases. En este caso el árbol ha diferenciado 4 casos distintos a la hora de realizar la clasificación, pero dos de los casos corresponden a la misma clase.

Construcción de un árbol de clasificación

A continuación, se explica cómo es la fase de entrenamiento que genera como resultado un árbol de clasificación. El proceso de construcción comienza por el nodo raíz. Al principio del proceso, el nodo raíz tiene asociadas todas la tuplas o patrones de entrenamiento. Ahora debemos dividir el conjunto de entrenamiento en un conjunto de particiones que serán los nodos hijos, haciendo que estos subconjuntos dividan de la mejor manera posible el conjunto del nodo padre, intentando que cada hijo solo contenga patrones pertenecientes a una sola clase. Para esta división debemos elegir un atributo discriminante en los patrones que divida la muestra de la manera más homogénea posible. El proceso de subdivisión de los nodos se realiza recursivamente y en el caso extremo, termina cuando los todos los nodos hijos solo contienen muestras pertenecientes a la misma clase. Como se verá más adelante, es deseable parar la subdivisión antes de que se llegue al caso extremo de subdivisión para evitar el fenómeno de “sobreentrenamiento” o “sobreajuste” del modelo.

Entrada: **Conjunto de casos de entrenamiento, *E*, con sus variables y su clase**
Salida: **Árbol de clasificación (*T*)**

comienzo
si **Todos los casos en *E* son de la misma clase C_j**
entonces
Resultado nodo simple etiquetado como C_j
si_no
comienzo
Seleccionar un atributo X_i con valores $X_{i1},\ldots, X_{il}$
Particionar *E* en $E_1,\ldots E_l$ de acuerdo con los valores de X_i
Construir subárboles $T_1,\ldots, T_l$ para $E_1,\ldots E_l$
El resultado final es un árbol con raíz X_i y subárboles $T_1,\ldots,T_l$
Las ramas entre X_i y los subárboles están etiquetados mediante $x_{i1},\ldots, x_{il}$
fin
fin

Figura 9 Algoritmo general de inducción de árboles de clasificación.

Una vez que se decide detener la subdivisión de un nodo, éste será considerado como nodo hoja y se etiquetará con la clase que más esté presente en el conjunto de tuplas de entrenamiento que han "caído" en esa subdivisión. También existe la posibilidad de asignar una probabilidad de pertenecer a una u otra clase presente en la subdivisión. De esta manera una vez construido el árbol, los nuevos ejemplos a catalogar irán "cayendo" desde el nodo raíz hasta los nodos intermedios hasta llegar a un nodo hoja que los clasifique o les asigne una probabilidad de pertenecer a cierta clase. El algoritmo de inducción de árboles de clasificación queda resumido en la Figura Figura **9** Algoritmo general de inducción de árboles de clasificación.

Árboles de regresión

Los árboles de regresión se diferencian de los árboles de decisión principalmente en la naturaleza de la variable a predecir. Cuando la variable a predecir es continua se utilizan dos tipos de árboles:

- Árboles de regresión (Ramakrishnan, 2009; Steinberg, 2009): guardan el valor promedio de los valores de las tuplas que clasifican en las hojas.

- Árboles de modelos (Wang y Witten, 1997): utilizan regresión lineal para predecir los valores de las clases. Cada hoja contiene un modelo lineal que calcula el valor de la clase.

En este tipo de árboles el resultado de la predicción es un número que intenta aproximarse lo mejor posible al valor real que le corresponde a cada ejemplo según sus características. Un ejemplo en el que nos sería útil usar árboles de regresión sería un problema en el que lo que queremos predecir es el valor del crédito que le corresponderá a un cliente según sus características personales, como son el salario, la edad, el número de hijos, etc. (Figura 10.).

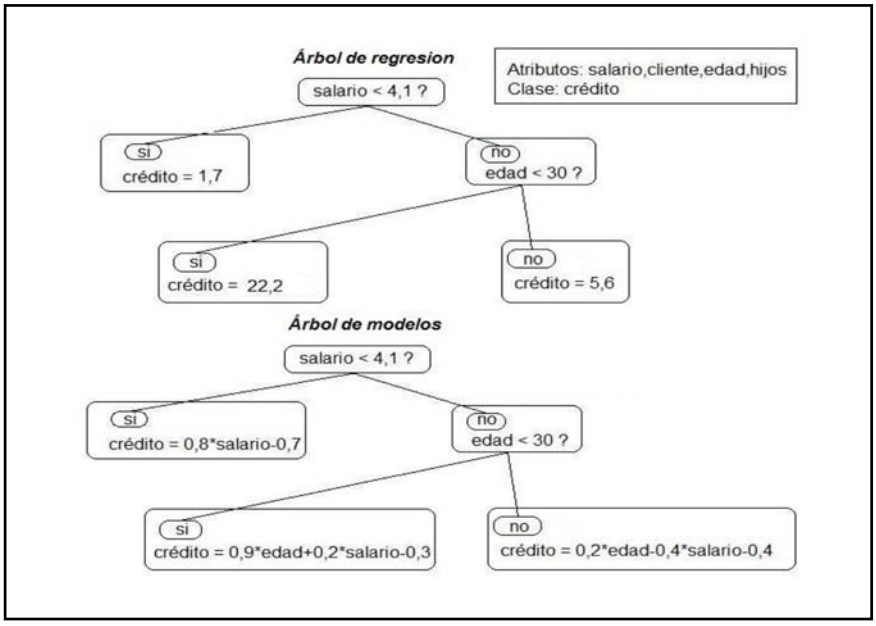


Figura 10 Ejemplo de árboles de regresión.

Árboles "sobreentrenados" y parsimoniosos

Como se comentó anteriormente, no siempre es deseable desarrollar el un árbol de tal modo que todos los nodos hoja sean homogéneos, haciendo que el error de entrenamiento se minimice. En principio, un error menor podría parecer un resultado

más deseable. Pero lo que realmente sucede con los árboles que consiguen ajustarse con precisión a las muestras de entrenamiento es que, al presentarles nuevos casos, pueden no funcionar bien y clasificar erróneamente. Cuando se produce esta situación se dice que el árbol está "sobre entrenado". Normalmente el "sobre entrenamiento" de un árbol está muy ligado a su complejidad (número de hojas).

Para evitar el "sobreajuste" conviene construir árboles parsimoniosos con una complejidad suficiente para resolver el problema y que sean capaces de clasificar con éxito nuevos casos no presentes en el entrenamiento (capacidad de generalización).

Los procedimientos que consiguen frenar el desarrollo de los árboles y generar árboles parsimoniosos se denominan procedimientos de poda. La poda puede aplicarse mientras se está desarrollando el árbol (pre-poda) o una vez desarrollado el árbol eliminando subárboles según un criterio determinado (post-poda).

Capítulo 3
Propuesta del modelo para la estimación del esfuerzo de desarrollo de software

Como hemos visto, en el capítulo anterior se detallaron las bases teóricas necesarias para esta investigación, como son: introducción al proceso de descubrimiento de conocimiento en bases de datos o KDD, minería de datos y sus herramientas, adicionalmente se describieron los diferentes métodos para la estimación de esfuerzo en el desarrollo de software, incluyendo los Árboles de clasificación y regresión.

Contando ya con el conocimiento del estado del arte y de las bases teóricas, podemos presentar en este capítulo los modelos propuestos para la estimación del esfuerzo de desarrollo de software. En la sección 3.1., se presenta un modelo basado en tareas de ingeniería y árboles de regresión. Mientras que en la sección 3.2., se detalla un modelo basado en la combinación de clasificadores y conteo de líneas de código.

A pesar que existen varios estándares internacionales que intentan normalizar el proceso de medición de software, no existe aún un consenso generalizado. Desde los orígenes de esta joven disciplina, se han intentado varias métricas, las cuales incluían, tanto contabilizar el número de líneas de código y comentarios, hasta considerar la influencia de la estructura del software sobre la fiabilidad del sistema.

Entre las principales métricas, se destacan las basadas en la complejidad del software, las basadas en el conteo de líneas de código y puntos de función, esta última, continúa siendo ampliamente empleada en la actualidad, debido a que se centra en las

funcionalidades entregadas al usuario. No obstante, la mayoría de estas métricas fueron concebidas antes de surgir el paradigma orientado a objetos, a partir de entonces, todas estas métricas han sido objeto de amplios estudios, adaptaciones y mejoras.

Sin embargo, la aparición del desarrollo ágil provocó un cambio en el panorama del desarrollo de software, esto se debe, en lo fundamental, a que el desarrollo ágil resulta una actividad adaptiva y no predictiva, lo cual hace que el proceso de estimación en un proyecto ágil sea totalmente diferente a la estimación de un proyecto tradicional, razón por la cual, en este capítulo se propone una nueva métrica de estimación del esfuerzo de desarrollo de software para metodologías ágiles, basada en tareas de ingeniería.

Actualmente el software es el elemento más costoso de la mayoría de los sistemas informáticos, un gran error en la estimación del esfuerzo puede ser lo que marque la diferencia entre beneficios y pérdidas.

Nuestras propuestas consisten en estimar con resultados aceptables, el esfuerzo de desarrollo en tareas de ingeniería de proyectos de software conocidos, aplicando técnicas de minería de datos (algoritmos de árboles de regresión), y a partir de los resultados obtenidos, analizar e interpretar el razonamiento que sigue el modelo con la finalidad de optimizar el tiempo de desarrollo y el diseño de las características de software de sus tareas de ingeniería.

3.1. Modelo de estimación basado en tareas de ingeniería

La estimación del esfuerzo de un proyecto de software se considera compleja, debido a que es un proceso altamente subjetivo (Nguyen-Cong y De Tran-Cao, 2013; Popli y Chauhan, 2014). Los árboles pueden procesar clases binarias o múltiples, así como atributos numéricos y nominales, e incluso valores faltantes. Además de resultar eficientes y poco consumidores de recursos, han sido empleados satisfactoriamente en la estimación del esfuerzo de desarrollo de software (Alsmadi y Nuser, 2013). Por tal motivo, se propone ETTpred (del inglés *Engineering Task and Tree-based predictor*), un

algoritmo capaz de estimar el esfuerzo de desarrollo ágil de software basado en árboles de regresión.

3.1.1.- Selección del clasificador base

ETTpred se entrenó con una base de datos basada en el desarrollo ágil de software. Para evaluar qué árbol es el más adecuado, fueron empleados en la experimentación los algoritmos basados en árboles más usados en la estimación del esfuerzo de desarrollo de software (Algabri et al., 2015; M. G. Almache et al., 2015; Borade y Khalkar, 2013; Dejaeger et al., 2012b; Kad y Chopra, 2012; Minku y Yao, 2013; Najadat et al., 2012; Waghmode y Kolhe, 2014; Wen et al., 2012):

- M5P: éste es un algoritmo de construcción de árboles de regresión lineal, basado en M5 de Quinlan (Ramakrishnan, 2009; Steinberg, 2009).

- M5Rules: genera una lista de regresión empleando la estrategia divide y vencerás. En cada iteración, construye un árbol modelo, empleando M5P y toma la mejor hoja como regla (Basten y Sunyaev, 2011).

- RandomTree: construye el árbol a partir de K atributos seleccionados aleatoriamente, luego los nodos se forman basado en un criterio de selección del mejor atributo. en cada nodo. Además, calcula la probabilidad de las clases mediante un procedimiento *holdout* (Dejaeger et al., 2012b). Puede ser empleado tanto para decisión como para regresión (Zhao y Zhang, 2007).

- REPTree: Construye un árbol de decisión/regresión empleando la información de varianza y realiza la poda usando como criterio la reducción del error. Funciona en dos fases (datos de aprendizaje y datos de poda). Primero se crea un conjunto de reglas que se sobre ajusta a los datos usados para el aprendizaje, después poda el conjunto de reglas usando ejemplares que no participaron en el aprendizaje (Zhao y Zhang, 2007).

3.1.2. Formalización y Algoritmo de ETTpred

En la formalización del algoritmo, tanto en el proceso de entrenamiento como en el proceso de predicción, se tendrán en cuenta las variables de entrada y salida. Ambos algoritmos serán escritos en pseudocódigo e incluirá el análisis de complejidad.

3.1.2.1. Construcción y entrenamiento de ETTpred

El proceso de entrenamiento y construcción de ETTpred se muestra en el algoritmo 1. ETTpred recibe como parámetros de entrada una lista de tareas de ingeniería y sus vectores de características, cada instancia perteneciente a las tareas de ingeniería se codifica, creando así el vector de entrada para el árbol. Posteriormente, se construye el árbol de regresión pasándole como parámetro el vector de entrada codificado y se devuelve el árbol como el modelo de estimación, este proceso se formaliza en el Algoritmo 1.

Algoritmo 1. ETTpred. Entrenamiento y construcción del modelo.

Entrada:
    tareas: lista de tareas de ingeniería.
    vector: Vector de características.
Salida:
    modelo: árbol de regresión.
Algoritmo:

```
1:  Para cada tarea en tareas
2:  |   Para cada variable en vector
3:  |   |   vectorEntrada[tarea] += Codificar(variable)
4:  |   Fin
5:  Fin
6:  modelo = ConstruirÁrbol(vectorEntrada)
7:  Retornar modelo
```

El primer ciclo (de la instrucción 1 a la 5) es lineal ***O(n)***, esto se debe a que depende del número de tareas que se empleen durante la construcción del predictor ***n***. Los siclos internos (instrucciones 2 y 3) recorren cada variable del vector inicial, realizando la codificación, por lo cual la complejidad de esta sección de código es constante: ***m***. Donde ***m*** se corresponde con el número de variables del vector. Por último, el segundo ciclo lo compone la instrucción 6, que es donde se construye el modelo. La construcción del

modelo equivale al coste de construir un árbol de regresión. Esto podría escribirse como ***O(m·n·log(n))***, donde ***m*** el número de variables y ***n*** el número tareas de ingeniería. Considerándose, entonces este segundo conjunto de instrucciones con complejidad superior a la logarítmica pero inferior a la cuadrática.

A partir de este análisis, podría decirse que el coste promedio de la construcción del estimador es ***O(n) +·O(m·n·log$_2$(n))***. Donde, la mayor complejidad es ***O(m·n·log$_2$(n))***, por lo que la complejidad ETTpred es **logarítmica.**

3.1.2.2.- Estimación del esfuerzo ETTpred

Una vez que se cuenta con el modelo ETTpred, el proceso para la estimación del esfuerzo de desarrollo de software se muestra en el Algoritmo 2. Donde, éste recibe como entrada el modelo ETTpred y el vector de características de la nueva orden de trabajo a estimar. Primeramente, se codifica el vector de características, creándose el vector de entrada. Se evalúa en el modelo ETTpred el nuevo vector de entrada y éste devuelve una regla en modo conjuntivo y el valor del tiempo estimado. Luego cada variable de la regla es sometida a un mecanismo de interpretación, el cual la convierte en una regla semántica (ver sección 4.4.3). Finalmente, el algoritmo retorna como respuesta la regla semántica y el tiempo estimado.

Algoritmo 2. ETTpred. Estimación del esfuerzo de desarrollo.

Entrada:
modelo: árbol de regresión.
vector: Vector de características de la nueva tarea de ingeniería.

Salida:
reglaSemantica: regla semántica de la estimación del tiempo de cumplimiento de la tarea de ingeniería.
tiempo: valor entero de la estimación del tiempo de cumplimiento de la tarea de ingeniería.

Algoritmo de construcción:

```
1:  vectorEntrada = Codificar(vector)
2:  regla, tiempo = EvaluarModeloLineal(modelo, vectorEntrada)
3:  Para cada variable en regla
4:  |       reglaSemantica += MecanismoInterpretación(variable)
5:  Fin
6:  Retornar reglaSemantica, tiempo
```

El proceso de codificación del vector de entrada (instrucción 1), es constante ***m***, ya que depende únicamente de la cantidad de variables que componen el vector. La estimación del esfuerzo de desarrollo de software (instrucción 2), se realiza evaluando el vector de entrada en el modelo del ETTpred. Lo cual significa, recorrer el árbol de regresión generado. Éste, es un proceso lineal ***O(n)***, que de pende de la profundidad del árbol ***n***. El último ciclo lo constituye el mecanismo de interpretación (de la instrucción 3 a la 5), es un proceso constante y depende de la dimensión de la regla ***m***. Por tanto, el coste promedio de la estimación empleando ETTpred es ***O(n)***, o lo que es igual, complejidad **lineal**.

3.2.- Modelo de estimación basado en líneas de código

La estimación del esfuerzo de un proyecto de software se considera compleja, debido a que es un proceso altamente subjetivo, esto unido a la gran variabilidad de los datos, lo convierte en un problema de espacio desconocido de clasificadores, donde es complicado encontrar el clasificador óptimo, por tal razón, proponemos una solución basada en la combinación de clasificadores (Kuncheva, 2004).

Como se muestra en la **Figura 11**, se propone un método que combina los resultados de un clasificador que realiza una predicción granular (primer nivel). En esta primera etapa solo determina si el proyecto de software será: orgánico (***O***); semi-libre (***SL***); o empotrado (***E***). Además, brinda un rango en que podría encontrarse el número de líneas de código que podría tener el proyecto. El segundo nivel actúa en función de los resultados de la primera. Donde, además de la información sobre el proyecto, se agrega la predicción sobre el rango de LOC del proyecto. En este nivel existe un clasificador especializado en cada tipo de proyectos, capaz de estimar el valor del esfuerzo que conllevaría el mismo.

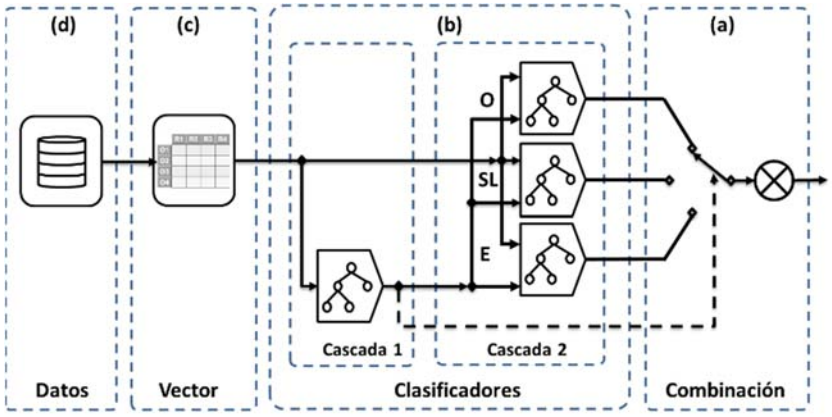


Figura 11 Esquema del modelo propuesto. (a) nivel de combinación de los resultados, (b) nivel de clasificadores base, (c) nivel de vector, y (d) nivel de datos.

Al igual que otros modelos de combinación de clasificadores, EEpred emplea diferentes algoritmos de construcción de los clasificadores base. No obstante, cada etapa es entrenada con bases de datos diferentes. Todo lo cual garantiza la diversidad. Donde, el primer nivel emplea la base de datos DS1. En la segunda etapa cada clasificador se entrena con DS2, DS3 o con DS4, en dependencia de si se especializarán en proyectos orgánicos, semi-libres o empotrados, respectivamente. Esto garantiza que cada clasificador logre una mejor estimación del esfuerzo en su categoría.

3.2.1. Análisis de la diversidad

Existen distintas estrategias para lograr la diversidad de los clasificadores base. La mayor parte de estas estrategias se basan en modificar el conjunto de entrenamiento de cada clasificador base. *Boosting*, realiza un entrenamiento iterativo variando los pesos de las instancias que emplea. *Bagging y Random Subspaces,* toman distintos subconjuntos para el entrenamiento. *Random Forests,* selecciona los atributos que se emplean en cada árbol.

La diversidad de EEpred se logra, de igual manera, incidiendo sobre los niveles de base de datos (Figura 11.D), clasificador base (Figura 11.B) y combinación (Figura 11.A). Sin embargo, en el nivel de extracción de rasgos (Figura 11.C), se mantiene el mismo vector de características para todos los clasificadores base. Esto fuerza a que cada

clasificador base se construya, únicamente, en dependencia de las instancias que recibe, logrando la deseada diversidad (Figura 11).

Este método presenta varias ventajas:

- Sencillez: no requiere de esfuerzo computacional extra, ni depende de configuraciones de estados anteriores para lograr la diversidad

- No ambiguo: dividiendo el espacio de búsqueda en regiones fijas y no presenta componente aleatorio alguno, lo que permite la reproducción exacta de la experimentación.

3.2.2. Selección del clasificador base

En la primera cascada, la predicción es tratada como un problema de clasificación. No obstante, una característica deseada del modelo propuesto es su capacidad explicativa, razón por la cual fueron escogidos algoritmos basados en árboles. Debido a que cada rama del árbol puede ser traducida como un conjunto de reglas de tipo ***Si–Entonces***, lo cual es fácilmente interpretable.

Por otra parte, los árboles pueden procesar clases binarias o múltiples, así como atributos numéricos y nominales, e incluso valores perdidos. Además de resultar eficientes y poco consumidores de recursos, han sido empleados satisfactoriamente en la estimación del esfuerzo de desarrollo de software (Alsmadi y Nuser, 2013).

Algoritmos de construcción de árboles empleados:

- J48: es la implementación del algoritmo C4.5 (Ramakrishnan, 2009) presente en la herramienta Weka. Este algoritmo se puede considerar un híbrido entre CART y C4.5. Puede trabajar con conjuntos de datos categóricos y continuos (Witten et al., n.d.).

- RandomTree (Zhao y Zhang, 2007).

- REPTree (Zhao y Zhang, 2007).

- M5P (Ramakrishnan, 2009; Steinberg, 2009).

Otros algoritmos de construcción de árboles no fueron tomados en cuenta debido a sus características propias. Siendo excluidos: ***ID3***, no concibe el trabajo con rasgos numéricos; ***ModelTree***, realiza un cambio de espacio de las variables numéricas a booleanas sintéticas y no toma en cuenta la dependencia entre rasgos; ***GATree***, en caso de tener los objetos descritos solamente por rasgos numéricos, este procedimiento sólo obtendría un árbol de un nodo y dos hojas y en presencia de ruido tiende a obtener clasificadores sobre entrenados; ***WDT***, toma en cuenta pesos, por lo cual sería altamente subjetivo; y, ***ODT***, no trabaja con datos cualitativos (Steinberg, 2009). De igual forma, no se incluyeron NBTree (Kohavi, 1996) y CART (Steinberg, 2009) por no estar incluidos en la versión 3.7.13 de Weka (Witten et al., n.d.).

En el segundo nivel la predicción se trata como un problema de regresión. Por lo que se realizó el análisis con los árboles de regresión RandomTree (Zhao y Zhang, 2007), REPTree (Zhao y Zhang, 2007) y M5P (Steinberg, 2009).

3.2.3. Combinación de los resultados

Existen varios métodos de combinación de la clasificación. Entre los que se destacan el voto mayoritario, voto mayoritario ponderado, recuento de Borda, *naive* Bayes, entre otros, ya sean a nivel abstracto, de ranking o de medición (Kuncheva, 2004). Para el modelo propuesto, se decidió por una combinación a nivel de medición, ya que se conoce previamente cuál es la efectividad de cada clasificador base del segundo nivel para cada uno de los tipos de proyectos. Con este método, cada predictor realiza su estimación del esfuerzo de desarrollo de software y el resultado de la combinación sería la selección del clasificador adecuado. Esta selección se realiza teniendo en cuenta la clasificación realizada por el primer nivel del multiclasificador (Figura 11).

3.2.4. Formalización y Algoritmo

Al igual que en el caso del estimador ETTpred, EEpred será formalizado tomando en cuenta: los parámetros de entrada y salida; el proceso de entrenamiento; el proceso de predicción; y, su análisis de complejidad.

3.2.4.1. Construcción y entrenamiento de EEpred

La construcción y entrenamiento del multiclasificador EEpred comienza con la subdivisión las instancias según el tipo de proyecto a que pertenezcan: orgánicos, semi-libres o empotrados. Donde, el primer nivel del estimador, es entrenado con todo el conjunto de entrenamiento, pero con clase discreta. Mientras que, en la segunda cascada, cada clasificador es entrenado con el subconjunto de instancia correspondiente a cada clasificador, pero con la clase numérica. Esto permite que el primer nivel sea empleado para predecir a qué tipo de proyecto pertenece el vector de entrada y el segundo nivel es el encargado de estimar el esfuerzo necesario para el desarrollo de dicho proyecto. El proceso de construcción y entrenamiento de EEpred, podría formalizarse de la siguiente manera (Algoritmo 3):

Algoritmo 3. EEpred: Entrenamiento y construcción del algoritmo

Entrada:
proyectos: lista de proyectos de entrenamiento.
Salida:
modelo: compuesto por 1 árbol de decisión y 3 árboles de regresión.
Algoritmo:

```
1:  Para cada proyecto en proyectos
2:  |   Si proyecto.LOC <= 50 entonces proyectos orgánicos += proyecto
3:  |   Sino proyecto.LOC <= 300 entonces proyectos_semi_libres += proyecto
4:  |   Sino proyectos_empotrados += proyecto
5:  Fin
6:  Para cada proyecto en proyectos
7:  |   modelo.decisión = ConstruirÁrbol(proyecto)
8:  Fin
9:  Para cada proyecto en proyectos_ orgánicos
10: |   modelo.orgánico = ConstruirÁrbol(proyecto)
11: Fin
12: Para cada proyecto en proyectos_ semi_libres
13: |   modelo.semi_libre = ConstruirÁrbol(proyecto)
14: Fin
15: Para cada proyecto en proyectos_empotrados
16: |   modelo.empotrado = ConstruirÁrbol(proyecto)
17: Fin
18: Retornar modelo
```

Para calcular el coste computacional de la construcción del algoritmo EEpred es necesario analizar cada paso del algoritmo independientemente. Pero, para hacer más simple el entendimiento, la complejidad del algoritmo será analizada a partir del análisis de sus dos cascadas de clasificadores base.

El primer ciclo (de la instrucción 1 a la 5) resulta lineal ***O(n)***, debido a que depende del número de proyectos ***n***, que se empleen durante la construcción del predictor. La construcción del primer nivel se encuentra en el siguiente ciclo (de la instrucción 6 a la 8). Donde se construye un árbol de decisión empleando todos los proyectos de la matriz de entrenamiento. Este proceso se describe como ***O(m·n·log(n))***, donde ***m*** el número de variables y ***n*** el número tareas de ingeniería.

En la construcción del segundo nivel (de la instrucción 9 a la 17), se encuentran tres ciclos iguales. Donde, en cada uno se construye un árbol de regresión empleando los proyectos correspondientes al tipo de proyectos correspondiente con cada estimador. Cada uno de estos ciclos tienen la complejidad ***O(m·n·log(n))***. donde ***m*** el número de variables y ***n*** el número tareas de ingeniería. Por lo que el coste promedio de la construcción del estimador EEpred es $O(m \cdot n \cdot log_2(n))$, considerándose **logarítmica.**

3.2.4.2.- Estimación del esfuerzo EEpred

La estimación del esfuerzo, empleando el modelo de estimación EEpred, es sencilla. Cuando se introduce los datos de entra del proyecto a estimar, el primer nivel de EEpred, decide si el tipo de proyecto es orgánico, semi-libre o empotrado y calcula el rango de líneas de código del mismo. El segundo nivel realiza la estimación basada en la información que brinda la primera sobre el número de líneas de código y los datos del proyecto. La selección del valor estimado final, se realiza en base a la clasificación brindada por la primera cascada. Este proceso podría formalizarse de la siguiente manera (Algoritmo 4):

Algoritmo 4. EEpred: proceso de predicción de mapas de contacto.

Entrada:
modelo: compuesto por 1 árbol de decisión y 3 árboles de regresión.
vector: Vector de características del nuevo proyecto.
Salida:
reglaSemantica: regla semántica de la estimación del tiempo de cumplimiento de la tarea de ingeniería.
tiempo: valor entero de la estimación del tiempo de cumplimiento de la tarea de ingeniería.
Algoritmo:

```
1:  tipo_proyecto, LOC = EvaluarModelo(modelo.decisión, vector)
2:  esfuerzo_orgánico = EvaluarModelo(modelo.orgánico, vector, LOC)
3:  esfuerzo_semi_libre = EvaluarModelo(modelo.semi_libre, vector, LOC)
4:  esfuerzo_empotrado = EvaluarModelo(modelo.empotrado, vector, LOC)
5:  En caso de que tipo_proyecto sea:
6:  |   orgánico: esfuerzo = esfuerzo_orgánico
7:  |   orgánico: esfuerzo = esfuerzo_semi_libre
8:  |   orgánico: esfuerzo = esfuerzo_empotrado
9:  Fin
10: Retornar esfuerzo
```

Como se muestra en el algoritmo 4, primeramente, se realiza la predicción del tipo de proyecto y de la cantidad de líneas de código (instrucción 1), posteriormente, se estiman los valores del esfuerzo para los tipos de proyectos orgánicos, semi_libre y empotrado, la estimación del esfuerzo de desarrollo de software (instrucciones de la 2 a la 4), se realiza evaluando el vector de entrada en el modelo del ETTpred; lo cual significa, recorrer el árbol de regresión generado. Éstos, son procesos lineales ***O(n)***, que de penden de la profundidad del árbol ***n***. En todos los casos, la estimación del esfuerzo de desarrollo de software, se realiza evaluando el vector de entrada en el modelo del ETTpred, junto con el número de líneas de código (LOC). Por último, se realiza la combinación de los resultados mediante selección, tomando en cuenta la clasificación realizada por el predictor de la primera cascada. El costo de la combinación es constante. Por tanto, el coste promedio de la estimación del esfuerzo de desarrollo de software empleando EEpred es ***lineal***, o lo que es igual ***O(n)***.

3.3. Metodología de estimación del esfuerzo basada en tareas de ingeniería

Dentro de las metodologías de estimación del esfuerzo de desarrollo ágil de software, más ampliamente difundidas, se encuentra la basada en puntos de historia. Ésta, no mide el esfuerzo en tiempo, sino en función de la complejidad de las historias de usuarios (H.U.) con respecto a otras. Lo que la convierte en una metodología dependiente del equipo de desarrollo y del planificador. Por lo que no puede ser comparada la estimación de un mismo proyecto por equipos diferentes.

La metodología de estimación del esfuerzo de desarrollo de software que se propone, en esta investigación, es independiente a la metodología de desarrollo de software empleada. Además, toma en cuenta las características del software a desarrollar y del personal que integrará el equipo de desarrollo. De manera que, intenta mitigar las limitaciones que presenta la estimación basada en historias de usuario.

La estimación en metodologías ágiles, intenta definir el plazo en que se desarrollarán las tareas en un sprint, estas tareas tienen como característica que son desarrolladas por un equipo y no por programadores individuales, por lo que es importante tomar en consideración las características de este equipo. Debido a que la duración de desarrollo de una tarea de ingeniería está en función de la experiencia del programador.

Este método también puede ser empleado para calcular el tiempo total de duración de un sprint o del proyecto completo, así como el costo total del mismo. Pero, ello depende de cómo se realiza la planificación (Figura 12.).

La Figura 12., realiza un acercamiento al proceso de planificación en la programación ágil. Donde, a partir de los requerimientos escritos por los usuarios, comienza el proceso de planificación de entregables. Una vez que se determinan los requerimientos que van a cada iteración, se determinan las tareas de ingeniería y se elaboran las órdenes de trabajo.

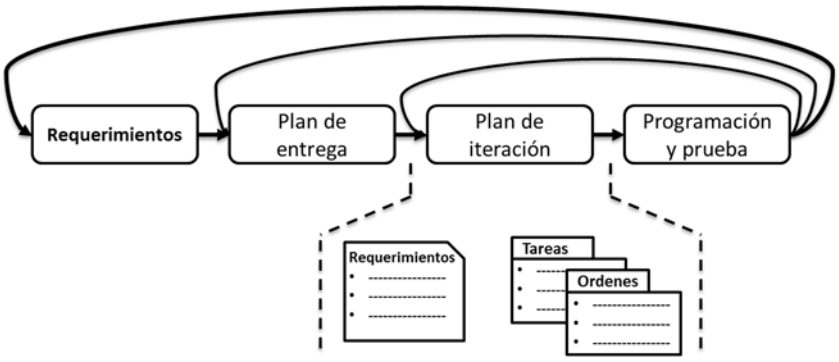


Figura 12 Diagrama de iteraciones del proceso de planificación en programación ágil.

Una técnica que ha sido ampliamente empleada es la de estimación por analogía. Para ello, se emplea la información de proyectos conocidos. Sin embargo, la gran variabilidad en los requerimientos de los proyectos hace que estas estimaciones presenten un amplio margen de error. Tomando en cuenta lo antes planteado, la metodología propuesta de estimación del esfuerzo, combina las ventajas de la estimación basada en analogía con elementos de la estimación basada en líneas de código. Por otra parte, recoge información de las características del equipo de desarrollo, para reducir su efecto en el tiempo de desarrollo. Y, se basa en la descomposición de las historias de usuario en sus tareas de ingeniería.

Este método, parte de la identificación de las historias de usuario del proyecto en una base de datos de plantillas. Este proceso se realiza a partir del empleo de una técnica estándar de alineamiento, la programación dinámica. En la cual se crea una matriz de similitudes y se calcula el nivel de parecido entre la historia de usuario objetivo y las historias de usuario en plantilla. Una vez identificada una historia de usuario en la plantilla, ésta se descompone en sus tareas de ingeniería y a partir de esta información, más la información de los miembros del equipo, se realiza la estimación del esfuerzo de desarrollo de la historia.

Las etapas del proceso para la estimación del esfuerzo basada en tareas de ingeniería (Figura 13.), pueden resumirse en:

- Identificación de plantillas. Se identifican historias de usuario conocidas que estén relacionadas con el proyecto a estimar. Para ello se emplean técnicas basadas similaridad y programación dinámica.

- Estimación. Se descompone el requerimiento plantilla en sus tareas de ingeniería, y a partir de la información del equipo de desarrollo, se realiza la estimación. Para ello se emplea el modelo heurístico que se propone en la sección 3.2.

- Cálculo del esfuerzo del proyecto. Una vez que se cuenta con el esfuerzo de todas las tareas de ingeniería, se estima el esfuerzo total del proyecto. Este proceso depende de cómo se planifique el desarrollo del mismo.

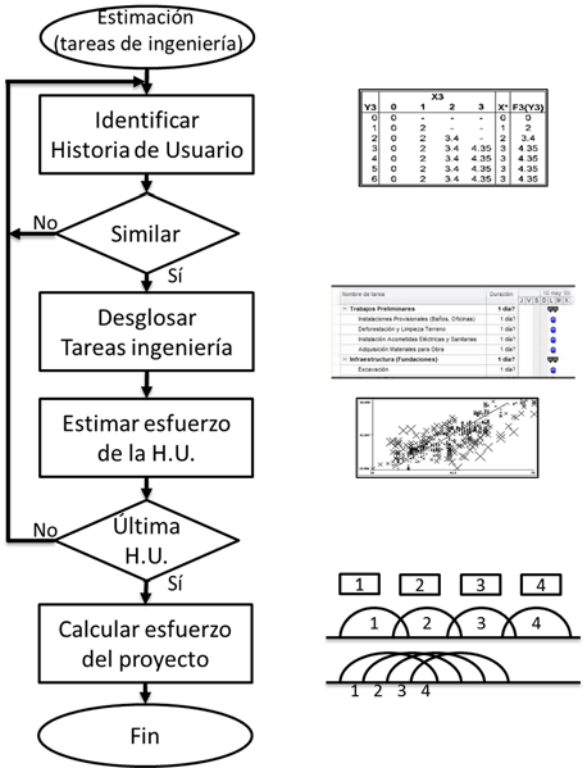


Figura 13 Algoritmo para la estimación del esfuerzo de desarrollo de software basado en tareas de ingeniería.

Como se observa en la Figura 13., la estimación del esfuerzo basado en tareas de ingeniería es un proceso iterativo. Donde, el cálculo del esfuerzo total consiste en la suma del esfuerzo calculado para cada tarea de ingeniería. Sin embargo, el cálculo del

tiempo total de duración del proyecto depende de cómo se planificarán los sprints, de forma secuencial, parcialmente solapados o con solapamiento total.

Un elemento importante en el modelo, lo constituyen los atributos. Cada atributo se cuantifica para un entorno de tarea de ingeniería. El significado de los atributos es el siguiente, según su tipo:

- **Complejidad** (Ordinal): Determina si se requieren gran cantidad de decisiones lógicas, complicados procedimientos matemáticos o difícil manejo de excepciones. Puede tomar los valores: ***muy simple, simple, promedio, compleja, muy compleja***.

- **conReuso** (Numérico): Porcentaje de código existente que puede ser utilizado en el desarrollo de la tarea.

- **ConocimientoAplicacion** (Ordinal): Experiencia con la que cuenta el programador en el desarrollo de aplicaciones y tareas similares a la que tiene que desarrollar. Puede tomar los valores: ***ninguna, muy poca, poca, promedio, alta, muy alta***.

- **ConocimientoLenguaje** (Ordinal): Experiencia con la que cuenta el programador en el uso del lenguaje empleado para el desarrollo de la tarea. Puede tomar los valores: ***ninguna, muy poca, poca, promedio, alta, muy alta***.

- **Fiabilidad** (Ordinal): Grado de precisión esperada, con que una tarea realice su función. Puede tomar los valores: ***muy poca, poca, promedio, alta, muy alta***.

- **HerramientasSoftware** (Ordinal): Empleo de herramientas de software que faciliten y reduzcan el tiempo de desarrollo de la tarea específica. Puede tomar los valores: ***ninguna, muy poca, poca, adecuada, alta, todas las posibles***.

En este modelo, la estimación se realiza en horas hombres. La cual puede ser convertida a meses hombres simplemente dividiendo la cantidad de horas entre 176, o a cualquier otra unidad de medición, con solo aplicar la razón adecuada.

3.4. Estimación del esfuerzo en el desarrollo ágil. Ejemplo prácticos

Con el objetivo de explicar el funcionamiento de la metodología propuesta, fueron diseñado dos casos de estudio. Para ello se tomó en cuenta, que los casos de estudio no deben ser grandes grupos de datos ni diseñarse en función de la aplicación de pruebas estadísticas. Además, la intención de éstos es ilustrativa, lo que significa que solo serán empleados para comprender el modelo, no para demostrar su efectividad. Un elemento importante a tomar en cuenta es que, para el diseño de los casos de estudio, fueron empleadas tareas de ingeniería de historias de usuario donde se manejan bases de datos.

Ejemplo práctico

Objeto de estudio: el proceso de estimación del esfuerzo de desarrollo de software basado en tareas de ingeniería.

Tipo de ejemplo: retrospectivo.

Objetivo: (***ilustrativo***) mostrar el funcionamiento de la metodología propuesta, a partir del empleo de una historia de usuario real.

Restricciones: Solo se mostrará el proceso de estimación del esfuerzo de una historia de usuario y no del sistema completo.

Aprobación ética: las tareas de ingeniería empleadas en este caso de uso fueron extraídas de historias de usuarios reales, de proyectos realizados por los centros de desarrollo de software de la Universidad de Ciencias Informáticas de las provincias de Las Villas y Ciego de Ávila en Cuba[4].

Planificación:

- ***Historia de usuario***: "Todos los usuarios deben tener un perfil y el derecho a verlo y modificarlo".

[4] Universidad de Ciencias Informáticas de Cuba, URL: http//:www.uci.cu

Funcionamiento del modelo:

El modelo de estimación propuesto funciona de la siguiente forma:

1. Identificación de la historia de usuario en la base de datos de plantillas del modelo:

   H.U. objetivo:

   - "Todos los usuarios deben tener un perfil y el derecho a verlo y modificarlo".

   H.U. plantilla:

   a) "Los usuarios cuentan con un perfil y tienen el derecho a verlo y modificarlo". (similaridad: 0,71).

   b) "Los usuarios del sistema pueden autenticarse y acceder a su perfil". (similaridad: 0,68).

2. Desglosar tareas de ingeniería para la opción (a):

   1. Crear prototipo de interfaz para la funcionalidad "Perfil de usuario".
   2. Integrar la funcionalidad Crear vista.
   3. Agregar a la interfaz editar tabla.
   4. Crear las URL's para gestionar datos en una tabla.
   5. Crear las URL para gestionar rol.
   6. Integrar funcionalidad "Dar de baja del sistema".
   7. Agregar objetos SQL al árbol.

3. Estimación del esfuerzo de cada tarea de ingeniería:

   a) Características del equipo de desarrollo: Se determinan las características de cada uno de los programadores a los que le fueron asignadas las tareas.

Por ejemplo:

- conocimientoAplicacion: ***alta → 5***
- conocimientoLenguaje: ***poco → 2***
- herramientasSoftware: ***adecuada → 3***

b) Vector formado para cada tarea de ingeniería:

Id	complejidad	conReuso	conocimientoAplicacion	conocimientoLenguaje	fiabilidad	herramientasSoftware	tiempoR
1	2,	0,	1,	2,	4,	0,	?
2	2,	34,	3,	3,	3,	2,	?
3	3,	50,	3,	3,	3,	2,	?
4	2,	35,	3,	3,	3,	1,	?
5	3,	45,	3,	3,	3,	3,	?
6	2,	30,	3,	3,	2,	2,	?
7	2,	26,	3,	3,	2,	2,	?

c) Esfuerzo estimado para cada tarea de ingeniería:

No	Tareas de Ingeniería	Esfuerzo
1.	Crear prototipo de interfaz para la funcionalidad “Perfil de usuario”.	10
2.	Integrar la funcionalidad Crear vista.	40
3.	Agregar a la interfaz editar tabla	40
4.	Crear las URL’s para gestionar datos en una tabla.	20
5.	Crear las URL para gestionar rol.	49
6.	Integrar funcionalidad “Dar de baja del sistema”.	27
7.	Agregar objetos SQL al árbol.	48

d) Esfuerzo estimado para la historia de usuario: Se calcula a partir de la suma total de todas las tareas de ingeniería.

- Esfuerzo total: ***10 + 40 + 40 + 20 + 49 + 27 + 48 = 234 horas / hombres***
- Lo que equivale a: ***1 mes y una semana / hombre***

3.5. Sumario

En este capítulo fueron formalizados los algoritmos propuestos, ETTpred (basado en tareas de ingeniería y árboles de regresión) y EEpred (basado en conteo de líneas de código y la combinación de clasificadores).

Se propuso una nueva metodología para la estimación del esfuerzo de desarrollo de software, basada en tareas de ingeniería. Esta metodología realiza la estimación en horas hombres, en vez de puntos de historias de usuario o puntos de tareas de ingeniería, sin embargo, no es dependiente del tipo de proyecto, el lenguaje o plataforma de desarrollo, ni de la experiencia del equipo de desarrollo. Es una metodología que puede catalogarse como de analogía, heurística, algorítmica y no paramétrica a la vez, debido a que se basa en tareas de ingeniería de proyectos ya resueltos, no calcula el esfuerzo a partir de los parámetros de entrada, sino que emplea algoritmos heurísticos.

Capítulo 4
Evaluación y validación del modelo propuesto

Como hemos visto, en el capítulo anterior fueron formalizados los algoritmos propuestos, ETTpred (basado en tareas de ingeniería y árboles de regresión) y EEpred (basado en conteo de líneas de código y la combinación de clasificadores), proponiéndose una nueva metodología que puede catalogarse como de analogía, heurística, algorítmica y no paramétrica a la vez, debido a que se basa en tareas de ingeniería de proyectos ya resueltos y no calcula el esfuerzo a partir de los parámetros de entrada, sino que emplea algoritmos heurísticos.

En este capítulo se presenta la metodología utilizada para evaluar los modelos anteriormente propuestos, describiéndose las bases de datos empleadas y mostrando los estadígrafos y pruebas estadísticas. Adicionalmente, se efectúa un estudio de la influencia de las variables sobre la estimación del esfuerzo, así como la validación de las propuestas presentadas en la investigación, los modelos EEpred y ETTpred. Se muestran los resultados experimentales para el análisis de la robustez y capacidad de generalización, así como del comportamiento del error y finalmente, se expone el mecanismo de interpretación de ambas propuestas.

4.1. Evaluación del modelo propuesto

En esta sección se presenta la metodología utilizada para evaluar los modelos propuestos, en 4.1.1. se describen las bases de datos empleadas y en 4.1.2. se muestran los estadígrafos y pruebas estadísticas

4.1.1. Bases de datos

Para realizar el análisis, fueron seleccionadas bases de datos públicas, extraídas del repositorio PROMISE (*PRedictOr Models In Software Engineering Software*) (Sayyad Shirabad y Menzies, 2005) y un repositorio de órdenes de trabajo obtenido de aplicaciones de software de base, implementadas por los centros de desarrollo de software de la Universidad de Ciencias Informáticas de las provincias de Las Villas y Ciego de Ávila en Cuba. Estas bases de datos fueron separadas en tres grupos, las basadas en: conteo de líneas de código, puntos de función y órdenes de trabajo.

4.1.1.1. Basadas en Líneas de Código

Como se muestra en la Tabla 4., este conjunto de datos recoge un amplio número de proyectos y atributos, los cuales representan un elevado número de casos útiles para la estimación del esfuerzo de desarrollo de software (Este conjunto de datos puede ser descargado desde el repositorio SEEBET[5]).

Tabla 4 Caracterización del set datos empleados en la experimentación.

Base de datos	Dimensiones		Esfuerzo			
	Proyectos	**Atributos**	**Min**	**Max**	**Med**	**Dev-est**
Cocomo81	63	17	5,9	11400	683,3	1821,6
Nasa_1	60	16	8,4	3240	406,4	657,0
Nasa_2	93	24	8,4	8211	624,4	1135,9
Cocomo_Sdr	12	25	1,0	22	5,7	6,8
Total a usar	**228**	**11**	**1,0**	**11400**	**550,8**	**1252,9**

Para entrenar el modelo propuesto, fueron escogidos 11 atributos comunes a todas las bases de datos, estos describen características de los proyectos en cuanto a: tipo de software, personal que participará en el proyecto y datos propios de la organización (Tabla 5). El valor de los atributos empleados es nominal, no obstante, debido a que los problemas relacionados con el hardware ya no se consideran una limitación, las variables relacionadas con estos rasgos fueron excluidas en esta investigación.

[5] https://sites.google.com/site /seebetrepository/

Tabla 5 Atributos escogidos para crear el modelo.

No	Atributo	Objetivo	Descripción
1	**Rely**	Software	Confiabilidad del software.
2	**Data**		Dimensión de la base de datos.
3	**Cplx**		Complejidad del software.
4	**Acap**	Equipo de desarrollo	Calificación de los analistas.
5	**Aexp**		Experiencia en el tipo de aplicación.
6	**Pcap**		Calificación de los programadores.
7	**Vexp**		Experiencia en la máquina virtual.
8	**Lexp**		Experiencia en el lenguaje de programación.
9	**Modp**	Proyecto	Prácticas de programación.
10	**Tool**		Empleo de herramientas de desarrollo.
11	**Sced**		Problemas de planificación.

A partir de estos atributos, se construyeron cuatro sets de datos diferentes. Esta distribución se corresponde con los modos de clasificación de los proyectos de software que sugiere el modelo COCOMO (Sheta, 2006) y los criterios establecidos en el 2012 por Galinina, Burceva, y Parshutin y por Bedini en el 2005. (Las bases de datos se adjuntan como material suplementario).

DS1: Incluye todos los proyectos (228 instancias) y la clase es discreta. Para ello, se hizo necesario realizar un procedimiento de discretización de los atributos de la base de datos cocomo81. Empleando como criterio la escala propuesta por el modelo COCOMO (A. F. Sheta, 2006). Este set de datos se emplea para entrenar el primer nivel del multiclasificador.

DS2: Incluye solo los proyectos catalogados como orgánicos, por debajo de las 50 mil líneas de código (74 instancias). Este set de datos se emplea para entrenar el clasificador especializado en proyectos orgánicos, en el segundo nivel del multiclasificador.

DS3: Incluye solo los proyectos catalogados como semi-libres, entre 50 y 300 mil líneas de código (105 instancias). Este set de datos se emplea para entrenar el clasificador especializado en proyectos semi-libres, en el segundo nivel del multiclasificador.

DS4: Incluye solo los proyectos catalogados como empotrados, mayores de 300 mil líneas de código (49 instancias). Este set de datos se emplea para entrenar el clasificador especializado en proyectos empotrados, en el segundo nivel del multiclasificador.

4.1.1.2. Basadas en Puntos de Función

El cálculo de los puntos de función conlleva un nivel elevado de definición y maduración del proyecto. Es una métrica que permite traducir en un número, el tamaño de la funcionalidad que brinda un producto de software. Se estima desde el punto de vista del usuario, a través de una suma ponderada de las características del producto. Resulta independiente de la tecnología utilizada para la construcción y explotación del software. Las variables empleadas para este estudio fueron cuatro:

- **Input**: Procesos en los que se introducen datos
- **Output**: Procesos en los que se envía datos al exterior de la aplicación
- **Enquiry**: Procesos consistentes en la combinación de una entrada y una salida
- **File**: Grupos de datos relacionados entre sí internos al sistema

En la Tabla 6., se muestra el conjunto de datos empleados para el análisis de los atributos relacionados con puntos de función (El conjunto de datos puede ser descargado del repositorio SEEBET[6]). Éste, reúne una amplia cantidad de proyectos, útiles para la estimación del esfuerzo y la comparación con las estimaciones basadas en líneas de código y órdenes de trabajo. (Las bases de datos se adjuntan como material suplementario).

Tabla 6 Caracterización del set datos empleados el análisis basado en puntos de función.

Base de datos	Dimensiones		Esfuerzo			
	Proyectos	Variables	mínimo	máximo	media	dev-est
Albrecht	24	8	0,5	105,2	21,88	28,42
China	499	19	26	54620	3921,05	6480,86
Total a usar	**523**	**4**	**0,5**	**54620**	**3742,12**	**6382,58**

[6] https://sites.google.com/site/seebetrepository/

4.1.1.3. Basadas en tareas de ingeniería

Para realizar la experimentación del modelo propuesto, fue empleado un repositorio de tareas de ingeniería obtenido de aplicaciones de software de base de datos (Las bases de datos pueden ser descargadas del repositorio SEEBET en: https://sites.google.com/site/seebetrepository/). Las cuales fueron implementadas por los centros de desarrollo de software de la Universidad de Ciencias Informáticas de las provincias de Las Villas y Ciego de Ávila en Cuba[7].

El cálculo del esfuerzo basado en ordenes de trabajo de proyectos de software está orientado mayormente a su empleo en prácticas de programación ágil. Donde, es imprescindible tener una medida de la productividad de los integrantes del equipo. Esta es una métrica que permite computar el esfuerzo en un producto de software independientemente de la tecnología y la secuencia empleada durante la construcción del mismo.

Tomando elementos del Modelo Constructivo de Costes (COCOMO) (Barry Boehm et al., 2000), se seleccionaron algunas variables como parte de la estimación del esfuerzo de desarrollo de software y se agregaron otras a partir de la experiencia de los equipos de desarrollo. Éstas, describen características de las tareas de ingeniería en cuanto a: tipo de tarea y personal integrante del proyecto (Tabla 7).

Tabla 7 Descripción de las variables empleadas para la estimación del esfuerzo basada en tareas de ingeniería.

N	Definición Variable	Nemotécnico	Tipo	Valores
1	Determina si se requieren gran cantidad de decisiones lógicas, complicados procedimientos matemáticos o difícil manejo de excepciones. Puede tomar los valores: ***muy simple, simple, promedio, compleja, muy compleja.***	complejidad	Ordinal	1-5

[7] Universidad de Ciencias Informáticas de Cuba, URL: http//:www.uci.cu

N	Definición Variable	Nemotécnico	Tipo	Valores
2	Porcentaje de código existente que puede ser utilizado en el desarrollo de la tarea.	conReuso	Numérico	0-100%
3	Experiencia con la que cuenta el programador en el desarrollo de aplicaciones y tareas similares a la que tiene que desarrollar. Puede tomar los valores: ***ninguna***, ***muy poca, poca, promedio, alta, muy alta***.	conocimientoAplicacion	Ordinal	0-5
4	Experiencia con la que cuenta el programador en el uso del lenguaje empleado para el desarrollo de la tarea. Puede tomar los valores: ***ninguna***, ***muy poca, poca, promedio, alta, muy alta***.	conocimientoLenguaje	Ordinal	0-5
5	Grado de precisión esperada, con que una tarea realice su función. Puede tomar los valores: ***muy poca, poca, promedio, alta, muy alta***.	fiabilidad	Ordinal	1-5
6	Empleo de herramientas de software que faciliten y reduzcan el tiempo de desarrollo de la tarea específica. Puede tomar los valores: ***ninguna***, ***muy poca, poca, adecuada, alta, todas las posibles***.	herramientasSoftware	Ordinal	0-5
7	Número de horas empleadas en el desarrollo de la tarea.	tiempoR	Numérico	≥ 1

4.2. Evaluación de los modelos propuestos ETTpred y EEpred

ETTpred se basa en un árbol que devuelve modelos de regresión lineal, mientras que, EEpred es un multiclasificador en cascada, donde cada nivel tiene diferentes responsabilidades, en el primer nivel se realiza clasificación y en la segunda regresión, es por ello que se hace necesario contar con diferentes estadígrafos, tanto discretos como

continuos, que permitan medir adecuadamente el desempeño en cada una de estas etapas.

4.2.1. Estadígrafos

Es importante tener en cuenta que muchas de las medidas comúnmente empleadas para evaluar la efectividad de los predictores no funcionan en predicción numérica, los principios básicos (empleando validación cruzada para la evaluación del desempeño), son igualmente aplicables a la predicción numérica. Pero la calidad de la medición ofrecida por la razón de error no es muy apropiada: los errores no están, simplemente, presentes o ausentes; ellos se presentan en diferentes magnitudes (Witten y Frank, 2005).

Es por ello que para la evaluación del desempeño del predictor en el primer nivel fueron empleadas medidas discretas. Éstas fueron: precisión (1), recuerdo o sensibilidad (2) y la media armónica (3).

$$P = \frac{TP}{TP + FP} \quad (1)$$

$$R = \frac{TP}{TP + FN} \quad (2)$$

$$Fm = 2 * \frac{P * R}{P + R} \quad (3)$$

donde ***TP*** (verdaderos positivos), proyectos bien clasificados, los ***FN*** (falsos negativos), proyectos que se clasificaron como incorrectos y no lo son y ***FP*** (falsos positivos), proyectos clasificados como correctos cuando en realidad no lo son.

Otra medida empleada para evaluar el desempeño del predictor en la primera cascada, es el área bajo la curva de operación (ROC). La cual permitirá conocer el desempeño del predictor para cada clase (García, Fernández, Luengo, y Herrera, 2009).

Error cuadrático medio (*Mean-squared error*): medida más comúnmente usada; en ocasiones tiende a brindar las mismas dimensiones que los valores predichos (4). Tiende a exagerar los valores de los outliers.

$$MSE = \frac{(p_1 - a_1)^2 + ... + (p_n - a_n)^2}{n} \quad (4)$$

Error relativo absoluto (Relative absolute error): es el error absoluto total con el mismo tipo de normalización (5). En estas tres medidas de error relativo, los errores son normalizados mediante el error del predictor simple, el cual predice los valores promedios.

$$MRE = \sum_i \left| \frac{p_i - r_i}{r_i} \right| \quad (5)$$

Coeficiente de Correlación (Correlation coefficient): mide la correlación estadística entre los a's y los p's (6). El coeficiente de correlación se expresa en valores en el rango de 1 para los resultados perfectamente correlacionados, 0 para la no correlación a -1 para los perfectamente correlacionados negativamente.

$$CC = \frac{S_{PA}}{\sqrt{S_P S_A}}, \quad donde \quad S_{PA} = \frac{\sum_i (p_i - \bar{p})(a_i - \bar{a})}{n-1}$$
$$S_p = \frac{\sum_i (p_i - \bar{p})^2}{n-1} \quad y \quad S_A = \frac{\sum_i (a_i - \bar{a})^2}{n-1} \quad (6)$$

* donde p son las predicciones y a son los valores actuales

En ocasiones el error relativo es más importante que el absoluto. Esto significa que si el 10% del error es igualmente importante tanto si es un error de 50 en una predicción de 500 o un error de 0.2 en una predicción de 2, entonces el error absoluto sería absurdo y el error relativo sería más apropiado. Este efecto debe tenerse en cuenta para usar los errores relativos en el cálculo del MSE o del MAE.

La correlación negativa no debe ocurrir en métodos de predicción razonables. La correlación es significativamente diferente de otras medidas debido a que es independiente de la escala. Si se tiene un conjunto de predicción específico, el error es inalterable si toda la predicción es multiplicada por un factor constante y los valores actuales se mantienen invariables. A diferencia del resto de las medidas de error donde los valores pequeños indican buen desempeño, el CC debe mostrar valores elevados (Witten y Frank, 2005).

4.2.2. Pruebas estadísticas

Para realizar las comparaciones entre los resultados de los diferentes algoritmos y determinar cuál es la mejor estrategia que funciona con la propuesta, resulta aconsejable la consideración de algún test estadístico (Demsar, 2006; Salvador García y Herrera, 2008). La selección de estos test está en dependencia de la naturaleza de los datos con que se cuenta. Existen contrastes que no necesitan establecer supuestos exigentes sobre las poblaciones de donde se extraen las muestras y ni que los datos sean obtenidos con una escala de medida de intervalo o razón, a los que se les llaman pruebas no paramétricas.

Teniendo en cuenta que en este caso no es posible determinar el cumplimiento de los supuestos de normalidad y homogeneidad de varianzas, serán empleadas entonces las pruebas no paramétricas (Figura 14). Primeramente, se aplica la prueba de Friedman con el objetivo de ver si existen diferencias significativas entre los algoritmos. Luego, si existen diferencias, se aplican las pruebas de *post-hoc* Bonferroni–Dunn, Holm, Hochberg y Hommel, como sugiere (Salvador García, Fernández, Luengo, y Herrera, 2010).

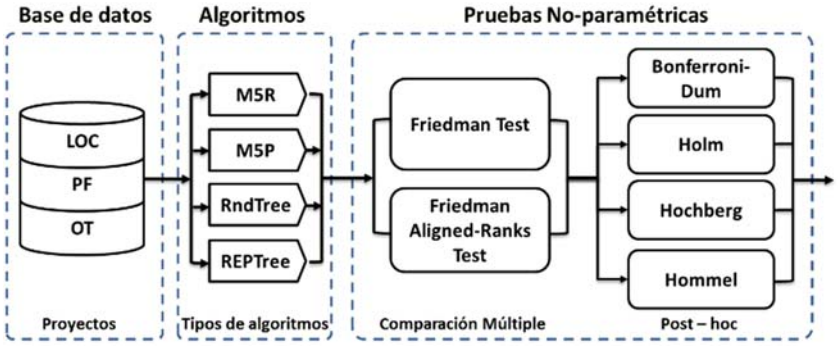


Figura 14 Algoritmo para la para la evaluación de los resultados.

4.2.2.1. Test de Friedman

Esta prueba puede considerarse como una extensión de la prueba de Wilcoxon (Davis y Goadrich, 2006; Demsar, 2006; Dietterich, 1997; Salvador García y Herrera, 2008; Sheskin, 2004), para el caso de más de dos muestras. En el caso de que las asunciones de la prueba ANOVA fuesen satisfechas, el análisis se realizaría de acuerdo a un diseño de ANOVA de dos factores sin repetición, en el que los factores serían respectivamente el conjunto de entrenamiento (bloques) y los algoritmos. Para el procesamiento estadístico se exigió una razón de confianza de 0,95.

4.2.2.2. Test de Bonferroni-Dunn

En el test de Bonferroni-Dunn se utiliza la t de Student convencional pero con unos niveles de confianza más exigentes en función del número de contrastes que se van a hacer (Salvador García et al., 2010).

Para este test se utiliza la probabilidad (p) que expresa el nivel de confianza dividida por el número de comparaciones previstas, así si el nivel de confianza es de 0,05 y se prevén realizar tres comparaciones, se utilizará como nivel de confianza 0,05/3 = 0,0167; en este caso 0,0167 equivale a un nivel de confianza de 0,05. También, si se conoce la probabilidad exacta (p), esta puede ser multiplicada por el número de contrastes para ver si llega a 0,05.

El problema de este contraste es que es muy conservador. Esto se traduce como que tiene poca potencia para rechazar la hipótesis nula cuando realmente es falsa, o sea, que

da muchos falsos negativos. Por tanto, la interpretación de un resultado depende de que el análisis se haga en solitario o junto con otros análisis.

4.2.2.3. Test de Holm, Hochberg y Hommel

El test de Holm se emplea para comparaciones de múltiples clasificadores. Éste ajusta el valor de α empleando un método descendente. Sean ***p1, …, pm*** valores ordenados de probabilidad (de mayor a menor) y ***H1, …, Hm*** las hipótesis correspondientes. El procedimiento de Holm rechaza ***H1*** para ***H(i-1)*** si ***i*** es el menor entero tal que ***pi > α/(m−i+1)*** (Salvador García et al., 2010).

Otras alternativas fueron desarrolladas por **Hochberg** y **Hommel**. Estos test son fáciles de realizar, la diferencia es que suelen ser más fuertes que **Holm** pero no es muy notable, por lo que los tres suelen tener resultados similares.

Los resultados de las pruebas de *post-hoc* serán mostrados en un diagrama de diferencias críticas. Estos diagramas representan el rango ordenado de la distancia media de significación estadística de cada método en una escala lineal. Las líneas gruesas indican los grupos de clasificadores que no presentan, entre ellos, diferencias significativas en su comportamiento.

4.3. Validación experimental de los resultados

4.3.1. Análisis del efecto de las variables sobre la estimación del esfuerzo

El primer paso que se propone realizar es un estudio, a partir de la aplicación de técnicas de minería de datos y aprendizaje automático, que permita establecer cuál es la influencia de cada una de las variables utilizadas en el proceso de estimación del esfuerzo de desarrollo de software.

4.3.1.1. Diseño experimental

La estimación del esfuerzo de un proyecto de software se considera compleja, debido a la gran variabilidad de los datos que intervienen en el proceso. Es por ello que,

para el desarrollo de la investigación, se siguió el diseño experimental que se muestra en la Figura 15. Y que cuenta con los siguientes pasos:

1) Análisis de la capacidad discriminante de las diferentes variables sobre la estimación del esfuerzo.

Comparar los perfiles de las variables en cada uno de los proyectos.

2) En caso de que los perfiles no sean discriminantes, determinar la influencia de cada variable sobre estimación del esfuerzo.

a) Emplear herramientas de teoría matemática de la información:

- Ganancia de la información.
- Razón de ganancia.

b) Emplear herramientas de aprendizaje automático para la selección de variables.

3) Realizar un análisis sistemático de la influencia de las variables en la estimación del esfuerzo:

a) Algoritmos interpretables:

- Basados en árboles.

4) Determinar la mejor estrategia de estimación del esfuerzo de desarrollo de software empleando herramientas de aprendizaje automático:

a) Algoritmos de aprendizaje más comunes:

- Redes neuronales.
- Máquinas de vectores soporte.
- Combinación de clasificadores.

b) Algoritmos interpretables:

- Basados en árboles.

5) Aplicar pruebas de significación estadística que permitan evaluar dichas estrategias.

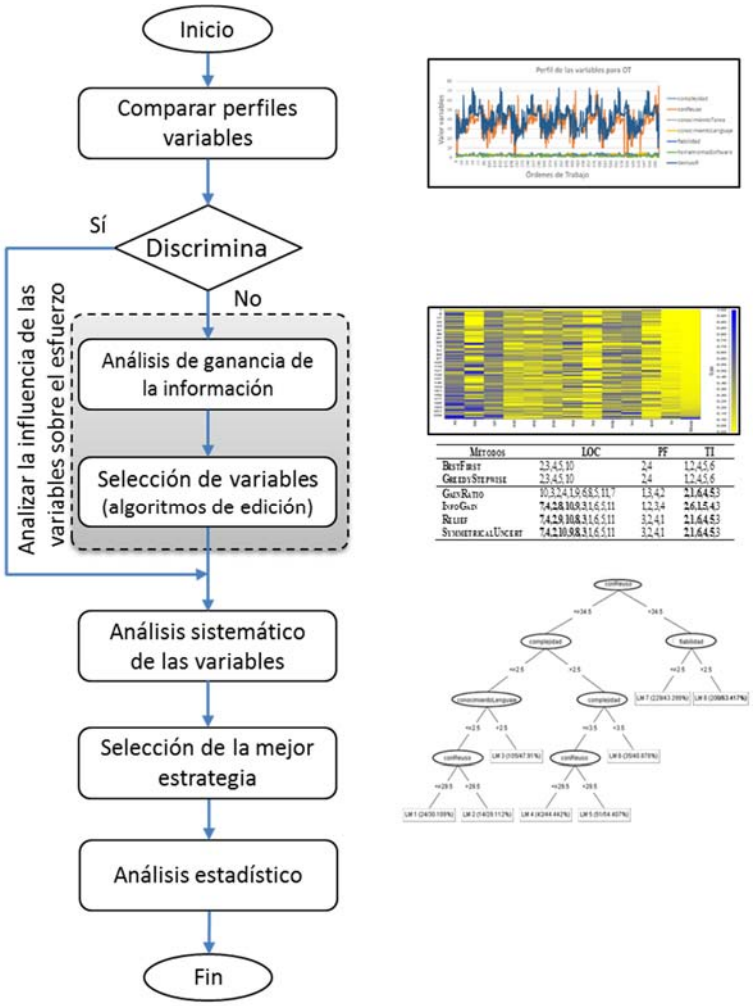


Figura 15 Diseño experimental. Describe la secuencia de pasos a seguir para el desarrollo de la investigación.

4.3.1.2. Análisis de perfiles y correlación

Para este análisis se utilizó el perfil de las variables, donde el eje de las abscisas está formado por los proyectos y el eje de las ordenadas por el valor que toma cada una de las variables. Los proyectos están ordenados en función del esfuerzo. Este proceso se realizó para los tres grupos de variables analizadas.

LOC: Cada una de las series constituye el perfil de las variables para los 228 proyectos y se realiza su análisis a partir de un diagrama de temperatura (Figura 16.). Para realizar el estudio, la matriz se creó normalizando todos los valores de las variables y se ordenó ascendentemente, en función del esfuerzo. Como se puede apreciar, es muy difícil establecer una clara correlación entre el comportamiento de las series (valores de las variables en cada proyecto) y el valor del esfuerzo de desarrollo de dichos proyectos.

Al analizar la Figura 16., se aprecia que una variable que se comporta de manera similar al esfuerzo es ***Rely***. Donde, a medida que aumenta Rely, aumenta el esfuerzo, pero con diferencias marcadas en las magnitudes de sus valores. Sin embargo, se logra identificar una correlación más directa entre el número de líneas de código generadas durante la elaboración del proyecto ***Loc*** y el esfuerzo necesario para su desarrollo.

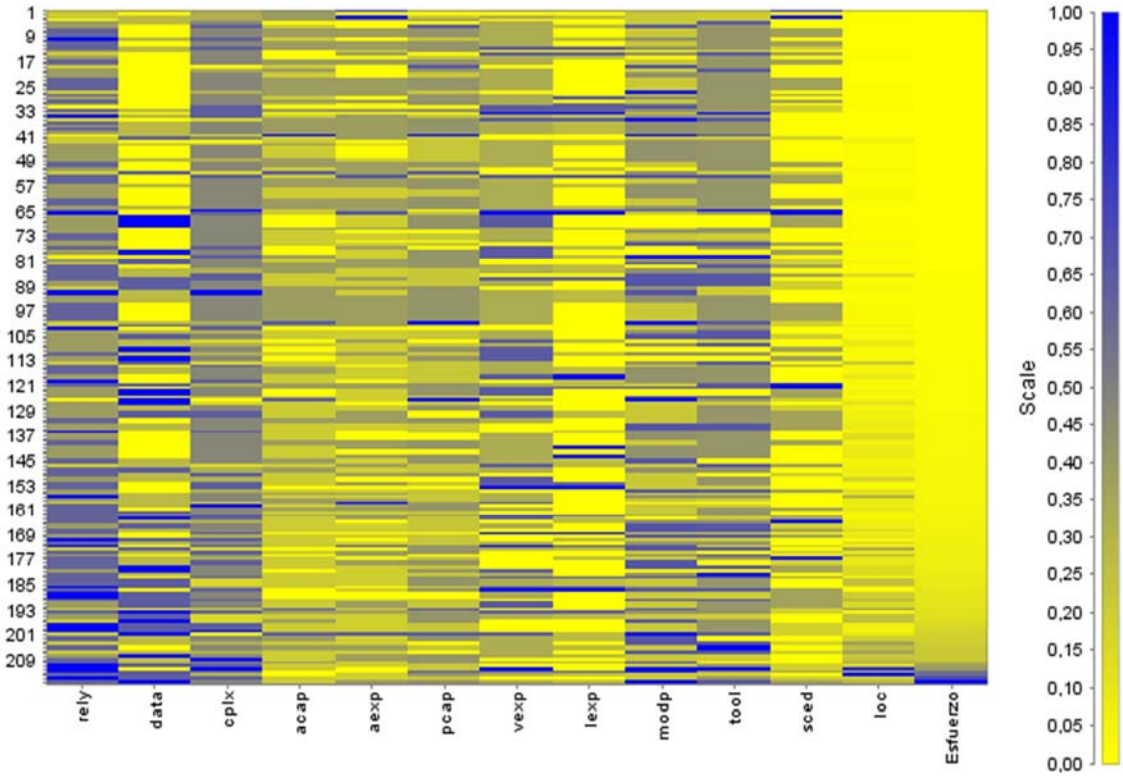


Figura 16 Diagrama de temperatura que representa el perfil de las variables para LOC. Para realizar el estudio, la matriz se normalizó y ordenó ascendentemente, en función del esfuerzo.

Lo que evidencia que, en este caso, el análisis del perfil de las variables no es determinante para la estimación del esfuerzo de desarrollo de software. Pues, éste parece depender exclusivamente de ***Loc*** y, en menor medida, de ***Rely***. Por tal motivo, se hace necesario realizar un estudio más profundo de la relación de las variables empleadas con el esfuerzo del proyecto.

PF: Igualmente, cada serie constituye el perfil de las variables para los 523 proyectos (Figura 17), esta figura muestra que el comportamiento de las variables para cada proyecto es similar. Presentándose las diferencias únicamente en la magnitud del valor de las mismas. De lo que se puede concluir que el análisis del perfil tampoco es determinante para la estimación del esfuerzo de desarrollo de software.

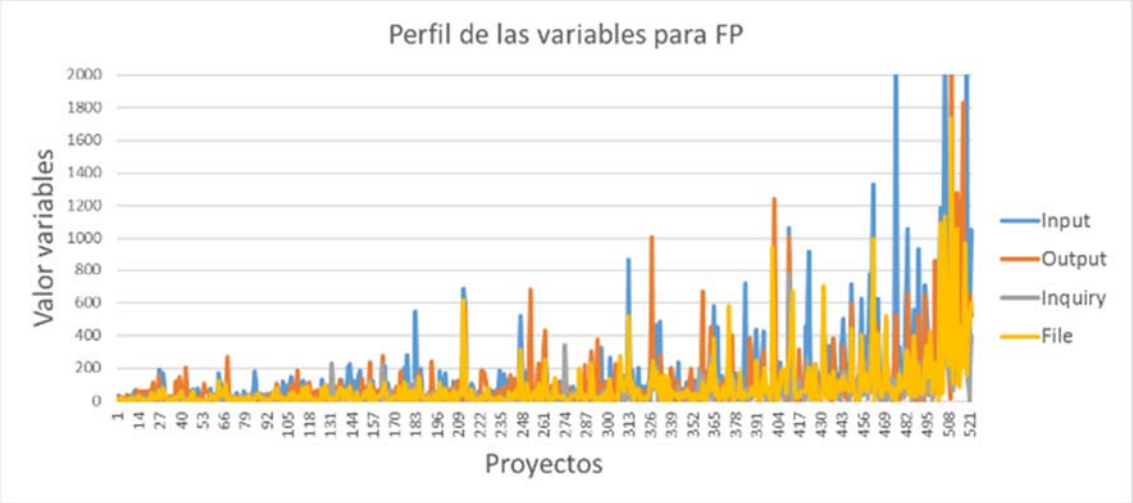


Figura 17 Perfil de las variables para FP. Muestra la alta correlación entre las variables.

TI: En este caso, el perfil de las variables se construye con el valor de 700 tareas de ingeniería (Figura 18.). Aquí se hace mucho más evidente la relación que existe entre ellas, por lo que tampoco el perfil resulta adecuado para la estimación del esfuerzo de desarrollo de software.

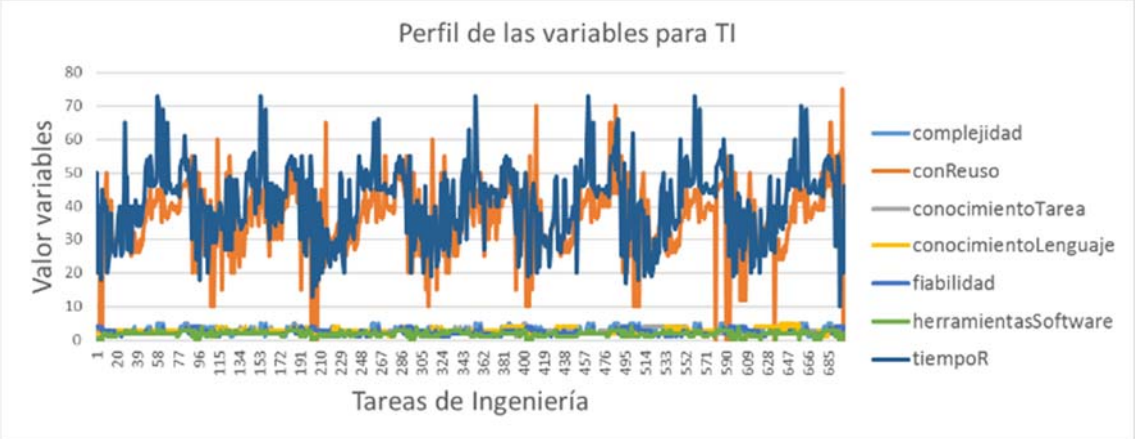


Figura 18 Perfil de las variables para TI. Muestra la alta correlación entre las variables

4.3.1.3. Influencia de las variables sobre el esfuerzo

Teniendo en cuenta que se está realizando un análisis exploratorio de datos, es necesario hallar las causas de la variabilidad del conjunto de datos y ordenar las variables por importancia para poder construir modelos predictivos. Es por ello que se realiza el análisis en función de la ganancia de la información y la razón de ganancia. Adicionalmente, fueron empleadas las técnicas de aprendizaje automático para la selección de variable: BestFirst y GreedyStepwise (Tabla 8.).

Tabla 8 Ranking de atributos para estimación del esfuerzo de desarrollo de software. Se resaltan las coincidencias en el orden de los atributos para cada base de datos.

No	Métodos	LOC	PF	TI
1	**BestFirst**	2,3,4,5,10	2,4	1,2,4,5,6
	GreedyStepwise	2,3,4,5,10	2,4	1,2,4,5,6
2	**GainRatio**	10,3,2,4,1,9,6,8,5,11,7	1,3,4,2	**2,1,6,4,5,3**
	InfoGain	**7,4,2,8,10,9,3,1,6,5,11**	1,2,3,4	**2,6,**1**,5,**4**,3**
	Relief	**7,4,2,9,10,8,3,1,6,5,11**	**3,2,4,1**	**2,1,6,4,5,3**
	SymmetricalUncert	**7,4,2,10,9,8,3,1,6,5,11**	**3,2,4,1**	**2,1,6,4,5,3**

La Tabla 8., muestra el orden de importancia de las variables en el proceso de estimación del esfuerzo, establecido como resultado del empleo de los métodos de evaluación de atributos. Como puede apreciarse, para cada base de datos, aparecen coincidencias en los rankings establecidos por estos algoritmos.

En el caso de la base de datos basada en LOC, los algoritmos de tipo goloso sugieren que las variables más influyentes son: (2) *Data*, (3) *Cplx*, (4) *Acap*, (5) *Aexp* y (10) *Tool*; mientras que los algoritmos basados en información de ganancia sugieren que el orden de importancia de estos atributos es: (7) *Vexp*, (4) *Acap*, (2) *Data*, (9) *Modp*, (10) *Tool*, (8) *Lexp*, (3) *Cplx*, (1) *Rely*, (6) *Pcap*, (5) *Aexp*, (11) *Turn*.

Por otra parte, para la base de datos basada en PF, los algoritmos golosos identifican como principales atributos a (2) *Output* y (4) *File*. Y, los algoritmos basados en la teoría de la información sugieren que el orden de importancia de las variables es: (3) *Enquiry*, (2) *Output*, (4) *File* y (1) *Input*.

Para la base de datos basada en TI, los algoritmos golosos sugieren: (1) *complejidad*, (2) *conReuso*, (4) *conocimiento-Lenguaje*, (5) *fiabilidad*, (6) *herramientasSoftware*. Por su parte, el orden sugerido por los algoritmos basados en teoría de la información, establecen el siguiente ranking: (2) *conReuso*, (1) *complejidad*, (6) *herramientasSoftware*, (4) *conocimiento-Lenguaje*, (5) *fiabilidad* y (3) *conocimientoTarea*.

En sentido general, los algoritmos de evaluación de variables basados en estrategias golosas, seleccionan menos variables, pero no difieren apreciablemente de los rankings establecidos por los algoritmos basados en la teoría de la información. Por tal motivo, se asumió el orden sugerido por los algoritmos basado en ganancia.

4.3.1.4. Análisis sistemático del orden de las variables

Poder determinar cuál es el orden de importancia de las variables a partir de la relación que existe entre éstas y el esfuerzo real, es una tarea importante. Esto se debe a que permitiría a los directores de proyecto tomar decisiones más adecuadamente. Sin

embargo, el análisis de la influencia de las variables sobre el esfuerzo, se realizó tomando en cuenta la influencia de las mismas sobre todo el conjunto de datos. Pero, esta no es la forma en que trabajan los algoritmos de aprendizaje automático. Debido a que cada vez que se realiza una partición de los datos basada en la información que aporta una de las variables, la influencia de las mismas sobre los subconjuntos resultantes varía considerablemente.

Esto justifica que se realice un estudio sistemático, a partir del análisis del orden de las variables establecido por varios algoritmos basados en reglas. Para ello, se tomaron en cuenta las reglas generadas por los árboles que mejor correlacionaron en la estimación del esfuerzo y que menor porcentaje de error relativo obtuvieron.

Para este estudio, fueron seleccionados los algoritmos basados en árboles de regresión: M5Rules (Ramakrishnan, 2009), M5P (Steinberg, 2009), RandomTree (Zhao y Zhang, 2007) y REPTree (Zhao y Zhang, 2007). Cada uno de los algoritmos son entrenados y evaluados en las bases de datos LOC, PF y TI y se evalúa el orden de las variables que éstos establecen. A partir de aquí, se puede sugerir cuál es el orden de importancia de dichas variables en la estimación del esfuerzo.

LOC: El resultado de la aplicación de estos algoritmos, muestran varios órdenes de importancia de las variables. Donde resaltan:

- *Rely*, (5) *Aexp*, (10) *Tool*, (7) *Vexp*, (8) *Lexp*, (4) *Acap*, (2) *Data* y (9) *Modp*.
- (9) *Modp,* (1) *Rely*, (8) *Lexp*, (2) *Data* y (10) *Tool*.
- (9) *Modp,* (10) *Tool,* (1) *Rely*, (2) *Data* y (8) *Lexp*.

De este análisis se puede inferir la importancia que juegan variables como ***Modp***, ***Rely***, ***Lexp***, ***Data*** y ***Tool***, con independencia del orden que ocupen en el proceso de la estimación. Por otra parte, estos algoritmos, en su mayoría, discriminan las variables: ***Cplx***, ***Pcap*** y ***Turn***.

PF: Cuando se analizan los resultados obtenidos de este estudio, se aprecia una ligera diferencia con los resultados propuestos por el análisis basado en teoría de la información. Donde, los árboles sugieren que el orden de importancia de las variables es: (2) *Output*, (3) *Enquiry*, (4) *File* y (1) *Input*. Intercambiándose las variables (2) *Output* y (3) *Enquiry.* Estos resultados son, relativamente, similares, sobre todo si se toma en cuenta que ambas variables logran el mayor protagonismo en la estimación del esfuerzo de desarrollo de software para cada uno de los algoritmos empleados en el estudio.

TI: En análisis de este estudio se realizó tomando como referencia, las reglas extraídas de los árboles más representativos: A partir de los cuales se puede identificar que el orden de importancia de las variables es: (2) *conReuso*, (1) *complejidad*, (3) *conocimientoLenguaje* y (5) *fiabilidad*. Si se toma en cuenta el orden establecido por los algoritmos de evaluación de variables se obtiene: (2) *conReuso*, (1) *complejidad*, (6) *herramientasSofware*, (3) *conocimientoLenguaje* y (5) *fiabilidad*. Criterios que son prácticamente coincidentes en ambos casos. Es por ello que, para la estimación del esfuerzo de desarrollo de software basado en órdenes de trabajo, se puede establecer que las variables relacionadas serían: ***conReuso***, ***complejidad***, ***conocimientoLenguaje*** y ***fiabilidad***.

4.3.1.5. Selección de la mejor estrategia

El último paso de este estudio es determinar cuál es la mejor estrategia para la estimación del esfuerzo de desarrollo de software. Para ello, es necesario poder determinar cuál es la variante (LOC, PF o TI) y cuál es el algoritmo más adecuado para realizar la estimación. Por tal motivo, fueron empleados algunos de los principales algoritmos de aprendizaje automático (Witten et al., n.d.), los cuales fueron evaluados en cada una de estas bases de datos en cuanto a su correlación y error relativo (Tabla 9.).

Tabla 9 Resultado de aplicar los algoritmos más populares en las bases de datos para estimación del esfuerzo de desarrollo de software. Donde R representa el ranking de los algoritmos/conjuntos de variables, CC es el coeficiente de correlación y MRE el error.

ALGORITMOS	LOC			PF			TI			R
	R	CC	MRE	R	CC	MRE	R	CC	MRE	
MLP	4	0,34	100,49	7	0,50	100,72	8	0,73	70,98	**7**
SVM	8	0,21	103,53	1	0,68	66,64	9	0,61	74,33	**6**
BAGGING	5	0,31	87,59	4	0,61	74,40	2	0,87	43,19	**2**
RANDOMCOMMITEE	2	0,39	70,73	6	0,50	86,77	3	0,86	43,20	**3**
M5RULES	7	0,31	104,06	3	0,62	73,00	7	0,83	50,32	**5**
M5P	6	0,31	94,52	2	0,64	71,81	4	0,84	48,62	**4**
RANDOMFOREST	1	0,42	75,49	5	0,61	75,85	1	0,89	40,95	**1**
RANDOMTREE	3	0,36	71,71	9	0,40	105,37	6	0,83	47,63	**6**
REPTREE	9	0,08	102,30	8	0,46	80,61	5	0,84	49,04	**9**
INFLUENCIA DE LAS VARIABLES	**3**	**0,30**	**90,05**	**2**	**0,56**	**81,69**	**1**	**0,81**	**52,03**	**////**

La Tabla 9. muestra que, los algoritmos logran mejor correlación y un error más bajo en la estimación del esfuerzo cuando se emplean tareas de ingeniería. Como promedio, los algoritmos logran estimar el esfuerzo de desarrollo de software con un 81% de correlación y un error relativo del 52% para esta base de datos. Mientras que si se emplea puntos de función solo se alcanza una correlación del 56% y el error relativo aumento al 81%, y al emplear el número de líneas de código, la efectividad de la estimación cae a solo un 30% de correlación y el error relativo llega a alcanzar el 90%.

Sin embargo, con el objetivo de determinar si existen diferencias estadísticamente significativas en el efecto de los diferentes conjuntos de datos en el desempeño de los algoritmos, fueron empleados múltiples pruebas estadísticas no paramétricas (Anexo A). Mediante la prueba de Friedman se comprueba la hipótesis nula de que las bases de datos permiten estimar de forma similar el esfuerzo. Hipótesis que fue rechazada, con un p-value de 0,0003. Por lo que fueron aplicadas las pruebas de post-hoc: Bonferroni-Dunn, Holm, Hochberg y Hommel, los cuales demostraron que existen diferencias estadísticamente significativas entre los algoritmos con un p-value ajustado de inferior a 0,1 para un nivel de significación igualmente de 0,1. Las cuales demostraron que estimar

el esfuerzo en tareas de ingeniería resulta más adecuado que la estimación empleando puntos de función o basado en líneas de código (Figura 19.)

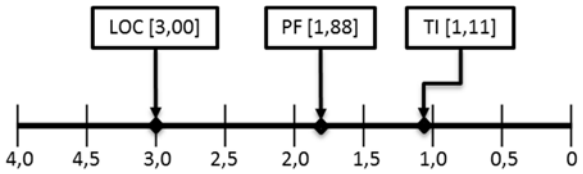


Figura 19 Representación visual de ordenamiento promedio de Friedman con un α = 0.10.

4.4. Resultados experimentales para el modelo EEpred

En este epígrafe se presentan los resultados de la experimentación realizada. Además, se realiza el análisis y discusión del comportamiento del error y del mecanismo de interpretación.

4.4.1. Evaluación de la capacidad de generalización del predictor

En este tópico se realiza el análisis y discusión de la validación interna del modelo EEpred. Donde, primeramente, expone el proceso de selección del clasificador base del modelo. Y, posteriormente, muestran los resultados del análisis de la robustez y capacidad de generalización del modelo.

4.4.1.1. Selección del clasificador base

Con el objetivo de seleccionar el clasificador base, en la primera cascada, fueron evaluados los algoritmos de construcción de árboles J48, RandomTree, y REPTree, en la base de datos DS1 (Tabla 10).

Tabla 10 Comparativa entre árboles de decisión. Donde P es la precisión, R el recuerdo, Fm es la media armónica, ROC la curva de operación y RMSE la raíz del error cuadrático medio.

Clasificadores	Reglas	Clasificaciones		Estadísticas				
		Buenas	Malas	RMSE	P	R	Fm	ROC
J48	20	156	72	100	0.64	0.68	0.65	0.65
RandomTree	**82**	**164**	**64**	**98,87**	**0.71**	**0.72**	**0.72**	**0.75**
REPTree	17	153	75	97,94	0.62	0.67	0.64	0.67

La Tabla 10 muestra que el comportamiento de estos algoritmos es muy similar. Sin embargo, tomando en cuenta las buenas y malas clasificaciones, así como la magnitud del RMSE, las medidas discretas, así como el área bajo la curva ROC, se determinó que RandomTree presenta mejor desempeño que J48, y REPTree.

Por otra parte, una de las características más importantes de los árboles es la capacidad de convertir su base de conocimientos en reglas entendibles. Por tanto, un indicador importante es el número de reglas que generan. Sin embargo, 82 no es un número de reglas que no pueda ser manejado.

De esta manera, tomando en cuenta que RandomTree es el mejor situado en cuanto a su desempeño y a que genera un número de reglas fácilmente manejable, se tomó la decisión de emplearlo como clasificador base de la primera cascada.

En el segundo nivel la estimación se trata como un problema de regresión. Por lo que se realizó el análisis con los árboles de regresión RandomTree, REPTree y M5P. En la Tabla 11 se muestran los resultados experimentales.

Tabla 11 Comparativa entre árboles de regresión. Donde R es el número de reglas, CC el coeficiente de correlación y RMSE la raíz del error cuadrático medio.

Clasificadores	Orgánico			Semi-libre			Empotrado		
	R	**CC**	**RMSE**	**R**	**CC**	**RMSE**	**R**	**CC**	**RMSE**
RamdonTree	53	0,34	99,90	58	0,36	93,72	53	0,34	99,90
REPTree	14	0,30	95,07	10	0,23	95,77	1	0,40	87,10
M5P	**1**	**0,48**	**93,21**	**1**	**0,46**	**87,57**	**1**	**0,49**	**87,86**

Aunque en todos los casos, M5P generó una regla única, mantuvo un coeficiente de correlación superior al del resto de los algoritmos y, de igual forma, el error fue menor. Por tal motivo, en la segunda cascada, responsable de estimar el esfuerzo del proyecto, se decidió el empleo del algoritmo M5P.

4.4.1.2. Análisis de la robustez y capacidad de generalización EEpred

Para analizar la robustez y la capacidad de generalización del algoritmo, EEpred fue sometido a un proceso de validación interna. Donde, se aplicó la validación cruzada para 10 muestras (*10-fold cross-validation*). Este procedimiento se realizó tomando en cuenta el tipo de software a predecir (Tabla 12.), por lo que se evaluó la capacidad de estimación de EEpred para proyectos que califican como orgánicos, semi-libres, empotrados y en general.

Tabla 12 Resultados experimentales del proceso de validación interna. Donde P es la precisión, R el recuerdo, Fm es la media armónica y ROC la curva de operación.

Proyectos	EEpred			
	P	R	Fm	ROC
Orgánico	0,78	0,82	0,80	0,76
Semi-libre	0,63	0,59	0,61	0,77
Empotrado	0,36	0,31	0,33	0,61
General	0.71	0.72	0.72	0.75

Como se puede observar en la Tabla 12., EEpred se comporta aparentemente de forma similar para los proyectos orgánicos y semi-libre, no siendo así para los empotrados cuya precisión cae drásticamente. Sin embargo, un análisis más detallado se puede realizar si se observa la Figura 20.

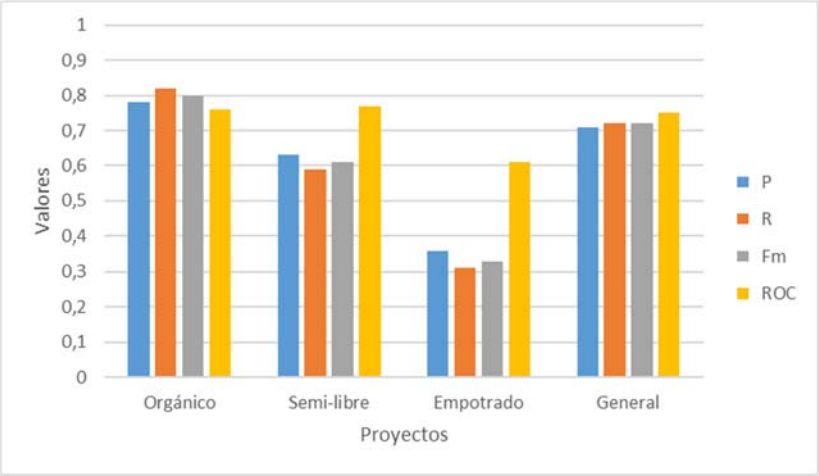


Figura 20 Diseño experimental de la evaluación del modelo propuesto.

La Figura 20., muestra que EEpred presenta mejor precisión (P) a la hora de identificar los proyectos orgánicos. Lo mismo sucede con la capacidad de recuerdo (R), área bajo la curva (ROC) y la media armónica (Fm). No obstante, el desempeño de EEpred para los proyectos de tipo semi-libre es superior al 60%. Sin embargo, un resultado interesante se muestra cuando EEpred es evaluado en una base de datos que contenga todo tipo de proyectos. Esto se debe a que, a pesar de su bajo desempeño en proyectos de tipo empotrado, éstos solo representan el 21% del total. Lo cual hace que el desempeño en general del algoritmo sea aceptable. Logrando una precisión de hasta un 78%.

4.4.2.- Comportamiento del error

Con el objetivo de estudiar el comportamiento del error del modelo propuesto, se construyó un diagrama de dispersión (Figura 21.), donde fueron graficados el valor del esfuerzo estimado en relación con el valor del esfuerzo real para cada proyecto

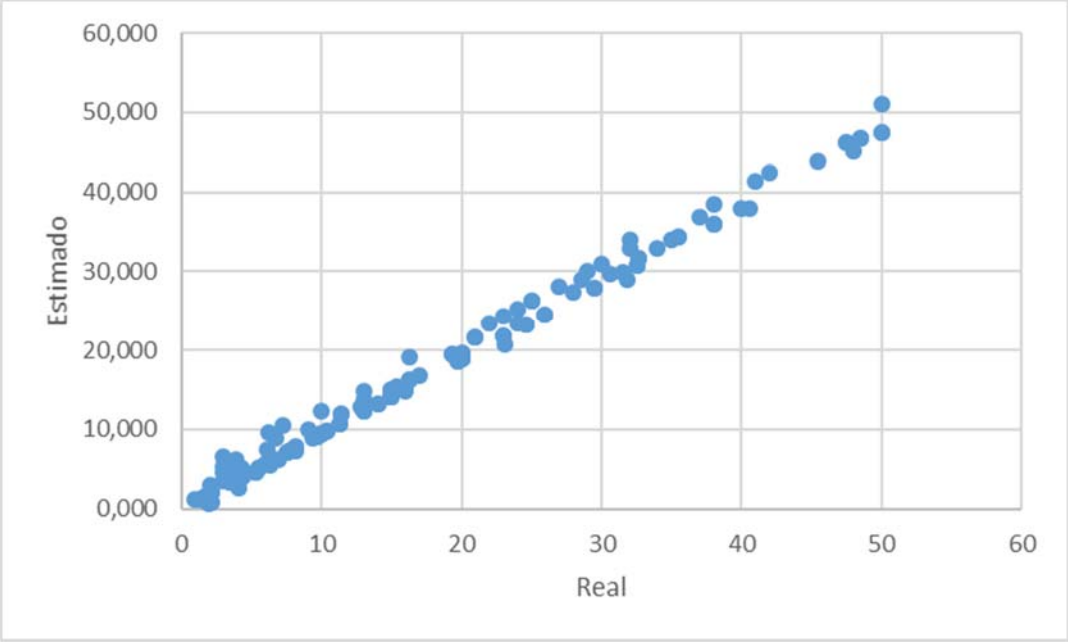


Figura 21 Diagrama de dispersión que muestra la calidad de la estimación del esfuerzo para proyectos orgánicos. Por el eje de las X se encuentra el esfuerzo real y por el eje de las Y el esfuerzo estimado.

La Figura 21., muestra que la estimación en proyectos orgánicos, se comporta alrededor de la pendiente ideal, este es un comportamiento deseable, sin embargo, se observa cómo se incrementa ligeramente la dispersión en los proyectos que requieren de un esfuerzo mayor. De igual manera, se evidencia que a medida que aumenta el esfuerzo requerido por el proyecto, EEpred tiende a sobreestimar el valor real.

Sin embargo, algo totalmente distinto sucede para los proyectos de tipo empotrado, donde se puede apreciar el alto nivel de error de EEpred (Figura 22.).

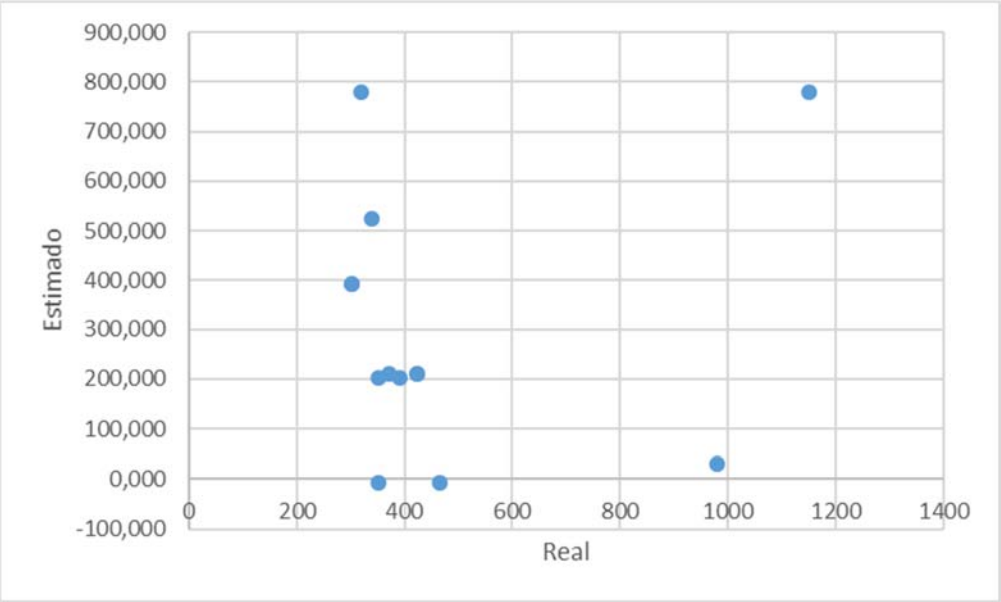


Figura 22 Diagrama de dispersión que muestra la calidad de la estimación del esfuerzo para proyectos empotrados. Por el eje de las X se encuentra el esfuerzo real y por el eje de las Y el esfuerzo estimado.

Como se puede observar en la Figura 22., en la mayoría de los casos, EEpred subestima el valor del esfuerzo necesario para los proyectos empotrados. Lo cual se debe al bajo número de muestras que tiene el predictor para entrenarse.

De este análisis se puede concluir que el modelo propuesto es capaz de estimar el esfuerzo con calidad en proyectos orgánicos y semi-libre, mientras que, para proyectos de tipo empotrado, el error aumenta significativamente.

4.4.3. Mecanismo de interpretación EEpred

Muchos de los modelos existentes para la estimación del esfuerzo de proyectos de software son cajas negras. Sin embargo, los árboles son capaces de describir explícitamente la naturaleza de su predicción.

Debido a que el árbol generado en el primer nivel tiene 82 reglas, a modo de ejemplo, se tomarán de muestras solo dos ramas del mismo:

vexp = n
| data = l
| | modp = n
| | | lexp = h
| | | | aexp = h : o (4/0)
| | | | aexp = vh : o (12/4)

De estas ramas del árbol se pueden extraer dos reglas de decisión, las que pueden ser transformadas en reglas fácilmente interpretables en solo tres pasos muy simples:

- **Conversión**: en este paso, cada rama del árbol se convierte en una proposición conjuntiva.

  a) ***(Vexp = n) ^ (Data = l) ^ (Modp = n) ^ (Lexp = h) ^ (Aexp = h) → O (4/0)***

  b) ***(Vexp = n) ^ (Data = l) ^ (Modp = n) ^ (Lexp = h) ^ (Aexp = vh) → O (12/0)***

- **Refinamiento**: en este paso, se buscan patrones coincidentes y se simplifica la proposición. Por ejemplo, ***(Aexp = h) → O ^ (Aexp = vh) → O*** por tanto, la proposición quedaría:

  (Vexp = n) ^ (Data = l) ^ (Modp = n) ^ (Lexp = h) ^ (Aexp = h, vh) → O (16/0)

- **Traducción**: en este paso, la proposición se traduce empleando el significado de las variables y se convierte en una regla de tipo ***Si–Entonces***.

"Si la experiencia en la máquina virtual es nominal **Y** la base de datos es pequeña **Y** la práctica de programación es nominal **Y** la experiencia del analista en este tipo de aplicación es alta o muy alta, ***Entonces*** el tipo de proyecto es **orgánico** y esfuerzo se estima por la ecuación (5)**"**.

$$\begin{aligned} Esfuerzo = {} & 117.0168 * rely = n, xh, h, vh \\ & + 176.7444 * cplx = vh, xh \\ & + 79.29 * vexp = xh, h, l, vl \\ & + 118.5762 * sced = vh, xh, h \\ & - 120.4674 \end{aligned} \tag{5}$$

Adicionalmente, cada una de estas reglas está acompañada por el valor de la cobertura y la precisión. Lo cual permite al especialista que la interpreta conocer su grado de certeza.

4.5. Resultados experimentales para el modelo ETTpred

Para analizar el comportamiento de ETTpred, primero se realizó la evaluación de las variables que intervinieron en el problema y la selección del clasificador base. Después, fue evaluada la robustez, capacidad de generalización y comportamiento del error de la propuesta. Por último, se realizó el análisis del mecanismo de interpretación generado por ETTpred.

4.5.1. Evaluación de las variables ETTpred

En este tópico se realiza el análisis y discusión de la validación interna del modelo ETTpred. Donde, primeramente, expone el proceso de selección del clasificador base del modelo. Y, posteriormente, muestran los resultados del análisis de la robustez y capacidad de generalización del modelo.

Tomando en consideración que la estimación del esfuerzo de desarrollo de software es un problema no resuelto (Nguyen-Cong y De Tran-Cao, 2013; Popli y Chauhan, 2014). El primer procedimiento que se realizó fue el análisis evaluativo de la influencia de cada variable en el problema. Para ello, se decidió emplear el algoritmo de evaluación de

atributos ***cfssubseteval***, el cual calcula la correlación de la clase con cada variable, seleccionando solo las variables con baja correlación entre ellas o poco redundantes. Por otra parte, los algoritmos basados en clasificadores o *wrapers*, que fueran capaces de trabajar con clases continuas. Ya que éstos, aunque muy costosos computacionalmente, devuelve el conjunto de variables que optimizan el comportamiento del algoritmo para el cual se ejecuta. (Witten et al., n.d.) (Tabla 13.).

Tabla 13 Resultados de la evaluación de las variables.

Métodos		Variables	Mérito
CfsSubsetEval	**BestFirst**	1, 2, 4, 5, 6	0,521
Wrapper	**M5Rules**	1, 2, 3, 5, 6	6,083
	M5P	1, 2, 3, 4, 5, 6	5,866
	RandomTree	1, 2, 5, 6	6,680
	REPTree	1, 2, 5, 6	6,473

La Tabla 13 muestra que, excepto para el M5P para el cual todas las variables resultan de interés, solo las variables 1, 2, 5, 6, resultan de mayor interés para estimación del esfuerzo de desarrollo de software. No obstante, la selección de variables varía en dependencia del algoritmo empleado. Por otra parte, el mérito o grado de fiabilidad de la selección de variables resulta similar para todos los algoritmos evaluados en el *wraper*. Teniendo en cuenta esto, se hace necesario determinar la mejor combinación variable – algoritmo – estimación del esfuerzo. Por lo que fueron creados nuevos vectores de entradas a los clasificadores, empleando las variables seleccionadas por cada uno de los algoritmos anteriores.

4.5.2. Relación variable – algoritmo – estimación del esfuerzo

Con el objetivo de seleccionar el clasificador con los mejores resultados en la estimación, fueron creados seis vectores de entrada. Un vector que emplea todas las variables del conjunto de datos original y un vector que emplea las variables propuestas por cada uno de los cinco algoritmos de evaluación de atributos descritos en la Tabla 3.

En la Tabla 14., se muestra el número de reglas generadas por cada algoritmo (R), su coeficiente de correlación (CC) y error relativo (MRE).

Tabla 14 Comparativa entre árboles de decisión y vectores de entrada.

Algoritmos	V(Completo)			V(BestFirst)			V(M5R)			V(M5P)			V(RndTreee)			V(REPTree)		
	R	CC	MRE	R	CC	MRE	R	CC	MRE	R	CC	MRE	R	CC	MRE	R	CC	MRE
M5Rules	6	0,84	49,07	5	0,84	48,23	7	0,84	49,45	6	0,84	49,07	7	0,83	50,85	7	0,83	50,85
M5P	12	0,85	47,24	7	0,85	46,57	11	0,85	48,47	12	0,85	47,24	8	0,84	49,59	8	0,84	49,59
RandomTree	1363	0,78	54,05	983	0,80	49,94	1030	0,81	51,43	1363	0,78	54,05	675	0,81	48,62	675	0,81	48,62
REPTree	153	0,84	47,50	182	0,84	47,99	152	0,83	49,07	153	0,84	47,50	143	0,83	49,34	143	0,83	49,34
Promedio	-	0,83	49,47	-	0,83	48,18	-	0,83	49,61	-	0,83	49,47	-	0,83	49,60	-	0,83	49,60

Como resultado se observa que, el predictor que mejor estima el esfuerzo de desarrollo de software es el M5P con una correlación del 85% y un error relativo de 47,24%. Sin embargo, en sentido general, todos los algoritmos se comportan de forma similar. Por lo que fue necesario realizar un estudio de significación estadística (Anexo 2). Mediante la prueba de Friedman se comprueba si existen diferencias estadísticamente significativas en la estimación del esfuerzo de desarrollo de software entre los algoritmos. La Figura 23. muestra un diagrama de distancia crítica con los resultados obtenidos después de aplicar la prueba de Friedman. La hipótesis nula fue rechazada con un *p-value* = $7{,}9 \times 10^{-8}$, para un $\alpha = 0{,}05$. Adicionalmente, se estableció en orden en el desempeño de los algoritmos.

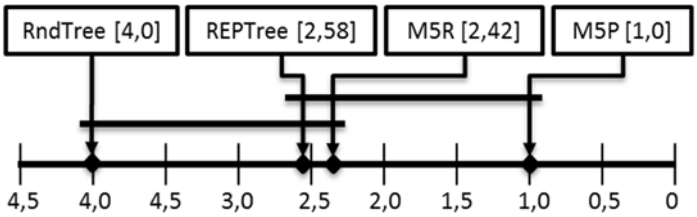


Figura 23 Diagrama de diferencias críticas para un $\alpha = 0{,}05$.

Una vez rechazada la hipótesis nula, fueron aplicadas las pruebas de *post-hoc* con $\alpha = 0{,}05$ y $\alpha = 0{,}10$, estas pruebas demostraron que solo existen diferencias, estadísticamente significativas, entre M5P y RandomTree. El ranking de Friedman ubica

primero al M5P, por lo que se decidió emplearlo en el modelo. Por otra parte, M5P obtiene resultados similares con independencia del vector de entrada que se emplee, por lo que se decidió por utilizar el vector que incluye todas las variables.

4.5.3.- Análisis de la robustez y capacidad de generalización ETTpred

Para analizar la robustez y la capacidad de generalización del algoritmo, ETTpred fue sometido a un proceso de validación interna. Donde, se aplicó la validación cruzada estratificada para 10 particiones (10-fold *cross-validation*). Este procedimiento se realizó tomando en cuenta el desempeño del algoritmo ante las bases de datos para la estimación el esfuerzo de desarrollo de software basado en líneas de código, puntos de función y tareas de ingeniería (Figura 24.).

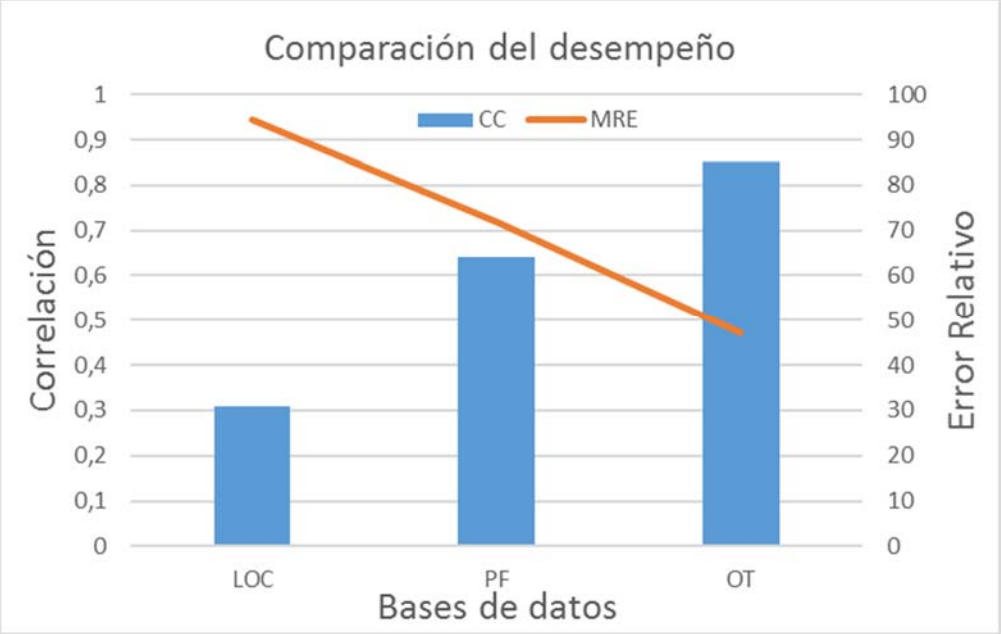


Figura 24 Gráfico de comparación del desempeño del algoritmo M5P en las bases de datos.

El eje de las abscisas está formado por las bases de datos LOC, PF, ET. El eje de las ordenadas principal (izquierda), representa la correlación de la estimación. El eje de las ordenadas secundario (derecha), representa la magnitud del error relativo de en la estimación.

Como se puede observar en el estudio comparativo, el desempeño del algoritmo M5P ante las diferentes bases de datos, es muy superior cuando se estima el esfuerzo de desarrollo de software empleando Tareas de ingeniería (ET) que basado en conteo de Líneas de Código (LOC) o Puntos de Función (PF). Por otra parte, el error que comete el algoritmo en la estimación es muy inferior cuando se trata de ET.

Comportamiento del error

Con el objetivo de estudiar el comportamiento del error del modelo propuesto, se construyó un diagrama de dispersión (Figura 25.). Donde se representa el valor del esfuerzo estimado en relación con el valor del esfuerzo real para cada tarea de ingeniería (orden de trabajo).

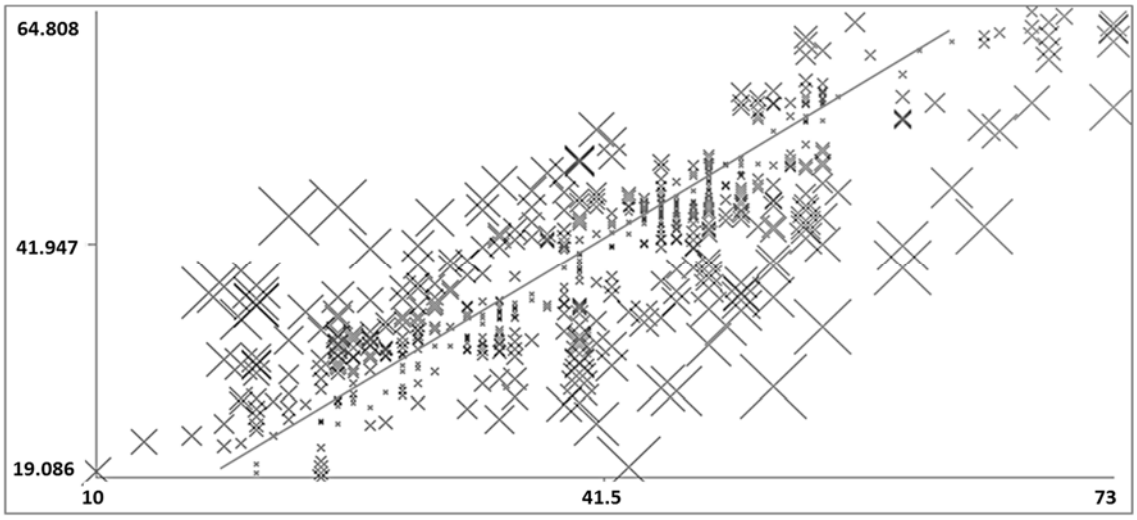
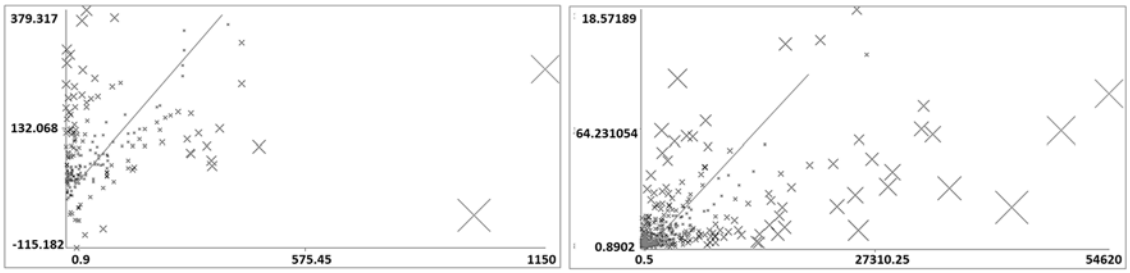


Figura 25 Diagrama de dispersión que muestra la calidad de la estimación del esfuerzo para tareas de ingeniería. Por el eje de las abscisas se encuentra el esfuerzo real y por el eje de las ordenadas, el esfuerzo estimado.

La Figura 25. muestra que la estimación en tareas de ingeniería se concentra alrededor de la pendiente ideal, este es un comportamiento deseable, no obstante, se observa el nivel de dispersión que genera el error en la estimación del esfuerzo de desarrollo de software. Sin embargo, algo totalmente distinto sucede cuando la estimación está basada en conteo de líneas de código o puntos de función (Figura 26.).



(a) (b)

Figura 26 Diagrama de dispersión que muestra la calidad de la estimación del esfuerzo. (a) Estimación basada en conteo de líneas de código. (b) Estimación basada en puntos de función. Por el eje de las abscisas, se encuentra el esfuerzo real y por el eje de las ordenadas, el esfuerzo estimado.

Como se puede observar en la Figura 26.a, el error cometido en la estimación basada en conteo de líneas de código muestra una gran dispersión. Cuando se trata de proyectos pequeños, el algoritmo sobrestima el valor del esfuerzo, sin embargo, en el caso de la estimación basada en puntos de función (Figura 26.b), se estiman valores muy bajos cuando se trata de proyectos grandes.

De este análisis se puede concluir que el modelo propuesto es capaz de estimar el esfuerzo de desarrollo de software, con mayor calidad cuando se trata de tareas de ingeniería. Siguiendo una distribución del error alrededor de la normal, con un valor máximo de 47,24%.

4.5.4.- Mecanismo de interpretación ETTpred

Muchos de los modelos existentes para la estimación del esfuerzo de proyectos de software son modelos de caja negra (Nguyen-Cong y De Tran-Cao, 2013; Dejaeger et al., 2012b; Saroha y Sahu, 2015). Sin embargo, los árboles son capaces de describir explícitamente la naturaleza de su predicción. ETTpred, en la base de datos de tareas de ingeniería empleada, es capaz de generar un árbol de regresión de 12 niveles de profundidad (Figura 27).

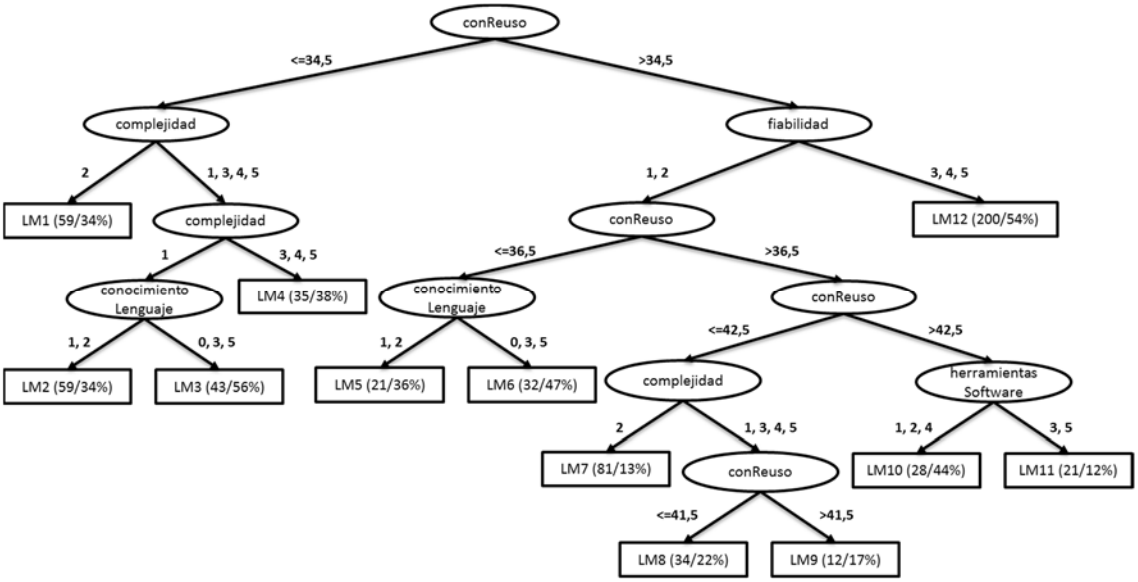


Figura 27 Árbol generado por el algoritmo M5P al ser entrenado en la base de datos ET.

De estas ramas del árbol se pueden extraer 12 reglas, derivando cada una en un modelo de regresión lineal. Las que pueden ser transformadas en reglas semánticas, interpretables en tres pasos:

1. **Conversión**: en este paso, cada rama del árbol se convierte en una proposición conjuntiva.

   (conReuso <= 34,5) ^ (complejidad = 1, 3, 4, 5) ^ (complejidad = 1) ^ (cocimientoLenguaje = 1, 2) → LM2 (59 / 34%)

2. **Refinamiento**: en este paso, se buscan reglas coincidentes o auto-contenidas y se simplifica la proposición. Por ejemplo:

   (complejidad = 1, 3, 4, 5) ^ (complejidad = 1)

   como la primera restricción contiene a la segunda, la proposición quedaría:

   (conReuso <= 34,5) ^ (complejidad = 1) ^ (cocimientoLenguaje = 1, 2) → LM2 (59 / 34%)

3. **Traducción**: en este paso, la proposición se traduce empleando el significado de las variables y se convierte en una regla de tipo ***Si–Entonces***.

   "Si el porcentaje de código existente que puede ser utilizado en la tarea es menor o igual a *34,5%* **Y** la complejidad de la tarea es simple **Y** el conocimiento del lenguaje es muy poco o poco, ***Entonces*** el esfuerzo se estima empleando la ecuación 2**"**.

tiempoR = 0.538 * complejidad=3,1,4,5 - 0.3714 * complejidad=1,4,5 + 1.3731 * complejidad=4,5 + 0.6132 * complejidad=5 + 0.0323 * conReuso + 3.793 * conocimientoTarea=0,4,3,2,5 - 0.183 * conocimientoTarea=3,2,5 - 2.5848 * conocimientoTarea=2,5 + 5.9747 * conocimientoLenguaje=4,0,3,5 + 0.2387 * conocimientoLenguaje=0,3,5 + 2.2402 * fiabilidad=2,4,5,3 - 1.5494 * fiabilidad=4,5,3 - 0.5522 * fiabilidad=5,3 + 3.0186 * herramientasSoftware=4,0,2,5,3 - 0.3603 * herramientasSoftware=2,5,3 + 0.2222 * herramientasSoftware=5,3 + 18.5301 (2)

Adicionalmente, cada una de estas reglas está acompañada por el valor de la cobertura y la precisión. Lo cual permite al especialista que la interpreta conocer su grado de certeza.

4.6. Sumario

En este capítulo fueron presentadas las bases de datos a emplear, así como las variables escogidas para la experimentación, de igual forma, se formalizó el tipo de experimentación que se realizará, así como, las pruebas de significación estadísticas a emplear y el método de interpretación de las mismas.

El método de estimación del esfuerzo de desarrollo de proyectos de software EEpred, fue sometido a un proceso de validación interna que permitió analizar su robustez y capacidad de generalización, así como el error en la predicción. Lo cual

demostró que EEpred es capaz de predecir el esfuerzo de proyectos ***orgánicos*** *y* ***semi-libre***, con una precisión de hasta el 78%.

El método de estimación del esfuerzo de desarrollo de proyectos de software ETTpred, fue sometido a un proceso de validación interna que permitió analizar su robustez y capacidad de generalización, así como el error en la predicción. Lo cual demostró que el método propuesto es capaz de predecir el esfuerzo con una precisión de hasta el 85% y un error relativo de 47,24%. De igual forma, se demostró la superioridad de la estimación del esfuerzo de desarrollo de software basado en tareas de ingeniería sobre las basadas en conteo de líneas de código y puntos de función.

La principal ventaja tanto de ETTpred como de EEpred, ante otros modelos, es su mecanismo de interpretación de los resultados sobre la base de un conjunto de reglas semánticas que son de fácil entendimiento. Lo cual los convierte en modelos entendibles para los expertos en el área de aplicación.

Capítulo 5
Conclusiones y trabajos futuros

5.1. Conclusiones

Producto de la investigación bibliográfica, se encontró la existencia de un amplio conjunto de repositorios de proyectos con características diferentes, donde no existe un consenso en las variables que se almacenan y se hace difícil la selección del mejor modelo de estimación, ya que las pruebas no son realizadas sobre los mismos datos.

En el desarrollo de la memoria, se encontraron una serie de aseveraciones que considero importantes comentar:

- Las pequeñas y medianas empresas de desarrollo de software, no cuentan con una herramienta estándar en la organización, la cual les permita hacer estimaciones de un proyecto de desarrollo de software y en cada área usan un método distinto para hacer el cálculo de las estimaciones.
- En la revisión de la literatura se encontraron muchos métodos de estimación, pero estos casi no se utilizan, y existen pocas herramientas que implementan estas metodologías. Los resultados de las estimaciones, independientemente del método usado, no van a ser exactos, siempre va a existir un porcentaje de incertidumbre.
- Se debe usar información histórica para obtener estimaciones más confiables y repetibles, debemos aprender de los errores del pasado para que estos no se vuelvan a cometer. Una buena estimación ahorra pérdidas económicas para la empresa.

En la presente investigación se propuso un nuevo método para la estimación del esfuerzo de desarrollo de software, especialmente diseñado para metodologías ágiles basada en tareas de ingeniería. Para ello, se introdujo una nueva base y se sugirieron nuevas métricas de evaluación del desempeño del equipo de trabajo y de evaluación del proyecto.

Además, se propusieron dos nuevos algoritmos de estimación del esfuerzo de desarrollo de proyectos de software: EEpred, basado en la combinación de clasificadores en serie; y ETTpred, basado en tareas de ingeniería y el árbol de regresión M5P, y diseñado especialmente para la estimación en el desarrollo ágil.

El algoritmo EEpred es un multiclasificador que funciona en serie y se basa en el conteo de líneas de código fuente. Primero realizando una clasificación granular, basada en un árbol de decisión, donde determina si el proyecto será ***orgánico***, ***semi-libre*** o ***empotrado***. Posteriormente, realiza una predicción fina, basada en árboles de regresión, devolviendo el valor del esfuerzo estimado. Éste, fue sometido a un proceso de validación interna que permitió analizar su robustez y capacidad de generalización, así como el error en la predicción. Lo cual demostró que EEpred es capaz de predecir el esfuerzo de proyectos ***orgánicos*** *y* ***semi-libre***, con una precisión de hasta el 78%.

El algoritmo ETTpred, el cual fue sometido a un proceso de validación interna que permitió analizar su robustez y capacidad de generalización, así como el error en la predicción. Lo cual demostró que el método propuesto es capaz de predecir el esfuerzo con una precisión de hasta el 85% y un error relativo de 47,24%. De igual forma, se demostró la superioridad de la estimación del esfuerzo de desarrollo de software basado en tareas de ingeniería sobre las basadas en conteo de líneas de código y puntos de función. Este método de estimación basada en tareas de ingeniería, está orientada a su empleo en metodologías ágiles.

La principal ventaja de EEpred y ETTpred, ante el resto de los modelos estudiados, es que ambos presentan un mecanismo de interpretación de los resultados sobre la base

de un conjunto de reglas semánticas que son de fácil entendimiento. Lo cual los convierte en modelos entendibles para los expertos en el área de aplicación.

5.2. Trabajos futuros

Luego de culminar el desarrollo de esta memoria, se pueden plantear las siguientes líneas de investigación futuras:

- Teniendo en cuenta que generalmente las técnicas de estimación basadas en modelos matemáticos son más simples de ajustar, pero no son más efectivas que las basadas en aprendizaje automático, se impone combinarlas en busca de un modelo híbrido adaptable y altamente configurable en cuanto sus restricciones iniciales.

- Integrar el conocimiento del experto en la generación de modelos predictivos mediante el empleo de lógica difusa para realizar interpretaciones lingüísticas de las entradas y respuestas obtenidas.

- Caracterización de nuevos atributos críticos y métricas especializadas para los requerimientos no funcionales, que podrían incidir notablemente en el nivel de precisión para las estimaciones realizadas por la propuesta presentada.

- Implementar una herramienta computacional de apoyo al modelo de estimación del esfuerzo de desarrollo, con interfaces amigables que permitan al experto extraer el conocimiento necesario para analizar e interpretar el razonamiento que sigue el modelo con la finalidad de optimizar el tiempo de desarrollo y el diseño de las características de software de sus tareas de ingeniería.

- En la administración de métodos ágiles, el tamaño del proyecto se mide en el esfuerzo total que se necesita para su desarrollo. Conforme a lo anterior, las medidas más comunes en métodos agiles son el esfuerzo y el tamaño, sin embargo, esto da pie a diversas pérdidas económicas por falta de medidas de costos, por lo que se plantea complementar el modelo presentado a través de la propuesta de un método capaz de estimar costos basados en evidencias históricas. Controlar y monitorizar los costos

periódicamente, logrará minimizar la pérdida de costos y ajustar el programa a los tiempos y compromisos con el cliente.

- Para poder realizar predicciones sobre características de un proyecto actual es necesario disponer de información adquirida de proyectos pasados sobre las variables a estimar y los factores que las afectan, estos datos permiten establecer un modelo de relaciones entre dichas variables que constituye la base del proceso de estimación. En la búsqueda de información realizada para esta investigación, no se encontró en repositorios de libre acceso, ningún *dataset* que contenga información de variables obtenidas a partir de proyectos desarrollados con metodologías ágiles; lo que dificulta probar y comparar los diferentes modelos de estimación. Por lo expuesto, se plantea crear un *plug-in* que se inserte en los programas de Gestión de Proyectos, con la finalidad de recabar y gestionar información de las variables utilizadas en las tareas de ingeniería, ampliando el número de instancias del *dataset* utilizado para el proceso de aprendizaje del modelo propuesto y de esta manera facilitar su uso para los investigadores de esta línea.

- A través de la ejecución de un Caso de Estudio, comprobar en la práctica cuan eficientes son los resultados del uso del modelo propuesto. Si al optimizar el diseño y el tiempo de desarrollo de las tareas de ingeniería, se logran cronogramas más acertados que permitan controlar mejor el proyecto, plazos de entrega y presupuestos más realistas, mayor satisfacción de los usuarios, estándares de calidad internacional que permitan mejorar la industria de desarrollo de software, logrando la competitividad internacional el país.

Referencias Bibliográficas

Abrahamsson, P., y Koskela, J. (2004). Extreme Programming: A Survey of Empirical Data from a Controlled Case Study. In *Proceedings of International Symposium on Empirical Software Engineering* (pp. 73–82).

Albrecht, A. J., y Gaffney, J. E. (1983). Software functions, source lines of codes and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering*, *9*(11), 639–648.

Algabri, M., Saeed, F., Mathkour, H., y Tagoug, N. (2015). Optimization of soft cost estimation using genetic algorithm for NASA software projects. In *2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW)* (pp. 1–4). IEEE. http://doi.org/10.1109/NSITNSW.2015.7176416

Aljahdali, S., y Sheta, A. F. (2010). Software Effort Estimation by Tuning COOCMO Model Parameters Using Differential Evolution. *EEE/ACS International Conference on Computer Systems and Applications (AICCSA)*. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5586985

Almache, M. G., Raura, G., Ruiz, J., y Fonseca, E. (2015). Modelo Neuronal de Estimación para el Esfuerzo de Desarrollo en Proyectos de Software (MONEPS). *Revista Latinoamericana de Ingeniería de Software*, *3*(3), 148–154.

Alsmadi, I. M., y Nuser, M. S. (2013). Evaluation of Cost Estimation Metrics: Towards a Uni fi ed Terminology. *Journal of Computing Anf Information Technology*, *21*(1), 23–34. http://doi.org/10.2498/cit.1002133

Axelrad, V., Granik, Y., Boksha, V. V., y Rollins, J. G. (1994). Cost and yield estimation in a virtual IC factory. In *Microelectronic Manufacturing* (pp. 102–108).

Balbín, D., Ocrospoma, M., Soto, E., y Pow-sang, J. A. (2009). TUPUX : An Estimation Tool for Incremental Software Development Projects, 39–43. http://doi.org/10.1109/AST.2009.25

Barcel, M. (n.d.). Estimación de costes de un proyecto informático Métricas de productividad y modelos.

Basha, S., y Ponnurangam, D. (2010). Analysis of Empirical Software Effort Estimation Models. *International Journal of Computer Science and Information Security*, *7*(3), 68–77. Retrieved from http://arxiv.org/abs/1004.1239

Basten, D., y Sunyaev, A. (2011a). Guidelines for Effort Estimation. *Computer*, 88–90.

Basten, D., y Sunyaev, A. (2011b). Guidelines for Software Development Effort Estimation. *Computer*, *44*(10), 88–90. http://doi.org/10.1109/MC.2011.315

Beck, K. (1999). Extreme programming. *Proceedings of the Conference on Technology of Object-Oriented Languages and Systems, TOOLS*, 411.

Bedini, A. (2005). *Gestión de Proyectos de Software* (2da ed.). Valparaiso, Chile, Chile: Universidad Técnica Federico Santa María. Retrieved from http://www.inf.utfsm.cl/~guerra/publicaciones/Gestion de Proyectos de Software.pdf

Boehm, B., Abts, C., y Chulani, S. (2000). Software development cost estimation approaches - A survey. *Annals of Software Engineering*, *10*, 177–205.

Boehm, B., Clark, B., Devnani-Chulani, S., Horowitz, E., Madachy, R., Reifer, D., … Engineering, S. (2000). COCOMO II Model Definition Manual. *University of Southern California*, *4*(1), 6–6. http://doi.org/10.1525/nr.2000.4.1.6

Boehm, B. W., y Valerdi, R. (2008). Achievements and challenges in cocomo-based software resource estimation. *Software, IEEE*, *25*(5), 74–83.

Borade, J. G., y Khalkar, V. R. (2013). Software Project Effort and Cost Estimation Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(8), 730–739. Retrieved from www.ijarcsse.com

Britto, R., Mendes, E., y Borstler, J. (2015). An Empirical Investigation on Effort Estimation in Agile Global Software Development. *2015 IEEE 10th International Conference on Global Software Engineering*, 38–45. http://doi.org/10.1109/ICGSE.2015.10

Britto, R., Usman, M., y Mendes, E. (2014). Effort Estimation in Agile Global Software Development Context. *XP 2014 Workshops, LNBIP 199*, 182–192. http://doi.org/10.1109/ICGSE.2015.10

Büyüközkan, G., y Ruan, D. (2008). Evaluation of software development projects using a fuzzy multi-criteria decision approach. *Mathematics and Computers in Simulation*, *77*(5–6), 464–475. http://doi.org/10.1016/j.matcom.2007.11.015

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., y Wirth, R. (2000). Crisp-Dm 1.0. *CRISP-DM Consortium*, 76. http://doi.org/10.1109/ICETET.2008.239

Chiu, N. H., y Huang, S. J. (2007). The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software*, *80*(4), 628–640.

Coad, P., Lefebvre, E., De Luca, J., y Luca, J. de. (1999). Java Modeling In Color With UML. In *Java Modeling In Color With UML: Enterprise Components and Process* (pp. 1–12).

Cockburn, A. (2004). Crystal Clear. A Human-Powered Methodology For Small Teams, including The Seven Properties of Effective Software Projects. *Integration The Vlsi Journal*, 39. http://doi.org/10.1111/j.1600-0560.2011.01732.x

Cordero Carrasco, R. J. (2013). *Una herramienta de apoyo a la estimación del esfuerzo de desarrollo de software en proyectos pequeños*.

Cristian A. Remón, P. T. (2010). Análisis de Estimación de Esfuerzo aplicando Puntos de Caso de Uso, 577–586.

Danh Nguyen-Cong, y De Tran-Cao. (2013). A review of effort estimation studies in agile, iterative and incremental software development. In *The 2013 RIVF International Conference on Computing y Communication Technologies - Research, Innovation, and Vision for Future (RIVF)* (pp. 27–30). IEEE. http://doi.org/10.1109/RIVF.2013.6719861

Davis, J., y Goadrich, M. (2006). The Relationship Between Precision-Recall and ROC Curves. In *23 rd International Con- ference on Machine Learning, Pittsburgh, PA, 2006.* (pp. 2–8).

Dejaeger, K., Verbeke, W., Martens, D., y Baesens, B. (2012). Data Mining Techniques for Software Effort Estimation: A Comparative Study. *IEEE Transactions on Software Engineering*, *38*(2), 375–397. http://doi.org/10.1109/TSE.2011.55

Demsar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, *7*, 1–30.

Dick, S., Meeks, A., Last, M., Bunke, H., y Kandel, A. (2004). Data mining in software metrics databases. *Fuzzy Sets and Systems*, *145*(1), 81–110. http://doi.org/10.1016/j.fss.2003.10.006

Dietterich, T. G. (1997). Statistical Tests for Comparing Supervised Classication Learning Algorithms 1 Introduction, 1–24.

Dolado, J. (2000a). A Validation of the Component-Based method for software size estimation. *IEEE Transactions on Software Engineering*, *26*(10), 1006–1021. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=879821

Dolado, J. (2000b). *medicion para la gestion en la ingenieria de software*. Ra-ma.

Engel, A., y Last, M. (2007). Modeling software testing costs and risks using fuzzy logic paradigm. *Journal of Systems and Software*, *80*(6), 817–835. http://doi.org/10.1016/j.jss.2006.09.013

Evett, M., Raton, B., Allen, E., Khoshgoftar, T., y Chien, P. (1998). GP-based software quality prediction. *Proceedings of the Third Annual Conference on Genetic Programming*, 60–65.

Fayyad, U., Piatetsky-Shapiro, G., y Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 37–54. http://doi.org/10.1145/240455.240463

Fenton, N. E., y Pfleeger, S. L. (1997). Software Metrics: A Rigorous and Practical Approach. *It Professional*. http://doi.org/10.1201/b17461

Fenton, N., y Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, *25*(5), 675–689.

Ferens, D. V. (1998). The conundrum of software estimation models. In *Aerospace and Electronics Conference, 1998. NAECON 1998.* (pp. 320–328). Proceedings of the IEEE National.

Finnie, G. R., y Wittig, G. E. (1996). AI tools for software development effort estimation. In *Software Engineering and Education and Practice Conference* (pp. 346–353). EEE Computer Society Press.

Frohnhoff, S., y Engels, G. (2008). Revised Use Case Point Method - Effort Estimation in Development Projects for Business Applications. In *Proceedings of the 11th International Conference on Quality Engineering in Software Technology*. Potsdam, dpunkt.verlag.

Galinina, A., Burceva, O., y Parshutin, S. (2012). The Optimization of COCOMO Model Coefficients Using Genetic Algorithms. *Information Technology and Management Science*, *15*(1), 15. http://doi.org/10.2478/v10313-012-0006-7

Ganesan, K., Khoshgoftar, T., Allen, E. (2000). Cased-based Software Quality Prediction. *International Journal of Software Engineering and Knowledge Engineering*, *10*(2), 139–152.

Garcia, A., Gonzalez, I., Colomo-Palacios, R., Lopez, J. L., y Ruiz, B. (2011). Methodology for software development estimation optimization based on neural networks. *Latin America Transactions, IEEE*, *9*(3), 384–398.

García, M. N. M., Quintales, L. A. M., Peñalvo, F. J. G., y Martín, M. J. P. (2004). Building knowledge discovery-driven models for decision support in project management. *Decision Support Systems*, *38*(2), 305–317. http://doi.org/10.1016/S0167-9236(03)00100-3

García, M., Peñalvo, F., y Martín, M. (2004). Mining interesting association rules for prediction in the software project management area. *Data Warehousing and Knowledge*, 341–350. Retrieved from http://www.springerlink.com/index/eak6h2buqtkwpq1p.pdf

García, S., Fernández, A., Luengo, J., y Herrera, F. (2009). A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Comput*, *13*, 959–977. http://doi.org/10.1007/s00500-008-0392-y

García, S., Fernández, A., Luengo, J., y Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, *180*(10), 2044–2064. http://doi.org/10.1016/j.ins.2009.12.010

García, S., y Herrera, F. (2008). An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, *9*, 2677–2694.

Garre, M., Cuadrado, J. J., Sicilia, M. A., Rodríguez, D., y Rejas, R. (2007). Comparación de diferentes algoritmos de clustering en la estimación de coste en el desarrollo de software, *3*(1), 6–22.

Hernández Orallo, J., Ramírez Quintana, M. J., y Ferri Ramírez, C. (2004). *Introducción a la Minería de Datos*. *Introduccion a la mineria de datos*.

Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. *Journal of Evolutionary Biology* (Vol. 12). Retrieved from http://books.google.com/books?id=R1ZyQgAACAAJ

Hirji, K. K. (1999). Discovering Data Mining : From Concept to Implementation. *Acm Sigkdd*, *1*(1), 44–45. http://doi.org/10.1145/846170.846181

Huang, S. J., Chiu, N. H., y Chen, L. W. (2008). Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research*, *188*(3), 898–909.

Huang, X., Ho, D., Ren, J., y Capretz, L. F. (2007). Improving the COCOMO model using a neuro-fuzzy approach. *Applied Soft Computing Journal*, *7*(1), 29–40. http://doi.org/10.1016/j.asoc.2005.06.007

Jodpimai, P., Sophatsathit, P., y Lursinsap, C. (2010). Estimating software effort with minimum features using neural functional approximation. In *International Conference on Computational Science and Its Applications* (pp. 266–273). IEEE.

Jones, C. (1996). How software estimation tools work. *American Programmer*, *9*, 18–27.

Jørgensen, M. (2013). Relative estimation of software development effort: It matters with what and how you compare. *IEEE Software*, *30*(2), 74–79. http://doi.org/http://doi.ieeecomputersociety.org/10.1109/MS.2012.70

Kad, S., y Chopra, V. (2012). Software Development Effort Estimation Using Soft Computing. *International Journal of Machine Learning and Computing*, *2*(5), 548–551. http://doi.org/10.7763/IJMLC.2012.V2.186

Khoshgoftaar, T. M., Geleyn, E., y Nguyen, L. (2003). Empirical case studies of combining software quality classification models. In *Quality Software, 2003. Proceedings. Third International Conference on* (pp. 40–49). http://doi.org/10.1109/QSIC.2003.1319084

Khoshgoftaar, T. M., y Lanning, D. L. (1995). A neural network approach for early detection of program modules having high risk in the maintenance phase. *Journal of Systems and Software*, *29*(1), 85–91. http://doi.org/10.1016/0164-1212(94)00130-F

Kohavi, R. (1996). Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In *Second International Conference on Knoledge Discovery and Data Mining* (pp. 202–207).

Krohn, U., y Boldyreff, C. (1999). Application of cluster algorithms for batching of proposed software changes. *Journal of Software Maintenance: Research and Practice*, *11*(3), 151–165. http://doi.org/10.1002/(SICI)1096-908X(199905/06)11:3<151::AID-SMR189>3.0.CO;2-G

Kumar, K. V., Ravi, V., Carr, M., y Kiran, N. R. (2008). Software development cost estimation using wavelet neural networks. *Journal of Systems and Software*, *81*(11), 1853–1867.

Kuncheva, L. I. (2004). Multiple Classifier Systems. In *Combining Pattern Classifiers. Methods and Algorithms* (1st ed., pp. 101–110). Wiley Interscience.

Layman, L., Williams, L., y Cunningham, L. (2004). Motivations and Measurements in an Agile Case Study. In *ACM SIGSOFT Foundation in Software Engineering Workshop Quantitative Techniques for Software Agile Processes (QTE? SWAP)*. Newport Beach, CA.

Letelier, P., Canós, M., Sánchez, E., y Penadés, M. (2003). Métodologías Ágiles en el Desarrollo de Software. *Valencia, Valencia, España*, 1–8. Retrieved from http://www.carlosfau.com.ar/nqi/nqifiles/XP_Agil.pdf

LEUNG, H., y FAN, Z. (2002). Software Cost Estimation. *Information and Software Technology*, *34*(10), 307–324. http://doi.org/10.1142/9789812389701_0014

Madachy, R. (2007). Distributed global development parametric cost modeling. In *Software Process Dynamics and Agility* (pp. 159–168). Springer Berlin Heidelberg.

Madachy, R., y Boehm, B. (2008). (). Comparative Analysis of COCOMO II, SEER-SEM and True-S Software Cost Models. In *USC-CSSE-2008-816, University of Southern California Center for Systems and Software Engineering*.

Manoj, R. (2012). Communication as an essential factor for effort estimation in GSD. *Trends in Innovative Computing*.

Mendonca, M., y Sunderhaft, N. L. (1999). *Mining Software Engineering Data: A Survey*. *Analysis* (Vol. 4000).

Minku, L. L., y Yao, X. (2013). Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, *55*(8), 1512–1528. http://doi.org/10.1016/j.infsof.2012.09.012

Mizell, C., y Malone, L. (2007). *A project management approach to using simulation for cost estimation on large, complex software development projects.*

Mohanty, S., Bisoi, A. . K. (2012). SOFTWARE EFFORT ESTIMATION APPROACHES – A REVIEW. *International Journal of Internet Computing*, *1*(3), 82–88.

Moreno García, M. N., y García-Peñalvo, F. J. (2005). Improving Estimations in Software Projects with Data Mining Techniques. *Projects y Profits*, *6*, 25–37.

Najadat, H., Alsmadi, I., y Shboul, Y. (2012). Predicting Software Projects Cost Estimation Based on Mining Historical Data. *ISRN Software Engineering*, *2012*, 1–8. http://doi.org/10.5402/2012/823437

Podgurski, A., Masri, W., McCleese, Y., Wolff, F. G., y Yang, C. (1999). Estimation of software reliability by stratified sampling. *ACM Transactions on Software Engineering and Methodology*, *8*(3), 263–283. http://doi.org/10.1145/310663.310667

Popli, R., y Chauhan, N. (2014). Cost and effort estimation in agile software development. In *Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on* (pp. 57–61). http://doi.org/10.1109/ICROIT.2014.6798284

Poppendieck, M., y Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. *Addilson-Wesley*. http://doi.org/10.1109/MC.2003.1220585

Prasad-Reddy, P., Sudha, K.R., Rama, P., y Ramesh, S. (2010). Software Effort Estimation using Radial Basis and Generalized Regression Neural Networks. *Journal of Computing*, *2*(5), 87–92.

Pyle, D., Editor, S., y Cerra, D. D. (1999). *Data Preparation for Data Mining*. *Order A Journal On The Theory Of Ordered Sets And Its Applications* (Vol. 17). http://doi.org/10.1080/713827180

Ramakrishnan, N. (2009). Chapter 1. C4.5. In *The Top Ten Algorithms in Data Mining* (pp. 1–19). Taylor y Francis Group, LLC.

Rubio, M. G. (2006). *Aplicación de técnicas de clustering para la estimación del esfuerzo en la construcción de proyectos software*.

Sahraoui, H. A., y Lounis, H. (1998). Reusability hypothesis verification using machine learning techniques: a case study. In *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No.98EX239)* (pp. 84–93). http://doi.org/10.1109/ASE.1998.732582

Saroha, M., y Sahu, S. (2015). Tools y methods for software effort estimation using use case points model – A review. In *International Conference on Computing, Communication y Automation* (pp. 874–879). IEEE. http://doi.org/10.1109/CCAA.2015.7148498

Sayyad Shirabad, J. ., y Menzies, T. J. (2005). *The PROMISE Repository of Software Engineering Databases*. Ottawa, Canada, Canada. Retrieved from http://promise.site.uottawa.ca/SERepository

Schwaber, K., y Beedle, M. (2001). *Agile Project Management with Scrum*. *Cdswebcernch*. http://doi.org/10.1201/9781420084191-c2

Sheskin, D. J. (2004). *Handbook of PARAMETRIC and NONPARAMETRIC STATISTICAL PROCEDURES* (Third Edit). Western Connecticut State University: Chapman y Hall/CRC.

Sheta, A. (2006). Software effort estimation and stock market prediction using takagi-sugeno fuzzy models. In *2006 IEEE Fuzzy Logic Conference* (pp. 579–586). Vancouver, BC, Canada,: Sheraton, Vancouver Wall Centre.

Sheta, A. F. (2006). Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects. *Journal of Computer Science*, *2*(2), 118–123.

Sheta, A., Rine, D., y A. Ayesh. (2008). Development of software effort and schedule estimation models using soft computing techniques. In *2008 IEEE Congress on Evolutionary Computation (IEEE CEC 2008)* (pp. 1283– 1289). Hong Kong.

Southwell, S. V. (1996). *Calibration of the Softcost-R Software Cost Model to the Space and Missile Systems Center (SMC) Software Database (SWDB)*.

Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: The Method in Practice*. *DSDM Dynamic Systems Development Method The Method in Practice*.

Steinberg, D. (2009). Chapter 10. CART: Classification and Regression Trees. In *The Top Ten Algorithms in Data Mining* (pp. 179–201). Taylor y Francis Group, LLC.

Suri, P. K., y Ranjan, P. (2012). Comparative Analysis of Software Effort Estimation Techniques. *International Journal of Computer Applications*, *48*(21).

Thwin, M. M. T., y Quah, T.-S. (2005). Application of neural networks for software quality prediction using object-oriented metrics. *Journal of Systems and Software*, *76*, 147–156. http://doi.org/10.1016/j.jss.2004.05.001

Tian, J., y Palma, J. (1998). Analyzing and improving reliability: A tree-based approach. *IEEE Software*, *15*(2), 97–104. http://doi.org/10.1109/52.663793

Tuya, J., Román, I. R., y Dolado Cosín, J. J. (2007). *Técnicas cuantitativas para la gestión en la ingeniería del software*. (J. Tuya, I. R. Román, J. J. Dolado Cosín, y M. Martines, Eds.) (1ra ed.). Madrid: Netbiblo S. L.

University of Southern California. (2000). *COCOMO II.2000.0 Software User Manual*. California. Retrieved from http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf

Waghmode, S., y Kolhe, K. (2014). A Novel Way of Cost Estimation in Software Project Development Based on Clustering Techniques. *International Journal of Innovative Research in Computer and Communication Engineering*, *2*(4), 3892–3899.

Wang, Y., y Witten, I. H. (1997). Induction of model trees for predicting continuous classes. *Proceedings of the 9th European Conference on Machine Learning Poster Papers*. Retrieved from http://researchcommons.waikato.ac.nz/handle/10289/1183

Webber, B. G. (1995). *A Calibration of the Revic Software Cost Estimating Model*.

Weiss, S. M., y Indurkhya, N. (1998). *Predictive data mining : a practical guide*. San Francisco: Morgan Kaufmann Publishers.

Wen, J., Li, S., Lin, Z., Hu, Y., y Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, *54*(1), 41–59. http://doi.org/10.1016/j.infsof.2011.09.002

Witten, I. H., y Frank, E. (2005). *Data Mining: Practical Machie Learning Tools and Techniques* (2th ed). Morgan Kaufmann.

Witten, I. H., Frank, E., Trigg, L., Hall, M., Holmes, G., y Cunningham, S. J. (n.d.). Weka: Practical Machine Learning Tools and Techniques with Java Implementations.

Zhang, S., Zhang, C., y Yang, Q. (2010). Data preparation for data mining. *Applied Artificial Intelligence*, *17*, 2003. http://doi.org/10.1080/08839510390219264

Zhao, Y., y Zhang, Y. (2007). Comparison of decision tree methods for finding active objects. http://doi.org/10.1016/j.asr.2007.07.020

Anexo A
Resultados de las pruebas estadísticas en orden de los rangos promedios de los modelos usados en la experimentación (TI, PF y LOC).

Rangos promedios del test de Friedman

Tabla 15 Orden promedio de los algoritmos.

Algorithm	Ranking
TI	1.1111
PF	1.8889
LOC	3

Estadística de **Friedman**, considerando una reducción del desempeño (distribuía acorde a un chi-cuadrado con 2 grados de libertad: 16.222222. *P-value* calculado por la prueba de Friedman: 3.001851594693905E-4.

Iman and Davenport: considerando una reducción del desempeño (distribuía acorde a un chi-cuadrado con 2 y 16 grados de libertad: 73.

P-value calculado: 9.053984463275025E-9.

Pruebas de Post hoc

Resultados alcanzados en las comparaciones post hoc para un α = 0.05 y α = 0.10 los *p-values* ajustados.

- ***P-values* para un *α* = 0.05**

Tabla 16 P-values para un α = 0.05.

i	algorithms	$z = (R0 - R_i)/SE$	*p*	Holm	Shaffer
3	LOC vs. TI	4.006938	0.000062	0.016667	0.016667
2	LOC vs. PF	2.357023	0.018422	0.025	0.05
1	PF vs. TI	1.649916	0.09896	0.05	0.05

Nemenyi's: rechaza la hipótesis nula con un *p-value* ≤ 0.016667.

Holm's: rechaza la hipótesis nula con un *p-value* ≤ 0.05.

Shaffer's: rechaza la hipótesis nula con un *p-value* ≤ 0.016667.

Bergmann's rechaza las hipótesis:

- **LOC vs. PF**
- **LOC vs. TI**

- ***P-values* para un α = 0.10**

Tabla 17 P-values para un α = 0.10.

i	algorithms	$z = (R0 - R_i)/SE$	*p*	Holm	Shaffer
3	LOC vs. TI	4.006938	0.000062	0.033333	0.033333
2	LOC vs. PF	2.357023	0.018422	0.05	0.1
1	PF vs. TI	1.649916	0.09896	0.1	0.1

Nemenyi's: rechaza la hipótesis nula con un *p-value* ≤ 0.033333.

Shaffer's: rechaza la hipótesis nula con un *p-value* ≤ 0.033333.

Bergmann's rechaza las hipótesis:

- **LOC vs. PF**
- **LOC vs. TI**
- **PF vs. TI**

- ***P-values* ajustados**

Tabla 18 P-values ajustados.

i	hypothesis	unadjusted *p*	*pNeme*	*pHolm*	*pShaf*	*pBerg*
1	LOC vs .TI	0.000062	0.000185	0.000185	0.000185	LOC vs .TI
2	LOC vs .PF	0.018422	0.055266	0.036844	0.018422	LOC vs .PF
3	PF vs .TI	0.09896	0.29688	0.09896	0.09896	PF vs .TI

Anexo B
Resultados de las pruebas estadísticas en orden de los rangos promedios de los algoritmos utilizados en la experimentación (M5P, M5Rules, REPTree, Random Tree).

Rangos promedios del test de Friedman

Tabla 19 Orden promedio de los algoritmos

Algoritmo	Ranking
M5P	1
M5Rules	2.4167
REPTree	2.5833
RandomTree	4

Estadística de **Friedman**, considerando una reducción del desempeño (distribuía acorde a un chi-cuadrado con 3 grados de libertad: 16.25. *P-value* calculado por la prueba de Friedman: 0.001007700631635.

Iman and Davenport: considerando una reducción del desempeño (distribuía acorde a un chi-cuadrado con 3 y 15 grados de libertad: 46.428571.

P-value calculado: 7.934887248378574E-8.

Pruebas de Post hoc

Resultados alcanzados en las comparaciones post hoc para un α = 0.05 y α = 0.10 los *p-values* ajustados.

- ***P-values* para un *α* = 0.05**

Tabla 20 P-values para un α = 0.05

i	algorithms	$z = (R0 - R_i)/SE$	*p*	Holm	Shaffer
6	M5P vs. RandomTree	4.024922	0.000057	0.008333	0.008333
5	M5P vs. REPTree	2.124265	0.033648	0.01	0.016667
4	M5Rules vs. RandomTree	2.124265	0.033648	0.0125	0.016667
3	M5Rules vs. M5P	1.900658	0.057347	0.016667	0.016667
2	RandomTree vs. REPTree	1.900658	0.057347	0.025	0.025
1	M5Rules vs. REPTree	0.223607	0.823063	0.05	0.05

Nemenyi's: rechaza la hipótesis nula con un *p-value* ≤ 0.008333.

Holm's: rechaza la hipótesis nula con un *p-value* ≤ 0.01.

Shaffer's: rechaza la hipótesis nula con un *p-value* ≤ 0.008333.

Bergmann's rechaza las hipótesis:

- **M5P vs. RandomTree**

- ***P-values* para un α = 0.10**

Tabla 21 P-values para un α = 0.10

i	algorithms	$z = (R0 - R_i)/SE$	*p*	Holm	Shaffer
6	M5P vs. RandomTree	4.024922	0.000057	0.016667	0.016667
5	M5P vs. REPTree	2.124265	0.033648	0.02	0.033333
4	M5Rules vs. RandomTree	2.124265	0.033648	0.025	0.033333
3	M5Rules vs. M5P	1.900658	0.057347	0.033333	0.033333
2	RandomTree vs. REPTree	1.900658	0.057347	0.05	0.05
1	M5Rules vs. REPTree	0.223607	0.823063	0.1	0.1

Nemenyi's: rechaza la hipótesis nula con un *p-value* ≤ 0.016667.

Holm's: rechaza la hipótesis nula con un *p-value* ≤ 0.02.

Shaffer's: rechaza la hipótesis nula con un *p-value* ≤ 0.016667.

Bergmann's rechaza las hipótesis:

- **M5P vs. RandomTree**

- ***P-values* ajustados**

Tabla 22 P-values ajustados

i	hypothesis	unadjusted *p*	*pNeme*	*pHolm*	*pShaf*	*pBerg*
1	M5P vs .RandomTree	0.000057	0.000342	0.000342	0.000342	0.000342
2	M5P vs .REPTree	0.033648	0.201888	0.168240	0.100944	0.100944
3	M5Rules vs .RandomTree	0.033648	0.201888	0.168240	0.100944	0.100944
4	M5Rules vs .M5P	0.057347	0.344081	0.172041	0.172041	0.114694
5	RandomTree vs .REPTree	0.057347	0.344081	0.172041	0.172041	0.114694
6	M5Rules vs .REPTree	0.823063	4.938380	0.823063	0.823063	0.823063