UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

AUTOMATIZACIÓN DE MODELOS DE ESTIMACIÓN UTILIZANDO APRENDIZAJE DE MÁQUINAS

T E S I S

QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
JESÚS IVÁN SAAVEDRA MARTÍNEZ

TUTORA:
M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

CIUDAD UNIVERSITARIA, CD. MX., 2020
DICIEMBRE

A mis padres, hermanos, familiares y amigos, por ustedes es posible este trabajo.
Gracias al Posgrado en Ciencia e Ingeniería de la Computación y a la Facultad de Ciencias por la formación que me han dado.
En verdad, gracias.
Yo.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Jesús Iván Saavedra Martínez, Ciudad Universitaria, Cd. Mx., 2020.

Resumen

La presente tesis de maestría presentada por Jesús Iván Saavedra Martínez, como requisito para obtener el título de Maestro en Ciencias e Ingeniería de la Computación, muestra el desarrollo y análisis de resultados de aplicar un sistema que automatiza la generación de modelos de estimación de software utilizando aprendizaje de máquinas. El sistema utiliza modelos algorítmicos, los cuales representan el enfoque más formal en la estimación de software, basados procedimientos como la regresión para producir un modelo construido por una o varias expresiones matemáticas que relacionan el esfuerzo con variables de los proyectos.

La eficiencia de aplicar aprendizaje de máquinas para generar modelos de estimación depende en gran medida del preprocesamiento de datos y de la configuración de los algoritmos utilizados. En los capítulos de esta tesis, se especifica cómo se automatizan las técnicas de preprocesamiento de datos, entrenamiento y evaluación de modelos. Asimismo, se muestran diferentes casos de estudio de aplicar los modelos automatizados, en los cuales principalmente se analiza la precisión de los modelos comparándolos con otros métodos de estimación.

Entre las técnicas de aprendizaje de máquinas utilizadas, debido a la naturaleza del problema, están las relacionadas con el aprendizaje supervisado y las redes neuronales artificiales.

Se considera que el uso de técnicas de Inteligencia Artificial en la estimación de software constituye un área de investigación en desarrollo y de interés para la Ingeniería de Software.

Palabras clave - estimación de software; estimación de proyectos; automatización de estimación; modelos de estimación; modelos algorítmicos; tamaño funcional; aprendizaje de máquinas; aprendizaje supervisado; algoritmos de regresión; redes neuronales;

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Presentación

Generar modelos de estimación confiables es una actividad importante en pequeñas y grandes organizaciones. El desarrollo de estos modelos se basa normalmente en información obtenida de proyectos pasados y requiere de un análisis profundo y preciso.

En la industria se han desarrollado un amplio número de prácticas para estimar proyectos de software, donde los modelos de algorítmicos representan el enfoque más formal y son los que han proporcionado resultados más confiables. Sin embargo, aún predomina el uso de prácticas no formales como el juicio de experto, que no permite la madurez de la Ingeniería de Software [1].

Hoy en día, la mayoría de las variables de los proyectos, con las que se cuenta a la hora de estimar, son de tipo categórica, como lo son la metodología de desarrollo, la arquitectura del sistema, el lenguaje de programación, etc [2]. En ocasiones se tienen variables de tamaño (basadas en Puntos de Función, Líneas de Código, etc.) y en otras no se tiene ninguna de ellas, lo cual representa un problema para algunos métodos de estimación.

El trabajo de esta tesis consiste en analizar la precisión de un sistema que genera un conjunto de modelos de estimación con los criterios de calidad más altos posibles (al preprocesamiento de datos aplicado). Este sistema automatiza el preprocesamiento de datos, entrenamiento y evaluación de modelos utilizando algoritmos de aprendizaje de máquinas.

1.2. Objetivos

Este trabajo tiene por objetivo principal mejorar la precisión de la estimación de proyectos de software mediante el desarrollo e implementación un sistema. De manera general, este sistema recibe como parámetros una base de datos y, mediante el uso de algoritmos de aprendizaje de máquinas, de manera automática realiza el

preprocesamiento de los datos, el entrenamiento y evaluación de modelos de estimación con los criterios de calidad más altos posibles. La eficiencia del sistema es analizada mediante 5 casos de estudio que evalúan la precisión de los modelos de estimación generados.

Los beneficios de contar con un sistema que genere de manera automática modelos de estimación son los siguientes:

- Mayor probabilidad de tener proyectos exitosos al conocer a detalle la confiabilidad de los modelos.
- Ahorro en tiempo y recursos, donde se suele invertir demasiado.
- No se necesita experiencia para estimar, al tener automatizada la estimación.
- Monitoreo y control factible de proyectos, al tener estimaciones confiables.

1.3. Motivación

En la actual industria de desarrollo software, en la que surgen avances tecnológicos frecuentes, las mejoras metodológicas y los cambios en las habilidades de los equipos de desarrollo demandan modelos de estimación que se adapten a entornos cambiantes a través del tiempo [3]. Los algoritmos de aprendizaje de máquinas tienen la capacidad de aprender y mejorar en función de las predicciones generadas a través de la estimación más cercana a los datos observados y cualquier conocimiento previo [4].

El uso de algoritmos de aprendizaje de máquinas en la estimación de software recientemente ha ganado una gran popularidad, esto a causa del amplio margen de error en las estimaciones de proyectos de los últimos años [5].

El proceso de aprendizaje de los algoritmos puede tomar de una tarea descriptiva o predictiva elegida sin supervisión o supervisada. El aprendizaje sin supervisión se basa en patrones que no están vinculados a ninguna característica en el conjunto de datos de entrenamiento, para fines tales como agrupación, reglas de asociación o detección de anomalías [6]. En cambio, el aprendizaje supervisado debe tener entradas definidas explícitamente correspondientes a salidas conocidas por una variable objetivo, la cual se utiliza para la clasificación o la regresión [7]. En ambos casos, existen numerosos algoritmos disponibles que se pueden aplicar según el resultado deseado.

Este trabajo se basa en la predicción del esfuerzo de desarrollo de proyectos de software utilizando algoritmos de regresión. Esto con la finalidad de analizar si con estas técnicas es posible generar modelos de estimación más precisos que los métodos de estimación más utilizados en la industria.

1.4. Planteamiento del problema

La estimación de proyectos de software es una actividad importante en pequeñas y grandes organizaciones alrededor del mundo. Se han realizado investigaciones en la estimación de proyectos de software durante más de 20 años y una gran cantidad de modelos han sido propuestos. La cuestión es: ¿qué tan eficiente es actualmente la estimación de software en la industria? La respuesta es: no muy eficiente [8].

De acuerdo con un estudio de la Government Accountability Office (GAO) [9], si se establecieran bases de referencia más realistas de los requerimientos, costos, cronogramas y riesgos durante las fases de planificación de los proyectos de software, se podrían evitar casi la mitad de los proyectos de TI cancelados o sobre-estimados.

De la misma forma, el Chaos Report 2020 del Standish Group [8] analiza los cinco factores principales encontrados para lograr proyectos exitosos, los cuales son los siguientes:

1. Participación del usuario
2. Apoyo de la gerencia
3. Especificación de requerimientos oportuna
4. Planificación adecuada
5. Expectativas realistas

En el caso del cuarto punto, una estimación precisa permite lograr una planificación adecuada, ya que el esfuerzo (o costo) estimado se puede distribuir en las diferentes actividades de las fases de desarrollo de un proyecto.

Con el apoyo de técnicas de aprendizaje de máquinas, este trabajo pretende mejorar la eficiencia de las estimaciones, ya que hoy en día la probabilidad de tener un proyecto exitoso no es la más adecuada.

1.5. Metodología

Para cumplir con los objetivos de este trabajo de tesis de maestría se plantearon los siguientes pasos a seguir:

1. Investigar el estado del arte relacionado a la estimación de software con aprendizaje de máquinas.
2. Investigar y establecer el marco teórico
    - La estimación de software, desde sus principios básicos hasta los métodos más utilizados en la industria.

   - El aprendizaje de máquinas, principalmente los algoritmos de regresión en aprendizaje supervisado.

3. Desarrollar un sistema que automatice la generación de modelos de estimación de software mediante la implementación de funciones que hagan las siguientes tareas:

   - Preprocesamiento de los datos (proyección, reducción, tratamiento de datos faltantes, normalización y codificación one-hot)
   - Entrenamiento de modelos utilizando algoritmos de aprendizaje de máquinas.
   - Evaluación de modelos con métricas de evaluación de errores.
   - Selección de modelos con los criterios de calidad más precisos.

4. Generar y evaluar un conjunto de modelos con el sistema desarrollado para probar su eficiencia.

5. Documentar resultados y conclusiones obtenidas.

1.6. Contribuciones

La principal contribución de este trabajo es el sistema desarrollado que permite generar de manera automática modelos de estimación de software con los criterios de calidad más altos posibles (al preprocesamiento de datos aplicado). Este sistema puede ser utilizado principalmente por administradores o líderes de proyectos para estimar el esfuerzo de desarrollo de los mismos. Sin embargo, puede ser utilizado también por usuarios, desarrolladores o cualquier involucrado en el proyecto, ya que no es necesario ser un experto en temas de medición y estimación de software para poder utilizarlo. De esta manera, la probabilidad de tener proyectos exitosos aumenta ya que una estimación confiable permite un monitoreo y control factible del proyecto.

Por otra parte, este trabajo muestra la eficiencia de utilizar aprendizaje de máquinas en la estimación de proyectos de software, lo que da un punto de referencia para futuras investigaciones relacionadas con este tema.

1.7. Estructura de la tesis

Este documento está dividido en 6 secciones principales.

1. Introducción. Al principio (Capítulo 1) se encuentra la Introducción de la tesis, la cual abarca las secciones de presentación, objetivos, motivación, planteamiento del problema, metodología, contribuciones y estructura de la misma.

2. Marco teórico. El capítulo 2 muestra el marco teórico relacionado con la estimación de software y el aprendizaje de máquinas, resaltando como los algoritmos de regresión de aprendizaje supervisado han tenido un gran impacto y como pueden ayudar a la Ingeniería de Software para mejorar la calidad de las estimaciones.

3. Estado del arte. En el capítulo 3 se presenta el estado del arte, revelando la situación actual de la estimación de software en la industria y como algunos estudios han demostrado cómo es posible mejorar la precisión de las estimaciones utilizando técnicas de aprendizaje de máquinas.

4. Metodología propuesta. El capítulo 4 muestra a detalle como fue desarrollado el sistema que automatiza la generación de modelos de estimación mediante el uso de funciones que realizan tareas específicas.

5. Resultados experimentales. En el capítulo 5 se muestran los resultados de 5 casos de estudio, los cuales analizan la eficiencia de modelos de estimación generados con el sistema desarrollado. Uno de los casos de estudio principales es el que compara estos modelos con los métodos de estimación más utilizados en la industria.

6. Por último, se presentan las conclusiones de la tesis, se indica si se cumplió con el objetivo al problema planteado y se da una breve descripción del trabajo futuro que se puede llevar a cabo a partir de esta investigación.

Capítulo 2

Marco Teórico

2.1. Fundamentos de estimación en Ingeniería de Software

2.1.1. Introducción a la Ingeniería de Software

2.1.1.1. Conceptos básicos

La Ingeniería de Software es la aplicación del método científico sobre la teoría y creación de conocimiento sobre si misma. Está dedicada al estudio de sus actividades y centrada en generar teorías, modelos explicativos o enunciados descriptivos sobre la práctica en el desarrollo de software. La IEEE (Institute of Electrical and Electronics Engineers) [10] define a la Ingeniería de Software como:

> *Definición.* La Ingeniería de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software.

La definición anterior menciona tres calificativos que pueden aplicarse a la ingeniería, definidos de la siguiente manera [11]:

- Sistemático: cuando sigue completamente un método.
- Disciplinado: cuando está bien organizado y sigue reglas o estándares.
- Cuantificable: si es capaz de ser medido o descrito como una cantidad.

La Ingeniería de Software aplica estándares y métodos que permiten cumplir de manera satisfactoria con sus objetivos fundamentales. Entre estos objetivos están [12]:

- Mejorar el diseño del software, adaptándose a las necesidades de las empresas.
- Promover mayor calidad al desarrollar aplicaciones.
- Brindar mayor exactitud en los costos y tiempo de desarrollo de los proyectos.

- Aumentar la eficiencia de los sistemas creando procesos que permitan medir la calidad del software.
- Mejor organización en los equipos de trabajo de las áreas de desarrollo de software.
- Detectar a través de pruebas posibles mejoras para un mejor funcionamiento del software.

Así mismo, la Ingeniería de Software trata fundamentalmente de actividades llevadas a cabo por personas que producen, usan o modifican artefactos. Esas actividades responden a planes parciales o totalmente prescritos y abarcan acciones con propósitos específicos.

Una secuencia de actividades para un propósito en especifico define un proceso. Los procesos fundamentales en la Ingeniería de Software incluyen actividades de gestión, producción, comunicación y documentación [12].

2.1.1.2. Procesos fundamentales de la Ingeniería de Software

Uno de los objetivos de la guía SWEBOK (Software Engineering Body of Knowledge) [13] establece la necesidad de especificar el alcance de la Ingeniería de Software. Por consiguiente, la guía define 15 áreas clave de conocimiento listadas en la Tabla 2.1, resaltando los procesos fundamentales del ciclo de vida del software de acuerdo con el estándar ISO/IEC/IEEE 12207 [14].

Procesos Fundamentales	Otras áreas
Requerimientos de software	Gestión de configuración de software
Diseño de software	Proceso de Ingeniería de Software
Construcción de software	Modelos y métodos de Ingeniería de Software
Pruebas de software	Calidad de software
Mantenimiento de software	Práctica profesional en Ingeniería de Software
Gestión de Ingeniería de Software	Economía de la Ingeniería de Software
	Fundamentos de computación
	Fundamentos de matemáticas
	Fundamentos de ingeniería

Tabla 2.1: Áreas clave de conocimiento de la guía SWEBOK

De acuerdo con metodologías tradicionales, las principales actividades del proceso de desarrollo son similares (o las mismas) que los procesos fundamentales. La Figura 2.1 muestra el proceso general de estas actividades.

- Requerimientos: Consiste en obtener y documentar los requerimientos necesarios para desarrollar un sistema. Un requerimiento de software [10] es la capacidad que debe alcanzar o poseer un sistema o componente para satisfacer un contrato, estándar, especificación u otro documento formal.

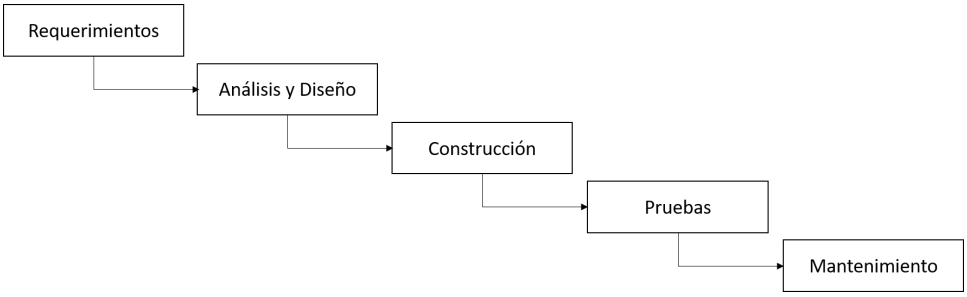

Figura 2.1: Actividades de una metodología de desarrollo tradicional

- Análisis y diseño: Pretende obtener una descripción de la "mejor" solución que dé cumplimiento a los requerimientos. Da la especificación de componentes o módulos del sistema y el modo en que estos se comunican. Define la arquitectura, los componentes, las interfaces y otras características del sistema.
- Construcción: Define un conjunto de actividades que engloban fundamentalmente la codificación, pero también la verificación, depuración y pruebas. El producto resultante es el entregable más importante de un proyecto de software.
- Pruebas: Tiene el objetivo de evaluar y mejorar la calidad del producto. El éxito de las pruebas de software está en la detección de errores en el proceso de desarrollo y en el software obtenido. Una prueba de software [10] es el proceso de analizar un componente para detectar las diferencias entre las condiciones existentes y los requerimientos.
- Mantenimiento: Es la modificación de un producto de software después de la entrega para corregir fallos, para mejorar su rendimiento o para adaptarlo a un entorno modificado [15]. Se considera que el mantenimiento ocurre durante todo el desarrollo del software.

2.1.1.3. Metodologías ágiles para el desarrollo de software

El PMI (Project Management Institute) [16] considera Agile como un paradigma definido por valores, guiado por principios y que se manifiesta a través de diferentes prácticas seleccionadas de acuerdo con las necesidades de los proyectos. Dichos valores y principios se presentan en el "Manifiesto ágil" [17], el cual consta de los siguientes cuatro puntos clave resumidos en la Figura 2.2:

- Individuos e interacciones sobre procesos y herramientas: da mayor énfasis a las relaciones entre los individuos involucrados en los procesos que a los procesos mismos o a las herramientas utilizadas. Este principio permite dar prioridad a las personas, la comunicación y la interacción, teniendo especialmente en cuenta la influencia que ejerce la toma de decisiones sobre cada uno de estos factores.

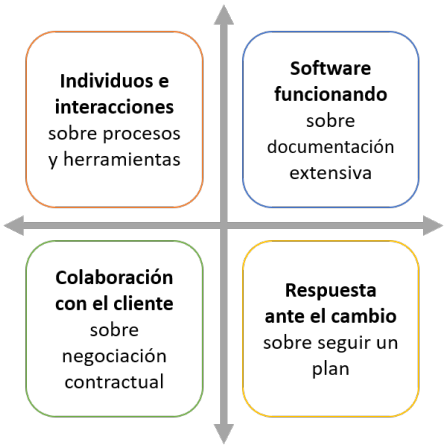

Figura 2.2: Puntos clave del manifiesto ágil

- Software funcionando sobre documentación extensiva: el objetivo fundamental del equipo de desarrollo es la entrega de soluciones de software de forma constante y en el menor intervalo de tiempo posible. La documentación se reduce a un nivel apropiado, evitando consumir tiempo que puede dedicarse al desarrollo, pruebas o puesta en marcha del software.
- Colaboración con el cliente sobre la negociación contractual: da preferencia a la cooperación y el trabajo en equipo sobre la estipulación de contratos estrictos y extensivos.
- Respuesta ante el cambio sobre seguir un plan: busca la adaptación a los cambios introducidos durante las fases del desarrollo de software de forma flexible y sin comprometer la calidad del producto. Para ello, el equipo del proyecto debe contar con las herramientas necesarias para realizar cambios en el momento en el que sean requeridos.

Las metodologías ágiles intentan ofrecer una respuesta a los negocios que requieren de procesos de desarrollo menos rígidos y más rápidos [18]. Una característica es reconocer que el desarrollo de software es un proceso empírico. El proceso de desarrollo de software debe procurar ser flexible y no estrictamente predictivo, dado que el mismo esta sujeto a una serie de cambios que en gran parte son impredecibles al inicio del desarrollo del proyecto. Por lo tanto, el desarrollo ágil requiere de pequeños ciclos de inspección y adaptación, con una constante retroalimentación en lugar de intentar contar con un proceso definido con alto nivel de exactitud y detalle desde las etapas iniciales del proyecto [19].

Scrum es uno de los métodos ágiles más populares. En realidad es un framework adaptable, iterativo, rápido, flexible y eficaz, diseñado para ofrecer la entrega de valor

constante a lo largo del proyecto. Scrum garantiza transparencia en la comunicación y crea un ambiente de responsabilidad colectiva y de progreso continuo [20].

Los procesos de Scrum abordan actividades específicas y el flujo de un proyecto. La Figura 2.3 muestra las 5 fases de Scrum. La principal diferencia con una metodología tradicional esta en tener entregas constantes de funcionalidad en lugar de una sola entrega al final del proyecto.

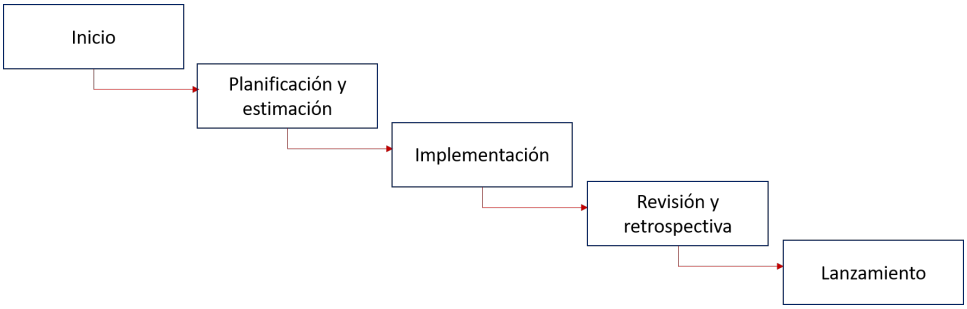

Figura 2.3: Puntos clave del manifiesto ágil

2.1.2. La medición del software

2.1.2.1. Fundamentos de medición

En la Ingeniería de Software, a menudo no se hacen estimaciones de esfuerzo y costo precisas, no se alcanza la calidad suficiente y/o se excede en el presupuesto de los proyecto [8]. La medición permite ganar comprensión acerca del proyecto, al proporcionar un mecanismo de evaluación objetiva.

> *"cuando puedes medir aquello de lo que hablas y expresarlo en números, sabes algo acerca de ello; pero cuando no puedes medir, cuando no puedes expresarlo en números, tu conocimiento es pobre e insatisfactorio"* [21]

La medición se aplica en el proceso de desarrollo de software con la intención de mejorarlo de manera continua. Además, puede utilizarse para apoyar en la estimación, control de calidad, valoración de productividad y control del proyecto.

Por otra parte, el término métrica es usado como referencia a múltiples conceptos. Por ejemplo, la calidad a ser medida, el procedimiento o resultado de la medición, etc. Abran [22] menciona que en la Ingeniería de Software el término es aplicado en los siguientes casos:

- La medición de un concepto. Por ejemplo, la complejidad ciclomática (métrica propuesta en 1976).
- Modelos de calidad. Por ejemplo, el estándar ISO 9126 – la calidad del producto de software.

- Modelos de estimación. Por ejemplo, los modelos algorítmicos y no algorítmicos, como la ecuación de esfuerzo de Halstead, COCOMO I y II, etc.

2.1.2.2. ¿Qué se puede medir en la Ingeniería de Software?

Una métrica debe poder medir adecuadamente el atributo de la entidad a medir, pero también debe definir cómo se va a realizar la medición. En la Ingeniería de Software encontramos tres tipos de entidades [12]:

- Productos: cualquier artefacto, entregable o documento que resulta de un proceso del desarrollo de software. Por ejemplo, la especificación de requerimientos, el código fuente, manuales de usuario, etc.
- Procesos: todas las actividades del desarrollo de software. Por ejemplo, los procesos fundamentales. Algunas de las métricas relacionadas con los procesos son el tiempo invertido en las actividades o el tiempo para corregir un defecto.
- Recursos: cualquier entrada de una actividad. Por ejemplo, el número de personas por proyecto, presupuesto, esfuerzo, herramientas utilizadas, etc.

Las entidades tienen o se les asignan atributos a medir, Sánchez et al. [12] distinguen entre atributos internos y atributos externos. Los internos son aquellos que se pueden medir directamente a partir de dicha entidad, mientras que los externos se relacionan con el entorno y son asociados con temas de calidad.

Los atributos internos del software se pueden dividir en factores técnicos y factores funcionales. La Figura 2.4 muestra algunas características de estos factores y para quiénes son significativos.

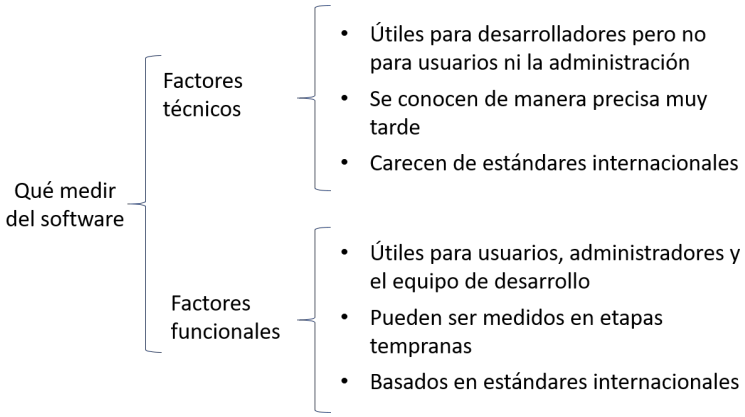

Figura 2.4: Factores ténicos y funcionales en la medición del software

2.1.2.3. Medidas de tamaño

La métrica más antigua para los proyectos de software es la de líneas de código (Lines of Code – LOC) [23]. Fue introducida por primera vez alrededor de 1960 y se utilizó para estudios económicos, de productividad y de calidad, cuándo el término software era prácticamente sinónimo de programa de computadora.

Por otra parte, las métricas de software orientadas a función usan una medida de la funcionalidad del software como un valor de normalización. El estándar ISO/IEC 14143 [24] define y da las características de medición de tamaño funcional.

> *Definición.* El tamaño funcional [25] se define como un tamaño de software derivado de cuantificar los Requerimientos Funcionales de Usuario.

La métrica orientada a función más utilizada en la industria es el Punto de Función (Function Point – FP) [26]. Esta métrica fue descrita en 1979 por Allan Albrecht, con el objetivo de medir el tamaño funcional del software que proporciona a los usuarios sin considerar el código fuente (como lo hace LOC). De manera general, el proceso de medición del método FP se basa en el conteo de funciones de datos (archivos lógicos internos y archivos de interfaz externa) y funciones transaccionales (entradas, salidas y consultas externas) que son ponderadas por un factor de ajuste de hasta $\pm 35\,\%$ [23].

De la misma manera, está el método COSMIC (Common Software Measurement International Consortium) [27] que publicó su primera versión del método en 1999 dando cumplimiento al estándar ISO/IEC 14143 para medir el tamaño funcional del software. Este método ha sido diseñado con la industria y por la industria para ser aplicable en dominios de aplicaciones de software de gestión, de tiempo real y algunos tipos de software científicos. El tamaño medido por el método COSMIC depende solo de los requerimientos funcionales y es independiente de cualquier requerimiento no funcional.

De igual importancia, los casos de uso son utilizados ampliamente como una forma para describir requerimientos que implican características y funciones del software. Estos se definen al principio del proceso de desarrollo, lo cual permite emplearlos para una estimación antes de iniciar actividades de construcción.

> *Definición.* Un caso de uso [28] especifica la manera de cómo utilizar un sistema para que un usuario pueda lograr un objetivo en particular.

Basado en el auge de UML (Unified Modeling Language) y considerando los casos de uso, Karner [29] desarrolló el método puntos de casos de uso (Use Case Points – UCP) en 1993. Esta métrica se basa en lógica y cálculos similares a los FP en concepto, pero no en detalles específicos. Los factores que intervienen en los puntos de casos de uso incluyen factores de complejidad técnica y de ambiente. Una vez calculados, los puntos de casos de uso se pueden utilizar para estimar el esfuerzo y los costos para el desarrollo de un proyecto [30].

2.1.3. El proceso de estimación

2.1.3.1. Introducción a la estimación

La definición de estimación según la guía PMBOK (A Guide to the Project Management Body of Knowledge) [16] es el siguiente enunciado:

> *"Una evaluación cuantitativa de la cantidad o resultado probable. Normalmente se aplica a los costos, recursos, esfuerzo y duraciones del proyecto y suele estar precedida por un modificador".*

Cuando se pide una estimación, comúnmente se está pidiendo un compromiso o un plan para cumplir con el desarrollo del proyecto. Es fundamental comprender lo que es una estimación, lo que no lo es y cómo hacer mejores estimaciones.

> *Definición.* Una estimación [31] es una predicción que tiene la misma probabilidad (distinta de cero) de estar por encima o por debajo del valor real.

Expertos de estimación han propuesto diversas definiciones de una buena estimación. McConnell [31] ha declarado que con una exactitud de $\pm 10\,\%$ es posible, pero sólo en proyectos bien controlados, de lo contrario, los proyectos tienen demasiada variabilidad para lograr ese nivel de precisión.

La mayor parte de los métodos de estimación tienen como objetivo predecir el costo y esfuerzo del proyecto. De acuerdo con Tuya [32], estos métodos pueden ser clasificados en cuatro categorías:

- Juicio de experto: basado en la experiencia de una persona.
- Analogía: los expertos comparan el proyecto a estimar con uno o más proyectos anteriores, intentando encontrar similitudes y diferencias.
- Descomposición: basado en un análisis de las características del proyecto, haciendo estimaciones individuales sobre los componentes del mismo.
- Modelos algorítmicos: basados en 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 en información obtenida de proyectos pasados.

Los modelos algorítmicos representan el enfoque más formal y son los que han proporcionado resultados más confiables. Estos modelos utilizan procedimientos como la regresión para construir un modelo con una o varias expresiones matemáticas que relaciona el esfuerzo con las variables del proyecto, y en algunos casos, varios factores de ajuste secundarios [32].

2.1.3.2. El proceso de estimación completo

El proceso de estimación completo contiene las siguientes seis fases [1], como se muestra en la Figura 2.5:

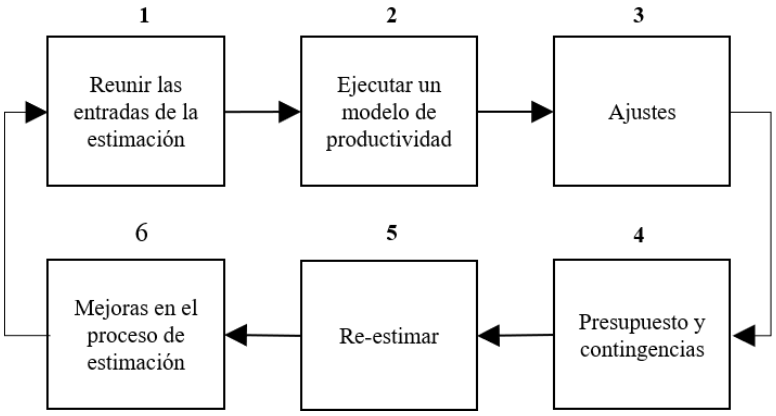

Figura 2.5: El proceso de estimación completo

1. Reunir las entradas de la estimación: puede incluir tanto variables continuas (como el tamaño funcional) como variables categóricas sobre el contexto de desarrollo del proyecto.
2. Ejecutar un modelo de productividad: generar o utilizar un modelo existente. .
3. Ajustes: tomar en cuenta variables e información no incluida en las entradas, como evaluación de riesgos o identificación de factores de incertidumbre.
4. Presupuesto y contingencias: tomar decisión sobre el presupuesto del proyecto.
5. Re-estimar: cuando sea requerido por el seguimiento y control del proyecto.
6. Mejoras en el proceso de estimación: fase a nivel de la organización que analiza la eficiencia del proceso de estimación una vez que un proyecto ha finalizado.

Los objetivos principales del proceso de estimación consisten en dar información acerca de lo siguiente:

1. Rango de posibles valores de la estimación.
2. Retroalimentación acerca de que tan precisa es la información.
3. Limitaciones de la información utilizada en la entrada y salida del proceso de estimación.
4. Análisis y mitigación de riesgos mediante documentación.

2.1.4. Modelos estimación de proyectos

2.1.4.1. Modelos de productividad y modelos de estimación

Los modelos de productividad típicamente son construidos utilizando datos de proyectos finalizados. Por lo tanto, la mayoría de los llamados "modelos de estimación" en la literatura, son en realidad modelos de productividad [1].

Asimismo, un modelo de productividad utiliza un análisis de regresión sobre los datos de los proyectos. Un ejemplo común es el modelo COCOMO [33], el cual calcula el esfuerzo del desarrollo de software en función del tamaño del programa expresado en líneas de código.

Algunos de los beneficios de los modelos de productividad basados en proyectos finalizados son los siguientes [1]:

- La eficiencia de los modelos puede ser analizada y descrita.
- Cualquiera puede utilizar los modelos para estimar futuros proyectos.
- Siempre que se ingresa la misma entrada al modelo, se obtendrá como resultado la misma salida.

Para obtener un modelo de estimación, es necesario evaluar la calidad del modelo de productividad utilizando un conjunto de proyectos que no fue incluido para generar el modelo. De esta manera, el modelo de productividad junto con los criterios de calidad dan como resultado un modelo de estimación. Algunas métricas de calidad se describen en la sección 2.2.5.4.

2.1.4.2. Incertidumbre en un modelo de estimación

La Figura 2.6 muestra una representación gráfica de un modelo de estimación, los ejes representan el tamaño y el esfuerzo de los proyectos. La gráfica muestra que la mayor parte de los datos no entran en la ecuación matemática pero si a cierta distancia de ella. Esto significa que el modelo no relaciona exactamente el tamaño con el esfuerzo, ya que algunos datos están cerca de la línea y otros muy separados de ella.

En algunas ocasiones, se pueden generar modelos en los que el esfuerzo disminuye con el aumento de los valores de las entradas. En este caso, la pendiente del modelo sería negativa. Siempre que se observen tales modelos se deben considerar lo siguiente:

- Verificar la calidad de las entradas utilizadas en la generación de los modelos de estimación.
- Tener cuidado de no utilizar estos modelos para rangos de valores negativos, ya que los valores estimados carecen de sentido.

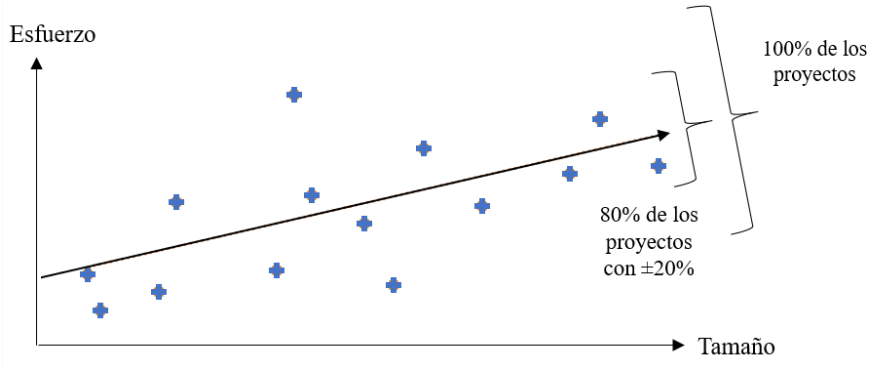

Figura 2.6: Incertidumbre en un modelo de productividad (adaptada de [1])

Cuando un modelo de estimación no puede modelar de la mejor manera los datos de entrada, puede ser necesario identificar más de un modelo para el mismo conjunto de datos. Los subconjuntos de datos para generar los diferentes modelos pueden tener algunas características en común, como el contexto de desarrollo, arquitectura, lenguaje de programación, sistema operativo, etc. La Figura 2.7 muestra un conjunto de datos en el que se identificaron dos subconjuntos de datos, cada una con su respectiva regresión.

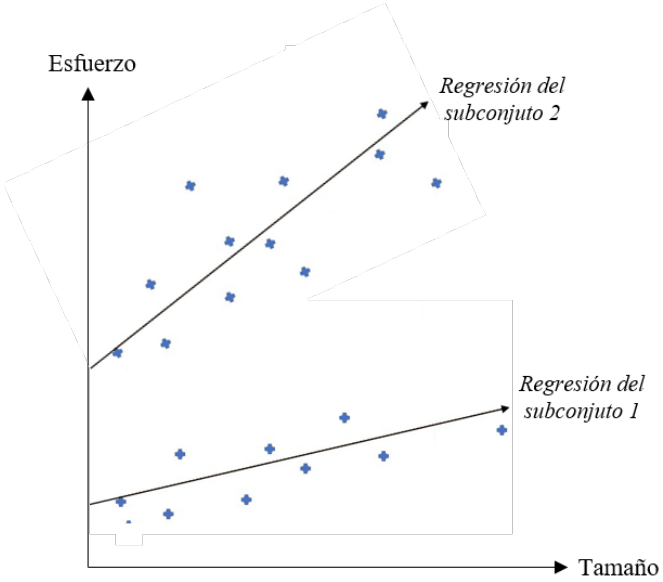

Figura 2.7: Subconjuntos de datos con su respectiva regresión (adaptada de [1])

2.2. Fundamentos de aprendizaje de máquinas

2.2.1. Introducción al aprendizaje de máquinas

En la actualidad, la cantidad de información que puede contener una base de datos excede la habilidad de las personas para analizarla sin el uso de técnicas de análisis automatizadas. Debido a la presencia de ruido, datos inconsistentes, datos faltantes, etc., es necesario aplicar técnicas de preprocesamiento sobre los datos [34].

El aprendizaje de máquinas es la ciencia de programar computadoras para que puedan aprender de los datos.

> *Definición.* Aprendizaje de máquinas [35] es el campo de estudio que brinda a las computadoras la capacidad de aprender sin ser programadas explícitamente.

En general, aprendizaje de máquinas puede ser aplicado a los siguiente [35]:

- Problemas donde las soluciones existentes requieren mucho ajuste manual o muchas reglas. Un algoritmo en aprendizaje de máquinas puede simplificar el código y dar un mejor rendimiento.
- Problemas complejos donde no hay una buena solución utilizando un enfoque tradicional, técnicas de aprendizaje de máquinas pueden encontrar una solución.
- Entornos en constante cambio donde aprendizaje de máquinas puede adaptarse fácilmente.
- Obtener ideas sobre problemas complejos y grandes cantidades de datos.

2.2.2. Tipos de sistemas de aprendizaje de máquinas

2.2.2.1. Problemas que aprendizaje de máquinas puede resolver

Los sistemas de aprendizaje de máquinas se pueden clasificar según la cantidad y el tipo de supervisión que reciben durante el entrenamiento. Existen tres categorías principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [35]. La Figura 2.8 muestra un diagrama de los tipos de sistemas de aprendizaje de máquinas.

2.2.2.2. Aprendizaje supervisado

Los algoritmos de aprendizaje de máquinas que han mostrado mejores resultados son aquellos que automatizan los procesos de toma de decisiones a partir de ejemplos conocidos. Esto se conoce como aprendizaje supervisado [36]. Su funcionamiento se basa en proporcionar las entradas y salidas correspondientes, y el algoritmo encuentra una manera de producir la salida deseada a partir de una entrada. En particular, los

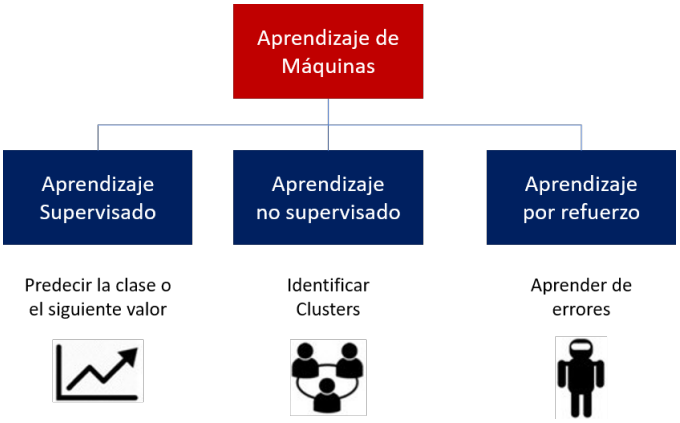

Figura 2.8: Tipos de sistemas en aprendizaje de máquinas

algoritmos pueden crear una salida para una entrada que nunca habían visto antes. Los datos de entrenamiento deben incluir forzosamente las salidas deseadas.

El problema para resolver más utilizado en aprendizaje supervisado es la clasificación. Un ejemplo de este sería entrenar un algoritmo con muchos correos electrónicos junto con su clase, spam o ham, y el algoritmo debe aprender a clasificar nuevos correos electrónicos.

Otros problemas comunes son los de predecir un valor numérico, conocidos como problemas de regresión. Un ejemplo sería predecir el costo de un automóvil dado un conjunto de variables (kilometraje, modelo, marca, etc.). Para entrenar el algoritmo, se le deben dar como entrada muchos ejemplos de automóviles, incluyendo sus variables y salidas deseadas (es decir, sus costos).

2.2.2.3. Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, en el aprendizaje no supervisado [35] los datos de entrenamiento no incluyen las salidas deseadas, es decir, el algoritmo trata de aprender y encontrar patrones a través de las entradas. Estos problemas aparentan ser más complejos que los anteriores. Sin embargo, los problemas más habituales en este tipo de aprendizaje son los de clúster. Estos buscan grupos de registros que son similares entre sí y, al mismo tiempo, diferentes del resto. Una vez obtenidos los grupos se les asigna una clasificación, lo que muchas veces lleva al descubrimiento de patrones desconocidos.

Del mismo modo, los algoritmos de visualización son un buen ejemplo de aprendizaje no supervisado. Estos algoritmos reciben una gran cantidad de datos complejos y generan una representación 2D o 3D de los datos, los cuales se pueden graficar fácilmente. De esta manera, se intenta preservar la mayor cantidad de estructura posible para comprender cómo se están organizando los datos y para tratar de identificar patrones inesperados.

La Figura 2.9 muestra una comparación entre el aprendizaje supervisado y el no supervisado utilizando el mismo conjunto de datos con algoritmos de entrenamiento distintos. Se puede observar que el aprendizaje supervisado conoce la clase a la que pertenecen los puntos e identifica una división entre ellos, mientras que el no supervisado identifica puntos céntricos entre los grupos de puntos para detectar algún patrón de comportamiento.

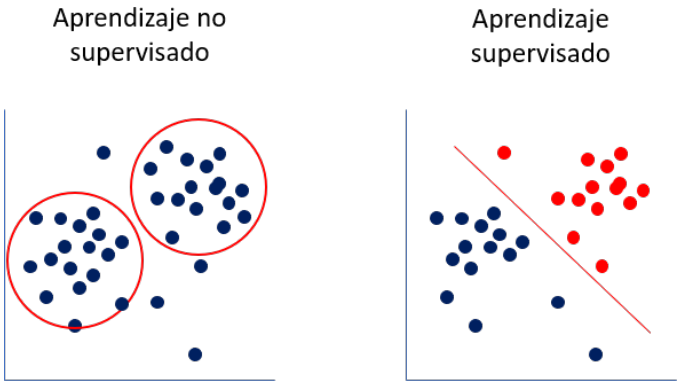

Figura 2.9: Comparación entre aprendizaje supervisado y no supervisado

2.2.2.4. Aprendizaje por refuerzo

El tercer tipo de aprendizaje de máquinas es conocido como aprendizaje por refuerzo. Este se utiliza con menor frecuencia que los dos anteriores pero es útil para determinar acciones sobre como debe actuar o comportarse un agente cuando se le dan recompensas o castigos. En general, al agente no se le indican las acciones que debe tomar, sino que él debe experimentar para encontrar las acciones que lo lleven a una mayor recompensa [37].

Por ejemplo, ¿cómo es que aprende una mascota a sentarse? Cada vez que lo hace correctamente se le da una galleta (un premio). Esta galleta es un refuerzo positivo para ella. Con el tiempo, la mascota habrá aprendido que al sentarse obtendrá una recompensa.

De la misma forma, un algoritmo puede aprender por refuerzo como ganar un juego. Por ejemplo, se le podría dar un refuerzo positivo (recompensa) cada vez que el algoritmo gana. La idea básica es aprender a realizar movimientos y analizar el refuerzo que proporcionan.

2.2.3. Algoritmos de aprendizaje supervisado para tareas de regresión

2.2.3.1. Modelos lineales

El aprendizaje de máquinas adoptó a los modelos lineales existentes. Sin embargo, debido a que aprender de los datos es una disciplina tan práctica, aprendizaje de máquinas separa los modelos lineales de todo lo relacionado con la estadística y conserva solo las fórmulas matemáticas [38].

De este modo, los modelos lineales esperan que el valor objetivo sea una combinación lineal de las variables. En notación matemática, si y es el valor predicho entonces [39]:

$$y(w, x) = w_0 + w_1 x_1 + ... + w_n x_n$$

el vector w $= (w_1, ..., w_n)$ es el coeficiente y w_0 es la variable independiente.

La regresión lineal (Linear Regression – LR) es uno de los casos más comunes de modelos lineales. Sin embargo, sigue siendo un algoritmo simple, comprensible pero efectivo para la predicción. Además, su entrenamiento es rápido, fácil de explicar a personas no técnicas y simple de implementar en cualquier lenguaje de programación. De hecho, LR suele ser de las primeras opciones a utilizar en tareas de regresión.

Asimismo, LR [38] funciona mediante la suma numérica de variables agregando un número constante, llamado sesgo. El sesgo representa la línea base de predicción cuando todas las variables tienen valores de cero. La fórmula general para una regresión lineal es la siguiente:

$$y = ax + b$$

En la expresión anterior, y es el vector de los valores de respuesta. Por ejemplo, los precios de las casas en una ciudad o las ventas de un producto. La x establece la matriz de variables a utilizar para predecir el vector y. La a representa el sesgo (una constante) y b representa un vector de coeficientes que un modelo de regresión lineal utiliza con el sesgo para generar la predicción.

En la Figura 2.10 se muestra el objetivo de un modelo de LR con fines de estimación [1]. La gráfica muestra que el modelo involucra dos parámetros fundamentales, b que se interpreta como un costo fijo incluido siempre como una condición inicial, y a que representa el costo variable de acuerdo con la cantidad de tamaño.

Existen diversas formas de generar una LR, entre ellas la más común es la LR por mínimos cuadrados ordinarios, la cual ajusta un modelo con coeficientes para minimizar la suma residual de cuadrados entre los objetivos observados y los objetivos predichos por la aproximación lineal. Sin embargo, existen otro tipo de LR que imponen una penalización sobre el tamaño de los coeficientes que minimizan la suma residual de los cuadrados o aquellas que estiman coeficientes dispersos [39].

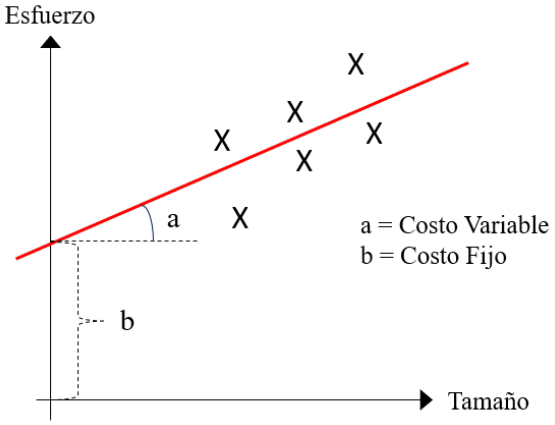

Figura 2.10: Modelo de regresión lineal con costos variables y fijos (adaptada de[[1])

2.2.3.2. Máquinas de vectores de soporte

Las máquinas de vectores de soporte (Support Vector Machines - SVM) [40] son un enfoque para el modelado predictivo basado en el aprendizaje por errores. SVM puede ser aplicado a problemas de clasificación y regresión. En términos matemáticos, se pueden definir comenzando con una fórmula de perceptrón y aplicar ciertas restricciones para tratar de obtener una línea (o hiperplano) de separación. Extendiendo la fórmula del perceptrón, la siguiente expresión sería verdadera para cada muestra i:

$$y(xTw + b) \geq M$$

La expresión anterior describe lo siguiente:

- La multiplicación (vector-vector) de la transpuesta del vector de variables x por un vector de coeficiente w se suma con un sesgo constante b.
- M es una restricción que representa el margen, este es positivo y el mayor valor posible para asegurar la mejor separación de las clases predichas.
- Entendiendo la expresión como una fórmula de distancia, M naturalmente se convierte en el margen.
- Como y puede ser solo -1 o 1, entonces puede tener un valor mayor o igual a 0 cuando las operaciones entre paréntesis don del mismo signo que y.
- El objetivo de la optimización se convierte en encontrar los pesos w que predicen correctamente cada muestra utilizando el mayor valor de M posible.

La Figura 2.11 muestra 4 gráficas de dispersión. Los datos linealmente separables se muestran en el cuadro A y los cuadros B, C y D muestran posibles líneas válidas

que separan las dos clases. ¿Cuál es la mejor? ¿Son B y C mejores que D? El objetivo es encontrar la línea (margen) que separe de la mejor manera las clases de los puntos. Los puntos más cercanos a la línea de separación se conocen como vectores de soporte. SVM maximiza la distancia desde la línea de separación hasta dichos vectores.

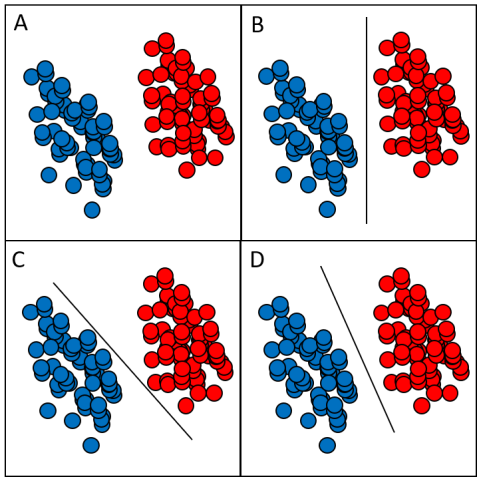

Figura 2.11: Ejemplo de máquinas de vectores de soporte

Las ventajas y desventajas que tiene SVM son las siguientes [41]:

- Ventajas:
  - Tienen un bajo error de generalización
  - Son computacionalmente económicos
  - Los resultados de aplicarlos son fáciles de interpretar
- Desventajas:
  - Son sensibles a los hiperparámetros y la elección del kernel (más información en la sección 4.4.2).
  - De forma nativa, solo maneja la clasificación binaria.

2.2.3.3. Árboles de decisión

Los árboles de decisión (Decision Tree - DT) [41] son una de las técnicas más utilizadas en aprendizaje de máquinas. Los DT tienen bloques de decisión (rectángulos) y bloques de terminación (óvalos) que indican cuándo se ha llegado a una conclusión. Las flechas derecha e izquierda que salen de los bloques de decisión se conocen como ramas y pueden conducir a otros bloques de decisión o a un bloque de terminación.

La Figura 2.12 muestra un diagrama de flujo que representa a un árbol de decisión. Se puede observar que el primer rectángulo comienza con “Peso > 150g”, si este es falso

pasa a un bloque de terminación donde la fruta es una uva, pero si es verdadero pasa al rectángulo de la izquierda y valida si la textura es rugosa, si lo es entonces es una naranja, sino es una manzana.

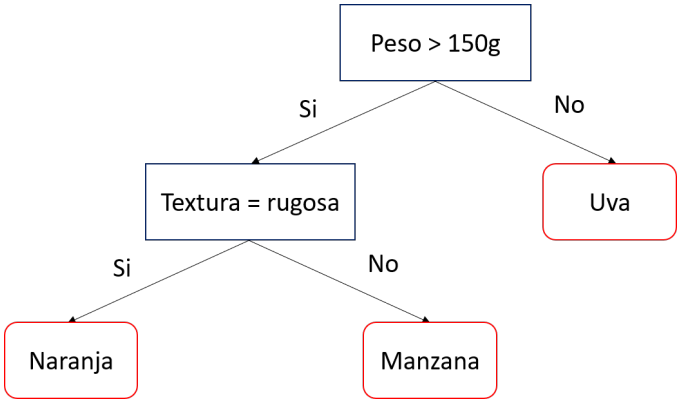

Figura 2.12: Ejemplo de árboles de decisión

Computacionalmente, los DT no utilizan muchos recursos y son fáciles de representar de manera gráfica. Sin embargo, son propensos al sobreajuste.

2.2.4. Modelos basados en redes neuronales

2.2.4.1. Redes neuronales artificiales

Las redes neuronales artificiales (Artificial Neuronal Networks - ANN) [42] se pueden interpretar como máquinas inteligentes que funcionan de manera similar al cerebro animal. Estas aprenden de la experiencia y rápidamente pueden resolver problemas complejos.

> *Definición.* Una red neuronal [43] es una conexión de nodos de procesamiento (red), cuya funcionalidad se basa en la transmisión de señales. La capacidad de procesamiento de la red se almacena en los pesos de conexión entre nodos obtenidos mediante un proceso de aprendizaje.

Las ANN pueden resolver diferentes tipos de problemas, entre los cuales están las tareas de generalización, optimización, reducción de datos, control y predicción en diversos escenarios. La elección del tipo de red depende del tipo de problema que se desea resolver. Actualmente, la red de propagación hacia atrás (Back Propagation Network - BPN) y la red de mapas autoorganizados (Self-Organizing Maps - SOM) son de las redes más utilizadas [42].

2.2.4.2. Neurona artificial

La neurona artificial [44] es aquella que intenta modelar el comportamiento de una neurona del cerebro humano. Asimismo, es la unidad con la que se construyen ANN. La Figura 2.13 muestra el modelo de una neurona artificial.

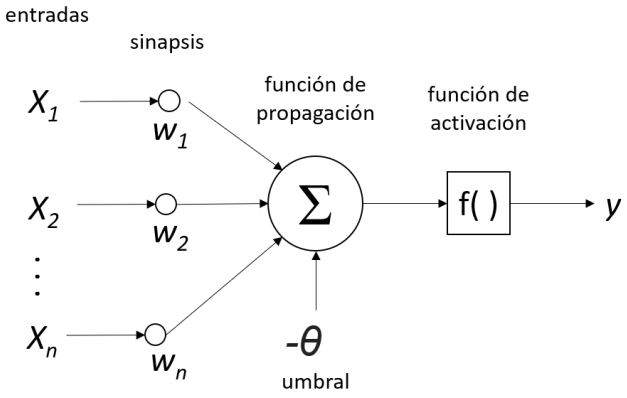

Figura 2.13: Modelo de una neurona artificial

Sus elementos son las siguientes:

- Conjunto de entradas x_i y pesos sinápticos w_i.
- Función de propagación $\sum$ definida a partir del conjunto de entradas y los pesos sinápticos.
- Función de activación $f()$ que representa la salida de la neurona y el estado de activación.
- Umbral $-\theta$ que representa el grado de inhibición de la neurona. Este puede modificar el resultado o imponer un límite al propagarse a otra neurona.

2.2.4.3. Perceptrón

El perceptrón multicapa [45] es un tipo de ANN formada por múltiples capas. El perceptrón es la base de este tipo de ANN. Visto desde el funcionamiento de una neurona narutal, las entradas del perceptrón actúan como señales de entrada hacia la neurona, la función de activación decide si se dispara o no la señal eléctrica y la salida, al activarse, envía información hacia otras neuronas.

La Figura 2.14 muestra un ejemplo de una ANN. Cada nodo representa una neurona y las flechas representan las conexiones entre ellas.

El proceso de aprendizaje de una ANN consiste en modificar los pesos de la red tomando en cuenta las entradas. Durante este proceso, se crean, se modifican o se eliminan conexiones.

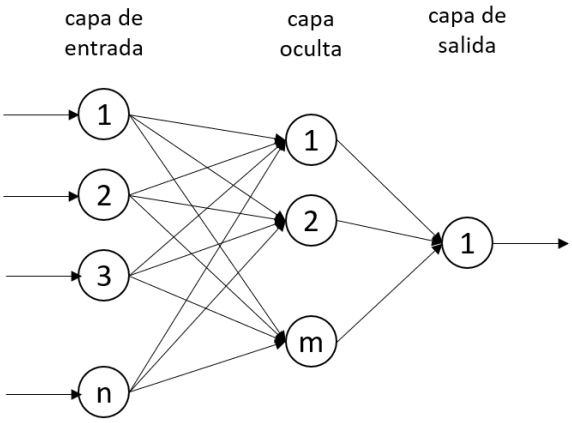

Figura 2.14: Ejemplo de ANN con su capa de entrada, oculta y de salida

2.2.5. Desarrollo de un modelo de predicción utilizando aprendizaje de máquinas

2.2.5.1. Recolección de datos

Cuando se trabaja utilizando aprendizaje de máquinas, lo más recomendable es experimentar con datos del mundo real para poder obtener resultados significativos. Un conjunto de datos se puede definir como una matriz, donde cada fila describe una muestra del mundo real y cada columna describe un único atributo de estas [46]. La Tabla 2.2 muestra un ejemplo de un conjunto de datos de frutas, indicando el nombre, color, diámetro y peso de estas.

Fruta	Color	Diámetro	Peso
Manzana	Rojo	70mm	252g
Limón	Amarillo	55mm	203g
Uva	Verde	27mm	9g
Manzana	Verde	66mm	241g
Fresa	Rojo	34mm	45g
Pera	Verde	61mm	97g

Tabla 2.2: Ejemplo de un conjunto de datos de frutas

La tarea principal en aprendizaje de máquinas es seleccionar un algoritmo de aprendizaje y entrenarlo con algunos datos. No obstante, las dos cosas que pueden salir mal son elegir un algoritmo o datos incorrectos. De hecho, se necesitan de muchos datos para que la mayoría de los algoritmos funcionen correctamente, incluso para problemas muy simples. Por ello, es crucial que los datos utilizados sean representativos para el análisis de resultados que se pretenda realizar.

2.2.5.2. Preprocesamiento de datos

La aplicación de los algoritmos de aprendizaje de máquinas requiere que los datos estén en un formato factible a través de un preprocesamiento de los datos. Estas técnicas incluyen proyección de datos, tratamiento de datos faltantes, normalización, codificación one-hot y reducción de datos.

La mayoría de los algoritmos trabajan mejor con números en lugar de texto. La *proyección de datos* consiste en convertir las variables categóricas de una representación textual a numérica [47]. La Tabla 2.3 muestra un ejemplo donde las variables categóricas Fruta y Color (de la Tabla 2.2) pasaron a una representación numérica. La asignación fue Manzana=0, Limón=1, Uva=2, Fresa=3, Pera=4 y Rojo=0, Amarillo=1, Verde=2.

Fruta	Color	Diámetro	Peso
0	0	70mm	252g
1	1	55mm	203g
2	2	27mm	9g
0	2	66mm	241g
3	0	34mm	45g
4	1	61mm	97g

Tabla 2.3: Proyección de datos aplicada al conjunto de datos de frutas

Si el conjunto de datos está lleno de errores, valores atípicos o ruido, esto causará que sea más difícil para los algoritmos detectar los patrones subyacentes, por lo que es menos probable que funcione correctamente. A menudo, vale la pena invertir tiempo limpiando los datos. Algunos ejemplos de esto son los siguientes [35]:

- Si existen muestras que son claramente atípicas, puede ser útil simplemente descartarlas o tratar de corregir los errores manualmente.
- Si en muestras faltan algunas variables, se debe decidir si ignorar la variable, ignorar la muestra, completar los valores faltantes (*tratamiento de datos faltantes*) o construir un modelo con la variable y uno sin ella.

Por otra parte, una transformación importante que se puede aplicar a las variables continuas es la *normalización* [48]. En general, los algoritmos no funcionan bien cuando las variables numéricas de entrada tienen escalas muy diferentes. Esta técnica transforma la escala de la distribución de una variable. En otras palabras, elimina las unidades de medida permitiendo comparar elementos de distintas variables y distintas unidades.

La Tabla 2.4 muestra el resultado de normalizar las variables Diámetro y Peso del conjunto de datos de frutas. Podemos observar que ahora estas variables están en un rango de 0 a 1 y sus respectivas unidades de medida fueron removidas.

De igual importancia, los algoritmos suponen que dos valores cercanos son más "parecidos"que dos distantes. Por ejemplo, 2 y 3 son más "parecidos"que 2 y 4, pero

Fruta	Color	Diámetro	Peso
0	0	1	1
1	1	0.65	0.79
2	2	0	0
0	2	0.9	0.95
3	0	0.16	0.15
4	1	0.79	0.36

Tabla 2.4: Normalización aplicada al conjunto de datos de frutas

esto no pasa con las variables categóricas, ya que valor numérico solo representa una asignación. Para este problema, una solución común es crear una variable binaria por categoría, asignarle a la muestra 1 cuando cumple la categoría y 0 en otro caso. Esto se conoce como one-hot encoding [35].

La Tabla 2.5 muestra el resultado de aplicar one-hot encoding al conjunto de datos de frutas. Se puede observar que se creó una variable por cada valor distinto que existía en Fruta y Color, y se eliminaron estas dos variables.

Manzana	Limón	Uva	Fresa	Pera	Rojo	Amarillo	Verde	Diámetro	Peso
1	0	0	0	0	1	0	0	1	1
0	1	0	0	0	0	1	0	0.65	0.79
0	0	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0.9	0.95
0	0	0	1	0	1	0	0	0.16	0.15
0	0	0	0	1	0	0	1	0.79	0.36

Tabla 2.5: Aplicación de one-hot encoding al conjunto de datos de frutas

Por último, la reducción de datos tiene como objetivo reducir el tamaño de las entradas de datos mediante la selección de variables [47]. La precisión de los algoritmos depende de que los datos tengan suficientes variables relevantes. Por lo tanto, es muy importante crear un conjunto de variables significativas para entrenar los algoritmos. Este proceso implica:

- Selección de variables: seleccionar las variables más útiles entre las existentes.
- Extracción de variables: combinar variables existentes para producir una más útil.
- Crear nuevas variables mediante la recopilación de nuevos datos.

2.2.5.3. Selección de algoritmos

Elegir correctamente un algoritmo permite generar un modelo capaz de realizar predicciones óptimas. La *validación cruzada* evalúa el rendimiento de generalización de un algoritmo por medio de k particiones de los datos. Al final, devuelve k resultados donde se puede analizar la media y desviación estándar, permitiendo verificar que el algoritmo no esté sobreajustado [35].

Generalmente, la selección de hiperparámetros (parámetros o argumentos de entrada de los algoritmos) tiene un efecto significativo en el éxito de los algoritmos. Un algoritmo mal configurado puede funcionar de manera poco eficiente, mientras que uno bien configurado puede llegar a lograr una precisión de predicción óptima [49]. La estrategia denominada *búsqueda en rejilla* evalúa todas las combinaciones posibles de valores de hiperparámetros mediante validación cruzada. De esta manera, se puede identificar el modelo con las mejores predicciones y los mejores hiperparámetros [47].

2.2.5.4. Entrenamiento y evaluación de modelos

Entrenar un algoritmo y probarlo con los mismos datos es un error metodológico, ya que un modelo que simplemente repita los valores de las muestras que acaba de analizar tendría una precisión de predicción óptima, pero los resultados no serían significativos. Este caso se le conoce como sobreajuste. Para evitarlo, una práctica común cuando se aplica aprendizaje supervisado es tener los datos divididos en dos subconjuntos; el primero se utiliza para el entrenamiento y el segundo para la evaluación del modelo [39].

Las métricas de calidad para evaluación de errores son esenciales para validar la confiabilidad y precisión de los modelos. Estas métricas evalúan el error de predicción para fines específicos. Algunas de las más utilizadas son las métricas de clasificación, métricas de regresión y métricas de agrupamiento [39].

Para tareas de clasificación, lo más común es evaluar la precisión de un modelo obteniendo la proporción del número de predicciones correctas. Por ejemplo, si un clasificador obtuvo 4 de 5 predicciones correctas entonces decimos que tiene un nivel de precisión de 0.8.

Por otra parte, el objetivo del agrupamiento es asociar objetos similares en el mismo grupo y objetos diferentes ubicarlos en otro. Por consiguiente, las métricas de validación usualmente están basadas en cohesión y separación. La primera valida que el miembro de cada grupo debe ser lo más cercano posible a los otros del mismo grupo, mientras que la segunda valida que los grupos estén ampliamente separados entre ellos.

En particular, para la evaluación de modelos de estimación de software se utilizan métricas de regresión, debido a que estos intentan estimar una variable continua.

Entre las métricas de regresión más utilizadas para evaluación de errores están las siguientes [1] [40][50]:

- Coeficiente de determinación R^2: Indica la proporción de la suma total de los cuadrados de la variable dependiente por las variables independientes en el modelo.

$$R^2 = \frac{SS_R}{SS_T}$$

donde SST es el total de la suma de los cuadrados y SSR es la regresión de la suma de los cuadrados. Es decir:

$$\frac{(n\sum xy - (\sum x)(\sum y))^2}{\left(n(\sum x^2) - (\sum x)^2\right)\left(n(\sum y^2) - (\sum y)^2\right)}$$

Un valor de R^2 cerca de 1 indica que existe una fuerte relación entre la variable independiente y la variable dependiente. Mientras que uno cercano a 0 estaría hablando de un modelo poco adecuado, pues la variabilidad del error sería muy grande.

- Error Absoluto (Absolute Error - AE): Mide la diferencia absoluta entre el valor real y la predicción. Si y'_i es la predicción de la muestra e y_i es el valor real correspondiente, entonces el error absoluto se define de la siguiente manera:

$$AE = |y_i - y'_i|$$

El mejor valor que puede tener AE es 0.

- Error Absoluto Medio (Mean Absolute Error - MAE): Indica la media de un conjunto de AE. Entonces, entonces el MAE estimado sobre n muestras se define de la siguiente manera:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y'_i|}{n}$$

El mejor valor que puede tener MAE es 0.

- Raíz del Error Cuadrático Medio (Root Mean Square Error - RMSE): Representa la raíz cuadrada del error cuadrático medio sobre n muestras. Entonces la $RMSE$ estimada se define de la siguiente manera:

$$RMSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - y'_i)^2$$

El mejor valor que puede tener $RMSE$ es 0.

- Magnitud del Error Relativo (Magnitude of relative error - MRE): Indica la divergencia entre los valores estimados por el modelo y los valores reales, expresados en porcentaje. MRE corresponde al valor absoluto de la diferencia entre el valor real y_i y el valor estimado y'_i dividido por el valor real y_i:

$$MRE = \left| \frac{y_i - y'_i}{y_i} \right|$$

El mejor valor que puede tener MRE es 0.

- Magnitud Media del Error Relativo (Mean Magnitude of Relative Error - MMRE): Indica la media de un conjunto de MRE. Entonces, entonces el $MMRE$ estimado sobre n muestras se define de la siguiente manera:

$$MMRE = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - y_i'|}{y_i}$$

El mejor valor que puede tener $MMRE$ es 0.

- Nivel de predicción (Prediction level - PRED): Indica la proporción de muestras que son menores o iguales a un porcentaje establecido (comúnmente 25 %) de MRE:

$$PRED\,(l) = \frac{k}{n}$$

donde k es el número de muestras en el conjunto de datos de tamaño n para los cuales su $MRE \leq l$.

El mejor valor que puede tener $PRED$ es 0.

Capítulo 3

Estado del arte

3.1. La realidad de la estimación de proyectos de software en los últimos años

3.1.1. Tendencias de éxito de proyectos de software

En la industria del software se han desarrollado un amplio número de prácticas para estimar proyectos. Sin embargo, muchas de ellas han dado resultados poco favorables, que representan pérdidas para las empresas. La Figura 3.1 muestra las tendencias de éxito de proyectos de software de los últimos años [8]. Se puede observar que en la actualidad ocurre lo siguiente con los proyectos:

- Alrededor del 50 % son subestimados, es decir, se invierte más de lo estimado.
- Cerca del 30 % son exitosos, terminando los proyectos de acuerdo con lo planeado.
- Cerda del 20 % son fracasados, es decir, estos fueron cancelados o no se utilizaron.

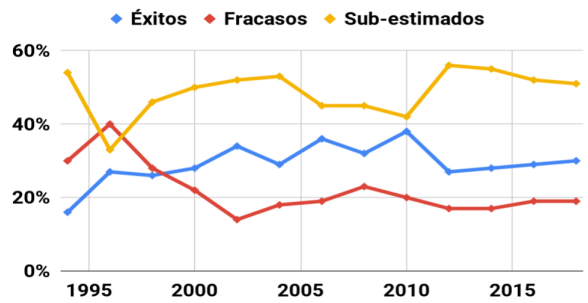

Figura 3.1: Tendencias de éxito de proyectos de software

3.1.2. La estimación de software en México

La Ingeniería de Software, como ya se ha mencionado, tiene un gran impacto en vida cotidiana de las personas. En México, se ha logrado crecer la industria de desarrollo de software, que en gran medida depende de la investigación y la educación en Ingeniería de Software [51]. Sin embargo, durante los últimos años México ha perdido posiciones en el aprovechamiento de tecnologías de la información, en parte debido a la falta de infraestructura [52].

La adopción de modelos de calidad en la industria de software tiene una relación importante entre el nivel de madurez y la mejora en las variables financieras y operativas de las empresas.

En la actualidad, la Asociación Mexicana de Métricas de Software (AMMS) [53] se encarga de promover, recolectar y difundir métricas para la Industria Mexicana de Desarrollo de Software (IMDS). La intención es definir métricas que cumplan con las siguientes características:

- Básicas: basadas en estándares internacionales.
- Transversales: útiles para todos los actores (usuarios, desarrolladores, etc.).
- Trascendentes: permiten la comparación a través del tiempo.

En 2018, la AMMS realizó el Estudio Línea Base de Productividad y Costo de la IMDS [54]. Este estudio proporciona una referencia de modelos formales fomentando las mejores prácticas de estimación. Asimismo, el estudio analiza una pequeña cantidad de datos (que son con los que se cuenta), mostrando que existe un alto porcentaje de proyectos fracasados en la IMDS.

Por otra parte, el análisis realizado en el estudio de la AMMS revela cuales son las variables que contribuyen, en mayor medida, a determinar el esfuerzo y costo necesarios para desarrollar los proyectos. Sin embargo, este estudio es solo una primera referencia, aún falta mucho por realizar.

3.2. Los métodos de estimación más utilizados

3.2.1. Clasificación de métodos de estimación

En la actualidad, el software se puede desarrollar siguiendo diversas metodologías. Factores como el tamaño del equipo, el nivel de madurez de la empresa, la experiencia de los miembros del equipo, el tipo de proyecto a desarrollar, etc., están directamente relacionados con el contexto de desarrollo del proyecto. Por ello, la forma de estimar un proyecto de puede variar dependiendo del tipo de desarrollo adoptado [55].

Una gran cantidad de métodos de estimación han sido propuestos por investigadores desde el inicio de la estimación de software como área de investigación. Los métodos

abarcan desde juicio de experto hasta aquellos compuestos por algoritmos complejos. Estos pueden utilizarse de forma aislada o combinados (híbridos) para aumentar su eficacia y precisión.

Los primeros enfoques para estimar el esfuerzo de desarrollo de software aparecieron a finales de la década de los 60, los cuales estaban basados fundamentalmente en la valoración de expertos [56]. En estos casos, un "experto" (persona con el conocimiento y experiencia suficiente) proponía una estimación del esfuerzo necesario para desarrollar el proyecto con base en su experiencia. Posteriormente, aparecieron los modelos formales tales como COCOMO y análisis de puntos de función (Function Point Analysis - FPA) [57].

Existen varios esquemas de clasificación de métodos de estimación de proyectos de software. La Figura 3.2 muestra una adaptación de la clasificación sugerida por Sehra et al. [58]. En esta se incluyen los métodos de juicio de experto, modelos formales e híbridos de los dos anteriores.

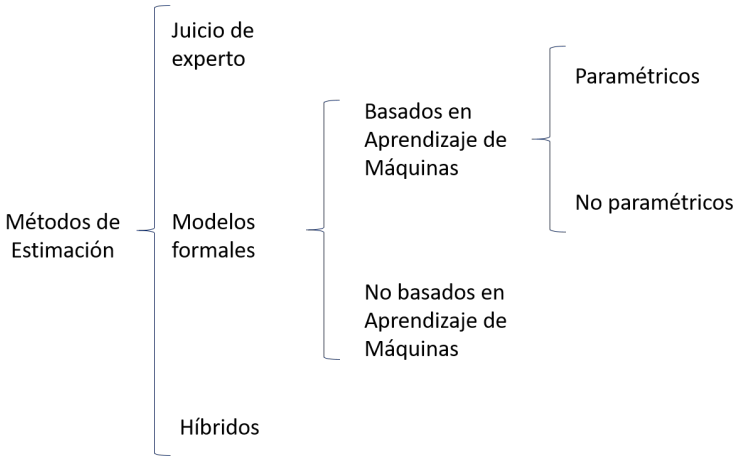

Figura 3.2: Clasificación de los métodos de estimación

Dentro de los modelos formales, están los basados en aprendizaje de máquinas, estos pueden ser paramétricos que disponen de un conjunto de datos completo o no paramétricos, que permiten trabajar con datos incompletos pero son sensibles a la selección de los parámetros.

Los modelos formales que no se basan en aprendizaje de máquinas, generalmente utilizan modelos matemáticos construidos utilizando datos de proyectos terminados. Estos modelos suelen utilizar fórmulas derivadas empíricamente para estimar el esfuerzo. Por ejemplo, una función de líneas de código o puntos de función de alguna métrica estandarizada (IFPUG, COSMIC, etc.). Un modelo de este tipo utiliza un análisis de regresión sobre los datos históricos de proyectos de software.

3.3. Estimación de software con aprendizaje de máquinas

3.3.1. Técnicas de aprendizaje de máquinas utilizadas en la estimación de software

En los últimos años, modelos basados en Inteligencia Artificial, fundamentalmente en técnicas de aprendizaje de máquinas, han sido utilizados para mejorar la precisión de las estimaciones. Estos modelos se basan en la aplicación de diferentes técnicas de extracción de conocimiento, con el objetivo de realizar estimaciones con mayor precisión [59].

Técnicas de aprendizaje supervisado son las que se utilizan con mayor frecuencia para la generación de modelos de estimación, cuya superioridad frente a otro tipo de métodos ha sido demostrada en numerosos estudios [3][60]. Sin embargo, estas técnicas no solo se utilizan en la estimación, sino también para predicción de defectos, complejidad, facilidad de reutilización, etc.

De igual importancia, los algoritmos de aprendizaje de máquinas más utilizados incluyen SVM, ANN y algoritmos lineales. Sin embargo, modelos basados en lógica difusa y algoritmos genéticos también han sido muy utilizados [3][61].

3.3.2. Estudios sobre la estimación de software con aprendizaje de máquinas

La Tabla 3.1 muestra un resumen de la literatura sobre la aplicación de enfoques basados en la estimación del esfuerzo de software que incluyen técnicas de aprendizaje de máquinas [61]. Se puede observar que desde los primeros estudios (de hace más de 20 años) ya se comenzaban a utilizar técnicas basadas en Inteligencia Artificial.

Los datos que se han utilizado con mayor frecuencia en recientes estudios se obtienen principalmente de la base de datos internacional gestionada por el International Software Benchmarking Standards Group (ISBSG) [26]. Anteriormente, los estudios realizados solían utilizar bases de datos incluidas en el repositorio PROMISE Software Engineering Repository [2].

Hoy en día, congresos relacionados a la Ingeniería de Software publican artículos sobre la estimación de proyectos de software con aprendizaje de máquinas [62][63].

No obstante, la estimación del esfuerzo de desarrollo de software sigue recayendo sobre los expertos, siendo las técnicas basadas en juicio de experto las más utilizadas en la actualidad [55]. Una razón de esto puede deberse a la recién popularidad del desarrollo ágil sobre las metodologías tradicionales, donde la estimación comúnmente se basa en estimaciones individuales de varios expertos [64].

Año	Autor	Título	Conferencia / Revista	Técnica
1995	Srinivasan K, Fisher D	Machine Learning Approaches to Estimating Software Development Efforts	IEEE Transactions on Software Engineering	ANN, CART
1997	Shepperd N. Schofield C	Estimating Software Project Efforts Using Analogies	IEEE Transactions on Software Engineering	OLS, Case based reasoning
1997	Witting G, Finnie G	Estimating Software Development Efforts with Connectionist Models	Information and Software Technology	ANN
2001	Burgess C.J., Lefley M	Can Genetic Programming Improve Software Effort Estimation? A Comparative Evaluation	Information and Software Technology	Genetic Algorithm
2002	Essam et.al.	Software Project Effort Estimation Using Genetic Programming	IEEE	Genetic Algorithm
2002	Idri et al.	Estimating Software Project Efforts by Analogy Based on Linguistic Values	Eighth IEEE International Symposium on Software Metrics	Fuzzy Analogy
2003	Huang X, Caretz L.F., and Ren J	A Neuro-Fuzzy Model for Software Cost Estimation	Third International Conference on Quality Software, (QSIC'03)	Neuro-Fuzzy, COCOMO II
2005	Song Q., Shepperd M., and Carolyn M	Using Grey Relational Analysis to Predict Software Efforts with Small Data Sets	IEEE International Software Metrics Symposium	Analogy, Grey Relational Analysis
2005	Bohem et. al.	Feature Subset Selection Can Improve Software Cost Estimation Accuracy	PROMISE'05	Feature Subset Selection (WRAPPER), COCOMO
2005	Sentas P, Angelis L, Stamelos I	Software Productivity and Effort Prediction with Ordinal Regression	Information and Software Technology	OLS regression, Ordinal regression
2006	Sheta A F	Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects	Journal of Computer Science	Genetic Algorithm, COCOMO
2006	Auer M, Trendowicz A, Biffl S, Haunschmid E, Graser B	Optimal Project Feature Weights in Analogy-Based Cost Estimation: Improvement and Limitations	IEEE Transactions on Software Engineering	Case base reasoning
2007	Baskeles B, Turhan B, Bener A	Software Effort Estimation Using Machine Learning Models	22nd International Symposium on Computer and Information Science	RBF, MLP, SVM, Decision Tree
2007	Chiu N, Huang S	The Adjusted Analogy-Based Software Effort Estimation Based on Similarity Distances	The Journal of Systems and Software	OLS regression, ANN, CART, case based reasoning

Año	Autor	Título	Conferencia / Revista	Técnica
2008	Keung J.W, Kitchenham A and Jeffery, D R	Analogy-X: Providing Statistical Inference to Analogy-Based Software Cost Estimation	IEEE Transaction on Software Engineering	Analogy, Jackknife
2008	Huang S J, Chiu N H and Chen L W	Integration of Grey Relational Analysis with Genetic Algorithm for Software Effort Estimation	European Journal of Operational Research, Elsevier	Grey Relational, Genetic Algorithm
2008	Kiran et.al.	Software Development Cost Estimation Using Wavelet Neural Networks	The Journal of Systems and Software	Wavelet Neural Network
2008	Azzeh M, Neagu D, and Cowling P	Improving Analogy Software Effort Estimation Using the Fuzzy Feature Subset Selection Algorithm	PROMISE'08	Feature Subset Selection, Fuzzy Algorithm
2010	Azzeh M, Neagu D and Cowling P	Fuzzy Grey Relational Analysis for Software Effort Estimation	Empir Software Eng, Springer	Analogy, Fuzzy Grey Relational
2010	Azzeh M, Neagu D and Cowling P	Analogy–Based Software Effort Estimation Using Fuzzy Numbers	The Journal of Systems and Software	Analogy, Fuzzy Numbers
2010	Chaudhary K	GA Based Optimization of Software Development Effort Estimation	IJCSI	SEL model, Walston- Felix model, COCOMO, Genetic Algorithm
2011	Hari et. al.	CPNA Hybrid Model for Software Cost Estimation	IEEE Explorer	COCOMO, PSO, Neural Network
2011	Song and Shepperd et al.	Predicting Software Project Effort: A Grey Relational Analysis Based Method	Expert System with Applications	Analogy, Fuzzy Numbers

Tabla 3.1: Resumen de la literatura sobre la aplicación de enfoques basados en la estimación del esfuerzo de software hasta el año 2011

Capítulo 4

Definición del sistema propuesto

4.1. Introducción del sistema

4.1.1. Descripción del sistema

En este capítulo se define el Sistema de Generación de Modelos de Estimación Automatizado (**A**utomated **E**stimation-Model **G**enerat**i**on **S**ystem - AEGiS) que con el uso de algoritmos de aprendizaje de máquinas genera un conjunto de modelos de estimación con los criterios de caldiad más precisos posibles (al preprocesamiento de datos aplicado). La principal contribución del sistema es que automatiza técnicas de preprocesamiento de datos, entrenamiento y evaluación de modelos, lo que permite generar de una manera relativamente simple modelos de estimación donde se conoce con exactitud el nivel de variabilidad de las predicciones.

La Figura 4.1 muestra el diagrama de flujo básico del funcionamiento principal de *AEGiS*. Lo primero que el sistema hace al recibir una base de datos de proyectos de software son las tareas relacionadas con el preprocesamiento de datos, lo siguiente es la selección de hiperparámetros de los algoritmos utilizando validación cruzada y por último el entrenamiento y evaluación de modelos.

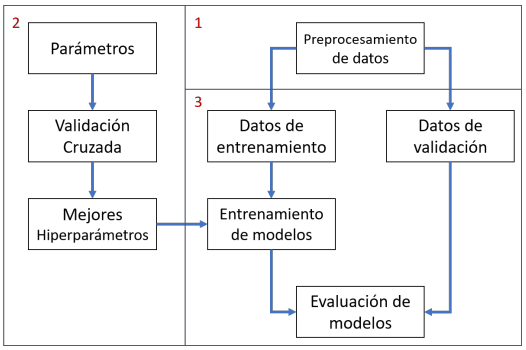

Figura 4.1: Diagrama de flujo de funcionamiento principal de *AEGiS*

4.2. Análisis de base de datos y herramientas de desarrollo

4.2.1. Descripción de los datos

Una de las tareas más importante antes de construir un modelo basado en aprendizaje de máquinas es comprender los datos con los que se está trabajando y cómo se relacionan con el problema que desea resolver. Es poco eficiente elegir aleatoriamente un algoritmo y entrenarlo con los datos, es necesario comprender lo que sucede con el conjunto de datos antes de comenzar a construir un modelo [36].

Para el desarrollo de esta sección se utiliza una base de datos de ejemplo que contiene 10 proyectos de software. Los resultados que se obtienen solo son de caracter ilustrativo. Para resultados con datos reales, revisar el Capítulo 5.

4.2.2. Herramientas de desarrollo

Python se ha convertido en un lenguaje de programación por defecto para muchas aplicaciones de ciencia de datos, ya que combina el poder de lenguajes de programación de propósito general con la facilidad de uso de lenguajes de dominio específico. Además, tiene bibliotecas para la carga de datos, visualización, estadísticas, procesamiento de lenguaje natural, procesamiento de imágenes, entre otras. Una de las principales ventajas de usar Python es la capacidad de interactuar directamente con el código, usando una terminal u otras herramientas como Jupyter Notebook [36].

Asimismo, Sklearn es una herramienta construida sobre librerías de Python, es simple y eficiente para el análisis predictivo de datos. También es reutilizable en diversos contextos, es de código abierto y comercialmente utilizable [39].

Por otra parte, Tensorflow es de las bibliotecas principales de código abierto para desarrollar y entrenar modelos de aprendizaje de máquinas, especialmente utilizado para ANN. Además, cuenta con varias herramientas flexibles, bibliotecas y recursos comunitarios que fácilmente permiten crear y desplegar aplicaciones [65].

4.3. Automatización del preprocesamiento de datos

4.3.1. Por qué automatizar el preprocesamiento de datos

Uno de los objetivos principales es automatizar el preprocesamiento de datos, ya que es una tarea donde se suele invertir bastante tiempo y recursos. En lugar de hacerlo manualmente, se pueden definir funciones para automatizarlo. Estas funciones tienen los siguientes beneficios y características [35]:

- Permiten realizar de manera automática preprocesamiento de datos en cualquier conjunto de datos.

- Se pueden reutilizar fácilmente en otros proyectos.
- Permiten crear varios modelos a la vez y poder comparar sus resultados.

Las siguientes subsecciones definen las funciones implementadas que utiliza el *AEGiS*. Estas funciones automatizan tareas específicas tanto en el preprocesamiento de datos, como en el entrenamiento y evaluación de modelos. Para cada una de ellas, se define el nombre, argumentos que recibe y valores que retorna la función.

4.3.2. Funciones de preprocesamiento de datos

4.3.2.1. Carga de base de datos

El primer paso para la automatización de modelos de estimación es cargar una base de datos de proyectos de software. Esta base de datos debe cumplir los siguientes requisitos mínimos:

- Estar en un formato *CSV* separado por comas
- El primer registro (primera fila) debe contener el nombre de las variables de los proyectos
- Contener la variable que se desea estimar (como esfuerzo, costo, tiempo, etc.)
- Contener al menos 30 proyectos

El último requisito es muy recomendable para generar modelos estadísticamente válidos. Sin embargo, no es indispensable, dependerá de la cantidad de datos con los que se cuenta y del nivel de confiabilidad que se desee obtener con los modelos de estimación.

Para cargar una base de datos, se define una función *import_db* que asigna a una variable *projects* la matriz de proyectos contenidos en la base de datos.

```
def import_db(database):
    ...
    return projects
```

Argumentos:	- **database**: archivo csv de la base de datos
Retornos:	- **projects**: matriz que contiene todos los proyectos de la base de datos

La Tabla 4.1 muestra 10 proyectos de software de una base de datos de ejemplo. Se puede observar que le falta información en algunos registros, representados con el caracter "?". Por ejemplo, para el primer proyecto la variable *Arquitectura* tiene un valor faltante.

Tipo de desarrollo	Tipo de aplicación	Arquitectura	Lenguaje de programación	Método de medición	Tamaño funcional	Esfuerzo
Nuevo desarrollo	Aplicación de negocios	?	Java2EE	IFPUG+4	1443	12987
Nuevo desarrollo	?	Cliente / Servidor	Javascript	IFPUG+4	961	11820
?	?	Cliente / Servidor	?	COSMIC	2044	26572
Mantenimiento	Tiempo real	Milticapas	Python	IFPUG+4	321	2728
Nuevo desarrollo	Aplicación de negocios	Cliente / Servidor	C++	FiSMA	1515	12120
Mantenimiento	Aplicación de negocios	Cliente / Servidor	?	COSMIC	432	5365
Mantenimiento	Aplicación de negocios	Milticapas	Java2EE	IFPUG+4	489	6846
Nuevo desarrollo	Tiempo real	Milticapas	Python	IFPUG+4	3006	38500
?	Aplicación de negocios	Cliente / Servidor	PHP	?	?	13849
Nuevo desarrollo	?	Cliente / Servidor	Javascript	COSMIC	1086	14237

Tabla 4.1: Base de datos de ejemplo con 10 proyectos de software

4.3.2.2. Proyección de datos

En la proyección de datos, primero se debe identificar el tipo de cada variable, indicando si es discreta o continua. Después, para las variables discretas se asigna un valor numérico por cada categoría de la variable, de tal manera que todas las cadenas de texto son reemplazadas por números. También se debe identificar la variable objetivo. Se define la función *handling_categorical_data* que realiza las acciones anteriores para convertir todos los datos en valores numéricos:

```
def handling_categorical_data(projects):
    ...
    return projects_hcd, var_types, discrete_vars
```

La Tabla 4.2 muestra la matriz de proyectos después de aplicar la proyección de datos. Se puede observar que la matriz ahora solo contiene números (ceros y unos para variables discretas) y valores de "?" en el caso de valores faltantes.

4.3.2.3. Tratamiento de datos faltantes

Para un funcionamiento adecuado de los algoritmos, las variables no pueden tener datos faltantes. Cuando esto ocurre, se los datos se pueden reemplazar o se puede eliminar la variable si tiene un alto porcentaje de datos faltantes. Para lo primero, se obtiene la media, la moda y el porcentaje de valores faltantes de cada variable. Se define

Argumentos:	- **projects**: matriz que contiene todos los proyectos de la base de datos
Retornos:	- **projects_hcd**: matriz de proyectos aplicada la proyección de datos - **discrete_vars**: diccionario que contiene la asignación numérica del valor de cada variable - **var_types**: diccionario que contiene el tipo de cada variable

Tipo de desarrollo	Tipo de aplicación	Arquitectura	Lenguaje de programación	Método de medición	Tamaño funcional	Esfuerzo
0	0	?	0	0	1443	12987
0	?	1	1	0	961	11820
?	?	2	?	1	2044	26572
2	2	2	3	0	321	2728
0	0	1	4	2	1515	12120
2	0	1	?	1	432	5365
2	0	2	0	0	489	6846
0	2	2	3	0	3006	38500
?	0	1	5	?	?	13849
0	?	1	1	1	1086	14237

Tabla 4.2: Matriz de proyectos después de aplicar la proyección de datos

la función *missing_values* que calcula el porcentaje de valores faltantes por variable y, dependiendo de este valor, realiza las siguientes acciones por variable:

- $< 33.33\,\%$ asigna la media para variables continuas y la moda para discretas
- $\geq 33.33\,\%$ y $\leq 66.66\,\%$ asigna la media para variables continuas y asigna una nueva categoría (por ejemplo "otros") a las variables discretas
- $> 66.66\,\%$ descarta la variable de los proyectos

```
def missing_values(projects_hcd, discrete_vars):
    ...
    return projects_mv
```

La Tabla 4.3 muestra la matriz de proyectos después de aplicar el tratamiento de datos faltantes. De esta manera, la matriz únicamente tiene números para representar la información de la base de datos.

Argumentos:	- **projects_hcd**: matriz de proyectos aplicada la proyección de datos - **discrete_vars**: diccionario que contiene la asignación numérica del valor de cada variable
Retornos:	- **projects_mv**: matriz de proyectos aplicado el tratamiento de datos faltantes

Tipo de desarrollo	Tipo de aplicación	Arquitectura	Lenguaje de programación	Método de medición	Tamaño funcional	Esfuerzo
0	0	1	0	0	1443	12987
0	0	1	1	0	961	11820
0	0	2	0	1	2044	26572
2	2	2	3	0	321	2728
0	0	1	4	2	1515	12120
2	0	1	0	1	432	5365
2	0	2	0	0	489	6846
0	2	2	3	0	3006	38500
0	0	1	5	0	1259	13849
0	0	1	1	1	1086	14237

Tabla 4.3: Matriz de proyectos después de aplicar el tratamiento de datos faltantes

4.3.2.4. Normalización

La normalización de variables continuas no toma en cuenta la variable objetivo, ya que es la variable que se desea predecir. En la matriz de proyectos la variable objetivo es *Esfuerzo*. Se define una función *normalize_continuous_vars* que normaliza las variables continuas, transformando todos los valores en un rango de 0 a 1. La variable *Esfuerzo* no fue normalizada ya que es la variable objetivo.

```
def normalize_continuous_vars(projects_mv, var_types):
    ...
    return projects_ncv
```

La Tabla 4.4 muestra la matriz de proyectos después de aplicar la normalización de variables continuas. De esta manera, las variables continuas (tamaño funcional) de la matriz están en un rango de 0 a 1.

La normalización de variables continuas y one-hot encoding se pueden aplicar en cualquier orden, no importa cual se aplica primero.

Argumentos:	- **projects_mv**: matriz de proyectos aplicado el tratamiento de datos faltantes - **var_types**: diccionario que contiene el tipo de cada variable
Retornos:	- **projects_ncv**: matriz de proyectos aplicada la normalización de variables continuas

Tipo de desarrollo	Tipo de aplicación	Arquitectura	Lenguaje de programación	Método de medición	Tamaño funcional	Esfuerzo
0	0	1	0	0	0.418	12987
0	0	1	1	0	0.238	11820
0	0	2	0	1	0.642	26572
2	2	2	3	0	0	2728
0	0	1	4	2	0.445	12120
2	0	1	0	1	0.041	5365
2	0	2	0	0	0.063	6846
0	2	2	3	0	1	38500
0	0	1	5	0	0.349	13849
0	0	1	1	1	0.285	14237

Tabla 4.4: Matriz de proyectos después de aplicar la normalización de variables continuas

4.3.2.5. Codificación one-hot

Para aplicar one-hot encoding se define la función *one_hot_encoding* que genera tantas nuevas variables como valores distintos de variables discretas.

```
def one_hot_encoding(project_mv, discrete_vars):
    ...
    return projects_ohe
```

Argumentos:	- **projects_mv**: matriz de proyectos aplicado el tratamiento de datos faltantes - **discrete_vars**: diccionario que contiene la asignación numérica del valor de cada variable
Retornos:	- **projects_ohe**: matriz de proyectos aplicado one-hot encoding

La Tabla 4.5 muestra algunas variables (de las 14 nuevas) de la matriz de proyectos

después de aplicar one-hot encoding. Se puede observar que se generó una nueva variable por cada valor distinto de las variables discretas. Estas nuevas variables solo pueden tener un valor de 0 y 1. Las variables discretas que se tenían anteriormente fueron eliminadas.

Tipo de desarrollo (Nuevo desarrollo)	Tipo de desarrollo (Mantenimiento)	...	Método de medición (COSMIC)	Método de medición (FiSMA)	Tamaño funcional	Esfuerzo
1	0	...	0	0	0.418	12987
1	0	...	0	0	0.238	11820
1	0	...	1	0	0.642	26572
0	1	...	0	0	0	2728
1	0	...	0	1	0.445	12120
0	1	...	1	0	0.041	5365
0	1	...	0	0	0.063	6846
1	0	...	0	0	1	38500
1	0	...	0	0	0.349	13849
1	0	...	1	0	0.285	14237

Tabla 4.5: Matriz de proyectos después de aplicar la normalización de variables continuas

4.3.2.6. Reducción de datos

Para seleccionar las variables que formarán parte de los modelos de estimación, es recomendable identificar aquellas con mayor correlación a la variable objetivo. Esta correlación se puede identificar antes o después de la normalización y one-hot encoding. Se define la función *var_correlations* que calcula las relaciones lineales entre las variables.

```
def var_correlations(projects_mv):
    ...
    return var_correlations
```

Argumentos:	- **projects_mv**: matriz de proyectos aplicado el tratamiento de datos faltantes
Retornos:	- **var_correlations**: arreglo con el coeficiente de correlación entre la variable objetivo y el resto de las variables

La Tabla 4.6 muestra la correlación entre *Esfuerzo* y el restode las variables. Se puede observar que la variable con mayor correlación al es *Tamaño funcional*.

Al identificar la correlación de las variables se puede aplicar lo siguiente:

Tipo de desarrollo	Tipo de aplicación	Arquitectura	Lenguaje de programación	Método de medición	Tamaño funcional
-0.618	0.303	0.336	0.116	-0.031	0.967

Tabla 4.6: Correlación entre Esfuerzo y el resto de las variables

- Generar modelos de estimación seleccionando las variables con mayor correlación.
- Eliminar las variables con la menor correlación.
- Seleccionar los proyectos más parecidos (en valores de las variables) a los proyectos que se desean estimar, tomando en cuenta los dos puntos anteriores.

Para revisar cómo se aplica la reducción de datos para generar modelos de estimación revisar la sección 5.3.

4.4. Automatización de entrenamiento y evaluación de modelos

4.4.1. Por qué automatizar el entrenamiento y evaluación de modelos

Como ya se ha mencionado anteriormente, la tarea donde se suele invertir más tiempo es en el preprocesamiento de datos. Sin embargo, seleccionar adecuadamente los algoritmos es de vital importancia para generar resultados significativos.

Contar con funciones que automaticen tareas de entrenamiento y validación, además de ahorrar tiempo y recursos, permite fácilmente entrenar un gran número de algoritmos con la finalidad de comparar cuáles de ellos obtienen los resultados más precisos.

4.4.2. Validación cruzada y selección de hiperparámetros

Antes de comenzar a entrenar modelos, es necesario asegurarse que estos no tiendan al sobre ajuste. Por ello, se utiliza la función *cross_val_score* de sklearn que recibe como argumentos un algoritmo, la matriz de proyectos y el número de particiones (5 por ejemplo) y retorna la validación cruzada. En esta validación se puede analizar principalmente la media y desviación estándar de la precisión del modelo. Una manera fácil para determinar el sobre ajuste es si se tiene una desviación estándar muy grande.

Por otra parte, para la identificación de mejores hiperparámetros de los algoritmos también se utiliza una función de sklearn, *fit* de *GridSearchCV* que recibe los mismos parámetros que *cross_val_score* y retorna el estimador del algoritmo con los mejores hiperparámetros encontrados.

Para automatizar la validación cruzada y la selección de mejores hiperparámetros se define la función *best_hyperparameters*, la cuál con la ayuda de *cross_val_score* y *fit* obtiene los mejores hiperparámetros de un conjunto de algoritmos:

```
def best_hyperparameters(projects_prep, algorithms):
    ...
    return hyperparameters
```

Argumentos:	- **projects_prep**: matriz de proyectos aplicado el preprocesamiento de datos - **algorithms**: arreglo que contiene los algoritmos que serán procesados
Retornos:	- **hyperparameters**: diccionario que contiene los mejores hiperparámetros de cada algoritmo

El conjunto de algoritmos que se utiliza para el entrenamiento y evaluación de modelos consta de los siguientes:

- Bayesian Ridge Regression (Bayesian-RR)
- Decision Tree Regressor (DTR)
- Elastic Net (eNet)
- Gradient Boosting Regressor (GBR)
- Linear Regression (LR)
- Multivariate Adaptive Regression Splines (MARS)
- Random Forest Regressor (RFR)
- Regresión Lasso (LR-Lasso)
- Regresión Ridge (LR-Ridge)
- Stochastic Gradient Descent Regressor (SGDR)
- Support Vector Regression (SVR)

4.4.3. Entrenamiento de modelos

El primer paso para el entrenamiento de los modelos es dividir la base de datos en dos; el conjunto de entrenamiento y el de validación. Para esto, se utiliza la función de sklearn *train_test_split* que recibe la matriz de proyectos (dividida en variables de entrenamiento y variable objetivo) junto con el porcentaje de datos que serán utilizados para la validación (por ejemplo 20 %) y retorna los siguientes conjuntos de datos:

- X_train: datos de entrenamiento de modelos
- X_test: datos de evaluación de modelos
- y_train: etiquetas de todos los datos en X_train
- y_test: etiquetas de todos los datos en y_train

Lo siguiente es entrenar los modelos con los respectivos conjuntos de datos. Para esto, se define la función *model_training* que utiliza los conjuntos de datos resultantes de *train_test_split*:

```
def model_training(X_train, y_train, hyperparameters):
    ...
    return trained_estimators
```

Argumentos:	- **X_train**: datos de entrenamiento de modelos - **y_train**: etiquetas de todos los datos en X_train - **hyperparameters**: diccionario que contiene los mejores hiperparámetros de cada algoritmo
Retornos:	- **trained_estimators**: diccionario que contiene los estimadores entrenados

4.4.4. Evaluación de modelos

Para la evaluación de un modelo de estimación se necesita generar un conjunto de predicciones realizadas con el sistema, para posteriormente comparar las predicciones con los datos reales. Para esto se define la función *model_evaluation* que realiza lo anterior y valida cada modelo con los criterios de calidad definidos en la sección 2.2.5.4:

```
def model_evaluation(X_test, y_test, trained_estimators):
    ...
    return estimator_evaluation
```

Argumentos:	- **X_test**: datos de evaluación de modelos - **y_test**: etiquetas de todos los datos en X_test - **trained_estimators**: diccionario que contiene todos los estimadores entrenados
Retornos:	- **estimator_evaluation**: diccionario que contiene todos los criterios de calidad de cada estimador

La Tabla 4.7 muestra los criterios de calidad resultantes de entrenar y evaluar modelos utilizando la matriz de proyectos de la subsección anterior (4.3.2) con los siguientes algoritmos:

Model	MAE	RMSE	MMRE	SDMRE	PRED
Bayesian-RR	***561***	***613***	0.087	0.041	***100 %***
DTR	819	925	0.062	0.048	***100 %***
eNet	655	778	0.043	0.049	***100 %***
GBR	683	709	0.051	0.018	***100 %***
LR	1567	1868	0.089	0.012	***100 %***
LR-Lasso	2070	2386	0.126	0.005	***100 %***
LR-Ridge	1194	1442	0.143	***0.003***	***100 %***
MARS	2067	2387	0.125	0.004	***100 %***
RFR	1027	1027	0.080	0.006	***100 %***
SGDR	1005	1006	0.055	0.028	***100 %***
SVR	710	727	***0.040***	0.010	***100 %***

Tabla 4.7: Criterios de calidad de los modelos de estimación

Se puede observar que el modelo *Bayesian-RR* obtuvo 2 de los criterios de calidad más precisos ($MAE = 561$ y $RMSE = 613$), mientras que *SVR* y *LR-Ridge* obtuvieron uno ($MMRE = 0.04$ y $SDMRE = 0.003$ respectivamente). Por otra parte todos los modelos obtuvieron el 100 % de precisión para el criterio de *PRED*.

4.4.5. Selección de mejores modelos de estimación

En general, se entiende por "mejor modelo" aquel modelo que en su conjunto tiene los criterios de calidad más precisos. En la Tabla 4.7 se puede observar que no existe un modelo que tenga todos los criterios más precisos. Por ello, se deben normalizar los criterios para poder compararlos entre si, ya que tienen escalas distintas.

Una normalización típica tiene la siguiente fórmula [66]:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

El concepto de razón (*ratio*) se emplea cuando se quiere definir una medida o norma para expresar los datos [67]. Para calcularla, se divide la cantidad que se quiere normalizar por cantidad normalizadora. Para comparar los criterios de calidad se calcula la razón de cada uno de la siguiente manera:

- si el mejor valor que puede tener el criterio de calidad es 0 (*MAE, RMSE, MMRE y SDMRE*):

$$X_{ratio} = \frac{X_{min}}{X}$$

- si el mejor valor que puede tener el criterio de calidad es 1 (*PRED*):

$$X_{ratio} = \frac{X}{X_{max}}$$

Después, se suma la razón de los criterios de calidad de cada modelo, el modelo que tenga la suma más alta será el modelo de estimación más preciso. Al ser evaluados con 5 criterios de calidad, la suma máxima que puede tener un modelo de estimación es 5, lo cual significa que todos sus criterios de calidad son los más precisos.

Se define la función *normalize_quality_criteria* que normaliza los criterios de calidad de un conjunto de modelos de estimación.

```
def normalize_quality_criteria(estimator_evaluation):
    ...
    return quality_criteria_normalized
```

Argumentos:	- **estimator_evaluation**: diccionario que contiene todos los criterios de calidad de cada estimador
Retornos:	- **quality_criteria_normalized**: diccionario que contiene todos los criterios de calidad normalizados y la suma de las razones de cada estimador

La Figura 4.2 muestra la suma de las razones de cada modelo (una vez normalizados) de manera gráfica. Se puede observar que *SVR* tiene los criterios de calidad más precisos, seguido de *eNet*. Por otra parte, *LR* obtuvo los criterios menos precisos, seguido de *LR-Lasso*.

Cabe mencionar que las razones calculadas solo son válidas para los criterios de calidad que se están comparando, por lo que sería incorrecto compararlas con otras razones de criterios de calidad de otros modelos.

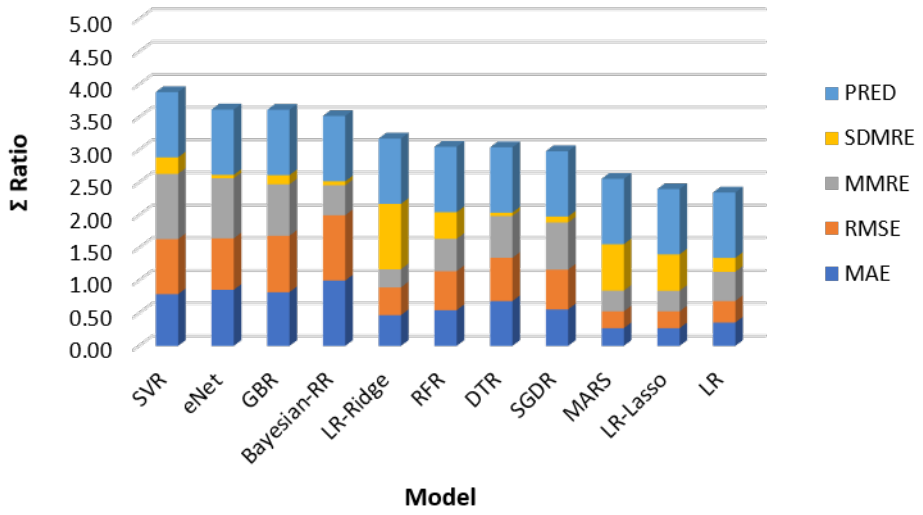

Figura 4.2: Suma de razones de cada modelo de la base de datos de ejemplo

4.5. Funcionamiento de AEGiS

4.5.1. Funcionamiento del sistema

En las secciones anteriores de este capítulo se definió un conjunto de funciones que automatizan el preprocesamiento de datos, entrenamiento y evaluación de modelos de estimación, las cuales son utilizadas por *AEGiS* mediante dos funciones principales, *data_preprocessing* y *model_generation*.

Se define la función *data_preprocessing* que recibe como argumento la base de datos de proyectos de software y se encarga de realizar todas las operaciones de preprocesamiento de datos. De manera general, llama a funciones previamente definidas generando 4 conjuntos de datos (en lugar de solo 1), los cuales son todas las combinaciones posibles de aplicar normalización de variables continuas y one-hot encoding, esto con el fin de analizar si mejora la precisión de los modelos aplicando dichas técnicas.

```
def data_preprocessing(database):
    # import database
    projects = import_db(database)

    # data preprocessing
    projects_hcd, var_types, discrete_vars =
                        handling_categorical_data(projects)
    projects_mv = missing_values(projects_hcd, discrete_vars)
    dataset = {
```

```
        'projects': projects_mv,
        'projects_ncv': normalize_continuous_vars(projects_mv,
                                                  var_types)
        'projects_ohe': one_hot_encoding(project_mv,
                                         discrete_vars)
        'projects_ncv_ohe': one_hot_encoding(project_ncv,
                                             discrete_vars)
    }

    return dataset
```

Argumentos:	- **database**: ruta del archivo csv de la base de datos
Retornos:	- **dataset**: conjuntos de datos preprocesados

Por otra parte, *model_generation* recibe los conjuntos de datos preprocesados y genera un conjunto de modelos de estimación con los criterios de calidad más precisos posibles (con los algoritmos seleccionados y el preprocesamiento de datos aplicado). De manera general, itera sobre los conjuntos de datos llamando a funciones previamente definidas para identificar los mejores hyperparámetros, entrenar, evaluar e identificar los mejores modelos de estimación.

```
def model_generation(dataset):
    # algorithm selection
    algorithms = {
        'Bayesian-RR': BayesianRidge(),
        'DTR': DecisionTreeRegressor(),
        ...
    }

    # model training and model evaluation
    for d in dataset:
        X_train, X_test, y_train, y_test =
                                        train_test_split(...)

        hyperparameters[d] = best_hyperparameters(dataset[d],
                                                  algorithms)

        models[d] = model_training(X_train, y_train,
                                   hyperparameters[d])

        estimators[d] = model_evaluation(X_test, y_test,
                                         models[m])
```

```
        automated_estimators[d] =
                    normalize_quality_criteria(estimators[d])

    return automated_estimators
```

Argumentos:	- **dataset**: conjuntos de datos preprocesados
Retornos:	- **automated_estimators**: diccionario que contiene los estimadores, sus criterios de calidad y la suma de sus razones

El código completo de las funciones del sistema *AEGiS* se puede consultar en el repositorio de Github https://github.com/ivansaavedra/, tomando en cuenta que es posible que existan diferencias en el código debido a las actualizaciones que pueda tener el sistema.

4.5.2. Elección de modelos y estimación

Elegir adecuadamente el modelo de estimación que se debe utilizar para estimar un proyecto de software es una tarea que debe considerar varios factores. Algunos de estos factores se describen a continuación:

- Modelo con la mayor suma de razones: los modelos que se han desarrollado a lo largo de este capítulo mostraron que *SVR* es el que tiene los criterios de calidad más precisos. Elegir entre los modelos con los mejores criterios de calidad es un buen comienzo.
- Peso de los criterios de calidad: al calcular la suma de las razones de cada criterio de calidad, se le asignó un peso equivalente de 20 % (por ser 5 criterios) a cada criterio de calidad. Sin embargo, dependiendo de los criterios a los que se desee dar mayor preferencia, los pesos pueden ser modificados. Por ejemplo, una organización busca tener siempre el *MAE* más bajo posible, entonces puede asignar un peso de 40 % a este criterio y 15 % a los demás. La suma de razones será distinta a la suma considerando todos los criterios con un peso de 20 %.
- Los n mejores modelos: al obtener la suma de las razones, puede ser posible tener 2 o más modelos con una precisión similar. Si esto ocurre, se puede utilizar más de un modelo para estimar un proyecto. La manera de hacerlo es calcular la estimación individual de cada modelo (de los n) y después obtener el promedio de las estimaciones. Esto es similar a *Voting Regressor* de sklearn [39].
- Juicio de experto: dentro del proceso de estimación, existe una fase donde se debe tomar en cuenta información no incluida en las entradas del modelo de estimación. Comunmente, un "experto" en estimación evalúa dicha información y toma una

decisión sobre el presupuesto del proyecto. Por ejemplo, si existen varios factores de riesgo que afectan al proyecto entonces puede ser necesario aumentar el costo del proyecto para mitigar los riesgos.

Capítulo 5

Resultados experimentales

5.1. Introducción

5.1.1. Descripción de los casos de estudio

Este capítulo presenta 5 casos de estudio que analizan los resultados de utilizar *AEGiS* con el fin de verificar su eficiencia en la estimación de proyectos de software. El resumen de los casos de estudio se describe a continuación:

1. Caso de estudio 1. En la literatura, existe un amplio número de métodos para estimar proyectos de software. En este caso de estudio se analiza y compara la eficiencia de estos métodos con los modelos generados con *AEGiS*.

2. Caso de estudio 2. Al generar modelos de estimación, puede ser necesario identificar más de un modelo para el mismo conjunto de datos, esto puede ser necesario cuando no se pueden modelar de la mejor manera los datos de entrada. Este caso de estudio analiza la eficiencia de utilizar particiones de la base de datos para generar modelos de estimación utilizando una base de datos interna de una organización.

3. Caso de estudio 3. Existe un amplio número de métodos para dimensionar un proyecto de software, entre los cuales destacan los estándares de medición de tamaño funcional (FSMM - Functional Size Measurement Methods). En este caso de estudio se analiza eficiencia de los métodos de medición en la estimación de proyectos de software.

4. Caso de estudio 4. El caso de estudio anterior muestra la importancia de contar con una variable de tamaño para generar modelos de estimación, ¿cómo podemos estimar si no contamos con esta variable? Este caso de estudio analiza la precisión que pueden llegar a tener los modelos que no toman en cuenta variables de tamaño.

5. Caso de estudio 5. Las ANN han sido utilizadas para la estimación de software. Sin embargo, *AEGiS* no las incluye en la generación de modelos debido al cuidadoso

análisis y la cantidad de procesamiento que pueden requerir. Este caso de estudio describe el entrenamiento de una ANN y sus resultados se comparan con los modelos de *AEGiS*.

5.1.2. Descripción de la base de datos

Los modelos generados en estos casos de estudio utilizan principalmente el repositorio ISBSG D&E Repository 2019 R1 [26], el cual cuenta con 9,178 proyectos de la industria de desarrollo de software a nivel internacional incluyendo proyectos de más de 30 países. Estos proyectos están bajo las siguientes características:

- Tipo de desarrollo: el 67.6 % son mantenimientos, el 30.6 % nuevos desarrollos y 1.1 % son proyectos que se desarrollaron nuevamente.
- Sector industrial: los sectores principales son comunicaciones, seguros y banca, los cuales cubren más del 50 % del total de proyectos.
- Marketing: el 91.6 % de los proyectos se desarrollaron para uso interno de la organización, mientras que el 8.1 % para uso externo. Por otra parte, el 34.5 % fue desarrollo propio y el 65.2 % por outsourcing.
- Tipo de aplicaciones: principalmente, el 90.5 % son aplicaciones de negocios y el 4.5 % son de tiempo real.
- Metodología de desarrollo: el 79.2 % de los proyectos utilizaron un modelo en cascada. Otras metodologías incluyen Agile y/o RUP (11.4 %).
- Arquitectura: principalmente, el 35.9 % de los proyectos tiene una arquitectura cliente-servidor y el 28 % multicapas.
- Tamaño funcional: el 73.5 % de los proyectos fueron medidos utilizando el método de medición IFPUG 4+, el 8 COSMIC, el 6.8 % NESMA y el 6.3 % FiSMA.
- Métodos de estimación: principalmente, el 54.5 % de los proyectos fueron estimados con métodos de estimación basados en el desglose de tareas, seguido de los métodos que se basan en el tamaño funcional. Los métodos de estimación que contiene la base de datos ISBSG serán referenciados a lo largo de este capítulo como "métodos de estimación tradicionales".

5.2. Caso de estudio 1: Comparación entre los métodos tradicionales y modelos de estimación automatizados

5.2.1. Descripción y análisis de los datos

En este caso de estudio se utiliza la base de datos ISBSG que cuenta con 922 proyectos con información en las variables *Effort estimate* y *Effort estimate method*. Estas variables son necesarias para este estudio ya que el objetivo es comparar la precisión entre los métodos de estimación tradicionales con los modelos generados con *AEGiS* (modelos automatizados).

Los proyectos incluidos en este estudio cuentan con 66 variables, entre las cuales están la variable objetivo *Normalised Work Effort* (relacionada con el esfuerzo real de desarrollo del proyecto) y las siguientes 3 variables que tienen la mayor correlación con esta variable objetivo:

- 0.52 - Max Team Size
- 0.24 - Functional Size
- 0.22 - Added count

Se puede observar que las variables con la mayor correlación a la variable objetivo son aquellas basadas en el tamaño del equipo y el tamaño funcional. Cabe resaltar que estas variables son continuas al igual que la variable objetivo.

5.2.2. Criterios de calidad de métodos de estimación tradicionales

Cuando se habla de métodos de estimación tradicionales es importante aclarar cuáles son estos métodos. La Tabla 5.1 muestra los métodos más utilizados de la base de datos ISBSG. Se puede observar que los métodos que predominan son los que se basan en el desglose de tareas (54.5 %), seguido de los basados en el tamaño funcional (9.5 %). Por lo tanto, estos dos métodos representan un 64.2 % de los proyectos. Sin embargo, en la base de datos se pueden identificar proyectos que fueron estimados utilizando ambos métodos, representando el 8.9 %. Por lo que, en su conjunto, el uso de estos métodos representa el 72.9 % de todos los métodos.

De los dos métodos anteriores, los que se basan en el desglose de tareas entran en la clasificación de descomposición, ya que hacen un análisis de las características (o tareas) del proyecto para obtener estimaciones individuales sobre los componentes del mismo. Por otra parte, los métodos basados en el tamaño funcional deberían entrar en la clasificación de modelos algorítmicos, los cuales aplican técnicas que identifican los factores clave que contribuyen al esfuerzo del proyecto y generan un modelo de algorítmico.

Variable	%
Task based & breakdown	54.5 %
FP & FSM based	9.5 %
GSMS Development+	8.9 %
FP & Task based*	8.7 %
SLC Roadmap	5.7 %
Last approved plan by the client	4 %
In house	2.7 %
System Life Cycle	1.3 %
Other project example	1.1 %
Calculated from person-day estimate	0.8 %

Tabla 5.1: Métodos de estimación tradicionales más utilizados en la base de datos ISBSG

Lo siguiente es obtener sus criterios de calidad utilizando el esfuerzo estimado y el esfuerzo real del proyecto. La Tabla 5.2 muestra los criterios de calidad de 5 métodos; 4 son los métodos tradicionales más utilizados y 1 que utiliza todos los métodos (All methods), es decir, utiliza todos los 922 proyectos. Se puede observar que la combinación de *Task based & FP based* obtuvo 4 de los 5 criterios de calidad más altos (*MAE, RMSE, MMRE y SDMRE*), mientras que *All methods* obtuvo solo 1 (*PRED*).

Method	MAE	RMSE	MMRE	SDMRE	PRED
Task & FP based	**1103**	**2112**	**0.282**	**0.251**	52 %
FP & FSM based	1774	3823	0.429	0.714	52 %
Task based & breakdown	1636	4253	0.422	1.913	59 %
All methods	2412	9626	0.322	1.105	**62 %**
GSMS Development+	5693	20270	0.529	1.100	47 %

Tabla 5.2: Criterios de calidad de métodos de estimación tradicionales

5.2.3. Criterios de calidad de modelos de estimación automatizados

Al tener automatizada la generación de modelos de estimación con *AEGiS*, solo es necesario pasarle como parámetro la base de datos, este se encarga de todo el preprocesamiento, generación y evaluación de modelos de estimación. La Tabla 5.3 muestra los 5 modelos que obtuvieron los mejores criterios de calidad. Podemos observar que el modelo entrenado con *LR* obtuvo 3 de los criterios de calidad más precisos (*MAE, MMRE y PRED*), mientras que el modelo entrenado con *LR-Lasso* obtuvo solo 2 (*RMSE y SDMRE*).

Model	MAE	RMSE	MMRE	SDMRE	PRED
LR	**891**	2168	**0.292**	0.846	**76 %**
LR-Lasso	896	**1371**	0.308	**0.267**	58 %
RFR	2429	7870	0.398	0.647	62 %
DTR	3167	10057	0.399	0.538	57 %
GBR	2574	7271	0.633	1.160	51 %

Tabla 5.3: Criterios de calidad de modelos de estimación automatizados

5.2.4. Comparación de criterios de calidad entre métodos tradicionales y modelos automatizados

A simple vista se puede observar que no existe ningún método/modelo que tenga todos los criterios más precisos. Por ello, se calcula la razón de todos los criterios, con el fin de poder comparar y analizar cuáles son los más precisos. La Figura 5.1 muestra de manera gráfica la suma de las razones de los criterios de calidad por cada método/modelo de estimación.

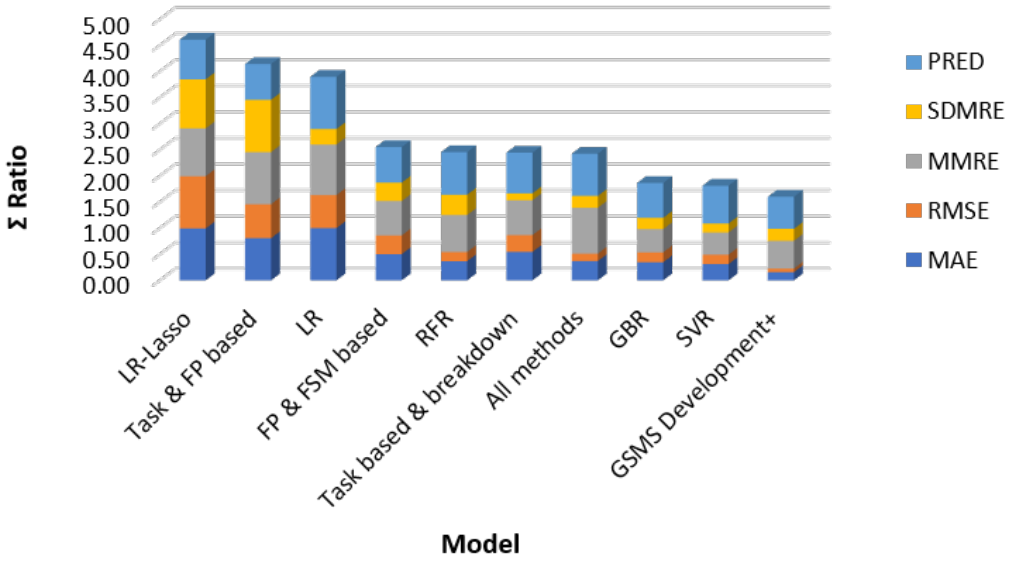

Figura 5.1: Suma de razones de los criterios de calidad de modelos automatizados y métodos tradicionales.

La gráfica muestra lo siguiente:

- El modelo entrenado con *LR-Lasso* es el más eficiente de los modelos automatizados y de los métodos tradicionales.
- El método tradicional más preciso es *Task & FP based*, el cual ocupa el segundo lugar de métodos/modelos más eficientes.
- El método *GSMS Development +* es el ménos eficiente de todos.
- El modelo entrenado con *SVR* es el modelo automatizado menos eficiente, ocupa el noveno lugar.
- El método tradicional más utilizado (*Task based & breakdown*) es de los métodos menos eficientes, ocupa el séxto lugar.
- De los 5 métodos/modelos más eficientes, 3 son modelos automatizados y 2 son métodos tradicionales.

Se puede concluir que *AEGiS* genera modelos de estimación eficientes, ya que el modelo entrenado con *LR-Lasso* (el modelo automatizado más preciso) fue mejor que todos los métodos de estimación tradicionales. La Figura 5.2 muestra el porcentaje de mejora de este modelo (de acuerdo con la razón calculada) respecto a los métodos tradicionales. Se puede observar que se obtuvo una mejora del 11 % respecto al método tradicional más preciso (*Task & FP based*) y una mejora del 90 % tomando en cuenta todos los métodos.

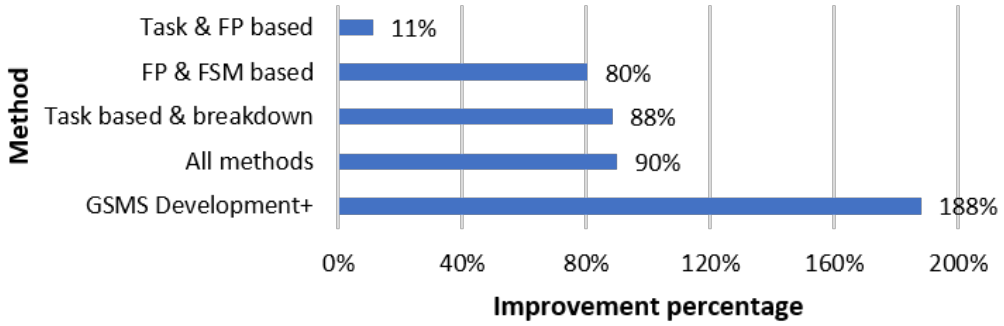

Figura 5.2: Porcentaje de mejora de *LR-Lasso* respecto a los métodos de estimación tradicionales

5.3. Caso de estudio 2: Eficiencia de utilizar particiones de la base de datos en modelos automatizados

5.3.1. Descripción y análisis de los datos

En este caso de estudio se utilizan dos bases de datos, la de ISBSG (externa) y una base de datos de una organización (interna) que contiene 216 proyectos con información en 15 variables, entre las cuales están la variable objetivo *Effort* (relacionada con el esfuerzo del proyecto) y las siguientes 3 variables que tienen la mayor correlación con esta variable objetivo:

- 0.98 - Functional Size
- 0.23 - Relative Size
- 0.04 - Primary Programming Language

Como ya se ha mencionado anteriormente, el tamaño funcional es una variable muy importante en la estimación de proyectos, en este caso, con una correlación de 0.98. El tamaño funcional de los proyectos de la base de datos interna fue dimensionado utilizando el método de medición de IFPUG.

Por otra parte, se entiende por *partición* a una división de una base de datos, de tal manera que se pueden obtener subconjuntos (particiones) de datos de la misma base de datos.

De este modo, el análisis de este caso de estudio consiste en generar modelos de estimación automatizados utilizando particiones de la base de datos externa para estimar proyectos de la base de datos interna. Esto con el fin de analizar si la eficiencia de los modelos de estimación automatizados puede llegar a mejorar utilizando particiones de la base de datos.

5.3.2. Generación de modelos de estimación automatizados utilizando particiones de la base de datos externa

5.3.2.1. Descripción de las particiones

A diferencia del caso de estudio anterior, el preprocesamiento de datos aplicado en este estudio incluye hacer particiones de los datos, ya que los proyectos de la base de datos interna contiene variables (en el contexto de desarrollo) que tienen el mismo valor. Por ejemplo, todos los proyectos fueron medidos con IFPUG y el sistema operativo principal fue Windows. Estas particiones de los datos permiten analizar si es posible generar modelos de estimación más eficientes.

Para seleccionar las particiones, es recomendable analizar la correlación que tienen las variables de los proyectos con la variable objetivo. Las siguientes variables categóricas son las que tienen la mayor correlación con el esfuerzo del proyecto:

- 0.16 - Relative Size (RS)
- 0.09 - Count Approach (CA)
- 0.06 - 1st Operating System (1OS)
- 0.02 - Languaje Type (LT)

La partición de la base de datos para la generación de modelos automatizados se llevará a cabo de la siguiente manera:

1. Los primeros modelos de estimación que se deben generar son los que consideran todos los proyectos de la base de datos externa, es decir, considerar toda la base sin ninguna partición (*NoP - No Partition*).
2. El siguiente paso es comenzar con la partición de la base de datos en el orden de las variables con mayor correlación. Por cada partición, se generarán modelos de estimación automatizados. Es importante resaltar que la partición i se hace a partir de la $i-1$.
3. Al final, los criterios de calidad obtenidos de todas las particiones se comparan y se analiza si la eficiencia de los modelos fue mejorando o no.

5.3.2.2. Modelos de estimación automatizados sin partición de los datos

Para calcular los modelos automatizados con los criterios de calidad más precisos, solo es necesario pasarle a *AEGiS* como parámetros la base de datos externa, indicar que genere los modelos con el 100 % de los datos (para el entrenamiento) y que la validación de los modelos la realice con la base de datos interna.

La Tabla 5.4 muestra los 3 modelos que obtuvieron los mejores criterios de calidad. Además, la tabla muestra el nombre de la partición y al final la suma de las razones de los criterios de calidad. Se puede observar que el modelo entrenado con *RFR* obtuvo todos los criterios más precisos, con una suma de $Ratio = 5.0$.

Partition	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
No Partition	RFR	**94**	**151**	**0.486**	**0.848**	**47 %**	**5.00**
No Partition	MARS	760	799	6.176	7.631	3 %	0.57
No Partition	SVR	1173	1214	7.562	7.410	0.5 %	0.39

Tabla 5.4: Criterios de calidad de modelos automatizados sin particiones

Una vez obtenidos los criterios de calidad de modelos sin particiones, el siguiente paso es comenzar con la partición de la base de datos en el orden de las variables discretas con mayor correlación. Por cada partición, se generarán modelos de estimación automatizados.

5.3.2.3. Partición 1: Tamaño relativo

La variable de los proyectos *Relative Size*, para los FSM principales, clasifica el tamaño funcional por tamaños relativos de la siguiente manera:

- $0 \leq XXS < 10$
- $10 \leq XS < 30$
- $30 \leq S < 100$
- $100 \leq M1 < 300$
- $300 \leq M2 < 1000$
- $1,000 \leq L < 3,000$
- $3,000 \leq XL < 9,000$
- $9,000 \leq XXL < 18,000$
- $18,000 \leq XXXL$

El proyecto de la base de datos interna con mayor tamaño funcional es de 417 PF, por lo que esta partición de los datos abarca aquellos proyectos con un tamaño relativo igual o menor que $M2$. Después de la partición, la base de datos externa cuenta con 8,191 proyectos.

La Tabla 5.5 muestra los 3 modelos que obtuvieron los mejores criterios de calidad. Se puede observar que nuevamente el modelo entrenado con *RFR* obtuvo todos los criterios más precisos, con una suma de $Ratio = 5.0$.

Partition	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
Relative Size	RFR	**118**	**165**	**0.729**	**1.147**	**23 %**	**5.00**
Relative Size	DTR	538	930	2.668	3.198	10 %	0.57
Relative Size	LR-Lasso	707	897	3.705	3.722	0.5 %	0.39

Tabla 5.5: Criterios de calidad de modelos automatizados con partición por tamaño relativo

5.3.2.4. Partición 2: Método de medición

La variable *CountApproach* es la que se relaciona con el método de medición de tamaño funcional del proyecto. Este valor puede estar especificado en diferentes unidades dependiendo del método de medición.

La base de datos de ISBSG cuenta con información de proyectos que fueron medidos utilizando cada uno de los FSMM. Sin embargo, el análisis de qué tan eficiente es utilizar cada uno de estos métodos se presenta en otro caso de estudio.

En el caso de los proyectos de la base de datos interna, todos fueron medidos con el método IFPUG, por lo que la siguiente partición de los datos consiste en seleccionar los proyectos que fueron medidos con este método. Después de la partición, la base de datos cuenta con 5,870 proyectos. Es importante resaltar que esta segunda partición se hace a partir de la primera partición, es decir, una vez que hizo la partición de la base de datos por tamaño relativo, se parte nuevamente pero ahora por método de medición.

La Tabla 5.6 muestra los 3 modelos que obtuvieron los mejores criterios de calidad. Se puede observar que una vez más el modelo entrenado con *RFR* obtuvo todos los criterios más precisos, con una suma de $Ratio = 5.0$.

Partition	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
Count Approach	RFR	**177**	**270**	**1.320**	**1.857**	**20 %**	**5.00**
Count Approach	DTR	251	363	2.011	2.819	19 %	3.67
Count Approach	LR-Ridge	463	484	3.787	4.324	4 %	1.92

Tabla 5.6: Criterios de calidad de modelos automatizados con partición por método de medición

5.3.2.5. Partición 3: Sistema operativo

La variable de los proyectos *1st Operating System* es aquella relacionada con el sistema operativo de tecnología principal utilizado para desarrollar el proyecto, es decir, el utilizado para la mayor parte del esfuerzo de construcción. Para la base de datos interna, todos los proyectos fueron desarrollados en *Windows*. Después de la partición correspondiente, la base de datos cuenta con 639 proyectos.

La Tabla 5.7 muestra los 3 modelos que obtuvieron los mejores criterios de calidad. En esta partición, el modelo más preciso fue entrenado con *MARS* obteniendo 4 de los criterios más precisos (*MAE, RMSE, MMRE y SDMRE*), mientras que *DTR* obtuvo 1 de los más precisos (*PRED*).

Partition	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
1st Operating System	MARS	**443**	**472**	**3.054**	**3.394**	0.5 %	**4.03**
1st Operating System	DTR	738	1401	6.610	14.132	**17 %**	2.64
1st Operating System	eNet	889	1133	6.529	9.026	13 %	2.51

Tabla 5.7: Criterios de calidad de modelos automatizados con partición por sistema operativo

5.3.2.6. Partición 4: Tipo de lenguaje

La última partición de los datos se realiza por la variable *Lenguage Type*, la cual define el tipo de lenguaje de desarrollo utilizado para el proyecto, por ejemplo, 3GL, 4GL, etc. Después de aplicar esta partición, la base de datos cuenta con 360 proyectos. Por ello, esta es la última partición que cuenta con una cantidad considerable de proyectos para generar los modelos de estimación.

La Tabla 5.8 muestra los 3 modelos que obtuvieron los mejores criterios de calidad. Se puede observar que el modelo más preciso fue entrenado con *DTR* obteniendo 3 de los criterios más precisos (*MMRE, SDMRE y PRED*), mientras que *SVR* obtuvo 2 de los más precisos (*MAE y RMSE*).

Partition	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
Lenguage Type	DTR	390	1582	**1.042**	**1.655**	**26 %**	**3.99**
Lenguage Type	SVR	**289**	**401**	2.565	4.511	14 %	3.30
Lenguage Type	eNet	553	728	4.370	7.268	3 %	1.64

Tabla 5.8: Criterios de calidad de modelos automatizados con partición por tipo de lenguaje

5.3.2.7. Análisis de eficiencia de las particiones de datos

Una vez calculados los criterios de calidad de los modelos automatizados por cada una de las particiones, el último paso es compararlos. Se puede observar que conforme se hicieron particiones en la base de datos la eficiencia de los modelos de estimación no mejoraba en todos los casos y los algoritmos que entrenaban a los modelos más precisos también iban cambiando.

La Tabla 5.9 muestra un cuadro comparativo respecto al porcentaje de mejora entre los modelos generados sin ninguna partición y todas las particiones realizadas, resaltando en color rojo si el porcentaje de mejora es negativo, es decir, si la precisión de los modelos fue menor.

La tabla muestra lo siguiente:

	NoP	RS	CA	1OS	LT
NoP	-	38 %	113 %	426 %	166.9 %
RS	-28 %	-	54 %	280 %	93 %
CA	-53 %	-35 %	-	147 %	25 %
1OS	-81 %	-74 %	-59 %	-	-49 %
LT	-63 %	-48 %	-20 %	97 %	-

Tabla 5.9: Cuadro comparativo de los porcentajes de mejora entre las particiones de datos

- La fila *NoP* tiene todos los porcentajes positivos, lo que significa que es mejor que todas las particiones.
- La columna *NoP* tiene todos los porcentajes negativos, lo que significa que las demás particiones son menos precisas que esta. Esto significa lo mismo que el punto anterior.
- La fila *1OS* tiene todos los porcentajes negativos, lo que significa que es peor que todas las particiones.
- Los porcentajes del lado de la diagonal izquierda son casi todos negativos (9 de 10), lo que significa que los modelos automatizados iban siendo menos precisos conforme se iban haciendo particiones en la base de datos.
- Los porcentajes del lado de la diagonal derecha son casi todos positivos (9 de 10), lo que significa que los modelos automatizados de la partición i fue más precisa que la $i + 1$.

Con los resultados anteriores, se puede concluir que las particiones de los datos no necesariamente mejoran la precisión de los criterios de calidad para los modelos de estimación automatizados utilizando técnicas de aprendizaje de máquinas. Por ello, es necesario verificar siempre si la precisión mejora o no cuando se hace una partición en la base de datos por alguna variable categórica de los proyectos.

Es importante resaltar que para este estudio se utilizaron 2 bases de datos (interna y externa), los criterios de calidad de las particiones de los datos dependen en gran medida de la información de la base de datos interna.

5.4. Caso de estudio 3: Importancia de las medidas de tamaño en la estimación de software

5.4.1. Descripción y análisis de las medidas de tamaño

Para este caso de estudio únicamente se utiliza la base de datos de ISBSG. Esta base cuenta con la variable *Count Approach* que describe la técnica utilizada para medir el proyecto. La Tabla 5.10 muestra las medidas de tamaño principales, el número de proyectos, número total de variables, las 3 variables con mayor correlación y su correlación con el esfuerzo del proyecto. En general, se puede observar que las variables con mayor correlación al esfuerzo son las relacionadas con el tamaño del proyecto.

Medida	**# Proyectos**	**# Variables**	**Variables con mayor correlación**	
			variable	**correlación**
COSMIC	733	40	Functional Size	0.6
			Added count	0.35
			COSMIC Read	0.25
FiSMA	580	32	Functional Size	0.61
			Data Quality Rating	0.22
			Relative Size	0.19
IFPUG	6744	36	Functional Size	0.4
			Adjusted Function Points	0.36
			Relative Size	0.15
IFPUG OLD	213	72	Adjusted Function Points	0.66
			Functional Size	0.59
			File count	0.58
MARKII	37	58	Development Platform	0.6
			Portability requirements	0.36
			Resource Level	0.35
NESMA	625	43	Functional Size	0.75
			Adjusted Function Points	0.67
			Relative Size	0.29
LOC	186	35	Max Team Size	0.43
			Language Type	0.26
			Team Size Group	0.23
Story Points	53	34	Used Methodology	-0.05
			Development Methodologies	-0.05
			Sprints / iterations	-0.08

Tabla 5.10: Descripción de proyectos y variables de medidas de tamaño

Para la mayoría de los proyectos, este es el método utilizado para medir el tamaño funcional. El tamaño de estos proyectos se especifica principalmente en la variable *Functional Size*. Para proyectos que utilizan otras medidas de tamaño (como *LOC*) los datos se encuentran en otras variables (como *Lines of Code*).

Cabe mencionar que algunos expertos no consideran a *Story Points* como una medida de tamaño, ya que generalmente determina el esfuerzo de una historia de usuario más no un tamaño de esta [68].

5.4.2. Modelos de estimación automatizados por medidas de tamaño

5.4.2.1. COSMIC

La unidad de medida de COSMIC es 1 CFP (COSMIC Function Point) que se define como el tamaño asignado a cada movimiento de datos (Entradas, Salidas, Lecturas y Escrituras) [27]. Para el método COSMIC, en la tabla anterior (3.10) se pueden observar que las variables con mayor correlación al esfuerzo son aquellas relacionadas al tamaño, las cuales son *Functional Size* que está relacionada con los CFP y también *COSMIC Read* relacionada con los movimientos de datos de lecturas.

La Tabla 5.11 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con COSMIC. Se puede observar que el modelo entrenado con *RFR* obtuvo 4 de los criterios de calidad más precisos (*MAE, RMSE, MMRE y PRED*), mientras que *DTR* obtuvo solo 1 (*SDMRE*).

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
COSMIC	RFR	**271**	**424**	**0.16**	0.2	**82 %**	**4.95**
COSMIC	DTR	378	645	0.19	**0.19**	77 %	4.16
COSMIC	GBR	509	947	0.47	0.65	57 %	2.31

Tabla 5.11: Criterios de calidad de modelos automatizados utilizando el método COSMIC

5.4.2.2. FiSMA

La unidad de medida de FiSMA es 1 Ffp (FiSMA Function Point) [69]. El tamaño de una pieza de software es la suma de los tamaños de los componentes funcionales base (BFC) por clase. El tamaño de un BFC depende de su tipo y la cantidad de elementos específicos de clase. Para FiSMA (al igual que con COSMIC), la variable con mayor correlación al esfuerzo es Functional Size determinada por Ffp.

La Tabla 5.12 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con FiSMA. Se puede observar que el modelo entrenado con *RFR* obtuvo 3 de los criterios de calidad más precisos (*MAE, RMSE y PRED*), mientras que *GBR* y *DTR* obtuvieron 1 (*MMRE y SDMRE* respectivamente).

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
FiSMA	RFR	**248**	**526**	0.155	0.345	**88 %**	**4.48**
FiSMA	GBR	426	1276	**0.144**	0.202	86 %	3.93
FiSMA	DTR	539	1265	0.187	**0.192**	78 %	3.53

Tabla 5.12: Criterios de calidad de modelos automatizados utilizando el método FiSMA

5.4.2.3. IFPUG

La unidad de medida de IFPUG es 1 FP (Function Point) [23]. El proceso de medición visto a un nivel de abstracción alto contempla dos partes; el cálculo de puntos de función no Ajustados y el valor de factor de ajuste de los FP. Al final se calculan los puntos de función ajustados multiplicando los dos factores anteriores.

La Tabla 5.13 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con IFPUG. Se puede observar que el modelo entrenado con *RFR* obtuvo 4 de los criterios de calidad más precisos (*MAE, RMSE, MMRE y PRED*), mientras que *DTR* obtuvo solo 1 (*SDMRE*).

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
IFPUG	RFR	**371**	**693**	**0.08**	0.14	**94 %**	**4.86**
IFPUG	DTR	599	1159	0.10	**0.12**	91 %	3.98
IFPUG	GBR	834	1679	0.57	1.38	61 %	1.73

Tabla 5.13: Criterios de calidad de modelos automatizados utilizando el método IFPUG

5.4.2.4. IFPUG OLD

IFPUG OLD incluye aquellos proyectos que fueron medidos utilizando versiones anteriores a la v4.0 del método IFPUG, la cual fue creada en 1994. La última versión de este método es la v4.3.1 [23].

La Tabla 5.14 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con IFPUG antes de la versión 4.0. Se puede observar que el modelo entrenado con *RFR* obtuvo 2 de los criterios de calidad más precisos (*MAE y RMSE*) al igual que *DTR* que obtuvo también 2 (*MMRE y SDMRE*), por otra parte, *GBR* obtuvo solo 1 (*PRED*).

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
IFPUG OLD	RFR	**1084**	**2177**	0.336	0.669	60 %	**4.37**
IFPUG OLD	DTR	1576	3411	**0.323**	**0.302**	56 %	4.21
IFPUG OLD	GBR	1168	2897	0.375	0.573	**63 %**	4.07

Tabla 5.14: Criterios de calidad de modelos automatizados utilizando el método IFPUG antes de la versión 4.0.

5.4.2.5. MARK II

La unidad de medida del método MARK II es 1 MkII Function Point o MkII FP Index [70]. Para calcularlos es necesario sumar todos los puntos de función MkII de todas las transacciones lógicas, las cuales tienen similitud a las transacciones del método IFPUG.

La Tabla 5.15 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con MARK II. Se puede observar que el modelo entrenado con *DTR* obtuvo 2 de los criterios de calidad más precisos (*MMRE y SDMRE*) al igual que *GBR* que obtuvo también 2 (*MAE y RMSE*), por otra parte, *SVM* obtuvo solo 1 (*PRED*). A diferencia de los métodos anteriores, *RFR* no fue el modelo más preciso, ni siquiera estuvo entre los 3 primeros.

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
MARK II	DTR	4681	7887	**0.32**	**0.27**	38 %	**3.53**
MARK II	GBR	**2104**	**2621**	0.91	1.47	38 %	3.28
MARK II	SVR	6972	9765	4.65	12.14	**50 %**	1.66

Tabla 5.15: Criterios de calidad de modelos automatizados utilizando el método MARKII

5.4.2.6. NESMA

El método de medición de tamaño funcional Nesma, también conocido como FPA, tiene como unidad de tamaño funcional 1 FP [71]. La guía de medición de FPA according to NESMA and IFPUG [72] indica que las reglas de medición y la terminología entre ambos métodos son casi las mismas. Sin embargo, NESMA tiene dos métodos adicionales muy adecuados para ser aplicados en etapas tempranas del ciclo de vida del proyecto.

La Tabla 5.16 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con NESMA. Se puede observar que el modelo entrenado con *RFR* obtuvo 4 de los criterios de calidad más precisos (*MAE, RMSE, MMRE y PRED*), mientras que *DTR* obtuvo solo 1 (*SDMRE*).

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
NESMA	RFR	**323**	**784**	**0.177**	0.260	**80 %**	**4.75**
NESMA	DTR	422	993	0.186	**0.196**	74 %	4.43
NESMA	GBR	542	1373	0.204	0.312	78 %	3.64

Tabla 5.16: Criterios de calidad de modelos automatizados utilizando método NESMA

5.4.2.7. LOC

La unidad de medida de este método es 1 LOC (Lines of Code). Esta es la métrica más antigua para los proyectos de software. Introducida por primera vez alrededor de 1960 y se utilizó para estudios económicos, de productividad y de calidad. Usualmente, se utiliza la variante *KLOC*, que son mil líneas de código [73].

La Tabla 5.17 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos medidos con LOC. Se puede observar que el modelo entrenado con *LR-Lasso* obtuvo 4 de los criterios de calidad más precisos (*MAE, RMSE, MMRE y PRED*), mientras que *MARS* obtuvo solo 1 (*SDMRE*). Los algoritmos que entrenaron a estos 3 modelos fueron totalmente distintos a los algoritmos de los métodos de medición anteriores.

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
LOC	LR Lasso	**784**	**1384**	**0.34**	0.84	**74 %**	**4.31**
LOC	MARS	2527	4661	0.42	**0.26**	37 %	2.92
LOC	LR Ridge	1959	3860	0.37	0.33	37 %	2.97

Tabla 5.17: Criterios de calidad de modelos automatizados utilizando LOC

5.4.2.8. Story Points

Los puntos de historia son la medida de tamaño más utilizada en las metodologías ágiles [74]. Estos expresan de forma relativa el tamaño de una historia de usuario [75]. A pesar de esto, en la Tabla 5.10 podemos observar que la variable relacionada con los puntos de historia no es una variable que tenga una alta correlación con el esfuerzo del proyecto.

La Tabla 5.18 muestra los 3 modelos que obtuvieron los mejores criterios de calidad utilizando proyectos desarrollados bajo metodologías ágiles. Se puede observar que el modelo entrenado con *LR-Ridge* obtuvo 3 de los criterios de calidad más precisos (*MAE, RMSE, y PRED*), mientras que *GBR y eNet* obtuvieron 2 (*MMRE, PRED y SDMRE, PRED* respectivamente). Los 3 modelos obtuvieron el mismo *PRED*.

Method	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
SP	LR Ridge	**850**	**1322**	0.21	0.26	**75 %**	**3.91**
SP	GBR	3003	4600	0.13	**0.10**	**75 %**	3.42
SP	eNet	4263	6230	**0.11**	0.16	**75 %**	3.04

Tabla 5.18: Criterios de calidad de modelos automatizados utilizando Story Points

5.4.3. Identificación de las medidas de tamaño más relevantes en la estimación

Una vez calculados los criterios de calidad de modelos automatizados utilizando las distintas medidas de tamaño, el siguiente paso es compararlos obteniendo la razón de cada criterio por cada modelo.

La Figura 5.3 muestra de manera gráfica la suma de razones de los criterios de calidad de modelos por cada medida de tamaño, mostrando únicamente los modelos más precisos de cada medida.

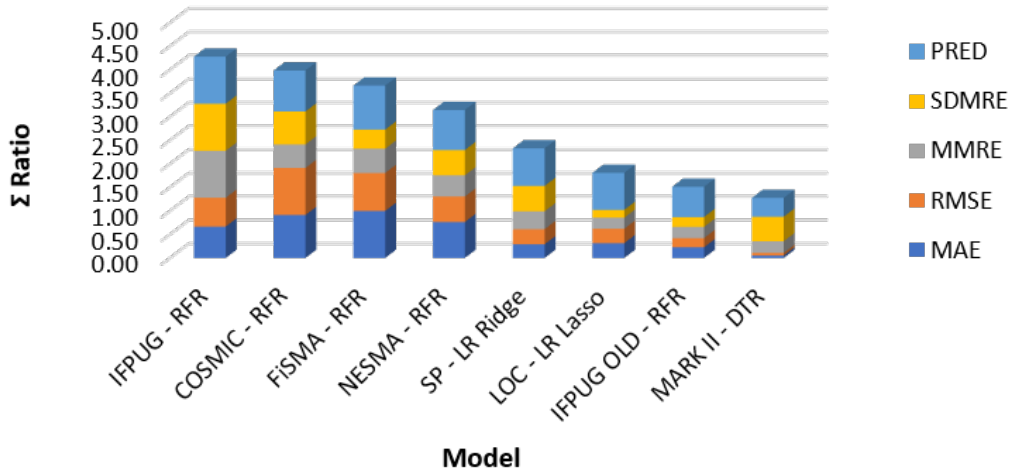

Figura 5.3: Suma de razones de los criterios de calidad de modelos por cada medida de tamaño.

La gráfica muestra lo siguiente:

- Por medidas de tamaño:
  - El método de medición IFPUG tiene los criterios de calidad más eficientes.

  - El método de medición COSMIC tiene el segundo modelo con los criterios de calidad más eficientes.
  - Las 4 medidas de tamaño más utilizadas para medir proyectos en la industria son las más eficientes (IFPUG, COSMIC, FíSMA y NESMA).
  - A excepción de MARK II, los estándares de medición de tamaño funcional son los que tienen modelos más eficientes.
  - MARK II tiene los criterios de calidad menos eficientes.
- Por modelos automatizados:
  - El algoritmo *RFR* entrenó a los 4 modelos más eficientes.
  - El algoritmo *DTR* que entrenó al modelo de MARK II tiene los criterios de calidad menos precisos.
  - Algoritmos basados en regresión lineal (*LR-Ridge* y *LR-Lasso*) entrenaron a los modelos de las medidas de tamaño que no se basan en la medición de tamaño funcional (SP y LOC)
  - *RFR* fue el algoritmo más preciso en varias ocasiones (5 veces), mientras que *LR-Ridge*, *LR-Lasso* y *DTR* solo aparecieron una vez.

Esta comparación de medidas de tamaño nos indica cómo contribuyen estas a la estimación del esfuerzo de proyectos. Sin embargo, para un análisis más exacto, sería de gran utilidad contar con proyectos que hayan sido medidos al mismo tiempo utilizando todas las medidas de tamaño, ya que en este caso de estudio para cada medida de tamaño los proyectos eran distintos.

5.5. Caso de estudio 4: Eficiencia de modelos de estimación sin utilizar variables de tamaño

5.5.1. Cuándo no es posible contar con una medición de tamaño funcional

La medición de tamaño funcional es una técnica utilizada para medir el tamaño del software cuantificando los Requerimientos Funcionales de Usuario (Functional Size Requirements – FUR) [25]. Como se ha demostrado en los casos de estudio anteriores, el tamaño funcional es una de las variables que tienen mayor correlación con el esfuerzo del proyecto. Sin embargo, una medición precisa requiere que sus FUR sean conocidos a nivel de detalle (granularidad) adecuado. Cuando el software carece de documentación y no es posible aplicar a detalle todas las reglas de medición entonces se deben utilizar otras técnicas para dimensionar los FUR [76].

Existen tres principales situaciones en las cuales no se puede medir con precisión el tamaño funcional [77]:

1. Cuando se necesita rápidamente una medida del tamaño, es decir, cuando no se tiene el tiempo suficiente para medir con detalle.

2. En etapas tempranas en el ciclo de vida de un proyecto antes de que los requerimientos se hayan especificado con suficiente detalle para una medición precisa.

3. En general, cuando la calidad de la documentación de los requerimientos no es lo suficientemente buena para una medición precisa.

Cuando se presentan algunas de las situaciones anteriores, los FUR se pueden medir utilizando métodos de aproximación. Estos métodos definen cómo estos pueden ser medidos en niveles de granularidad superiores. Cualquier aproximación de tamaño funcional es el resultado de una compensación entre la facilidad y rapidez de la medición frente a la pérdida de precisión.

5.5.2. Descripción y análisis de los datos

Este caso de estudio analiza la eficiencia de los modelos de estimación cuando los proyectos no cuentan con una variable del tamaño funcional. Este análisis se basa en los siguientes dos subconjuntos de la base de datos ISBSG:

1. Proyectos que no cuentan con información en la variable Functional Size.

2. Proyectos con información en las variables *Effort estimate y Effort estimate method* pero omitiendo las variables relacionadas con el tamaño funcional.

El primer subconjunto de datos cuenta con 990 proyectos que tienen 27 variables con suficiente información. Entre estas variables, las que tienen mayor correlación con el esfuerzo del proyecto son las relacionadas con el tamaño del equipo de desarrollo y el tamaño relativo del proyecto, es decir, si el proyecto tiene una clasificación de tamaño S, M, L, etc. El segundo subconjunto es el mismo del primer caso de estudio, simplemente se omitieron las variables relacionadas con el tamaño funcional del proyecto.

Utilizando los dos subconjuntos anteriores, se generarán modelos automatizados, se compararán con los métodos tradicionales y con los modelos automatizados del primer caso de estudio que si toman en cuenta variables de tamaño funcional.

5.5.3. Criterios de calidad de modelos automatizados con proyectos sin tamaño funcional

La Tabla 5.19 muestra los 3 modelos que obtuvieron los mejores criterios de calidad con el primer subconjunto de datos (proyectos sin tamaño funcional). Se puede observar que *DTR* obtuvo 3 de los criterios más precisos (*MMRE, SDMRE y PRED*). Sin embargo, este no fue el más preciso, ya que *GBR* obtuvo un *Ratio* más alto con 2 criterios más precisos (*MAE y RMSE*).

Subset	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
Projects without functional size	GBR	**2323**	**6475**	0.632	1.203	32 %	**4.43**
Projects without functional size	RFR	2872	13425	0.590	0.860	33 %	4.06
Projects without functional size	DTR	4084	13422	**0.575**	**0.730**	**35 %**	4.05

Tabla 5.19: Criterios de calidad de modelos automatizados de proyectos sin tamaño funcional

Por otra parte, la Tabla 5.20 muestra los 3 modelos que obtuvieron los mejores criterios de calidad con el segundo subconjunto de datos, proyectos con información en el esfuerzo estimado pero omitiendo variables de tamaño funcional. Se puede observar que *RFR* obtuvo 3 de los criterios más precisos (*MMRE, SDMRE y PRED*), mientras que *GBR* obtuvo solo 2 (*MAE y RMSE*).

Subset	Model	MAE	RMSE	MMRE	SDMRE	PRED	Ratio
Effort estimated without functional size	RFR	2752	8179	**0.550**	**0.739**	**42 %**	**4.58**
Effort estimated without functional size	DTR	3182	7984	0.572	0.974	37 %	4.09
Effort estimated without functional size	GBR	**2628**	**6869**	0.857	1.769	41 %	3.78

Tabla 5.20: Criterios de calidad de modelos automatizados de proyectos con esfuerzo estimado omitiendo el tamaño funcional

5.5.4. Comparación de criterios de calidad de modelos automatizados con y sin variables de tamaño funcional

La Figura 5.4 muestra de manera gráfica la suma de las razones de los criterios de calidad del modelo automatizado y el método tradicional más preciso del primer caso de estudio, así como también muestra el modelo más preciso del subconjunto 1 (NoMeasured) y subconjunto 2 (NoFS) de este caso de estudio. Se puede observar que la precisión de los modelos automatizados que no consideran variables de tamaño funcional es aproximadamente la mitad de aquellos modelos que si consideran estas variables.

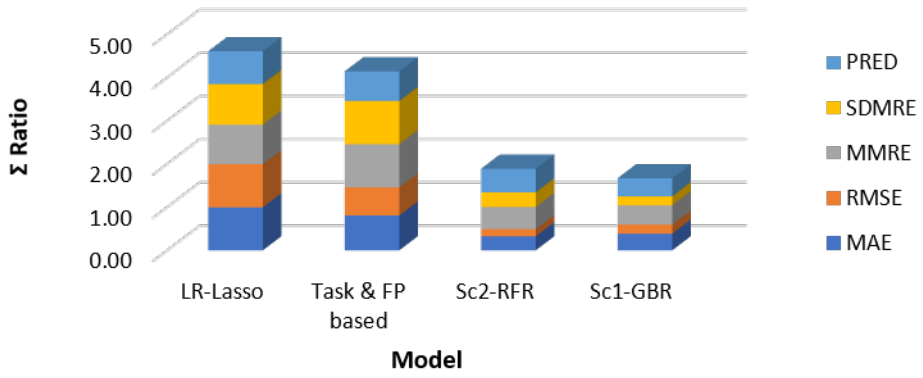

Figura 5.4: Suma de razones de los criterios de calidad de modelos sin considerar variables de tamaño funcional.

El mejor modelo automatizado del primer caso de estudio sigue siendo el más eficiente, el cual utiliza variables de tamaño funcional. Este modelo es 144 % y 176 % (respectivamente) más eficiente que los modelos del segundo y primer subconjunto de datos. Por otra parte, *Task & FP based* es 119 % y 148 % más eficiente que estos.

5.6. Caso de estudio 5: Entrenamiento de una red neuronal para la estimación de software

5.6.1. Descripción de los datos de proyectos

En este caso de estudio se analiza el resultado de entrenar una ANN con los datos de proyectos utilizados en el primer caso de estudio, es decir, proyectos que tienen información en el esfuerzo real y el esfuerzo estimado del proyecto.

Lo anterior se hace con el objetivo de analizar una vez más la precisión de *AEGiS*, en esta ocasión comparandolo la precisión de los criterios de calidad de una ANN. Adicionalmente, se comparan también con los criterios de calidad de los métodos de estimación tradicionales.

5.6.2. Descripción de los datos del entrenamiento

El primer paso para el entrenamiento de una red neuronal es crear el modelo, este fue definido con 4 capas ocultas. Lo siguiente fue definir el número de épocas y la tasa de aprendizaje, los cuales fueron $500,000$ y 0.000001 respectivamente.

Por otra parte, los parámetros de la red se definieron de la siguiente manera; el número de neuronas para la capa de entrada y las capas ocultas fue igual a 512 y el número de clases de la capa de salida fue igual a 1.

Por último, para la función de costo se utilizó la función de pérdida de *MAE* y el algoritmo de optimización utilizado fue *Adadelta*, este es un método de descenso del gradiente que se basa en la tasa de aprendizaje adaptativo por dimensión [65].

Cabe mencionar que se entrenó un conjunto de ANN con distintos parámetros, funciones de pérdida y algoritmos de optimización. Sin embargo, los datos anteriormente descritos fueron los que proporcionaron mejores criterios de calidad de una de las ANN entrenadas.

5.6.3. Criterios de calidad de la red neuronal

Una vez entrenada la red neuronal con los datos de entrenamiento, esta se debe utilizar para estimar los datos de validación. La Tabla 5.21 muestra los criterios de calidad de la ANN.

Subset	Model	MAE	RMSE	MMRE	SDMRE	PRED
Effort estimated	ANN	1317	**1104**	0.33	0.50	32 %

Tabla 5.21: Criterios de calidad de la ANN entrenada

Si se analizan y comparan uno a uno los criterios de calidad de la ANN con los del modelo más preciso de *AEGiS* y con el método tradicional más preciso, la ANN solo es mejor para el caso del criterio *RMSE*. Sin embargo, lo importante es comparar todos los criterios de calidad obteniendo sus razones.

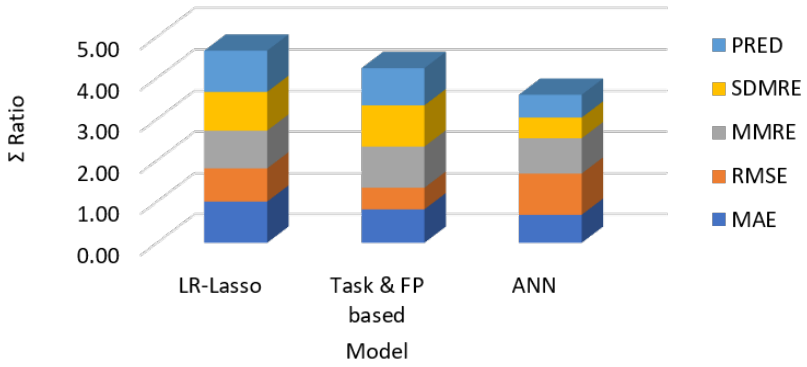

Figura 5.5: Suma de razones de los criterios de calidad de modelo automatizado, método tradicional y red neuronal más precisa.

La Figura 5.5 muestra de manera gráfica la suma de las razones de los criterios de calidad del modelo automatizado, el método tradicional y la ANN más precisa. Se puede observar la ANN no fue más eficiente que el modelo automatizado más preciso generado con *AEGiS* y esta tampoco fue más eficiente que el método tradicional más preciso.

Con la suma de las razones de los criterios de calidad, se puede calcular el porcentaje de mejora de un modelo respecto a otro. De esta manera, el modelo automatizado más preciso fue 30 % más eficiente que la ANN y el método tradicional más preciso fue 18 % más eficiente.

En conclusión, este primer intento por entrenar una red neuronal para estimar proyectos de desarrollo de software no obtuvo mejores resultados que los obtenidos previamente. Sin embargo, esto no significa que una red neuronal no pueda llegar a obtener criterios de calidad más precisos. Es importante resaltar que este estudio tuvo limitaciones respecto a tiempo y recursos, ya que por lo general entrenar redes neuronales requiere de un alto poder de cómputo y de varios días, incluso semanas de experimentar con los parámetros de entrada y algoritmos de optimización.

Conclusiones

Conclusiones generales

El objetivo principal de este trabajo de tesis fue mejorar la precisión de la estimación de proyectos de software mediante el uso de técnicas de aprendizaje de máquinas. Dada la naturaleza del problema, estas técnicas se basaron principalmente en tareas de regresión de aprendizaje supervisado.

Asimismo, este trabajo no pretendía solo entrenar algoritmos de aprendizaje de máquinas esperando obtener resultados favorables, sino que pretendió llevar a cabo el proceso de generación de modelos de estimación en una manera automatizada. Lo anterior se realizó mediante el desarrollo de un sistema llamado *AEGiS* (por sus siglas en inglés).

De manera general, la implementación del sistema se basó en la definición de un conjunto de funciones que realizan una tarea especifica de preprocesamiento de datos, entrenamiento y/o validación de modelos. Adicionalmente, el sistema también tiene la funcionalidad de comparar varios modelos de estimación mediante técnicas de normalización, con la finalidad de analizar cuál es el modelo con los criterios de calidad más precisos.

A pesar de que existen múltiples bibliotecas de código abierto para el aprendizaje de máquinas, es necesario tener cierto nivel de experiencia con el lenguaje de programación asociado. Aunque también existen varias funciones que realizan cierto preprocesamiento de datos, es importante tener cuidado con los tipos de estructuras de datos que reciben y retornan estas funciones, ya que es muy común obtener errores de ejecución por usar tipos de datos incorrectos.

Una de las principales ventajas de utilizar *AEGiS* es que no se necesita experiencia ni conocimientos relacionados con aprendizaje de máquinas, ni con un lenguaje de programación específico, ni mucho menos se necesita ser un experto en temas de estimación de software, ya que este sistema funciona de manera automatizada, solo es necesario pasarle como parámetro una base de datos de proyectos de software y este generará los modelos de estimación más precisos posibles (de acuerdo con el preprocesamiento que tiene definido).

Otra de las ventajas que se tiene al utilizar este sistema es que al funcionar de

manera automatizada se tiene un ahorro principalmente en tiempo y recursos, ya que generar modelos de estimación suele ser una tarea que requiere un análisis riguroso, principalmente para tareas de preprocesamiento de datos.

Por otra parte, es necesario analizar la eficiencia del sistema para saber si es adecuado utilizarlo en la generación modelos de estimación. Para esto, se realizaron 5 casos de estudio con el fin de verificar su eficiencia en la estimación de proyectos de software.

El primer caso de estudio posiblemente sea el más importante, ya que consiste en generar modelos de estimación automatizados y compararlos con los métodos de estimación más utilizados en la industria. Los resultados de este estudio mostraron que uno de los modelos automatizados tuvo criterios de calidad más precisos que los métodos tradicionales. En particular, este modelo obtuvo una mejora del 11 % respecto al método tradicional más preciso. Este porcentaje se puede interpretar de varias maneras, pero para el objetivo de la tesis es un buen porcentaje debido a que se pretendía mejorar la precisión de las estimaciones.

El segundo caso de estudio es un intento de analizar que tan adecuado es hacer particiones a la base de datos para generar modelos de estimación, tal como lo proponen algunos autores y han demostrado dar buenos resultados. Sin embargo, este estudio demostró que cuando se utilizan algoritmos de aprendizaje de máquinas puede no ser necesario hacer particiones, ya que los algoritmos presentan mejores resultados cuando reciben los datos completos y no particiones de los mismos. En este estudio se hicieron 4 particiones, en 3 de ellas la precisión de los modelos de estimación iba empeorando con cada partición. Al final, los modelos generados sin ninguna partición fueron los más precisos.

Para el tercer caso de estudio también se utilizaron particiones de los datos, con el objetivo de analizar la contribución que tienen las medidas de tamaño en la estimación de software. En este estudio, a diferencia de otros que también han demostrado la importancia de usar medidas de tamaño, se analizó cuales son las medidas que tienen la mayor correlación con el esfuerzo del proyecto. Los resultados del estudio fueron favorables para los métodos de medición de tamaño funcional, en los que IFPUG y COSMIC obtuvieron la correlación más alta con el esfuerzo del proyecto, no por nada son los dos métodos más utilizados en la industria.

Respecto al caso de estudio anterior, ¿qué pasaría si no contaramos con esas medidas de tamaño? El cuarto caso de estudio analiza la eficiencia de los modelos de estiamción automatizados sin tomar en cuenta estas medidas. Para esto, se analizaron también aquellos proyectos de la base de datos que no contaran con información relacionada a medidas de tamaño. Los resultados del estudio mostraron que los modelos de estimación que se pueden generar son poco eficientes, ya que si los comparamos con los modelos que si toman en cuenta estas medidas, se tiene menos de la mitad de precisión, es decir, utilizar medidas de tamaño permite generar modelos con más del doble de precisión.

El último caso de estudio demostró desde otra perspectiva la eficiencia de los modelos de estimación automatizados. El estudio consistió en entrenar una red neuronal que fuera capaz de estimar proyectos de software. La eficiencia de la red neuronal no

fue la mejor, ya que comparándola con el modelo automatizado y el método tradicional más preciso, la red fue la más baja. Sin embargo, se debe tener en cuenta que el entrenamiento de una red neuronal es una tarea computacionalmente costosa y que requiere de mucho tiempo para experimentar con los datos. Por esta razón, *AEGiS* no incluye en su funcionamiento el entrenamiento de redes neuronales, ya que intenta generar modelos de estimación de una manera relativamente rápida.

En general, se puede concluir que se cumplió con el objetivo principal de la tesis. El sistema que se desarrolló es capaz de generar modelos de estimación eficientes, los cuales pueden ser utilizados por administradores, líderes, usuarios, desarrolladores o cualquier involucrado en el proyecto. De esta manera, se aumenta la probabilidad de tener proyectos exitosos, ya que una estimación confiable donde se conoce la variabilidad de esta permite que durante el desarrollo del proyecto se pueda llevar un monitoreo y control factible.

Adicional a lo anterior, el sistema desarrollado no solo es capaz de estimar proyectos de software. En realidad, el sistema puede estimar cualquier variable continua de un conjunto de datos, siempre y cuando este conjunto cumpla los requisitos mínimos (descritos en el Capítulo 4). Por ejemplo, se puede utilizar para estimar el costo de una casa o de un automóvil, incluso de puede utilizar para diagnosticar la diabetes, solo se debe tener en cuenta que es necesario verificar que las métricas de evaluación de errores son relevantes para el problema que se desea atacar.

Trabajo futuro

Este trabajo de tesis es solo el comienzo de algo que puede explotarse en muchos aspectos. Si bien, el sistema *AEGiS* es capaz de generar modelos de estimación eficientes, la manera de generar y evaluar los modelos se puede hacer de una manera dinámica. Actualmente el sistema evalua los modelos con 5 criterios de calidad y después los normaliza para poderlos comparar con los de otro modelo, esta comparación asigna un peso equivalente a cada criterio (del 20 %). Sería muy adecuado poder configurar el peso de los criterios de calidad. Por ejemplo, puede ser necesario generar modelos con el *PRED* más alto debido a que eso permite garantizar que la estimación tendrá una variabilidad entre el 25 %, entonces el sistema debería poder asignarle al *PRED* un peso más alto (por ejemplo 50 %). De la misma forma sería muy útil poder seleccionar los criterios con los que se desean evaluar los modelos, ya que algunos de estos pueden no ser relevantes para alguna empresa y puede haber otros que si lo sean y que no estén incluidos en el sistema, entonces también sería importante poder definir métricas de evaluación propias.

Por otra parte, el sistema tiene definido un conjunto de algoritmos con los que genera y evalua modelos de estimación, indicando cuales son los que tienen criterios de calidad más precisos. De igual importancia, sería muy adecuado que se pudieran seleccionar los algoritmos con los que se desean entrenar los modelos y que el sistema fuera capaz de guardar los modelos previamente generados. Asimismo, el sistema podría surgerirle al usuario cuales son los algoritmos que por lo general entrenan a los modelos

más precisos. Por ejemplo, de todos los modelos generados en los 5 casos de estudio, el algoritmo que entrenó el mayor número de modelos más precisos fue *RFR*, el cual entrenó a 9 de los 16 totales.

Una vez que el sistema genera un modelo de estimación, se puede utilizar para estimar uno o varios proyectos de software. Algo que se podría hacer todavía más automatizado, sería que desde el principio se pudiera pasar como parámetros los proyectos que deseamos estimar y que el sistema, además de retornar los modelos más precisos, nos indique las estimaciones de los proyectos resultantes de utilizar los modelos. Lo anterior haría todavía más fácil la tarea de estimar nuevos proyectos de software. Sin embargo, requiere mayor trabajo para el sistema ya que deberá analizar las variables con las que cuentan los nuevos proyectos para generar los modelos. Adicional a esto, el sistema también podría no solo analizar las variables con mayor correlación, sino también analizar los valores de las variables que tienen mayor correlación, así como se mostró en el tercer caso de estudio con los métodos de medición de tamaño funcional.

Hasta este punto se han descrito posibles mejoras que se le pueden hacer al sistema. Sin embargo, estas también indican algunas limitaciones que tiene y que se deben tener en cuenta a la hora de utilizarlo.

Visto desde otra perspectiva, actualmente el sistema tiene la función de un backend, ya que se encarga de realizar todo lo invisible para el usuario, es decir, toma los datos, los procesa y envía envía una respuesta. Un posible trabajo futuro sería desarrollar el frontend, el cual consistiría en una interfaz gráfica donde el usuario pudiera seleccionar y configurar todo lo descrito anteriormente para generar modelos de estimación y poder estimar nuevos proyectos.

Por último, se pueden explorar otras técnicas para generar modelos de estimación, así como se intentó en el quinto caso de estudio. De la misma forma, se pueden entrenar redes neuronales pero esta vez aumentando los niveles de conexión, es decir, agregar capaz ocultas, se pueden aumentar capaz sinápticas, aumentar el número de épocas y también se puede probar con otros algoritmos de optimización y funciones de pérdida. Los resultados obtenidos se pueden comparar principalmente con los métodos de estimación tradicionales y también con los modelos automatizados de *AEGiS*. En caso de que la red neuronal tenga mejores resultados, se necesitará analizar cuál es el respectivo porcentaje de mejora y se deberá tomar en cuenta que el proceso de entrenamiento es menos eficiente que el de los modelos automatizados. En otras palabras, se debe analizar en qué aspectos se está ganando eficiencia y en cuales se está perdiendo.

Artículos publicados

Durante el primer año de desarrollo de este proyecto de tesis, se redactó el artículo titulado *"Estimación del esfuerzo de proyectos de software con algoritmos de aprendizaje de máquinas"* presentado en *The 7th edition of the International Conference in Software Engineering Research and Innovation (CONISOFT'19)* y publicado en noviembre de 2019 en la *Revista Electrónica de Computación, Informática, Biomédica y Electrónica.* Los autores principales fueron Jesús Iván Saavedra Martínez, María Guadalupe Elena

Ibargüengoitia González y Gibran Fuentes Pineda. El artículo presentó una comparación entre modelos de estimación basados en modelos estadísticos y modelos generados a partir de algoritmos de regresión de aprendizaje de máquinas.

En el segundo año de desarrollo de la tesis, se redactó un segundo artículo titulado *"Analysis of automated estimation models using machine learning"* que será presentado en noviembre de 2020 en *The 8th International Conference in Software Engineering Research and Innovation (CONISOFT'20)* y posteriormente indexado en la edición especial de una revista de la *IEEE*. Los autores principales fueron Jesús Iván Saavedra Martínez, Francisco Valdés Souto y Moisés Rodríguez Monje. El artículo muestra la aplicación del sistema que de manera automatizada genera modelos de estimación con el objetivo de comparar estos modelos con los métodos de estimación tradicionales y las técnicas más comunes para estimar proyectos de software.

Agradecimientos

El trabajo de este proyecto de tesis fue apoyado por *Grupo Alarcos* de la Universidad de Castilla-La Mancha, España, en una estancia de investigación llevada a cabo los 3 primeros meses del año 2020. En particular se agradece al Dr. Mario Piattini por la aceptación de la estancia en la universidad y al Dr. Moisés Monje por su asesoramiento y colaboración en los resultados experimentales de este trabajo.

De la misma manera, se agradece a la organización interna que proporcionó los datos de sus proyectos para poder llevar a cabo el análisis de uno de los casos de estudio.

This research is part of the DQIoT project (INNO-20171086), funded by CDTI; ECD project (PTQ-16-08504), funded by the "Torres Quevedo" Program of the Spanish Ministry of Economy, Industry and Competitiveness; and the TESTIMO project (La Consejería de Educación, Cultura y Deportes de la Junta de Comunidades de Castilla-La Mancha, and the Fondo Europeo de Desarrollo Regional FEDER, SBPLY/17/180501 /000503).

Bibliografía

[1] A. Abran, *Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers*. Hoboken, New Jersey: Wiley, IEEE Press, 2015. xiii, xiii, xiii, 1, 15, 16, 17, 21, 22, 29

[2] J. S. Shirabad and T. J. Menzies, *The PROMISE Repository of Software Engineering Databases*. http://promise.site.uottawa.ca/SERepository": School of Information Technology and Engineering, University of Ottawa, Canada", 2005. 1, 36

[3] P. Pospieszny, B. C. Chrobot, and A. Kobylinski, *An effective approach for software project effort and duration estimation with machine learning algorithms*. The Journal of Systems and Software, 137, 2018, pp. 184-196. 2, 36

[4] T. Mitchell, *Machine learning*. New York: McGraw Hill, 2017. 2

[5] K. Srinivasan, *Machine learning approaches to estimating software development effort*. IEEE Transactions on Software Engineering, 1995. 2

[6] G. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. John Wiley Sons, 2011. 2

[7] K. Cios, *Data Mining A Knowledge Dis- covery Apporach*. New York, USA: Springer Science, 2007. 2

[8] Standish-Group, *The CHAOS Report: Decision Latency Theory*. 2019. 21 - Mayo - 2019. 3, 11, 33

[9] Price-Systems, *A Cracked Foundation: A Critical Look at the Role of Baselining in Government IT Project Management*. Mount Laurel, New Jersey: GAO Testimony Before US Senate, 2006. 3

[10] IEEE, *Standard glossary of software engineering terminology*. New York: Institute of Electrical and Electronics Engineers, 1991. pp. 28-29. 7, 8, 9

[11] W. D. Halsey, *Macmillan Dictionary*. London: Macmillan, 1973. 7

[12] S. Sánchez, M. Sicilia, and D. Rodríguez, *Ingeniería del Software. Un enfoque desde la guía SWEBOK.* México: Primera ed., Alfaomega Grupo Editor, S.A. de C.V, 2012. 7, 8, 12

[13] P. Bourque and R. E. Fairley, *Software Engineering Body of Knowledge.* versión 3.0: IEEE, 2014. pp. 31. 8

[14] ISO/IEC/IEEE-12207:2017, *Systems and software engineering — Software life cycle processes.* 2017. 8

[15] IEEE 1219-1998, *IEEE Standard for Software Maintenance.* Institute of Electrical and Electronics Engineers, 1998. 9

[16] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK).* 5 ed., Newtown Square, Pennsylvania USA: PMI Publishing Division, 2000. pp. 596. 9, 14

[17] *"Manifiesto por el Desarrollo Ágil de Software", agilemanifesto.org, 2020. [Online]. Available: https://agilemanifesto.org/iso/es/manifesto.html. [Accessed: 23- Jun- 2020].* 9

[18] P. Abrahamsson, K. Conboy, and X. Wang, *Lots done, more to do: The current state of agile systems development research.* European Journal of Information Systems, 2009, pp. 281. 10

[19] L. Williams and A. Cockburn, *Agile software development: It's about feedback and change.* Computer, 2003, pp. 39. 10

[20] K. Schwaber and J. Sutherland, *The Scrum Guide.* Developed and sustained by Scrum creators, November 2017, pp. 3-5. 11

[21] W. Thomson, *Lecture to the Institution of Civil Engineers, Electrical Units of Measurement.* London: Popular Lectures and Addresses, May 1883, pp. 80-81. 11

[22] A. Abran, *Software Metrics and Software Metrology.* Hoboken, N. J.: Wiley, IEEE Press, 2010. pp. 328. 11

[23] The International Function Point Users Group, *Function Point Counting Practices Manual.* Release 4.3.1: IFPUG, January 2010. pp. 370. 13, 71

[24] ISO/IEC 14143-6:2012, *Information technology – Software measurement – Functional size measurement – Part 6: Guide for use of ISO/IEC 14143 series and related International Standards.* 2012. 13

[25] ISO/IEC 14143-1:2007, *Information technology – Software measurement – Functional size measurement – Part 1: Definition of concepts.* 2017. 13, 75

[26] International Software Benchmarking Standards Group, *ISBSG Repository R1 - Field Descriptions.* Software project data, 2019. 13, 36, 58

[27] COSMIC Measurement Practice Committee, *COSMIC Measurement Manual-version 5.0 – Part 1: Principles, Definitions & Rules*. March 2020. 13, 70

[28] I. Jacobson, I. Spence, and K. Bittner, *USE-CASE 2.0, The Guide to Succeeding with Use Cases*. Ivar Jacobson International, 2011. pp. 55. 13

[29] G. Karner, *Resource Estimation for Objetory Projects*. Objetive Systems SF AB, 1993. 13

[30] M. Chemuturi, *Software Estimation Best Practices, Tools Techniques: A Complete Guide for Software Project Estimators*. Illustrated ed., J. Ross Publishing, 2009. pp. 320. 13

[31] S. McConnell, *Software Estimation: Demystifying the Black Artare*. the University of Californiaa: Illustrated ed., Microsoft Press, Nov 2009. pp. 308. 14

[32] J. Tuya, I. R. Román, and J. D. Cosín, *Tecnicas cuantitativas para la gestion en la ingenieria del software*. Oleiros, La Coruna: Netbiblo, 2007. 14

[33] R. S. Pressman, *Ingeniería del software: un enfoque práctico*. Quinta ed., D. Ince, Ed., Madrid: McGraw-Hill, 2003. pp. 601. 16

[34] C. Zhang and Q. Yang, *Data Preparation for Data Mining*. University of Technology Sydney — UTS: Centre for Quantum Computation and Intelligent Systems (QCIS), May 2003, pp. 377. 18

[35] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd Edition, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., June 2019, pp. 30. 18, 19, 27, 28, 29, 40

[36] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python*. First Edition, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., October 2016, pp. 2. 18, 40

[37] K. P. Murphy, *Machine learning : a probabilistic perspective*. Massachusetts London, England: The MIT Press Cambridge, 2012, pp. 2. 20

[38] J. P. Mueller and L. Massaron, *Machine learning for dummies*. Hoboken, New Jersey: John Wiley Sons, Inc., 2016, pp. 258. 21

[39] Scikit-learn.org, *scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation", 2020. [Online]*. Available: https://scikit-learn.org/stable/. [Accessed: 04- Jun- 2020]. 21, 29, 40, 54

[40] J. D. Kelleher, B. M. Namee, and A. D'Arcy, *Fundamentals of machine learning for predictive data analytics*. Massachusetts Institute of Technology: The MIT Press, 2015, pp. 436. 22, 29

[41] P. Harrington, *Machine Learning In Action.* Shelter Island, N.Y: Manning, 2012, pp. 102. 23

[42] S. Lek and J.-F. Guegan, *Artificial Neuronal Networks: Application to Ecology and Evolution.* New York: Springer-Verlag Berlin Heidelberg, 2000, pp. 3. 24

[43] K. Gurney, *An introduction to neural networks.* University of Sheffield, London and New York: UCL Press, 1997, pp. 13. 24

[44] W. Pitts and W. S. McCulloch, *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics 5, 1943. 25

[45] M. Nielsen, *Neural Networks and Deep Learning.* Determination Press, 2015. 25

[46] D. Conway and J. M. White, *Machine Learning for Hackers.* 1005 Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc, February 2012, pp. 31. 26

[47] N. Mittas and L. Angelis, *LSEbA: least squares regression and estimation by analogy in a semi-parametric model for software cost estimation.* Empirical Software Engineering, 2010, pp. 15. 27, 28, 29

[48] J. Z. Ruiz, *Comparativa y análisis de algoritmos de aprendizaje automático para la predicción del tipo predominante de cubierta arbórea.* Universidad Complutense de Madrid: Departamento de sistemas informáticos y computación, 2018. 27

[49] J. Bergstra, D. Yamins, and D. Cox, *Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms.* Austin, Texas: Proceedings of the 12th Python in Science Conference (SciPy 2013), 2013. 29

[50] J. O. Rawlings, S. G. Pantula, and D. A. Dickely, *Applied Regression Analysis: A Research Tool, Second ed.* Department of Statistics, North Carolina State University: Springer Science Business Media, 2001, pp. 660. 29

[51] R. A. Vera, *Ingeniería de Software en México: Educación, Industria e Investigación.* Impreso en México: Academia Mexicana de Computación, A.C., 2018. 34

[52] IMCO-AMITI, *Select - Mapa de ruta 2025: para transformar a México a través de la adopción de Tecnologías de la Información y Comunicación.* 2013. 34

[53] Amms.org.mx, *AMMS – Asociación Mexicana de Métricas de Software, 2020. [Online].* Available: https://amms.org.mx/. [Accessed: 02- Jul- 2020]. 34

[54] Asociación Mexicana de Métricas de Software, *Estudio Línea Base de Productividad y Costo de la Industria Mexicana de Desarrollo de Software.* 2018. 34

[55] R. Britto, E. Mendes, and J. Börstler, *An empirical investigation on effort estimation in agile global software development.* IEEE 10th international conference on global software engineering, 2015, pp. 38. 34, 36

[56] E. A. Nelson, *Management handbook for the estimation of computer programming costs.* Santa Monica, California: System Development Corporation (SDC), 1967. 35

[57] B. W. Boehm, C. Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece, *Software Cost Estimation with Cocomo II with Cdrom.* Prentice Hall PTR: 1st ed. Upper Saddle River, NJ, USA, 2000. 35

[58] S. K. Sehra, Y. S. Brar, N. Kaur, and S. S. Sehra, *Research patterns and trends in software effort estimation.* Information and Software Technology, Volume 91, November 2017. 35

[59] L. Ferreira, D. Galvez, L. A. Quintero, and J. Vargas, *Estimación del esfuerzo en proyectos de software utilizando técnicas de inteligencia artificial.* Universidad Central "Marta Abreu" de Las Villas: Revista Cubana de Ciencias Informáticas, 2014, pp. 1. 36

[60] M. Mendonca and Sunderhaft, *Mining Software Engineering Data: A Survey Analysis.* (Vol. 4000), 1999. 36

[61] G. Nagpal, M. Uddin, and A. Kaur, *A Comparative Study of Estimation by Analogy using Data Mining Techniques.* Inf Process Syst, Vol.8, No.4, 2018, pp. 3-6. 36

[62] Conisoft.org, *CONISOFT - 2020. [Online].* Available: http://conisoft.org/2020/. [Accessed: 02- Jul- 2020]. 36

[63] Sistedes.es, *JISBD - Sistedes, 2020. [Online].* Available: https://www.sistedes.es/jornadas/jisbd. [Accessed: 02- Jul- 2020]. 36

[64] J. G. Borade and V. R. Khalkar, *Software Project Effort and Cost Estimation Techniques.* International Journal of Advanced Research in Computer Science and Software Engineering, 2015, pp. 38. 36

[65] Tensorflow.org, *"TensorFlow", 2020. [Online].* Available: https://www.tensorflow.org/. [Accessed: 06- Jul- 2020]. 40, 79

[66] D. Freedman, R. Pisani, and R. Purves, *Statistics: Fourth International Student Edition.* W. W. Norton & Company, 2007. 50

[67] V. Ulloa, *Estadistica Aplicada a la Comunicacion.* UNAM, 2006, pp. 48. 51

[68] S. Trudel, *Agile COSMIC A Good Integration.* Congreso Nacional de Medición y Estimación de Software (CNMES) 2019, October 2019. 70

[69] ISO/IEC 29881:2008, *Information technology – Systems and software engineering – FiSMA 1.1 functional size measurement method.* 70

[70] UKSMA Metrics Practices Committee, *Mk II Function Point Analysis, Counting Practices Manual Version 1.3.1.* September 1998, pp. 100. 72

[71] ISO/IEC 24570:2005, *Software engineering – NESMA functional size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis.* 72

[72] NESMA, *FPA according to NESMA and IFPUG.* July 2015. 72

[73] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. Boehm, *A SLOC Counting Standard.* University of Southern California: Center for Systems and Software Engineering, 2007. 73

[74] M. Usman and R. Britto, *Effort estimation in co-located and globally distributed agile software development: A comparative study.* joint conference of the international workshop on software measurement and the international conference on software process and product measurement (IWSM-MENSURA), 2016, pp. 219–224. 73

[75] E. Coelho and A. Basu, *Effort estimation in agile software development using story points.* International Journal of Applied Information Systems (IJAIS), 2012. 73

[76] J.-M. Desharnais and A. Abran, *Approximation techniques for measuring Function Points.* Proc, in 13th Inter. Workshop on Software Measurement (IWSM 2003), 2003. 75

[77] COSMIC Measurement Practice Commitee, *COSMIC Guideline for Early or Rapid Functional Size Measurement using approximation approaches.* 2015, pp. 46. 75