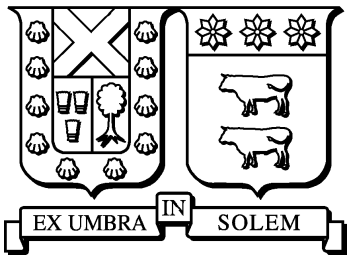
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE



"ESTIMACIÓN DE ESFUERZO EN PROYECTOS DE SOFTWARE A PARTIR DE HISTORIAS DE USUARIO"

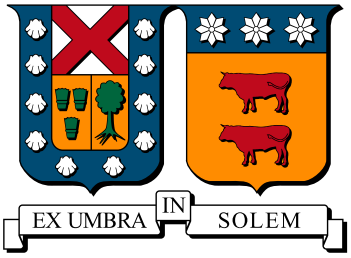
FELIPE IGNACIO MORALES MATURANA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: RICARDO ÑANCULEF

SEPTIEMBRE 2019

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



"ESTIMACIÓN DE ESFUERZO EN PROYECTOS DE SOFTWARE A PARTIR DE HISTORIAS DE USUARIO"

FELIPE IGNACIO MORALES MATURANA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: RICARDO ÑANCULEF
PROFESOR CORREFERENTE: JOSÉ LUIS MARTÍ

SEPTIEMBRE 2019

Agradecimientos

A mi madre Carmen Gloria por el apoyo y amor incondicional que me ha brindando durante toda mi vida, y por ser el pilar fundamental de lo que hoy soy como persona y profesional. En memoria de mi padre Jorge, que a pesar de eventos muy turbulentos durante el transcurso del proceso, logró dejarme por siempre la meta de alcanzar la mayor felicidad posible. En memoria de mi abuelita Rosa, cuyo cariño, bondad, dulzura y amor quedarán por siempre en mis recuerdos. A mis tíos, quienes fueron un apoyo fundamental en momentos difíciles. A mi prima Karen, por estar también ahí y recordarme lo lindo que es simplemente vivir.

A mis queridos amigos de la universidad que me entregaron momentos inolvidables durante mi estadía en ella y quienes tendrán por siempre un lugar en mi corazón: Jose, Dani, May, Gustavo, Marco y Felipe. A mis amigos de toda la vida, Nini y Beto, por siempre estar ahí cuando más lo necesitaba.

A Nursoft por confiar en mi en la realización de este trabajo. A mis compañeros y amigos de Nursoft, que me han ayudado a no temer quien soy realmente y a siempre mejorar profesionalmente.

A Jose, por su cariño y apoyo durante los días más difíciles durante la realización de este trabajo.

Finalmente, a todos los profesores que me formaron para convertirme en un profesional, en especial al profesor José Luis Martí por ser el principal apoyo durante toda la carrera y al profesor Ricardo Ñanculef por presentarme el interesante mundo de las máquinas de aprendizaje.

Resumen

En la industria del desarrollo de software, la estimación inicial de la duración de los proyectos es de suma importancia para garantizar su éxito y es una de las actividades más complejas de realizar correctamente. Alrededor del 60 % de los proyectos fracasa por no cumplir con expectativas temporales, por lo que se ha hecho énfasis en poder generar mejores métodos de predicción. Sin embargo, existen falencias en la mayoría de los métodos que no permiten que sean implementados en la realidad.

En este trabajo se propone un *framework* de pronóstico de esfuerzo en proyectos de software, que utiliza la información de las historias de usuario y métodos de aprendizaje automático. Para esto se proponen además aspecto clave de las historias de usuario que se relacionan con el esfuerzo que requieren para ser terminadas.

Finalmente, se valida el framework con un dataset de proyectos que utilizan metodologías ágiles para su realización, comparando su rendimiento con el estado del arte y el proceso de estimación clásico. En base a los resultados, se concluye respecto a la información que es relevante para pronosticar un proyecto y el rendimiento general de la propuesta.

Abstract

In the industry of software development, the initial estimation of the duration of projects is one of the most important and difficult tasks in the planning phase. Around 60 % of projects fails due to unfeasible deadlines, so a lot of research has been done in order to mitigate it. Most of this research, however, presents a degree of problems that makes it rather impossible to implement in practice.

In this work, an effort forecasting framework is proposed that uses information of the user stories to perform predictions with modern machine learning techniques. A set of key aspects of the user stories is also proposed with the motivation to recognize useful information for the predictive process.

Finally, the framework is validated on a modern dataset of projects that use agile methodologies in their implementations, comparing its performance with the state of the art and the current classic forecasting process. Based on the results obtained, conclusions are made about the relevance of the proposed aspects of the user stories and the overall performance of the framework.

Índice de Contenidos

Glosario

- **Esfuerzo**: cantidad de horas humanas requeridas para finalizar una actividad particular.
- **Historia de usuario (*US*)**: corresponde a una descripción en lenguaje natural que define las características de una funcionalidad que compone a un sistema.
- **Heurística**: corresponde a un proceso específico recomendado que no es posible validar o que está validado de manera empírica, sin ser aún demostrado.
- **Backend**: se refiere a la capa de lógica de un sistema en particular.
- **Frontend**: se refiere a la capa general de presentación en un sistema particular.

Capítulo 1

Introducción

El éxito en los proyectos de desarrollo de software depende del equilibrio de 3 variables fundamentales: el costo, el alcance y su duración. Esta última debe ser definida al inicio de los proyectos independientemente de la metodología de desarrollo que se esté utilizando, y sin embargo es la que presenta la mayor tasa de problemas en la industria. Alrededor del 60 % de los proyectos no logra finalizar dentro de los plazos que inicialmente fueron propuestos [23], lo cual se explica por el gran nivel de incertidumbre que se tiene al inicio de ellos en comparación con estimaciones realizadas más tarde en el desarrollo [12].

La estimación del tiempo de un proyecto se traduce en estimar el **esfuerzo** total requerido para realizarlo, que se define como la cantidad de horas humanas que se deben invertir para desarrollar todas las funcionalidades que lo componen. El proceso de pronóstico del esfuerzo ha sido exhaustivamente estudiado desde varios enfoques, con métodos y metodologías asociadas a juicios expertos, modelos paramétricos y aprendizaje automático [42], teniendo la mayoría de estas investigaciones problemas debido a la antigüedad de los datos que se analizan, a la baja presencia de ellas en la industria, y por último y más importante, a su reducida aplicabilidad a desarrollos que utilizan metodologías ágiles, las cuales poco a poco se han convertido en un estándar común en la industria. Lo anterior se debe principalmente debido a que se utilizan datos de los proyectos que no están disponibles al inicio en metodologías ágiles, en donde la planificación tiende a ser cortoplacista y no suele haber un rumbo claro sobre la totalidad de funcionalidades que se implementarán. Surge entonces la

necesidad de explorar nuevas formas compatibles con el agilismo para predecir el esfuerzo. Es también relevante considerar que lo único certero respecto a la estimación de esfuerzo en proyectos de software, es que en definitiva ésta jamás será certera [50]. Por lo tanto, es de esperar que todas las estimaciones de esfuerzo tengan un grado de incertidumbre en sus pronósticos. Luego, es necesario acompañar el desarrollo de un método predictivo junto a la definición de una estrategia para transparentar esta incertidumbre, sobre todo considerando que es importante que los clientes dueños de los proyectos se puedan alinear con la idea de procesos inciertos [6].

En este trabajo se utilizó un dataset actualizado de proyectos de software sobre el cual se propuso un método de aprendizaje automático para realizar pronósticos del esfuerzo requerido para completarlos, basado en las historias de usuario de cada uno. Estas últimas corresponden a la forma de documentar funcionalidades que se sugiere en las metodologías ágiles modernas. El dataset tiene en total 12 proyectos de la empresa de desarrollo de software *Nursoft* y 326 historias de usuario con atributos en bruto presentes en la base de datos donde residen, a los cuales se referenciará de ahora en adelante como **atributos base**. Sobre estos se propuso además la incorporación de nueva información que mejore la precisión del pronóstico completo.

1.1. Objetivos

1.1.1. Objetivo general

- Validar un método de aprendizaje automático para pronosticar al inicio de proyectos de desarrollo de software el esfuerzo requerido para finalizarlos, a partir de atributos relevantes propuestos de las historias de usuario que los componen.

1.1.2. Objetivos específicos

- Construir una herramienta que permita a Nursoft agilizar futuras investigaciones sobre estimación de esfuerzo usando aprendizaje automático.

- Explorar y seleccionar distintos modelos de aprendizaje para realizar el pronóstico.
- Evaluar el impacto de la ingeniería de atributos en el rendimiento de la solución propuesta.
- Definir una estrategia para estimar la incertidumbre del pronóstico.
- Mejorar el rendimiento de la metodología actual basada en Planning Poker utilizada en Nursoft.
- Comparar el rendimiento obtenido con investigaciones similares del estado del arte.

1.2. Metodología

La metodología utilizada en el presente trabajo se basa en CRISP-DM, propuesta por Wirth et al. [52], que define una serie de pasos que permiten construir y evaluar modelos sobre conjuntos de datos que se desean explorar. Se definen en ésta las etapas de entendimiento del negocio, entendimiento de los datos, preparación de los datos, modelamiento de los datos, evaluación del modelo y la puesta en producción de los modelos.

En primer lugar, la etapa de entendimiento del negocio se realiza explorando el estado del arte de estimación del esfuerzo en proyectos de software y realizando un análisis del proceso actual que realiza la empresa de la cual provienen los datos.

Luego, y para poder entender la estructura de los historias de usuario que se utilizarán para realizar predicciones, se realiza un análisis de los atributos de las historias de usuario disponibles en los proyectos, enfocándose en la utilidad que tienen estos en los pronósticos. Se proponen además un conjunto nuevo de atributos relevantes basados en las características más importantes que afectan el esfuerzo requerido para realizar funcionalidades.

A continuación, se agregan los atributos propuestos previamente a las historias de usuario y se transforman estas a formatos que se pueden utilizar en técnicas de aprendizaje automático.

Subsiguientemente, se exploran, evalúan y seleccionan distintos modelos de aprendizaje automático para construir el *framework* de pronósticos de esfuerzo, haciendo énfasis en modelos que realicen selección de atributos de manera inherente para poder visualizar y analizar la importancia de cada uno de los atributos utilizados para el pronóstico de un proyecto.

Posteriormente, se exploran y refinan métodos para poder obtener distribuciones de probabilidad para visualizar la incertidumbre del pronosticador seleccionado durante el pronóstico de un proyecto.

Finalmente, se analizan los resultados finales de los pronósticos obtenidos realizando simulaciones con los proyectos terminados en la empresa *Nursoft* y se realiza una comparación con el estado del arte y el proceso actual de la empresa, utilizando métricas estándar en el rubro.

1.3. Estructura de la memoria

En el capítulo 2 se define con más detalle la problemática a tratar y el contexto en el cual se estará validando el método propuesto, explicando la procedencia de los datos que se utilizarán para realizar el pronóstico del esfuerzo y los procesos internos actuales que se usan para pronosticar.

En el capítulo 3 se resume la literatura referente a metodologías ágiles y máquinas de aprendizaje, explicando cómo se pre-procesan los datos, cómo se describen los modelos probados para la predicción y cómo son estos seleccionados. Además, se explican los métodos utilizados para visualizar la incertidumbre de los pronósticos.

En el capítulo 4 se hace una revisión exhaustiva del estado del arte de la problemática de predecir el esfuerzo de proyectos de software, haciendo énfasis en los distintos métodos para realizar pronósticos, en los atributos y características interesantes que utilizan los estudios para sus pronósticos y en las métricas estándares definidas en el rubro para realizar comparaciones correctas.

En el capítulo 5 se presentan los resultados obtenidos al realizar pronósticos con el *framework* y se visualiza el rendimiento obtenido.

Finalmente, en las conclusiones se analizan los resultados obtenidos, realizando una comparación con el estado del arte y el proceso actual utilizado en la empresa de donde provienen los datos.

Capítulo 2

Problema y Contexto

En todo tipo de proyectos es necesario realizar una estimación inicial de su duración. Aquellos enfocados en la construcción de software no están exentos a esta regla, siendo particularmente en este tipo de proyectos una tarea compleja considerando la inherente incertidumbre que existe en el rubro. Esta incertidumbre proviene desde varios factores que han sido abordado en diversos estudios [42], entre los cuales se destacan la existencia de requerimientos iniciales inmaduros, falta de visión del producto final, riesgos no contemplados, factores técnicos y factores externos que no son inicialmente identificados. Aún con la alta incertidumbre que es asumida en la industria, sigue siendo necesario estimar la variable temporal al inicio debido a que no es factible iniciar un desarrollo de software sin tener al menos una noción de los plazos iniciales de término, ya que de esto dependen otras variables, tal como el cálculo de costo monetario y humano del proyecto y su factibilidad. Además, ha sido comprobado en varios estudios que es ésta la variable que tiene mayor impacto en el éxito de los proyectos o en su eventual fracaso [42] [13].

La investigación en el área de la estimación y pronosticación del tiempo necesario para finalizar los proyectos se ha intensificado en la última década, en donde se han obtenido métodos basados en juicio experto, métodos por analogías con proyectos similares, métodos que utilizan modelos paramétricos y métodos que utilizan aprendizaje automático. Las técnicas de esta última índole han probado ser las más eficientes y con menos sesgo que las demás [42],

a expensas de una reducción en la transparencia de cómo se está en verdad realizando la predicción (*Black-Box effect*), especialmente en métodos basados en redes neuronales [30]. Todas estas técnicas buscan estimar el tiempo en función del *esfuerzo* requerido para finalizarlo, enfocándose en la obtención de una estimación precisa y única sin prestar mucha atención a la variabilidad de los pronosticadores.

En particular, se tiene que en la empresa de desarrollo de software *Nursoft*, que se especializa en desarrollo de software a medida, se tiene un error más bajo que la media observable en la industria. En esta empresa, se utiliza la metodología de juicio experto *Planning Poker*, que es un estándar en las metodologías modernas de desarrollo (metodologías ágiles).

El esfuerzo de un proyecto de software en la empresa sigue las convenciones generales de la industria, y se define como la cantidad de horas humanas que se deben invertir para su finalización. Esto considera todos los distintos roles que participan en el desarrollo, tales como desarrolladores, diseñadores, arquitectos de software, encargados de calidad y administradores de operaciones. La estimación del esfuerzo total del proyecto se realiza sumando la estimación del esfuerzo de cada uno de las funcionalidades que lo componen. Dada la utilización del modelo ágil en los procesos, los requerimientos se representan como historias de usuario [37]. La documentación en el formato de historias de usuario corresponde a una explicación en lenguaje natural sobre lo que se desea construir y define una cierta cantidad de características particulares que se deben cumplir. El esfuerzo de cada historia de usuario se traduce a partir de una métrica especial de la metodología *Scrum*, llamado puntos de historia de usuario. Estos puntos son consensuados y generalmente se interpreta que una unidad de punto de historia de usuario equivale a un número horas de trabajo humano específicas.

El proceso de estimación actual consiste en realizar el proceso de *planning poker* para un proyecto, y luego definir el valor en horas de cada punto asignado. Generalmente, se utiliza la métrica de que cada punto equivale a 5 horas de esfuerzo humano. Por otro lado, el proceso actual del registro del esfuerzo real requerido en cada historia de usuario se realiza a través de la plataforma *BlackSheep*. El proceso se explica con mayor detalle en el anexo A.

Dado el contexto anterior, surge la necesidad de definir un nuevo proceso de pronóstico que

permita disminuir el error entre los pronósticos de esfuerzo iniciales y el esfuerzo total final requerido para el proyecto. Dicho proceso debe considerar toda la información relevante de las historias de usuario, asegurándose de que dicha información pueda ser fácilmente deducible en potenciales nuevos proyectos. Dicho en otras palabras, es necesario que la información del pasado a utilizar para pronosticar, pueda ser añadida a los proyectos antes de que inicie el desarrollo. El proceso debe poder proveer además una representación gráfica de la distribución de probabilidad de la estimación para visibilizar y transparentar la incertidumbre del proceso de desarrollo. Por último, éste debe poder reducir el error en la empresa *Nursoft* en comparación al proceso actual de pronóstico que se utiliza.

Capítulo 3

Marco Teórico

3.1. Metodologías contemporáneas de desarrollo

En la industria de desarrollo de software se han creado diversas metodologías, modelos y técnicas para poder ordenar un proceso que se caracterizaba en sus inicios por ser muy desordenado, desestructurado y caótico, y en donde abundaban los retrasos y clientes insatisfechos, a una magnitud tan grave que incluso en el año 1968 se comenzó a hablar de una crisis [38]. Fue gracias a esta crisis y a numerosos y catastróficos fallos que el área de desarrollo y gestión de proyectos de software comenzó a evolucionar y madurar hasta encontrar formas de ordenar los procesos [45]. Esta estructuración de procesos se formalizó en los años 70's y 80's, para establecerse como una disciplina fija con el nombre de Ingeniería de Software, que se enfoca en mejorar los procesos de todos los aspectos de la producción de software, desde las primeras fases de consolidación del alcance de los productos, hasta su mantención una vez desarrollados.

Para efectos del presente marco teórico y con el objetivo de clarificar conceptos que suelen mezclarse en la práctica, es importante tener en mente que un modelo es muy distinto a una metodología, en el sentido de que el primero describe lo que se debe hacer, mientras que el segundo cómo hacerlo. Un modelo es entonces descriptivo, mientras que una metodología es prescriptiva.

Los avances más significativos y revolucionarios en ingeniería de software corresponden a aquellos que han definido y estructurado todo el ciclo de vida de los proyectos. Entre los principales modelos que se han propuestos para definir cómo debe ser su ciclo de vida, se destacan *Waterfall*, *V-model*, *Spiral Model* y *Agile* como los más relevantes [45]. Este último es el más reciente y el que presenta el mayor porcentaje de éxito [3] en dónde el 39 % de los proyectos finaliza sin dificultades, mientras que en el resto de los modelos solo el 11 % lo hace. Además este tiene un alto porcentaje de adopción en la industria moderna, incluyendo empresas que se forman aplicando las metodologías del modelo desde el inicio y empresas que poco a poco han ido migrando a las metodologías ágiles en busca de hacer más eficientes y adaptivos sus procesos.

3.1.1. Modelo clásico

El modelo en cascada (*Waterfall*) es considerado como clásico en el área de desarrollo de software y corresponde a uno de los primeros intentos de darle estructura el ciclo de vida de proyectos de software. Se caracteriza por definir un proceso lineal de desarrollo en donde es fundamental que los requerimientos del software que se quiere construir estén definidos a priori [7].

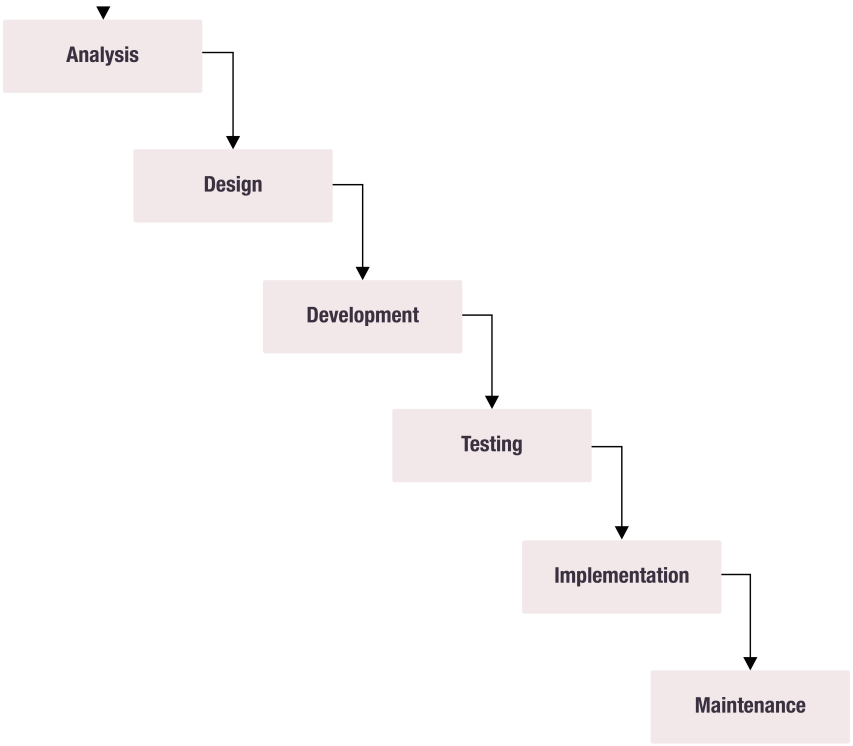


Figura 3.1: Modelo de ciclo de vida en cascada [7].

En este modelo las etapas están bien definidas, sin que ocurra traslape. En resumen, el software es primero definido y diseñado según los requerimientos previamente analizados y definidos por le cliente y el equipo, luego es desarrollado en base a dicha definición, y por último es puesto en producción y entra así a un ciclo de mantención.

Los proyectos que utilizan el modelo en cascada son relativamente más fáciles de interpretar. Sin embargo, en este modelo resulta ser muy importante recolectar correcta y prontamente todos los requerimientos que el cliente, la organización o quién sea que lo desea construir tenga en mente. Por lo mismo, una de las desventajas más características y el por qué la industria está generalmente abandonándolo, es que aquellos requerimientos que se dejan fuera no pueden ser incorporados de manera fácil [7]. Muchas veces es necesario realizar un nuevo proyecto para poder iterar sobre los construidos anteriormente. Por otro lado, los clientes y los impulsores de la construcción de software son humanos y tienden a cambiar de

opinión durante el proceso de desarrollo respecto a varias funcionalidades, lo que hace que cambios que pueden ser fundamentales para el éxito del proyecto no puedan ser incorporados prontamente.

3.1.2. Modelo ágil

Si bien han surgido varios modelos que buscan eliminar o mitigar los problemas definidos anteriormente en el modelo clásico, el más relevante corresponde al modelo ágil de desarrollo. Este modelo se enfoca en la capacidad de recibir requerimientos de forma adaptiva para solventar todos los problemas anteriores.

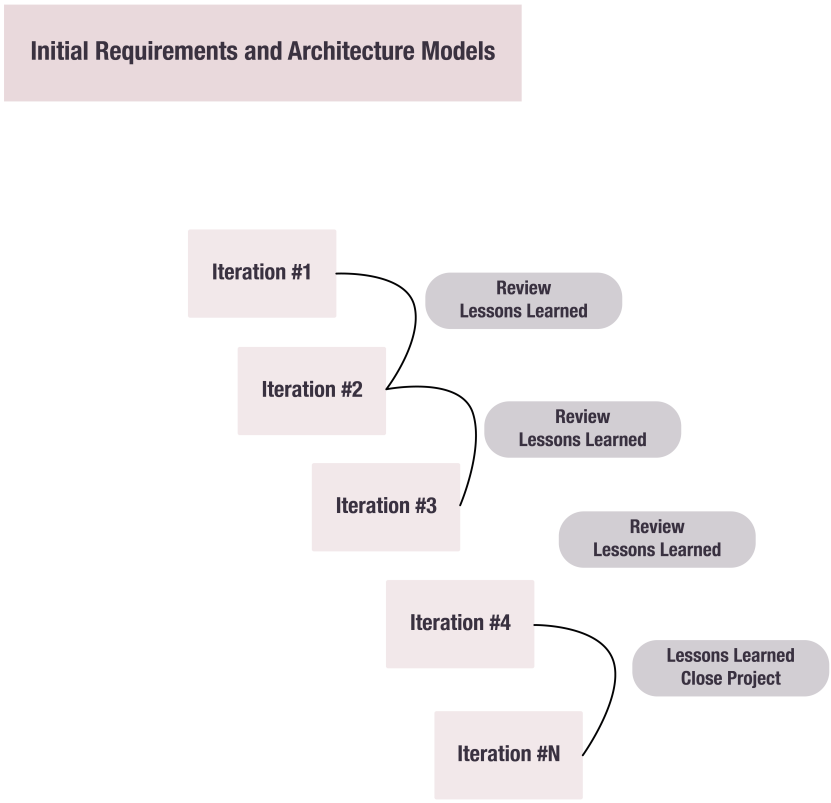


Figura 3.2: Modelo de ciclo de vida Ágil [7].

Las metodologías del modelo Ágil (o metodologías ágiles), responden al llamado *Agile Manifesto*, que corresponde a un documento en donde se encuentra la definición y especificación de lo que significa que una metodología sea ágil. Dicho manifiesto se reduce a 12 principios específicos que buscan aumentar la calidad del software que se produce, centrándose en la adaptabilidad, la entrega de valor continuo y equipos autónomos [8]. Hoy en día existe una tendencia por la industria a adoptar las metodologías ágiles como estándar en sus procesos, en donde la metodología de *Scrum* es la que lidera dicha tendencia [37]. La acelerada adopción de metodologías ágiles por parte de las empresas se debe al gran éxito que ha tenido el modelo ágil a la hora de asegurar que los proyectos de desarrollo de software terminen a tiempo y dejando satisfechos a los clientes, además de la gran versatilidad y adaptabilidad a los cambios en los requerimientos que ofrece.

Incluso con la adopción de metodologías ágiles, los proyectos deben de igual manera manejar correctamente su alcance temporal para lograr entregar las soluciones a tiempo. Esto es especialmente importante considerando que el pronóstico certero de la duración inicial de un proyecto aumenta significativamente la probabilidad del mismo de ser completado [42]. No adoptar una metodología de predicción de esfuerzo inicial puede traer consecuencias de distinta índole, como por ejemplo el retraso de proyectos que dependen de la entrega, deteriorar la percepción de excelencia que tienen los clientes, y en el peor caso, que esto provoque una pérdida de competitividad en la industria. Lo anterior se hace mucho más relevante considerando que para el año 2015, alrededor del 40 % de los proyectos de software presentaron atrasos relevantes con respecto a las estimaciones iniciales de tiempo y/o esfuerzo que se realizaron. Producto de estos atrasos, alrededor del 20 % se vio afectado en mayor magnitud, causando el fallo del proyecto [13] [23].

3.2. Máquinas de aprendizaje

Un algoritmo de aprendizaje automático o de máquina de aprendizaje corresponde a aquel que puede aprender de un conjunto de datos. Se dice que un algoritmo aprende de una experiencia E obtenida al realizar una tarea T con medida de rendimiento P, si su rendimiento en la tarea T mejora por la experiencia E. Existen una gran variedad de experiencias E, tareas

T y formas de medir el rendimiento de ellas P [40].

Se dice que el proceso de aprendizaje de un algoritmo es supervisado si este conoce el resultado que se espera luego de realizar una tarea en particular. Estos algoritmos se enfocan en observar varias muestras de un vector aleatorio x y un vector de valores asociados y para poder predecir y con x, usualmente estimando $p(y|x)$. El término supervisado proviene de la analogía de que el vector y se puede interpretar como las respuestas correctas a un problema que son proveídas por un instructor o profesor, quien le va mostrando a la máquina qué hacer y qué rendimiento está teniendo. Un algoritmo no supervisado, por otra parte, es aquel en donde el algoritmo no tiene claridad sobre el resultado que se desea obtener y generalmente intenta aprender de manera explícita o implícita la distribución de probabilidad [22] $p(x)$, o algún conjunto interesante de propiedades de dicha distribución. Siguiendo la analogía anterior, en este tipo de aprendizaje no existe el instructor, y el algoritmo debe poder de igual manera aprender algo acerca de la tarea que está realizando.

Las tareas que el algoritmo debe aprender a mejorar son generalmente clasificadas en dos grandes familias: tareas de clasificación y tareas de regresión. En las del primer tipo, un algoritmo debe decidir entre un total de k categorías a la cual un conjunto de datos pertenece. Para resolver esto, el algoritmo generalmente debe resolver la función $f : \mathbb{R}^n \to \{1, \cdots, k\}$ que transforma una serie de características (representadas en un vector) a una categoría específica. Un ejemplo de este tipo de tareas es el reconocimiento de objetos y patrones en imágenes, donde el ejemplo clásico es el reconocimiento de números escritos a mano. Algoritmos de regresión, por otro lado, corresponden a aquellos cuya tarea es predecir un valor continuo a partir de algún conjunto de datos. Generalmente, el algoritmo busca la función $f : \mathbb{R}^n \to \mathbb{R}$. Un ejemplo de esto es la predicción de esfuerzo en proyectos de software a partir de características específicas.

En algoritmos supervisados, entrenar un modelo se define como la acción de entregar características de entrada X y resultados esperados y para que este pueda encontrar la función f que pueda predecir $p(y|x)$ de la forma más certera posible.

3.2.1. Entrenamiento

Cuando un modelo de aprendizaje automático supervisado aprende patrones de un conjunto específico de datos, se dice entonces que el modelo pasa por un proceso de entrenamiento. En él, el modelo ajusta las variables libres de su implementación (parámetros) utilizando como referencia el valor real de estos, con el objetivo de poder predecir de la forma más general y precisa nuevas observaciones.

Para poder realizar la correcta evaluación de los modelos, es necesario reservar un porcentaje del conjunto de datos que no sea utilizada para el entrenamiento de ellos. La forma más común de realizar esto es realizar la división del conjunto de datos en un subconjunto de entrenamiento (que será utilizado para entrenar el modelo) y un subconjunto de test (que será utilizado para evaluar el rendimiento del modelo a través del error obtenido al predecir este conjunto). Es también bastante común reservar otro subconjunto adicional, generalmente llamado de validación, para realizar evaluaciones intermedias en los modelos para, por ejemplo, seleccionar los hiper-parámetros más adecuados al problema y poder mantener el conjunto de test como una instancia única de evaluación de la generalidad de la solución se está proponiendo.

Cross-Validation

Realizar una división del conjunto de datos para obtener subconjuntos de entrenamiento, test y validación no es siempre factible o lo más óptimo, sobretodo cuando se tiene una baja cantidad de observaciones. Esto hace que, por ejemplo, los modelos sean entrenados con cantidades de datos demasiado pequeñas o evaluados en cantidades pequeñas también. Una alternativa bastante usada corresponde al método de *K-Fold Cross Validation*, en donde se toma todo la totalidad del conjunto de datos y se generan K instancias distintas denominadas $folds$. En cada uno de los $folds$, se realiza una división de los datos para obtener un conjunto de entrenamiento y de validación (o test si se utiliza para realizar evaluación de modelos) particular a cada una de los $folds$, haciendo que el rendimiento sea el promedio del error obtenido en el conjunto de evaluación (test o validación) de todos ellos. Este método es comúnmente utilizado en el proceso de evaluación de hiper-parámetros, pero puede ser de

igual forma utilizado en el proceso de evaluación de modelos siempre y cuando no se realice evaluación de hiper-parámetros en el mismo conjunto para asegurar que se está evaluando correctamente sobre datos que no hayan sido utilizados para tomar decisiones previas.

3.2.2. *Overfitting* y *underfitting*

Overfitting corresponde al fenómeno en el cual un modelo se ajusta demasiado bien para un conjunto específico de entrenamientos, lo que causa que pierda generalidad y no sea certero para observaciones nuevas o que el modelo no pudo ver. Esto se ve generalmente reflejado en el error de test, en donde el overfitting se observa generalmente cuando el error de entrenamiento mejora cada vez más, y el error de testing empeora. *Underfitting*, por otro lado, ocurre cuando el error de entrenamiento es muy grande, lo que representa que el modelo no puede capturar el patrón y la correlación entre el valor esperado y los atributos que componen a las observaciones.

3.2.3. Sesgo versus varianza

En modelos de aprendizaje estadístico se puede demostrar matemáticamente [1] que el valor esperado de la variable que se quiere predecir en el conjunto de test depende de dos variables fundamentales: La varianza y el sesgo. La primera se refiere al error producido por la sensibilidad a que cambie la estimación $\hat{y}$ si cambiamos el conjunto de datos utilizado para entrenamiento, lo que termina haciendo que el modelo aprenda el ruido asociado al conjunto de datos (*overfitting*). Por otro lado, el sesgo corresponde al error que proviene de supuestos erróneos que se hacen del conjunto de datos, que puede causar que el modelo no pueda detectar correctamente relaciones relevantes entre los atributos que se utilizan *underfitting*. Ambos tipos de error se pueden considerar como fuerzas opuestas y se puede visualizar esto matemáticamente a través de la ecuación que define el valor esperado del error de una

[1] Al menos para el error cuadrático

estimación:

$$E[(y - \hat{y}(x))^2] = Var\hat{y}(x)) + [Bias(\hat{y}(x))]^2 + \sigma^2 \tag{3.1}$$

Donde Var es la varianza del modelo, definida por:

$$Var(\hat{y}(x)) = E[\hat{y}(x)^2] - E[\hat{y}(x)]^2 \tag{3.2}$$

, $Bias$ el sesgo definido por:

$$Bias(\hat{y}) = E[\hat{y}(x)] - y \tag{3.3}$$

y σ^2 la varianza de la función real y. Se puede observar que el error óptimo se obtiene cuando se logra reducir tanto el sesgo como la varianza, lo cual en la práctica no es trivial de lograr debido a la naturaleza muchas veces contradictoria de estos.

3.2.4. Maldición de la dimensionalidad

La colectivamente conocida **maldición de la dimensionalidad** define varios fenómenos que ocurren cuando se analizan espacios con altas dimensiones, dentro de los cuales el más común corresponde al hecho de que el volumen del espacio en que viven los datos aumenta exponencialmente cuando aumenta el valor de la dimensionalidad (figura 3.3), provocando que existan una mayor cantidad de valores nulos (datos dispersos) al tener que existirá una mayor cantidad de instancias en donde para un atributo particular x se tenga un valor nulo en z. Por

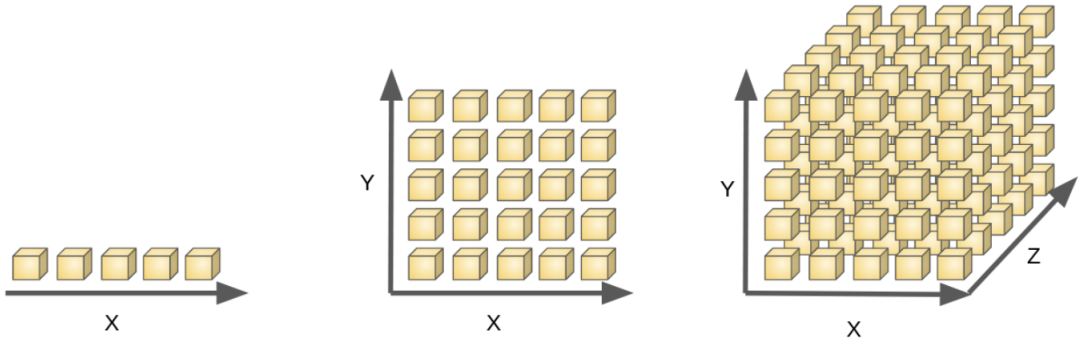


Figura 3.3: La imagen representa la velocidad con que aumenta el volumen de un espacio cuando aumentan la dimensionalidad. Se puede visualizar el cambio cuando la dimensión es compuesta por el eje x, para luego componerse por los ejes y y z.

En máquinas de aprendizaje lo anterior es bastante problemático ya que tener datos dispersos hace que los modelos de aprendizaje no puedan interpretar correctamente relaciones entre características en las observaciones de entrada, resultando en modelos clásicos y complejos que no convergen o que presentan *underfitting* u *overfitting*. En otras palabras, los valores nulos propios de un espacio disperso añaden ruido adicional al conjunto de datos de que la mayoría de los modelos no puede obviar, causando la necesidad de tener la menor cantidad de características posibles en los datos para reducir así el valor de la dimensión del espacio.

3.2.5. Regularización

La regularización corresponde a métodos propios de cada uno de los modelos existentes para evitar el *overfitting*. Generalmente, estos métodos implican añadir penalizaciones bajo ciertos escenarios y supuestos que permiten al modelo reducir el sobre-ajuste a los datos utilizados para el entrenamiento.

3.3. Ingeniería de atributos

Antes de utilizar datos como input en modelos de máquinas de aprendizaje para tareas de cualquier índole, es necesario preprocesarlos para reducir el ruido que estos puedan contener y para ayudar a los modelos a reconocer patrones de manera eficiente, y así aprender mejor. Se define como **atributo** la representación numérica de algún aspecto específico de los datos que se quieren utilizar como entrada. Por ejemplo, un atributo de una persona es su edad, que generalmente oscila en un rango de [0, 100], con excepciones en donde existen registros de gente que ha vivido más de 100 años. La **ingeniería de atributos** es el acto de extraer atributos de los datos en bruto y transformarlos a formatos que se puedan utilizar en modelos de aprendizaje automático, ya que una buena parte de estos no pueden manejar datos que no sean numéricos, tales como listas, objetos, texto, imágenes, audio, data binaria, etc. Este proceso se realiza antes de entrenar los modelos de máquinas de aprendizaje y es crucial debido a que la dificultad de entrenamiento puede ser reducida enormemente con los atributos correctos, que resulta en modelos más precisos o en procesos computacionales más baratos. El proceso es de suma importancia y generalmente se tiene que la mayor cantidad de tiempo en la utilización de modelos de máquinas de aprendizaje se invierte en él.

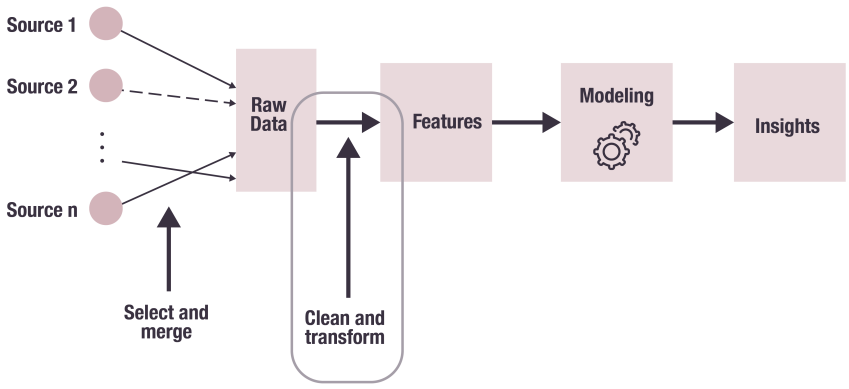


Figura 3.4: Lugar de la ingeniería de atributos en un flujo común de aprendizaje automático [54].

En un flujo natural de un proceso de aprendizaje de datos tal y como se muestra en la figura

3.4, la ingeniería de atributos se encuentra entre los datos en bruto y la obtención de atributos como tal.

3.3.1. Transformación de atributos

Los datos en bruto que se utilizan de entrada para un algoritmo de aprendizaje automático pueden venir en formatos incompatibles o inconsistentes, o simplemente en formatos que pueden incluso generar sesgo e incerteza en los modelos utilizados. Por esto, es necesario transformarlos en atributos numéricos que sean compatibles con la teoría matemática y estadística que soporta a los modelos y sobre los cuales es posible simular aprendizaje. Existen distintas técnicas para transformar datos numéricos, categóricos, de texto y de tipo más complejos, como por ejemplo imágenes o grabaciones multimedia. Dado el alcance del presente trabajo, solo se definirán los métodos relevantes que se utilizan para manejar datos numéricos y categóricos. Para el resto, se recomienda [54] como una introducción inicial.

Datos numéricos

Los datos numéricos pueden técnicamente ser usados directamente en la mayoría de los modelos sin necesidad de ser procesados. Es necesario sin embargo, asegurarse de que todas las observaciones que se utilizan estén completas y dentro de rangos lógicamente posibles.

Para atributos que representan contadores, es necesario analizar si utilizarlos directamente traerá problemas de rango de valores muy altos. Por ejemplo, el contador de visitas de un sitio web puede rápidamente aumentar a grandes valores de magnitud, lo cual puede producir un sesgo en los modelos de máquina de aprendizaje que utilizan la distancia euclidiana para su entrenamiento, y por tanto son más sensibles a la hora de detectar si un atributo tiene más importancia que otro o no. Además de esto, muchos algoritmos utilizan operaciones matemáticas que hacen que los valores aumentan o disminuyan su valor en varias órdenes de magnitud, imposibilitando entonces la convergencia de estos o alcanzado los límites computacionales de magnitud de valores numéricos [9]. Para solventar el problema anterior, se suele utilizar la técnica de cuantización, que consiste en crear intervalos

de distintas ordenes de magnitud y hacer que el atributo indique a cuál de estos intervalos pertenece un dato específico. Por ejemplo, la data en bruto podría traer un contador de visitas de $109,342,423 \sim 1{,}09 \cdot 10^8$ y el atributo terminar siendo 8, que corresponde a su orden de magnitud.

Dentro del mismo concepto de atributos numéricos con rangos muy altos, se puede aplicar la transformación logarítmica $log_a(a^x) = x$, que corresponde al inverso de la función exponencial. Esta simple y conocida transformación es una herramienta potente para trabajar atributos que tienen distribuciones con colas pesadas, que pone un mayor peso de probabilidad en la cola de la distribución respecto a la gausiana. Es posible generalizar la transformación logarítmica a toda la familia de transformaciones exponenciales definiendo la de *Box-Cox* que se define por la fórmula:

$$\tilde{x} = \frac{\frac{x^\lambda - 1}{\lambda}}{ln(x)} \tag{3.4}$$

Esta transformación define a toda la familia de transformaciones exponenciales que buscan reducir la varianza de los atributos. La idea es encontrar el parámetro λ por análisis de máxima verosimilitud o por métodos gaussianos que estabilicen la varianza. Los detalles de este método escapan al alcance del presente trabajo y es posible obtener más información en [26].

Datos categóricos

Una variable o dato categórico es utilizado generalmente para representar el valor que tiene un atributo en un conjunto de categorías. Por ejemplo, una categoría podría representar la nacionalidad de una persona, en donde los posibles valores equivalen a todas las nacionalidades existentes en el mundo. Este tipo de atributos no pueden ser ingresados de forma directa a los modelos ya que la cardinalidad del tipo de datos enumerados genera sesgos en los modelos. Siguiendo el ejemplo de la nacionalidad $N \in \{\text{Suecia}, \text{Inglaterra}, \cdots, \text{Chile}\}$, una solución ingenua podría ser convertir estos valores a $N' \in \mathbb{R}$, en donde se le asigna un número real incremental a cada valor de la categoría. El problema de lo anterior es que los algoritmos de aprendizaje podrían erróneamente correlacionar el comportamiento del valor

que se quiere predecir al orden en que se definen las categorías [29].

La codificación *One-Hot*, permite transformar datos categóricos a múltiples atributos binarios que indican la presencia o no de una categoría en particular y soluciona la problemática anterior. Matemáticamente hablando, se convierte un dato que puede tomar valores en $V \in \{c_1, c_2, \cdots, c_n\}$ a un vector de atributos de la forma $X \in \mathbb{B}^n$. Por ejemplo, el dato $V = c_2$ se transformaría al atributo $X = (0, 1, 0, \cdots, 0)$. La principal desventaja de este método es que aumenta el número de atributos de los datos de entrada, ya que cada valor de la categoría se transforma en un nuevo sub-atributo cuya correlación se invisibiliza para los modelos.

Normalización de atributos

La normalización de atributos corresponde al proceso de cambiar su escala original de valores al multiplicarlos por alguna constante específica sin alterar su dimensión. Se realiza generalmente para cada atributo en particular y las técnicas más populares corresponden a la normalización min-max, a la normalización por varianzas (o estandarización) y al escalamiento por norma l^p. El proceso de normalización es bastante estándar en el entrenamiento sobre todo cuando se quieren utilizar modelos de índole lineal, ya que son los más sensibles a las escalas. Las técnicas vistas anteriormente también pueden considerarse cambios de escala, pero son generalmente parte de un proceso que depende mucho más de la naturaleza de los datos y la información que están representando.

La **escala min-max** convierte la escala original de los atributos de $\mathbb{R}$ al dominio $[0, 1]$, según:

$$\hat{x} = \frac{x - min(x)}{max(x) - min(x)} \tag{3.5}$$

en donde $\hat{x}$ corresponde a las observaciones transformadas.

La **estandarización**, por otro lado, transforma los valores de las observaciones para un atributo en particular, para que así la media de estos sea igual a 0, y la varianza igual a 1. La

transformación se define como:

$$\hat{x} = \frac{x - \mu(x)}{\sigma(x)} \tag{3.6}$$

en donde $\mu(x)$ es la media de las observaciones en un atributo específico, y $\sigma(x)$ su desviación estándar.

Es importante considerar que el escalamiento por min-max o por varianza pueden convertir conjuntos de datos inicialmente dispersos en densos, por lo que su uso debe ser previamente analizado.

La **normalización por norma** l^p se define según:

$$\hat{x} = \frac{x}{||x||_p} \tag{3.7}$$

En donde generalmente se utiliza la norma l^2 que corresponde a la distancia euclidiana desde el origen a un punto en el espacio y es la normas más frecuentemente usada en máquinas de aprendizaje. La norma l^1, por otro lado, es utilizada para discriminar entre elementos que son muy pequeños (pero no son cero) y los que sí lo son.

3.3.2. Selección de atributos

La selección de atributos corresponde al proceso de determinar aquellos atributos que deben ser incluidos en el entrenamiento y validación del modelo a utilizar [29]. El proceso de selección se enfoca principalmente en eliminar aquellos atributos que no aportan información a la predicción que se desea realizar o cuya información ya está presente en otros atributos (o en una combinación de varios atributos) haciendo que sean redundantes. Muchos modelos, especialmente aquellos basados en regresiones lineales, estiman parámetros para cada atributo de los datos, lo que tiene como consecuencia que la presencia de atributos sin información aumente la incertidumbre y reduzca así la efectividad de los modelos tal y como se puede observar en la figura 3.6. En la figura, los modelos basados en árboles de decisión se ven menos

afectados por la dimensión de los atributos debido a la naturaleza de sus implementaciones.

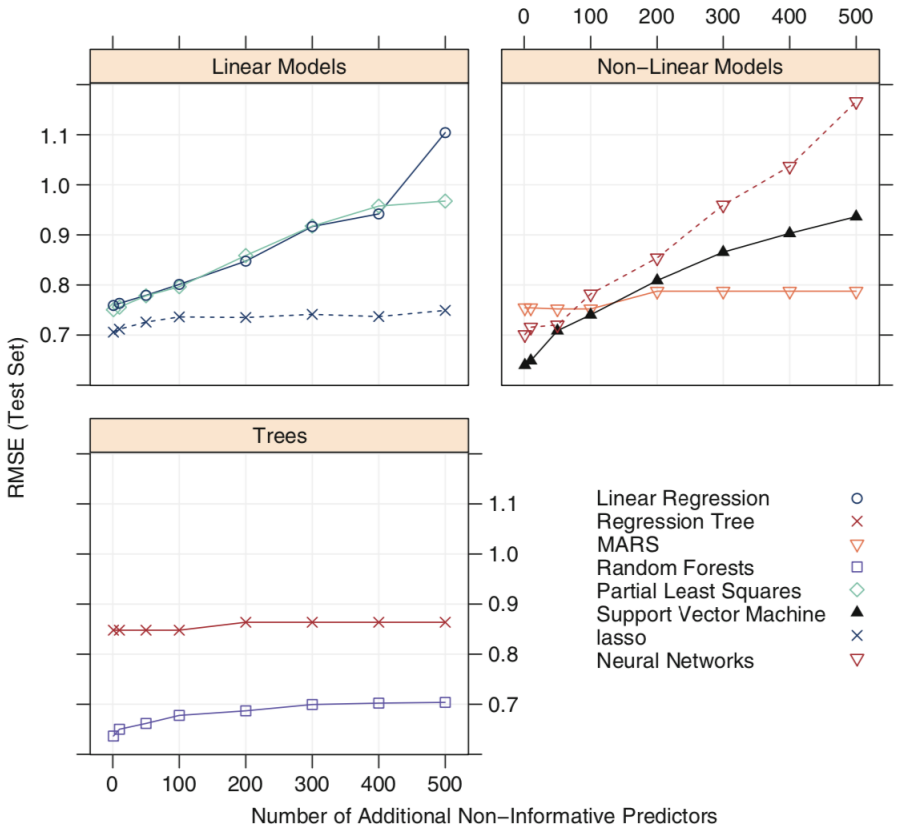


Figura 3.5: Comportamiento del error cuando aumenta la cantidad de atributos que no aportan información a las predicciones [29]. Es importante notar que la dimensión no afecta en gran medida a algoritmos basados en árboles.

La mayoría de los enfoques para reducir la cantidad de atributos que se desea utilizar para un modelo en particular se puede clasificar en tres grandes categorías: métodos envolventes (*Wrapping methods*), los métodos de filtrado y métodos incrustados.

Métodos envolventes

Los métodos envolventes son algoritmos de búsqueda que buscan encontrar la combinación de atributos que optimizan al algoritmo de máquinas de aprendizaje que se quiere utilizar. Estos métodos eliminan, añaden y recombinan los atributos de tal forma de poder buscar

entre todas las combinaciones posibles o en combinaciones guiadas por algún algoritmo de inteligencia artificial [29].

El método más simple para buscar atributos corresponde a *Classical forward selection*. En este algoritmo se tiene inicialmente un conjunto de atributos seleccionados vacío A_s y un conjunto de atributos candidatos por seleccionar A_c. En cada iteración, se generan varias instancias del predictor en donde cada una considera un atributo individual del conjunto de atributos candidatos A_c. Para cada instancia, se realiza un contraste de hipótesis estadístico sobre el predictor para determinar así la relevancia de cada uno de los atributos en la predicción. Se selecciona aquel atributo más significativo y se añade al conjunto de atributos seleccionados A_s. Como el objetivo es reducir la cantidad de atributos, se suele definir un mínimo de significancia para añadir atributos a A_s, que si no se cumple hace que el algoritmo termine de realizar evaluaciones y entregue el conjunto final. De esta forma, se genera un conjunto de atributos que son estadísticamente significativos y que se espera sea más pequeño que el conjunto inicial de atributos candidatos.

```
1  Create an initial model containing only an intercept term.
2  repeat
3  |  for each predictor not in the current model do
4  |  |  Create a candidate model by adding the predictor to the
   |  |  current model
5  |  |  Use a hypothesis test to estimate the statistical significance
   |  |  of the new model term
6  |  end
7  |  if the smallest p-value is less than the inclusion threshold then
8  |  |  Update the current model to include a term corresponding to
   |  |  the most statistically significant predictor
9  |  else
10 |  |  Stop
11 |  end
12 until no statistically significant predictors remain outside the model
```

Figura 3.6: Algoritmo de Classical forward selection [29].

La generación de combinaciones de atributos es un problema de inteligencia artificial y se puede abordar con métodos de búsqueda completa e incompleta, tales como algoritmos

genéticos, *simulated anhealing*, búsqueda por colonias de hormigas, búsqueda por partículas, *wolf-search*, entre muchos otros más. La definición de estos métodos queda fuera del alcance del presente trabajo [29].

La principal desventaja de métodos de esta categoría es que son computacionalmente costosos y requieren un gran esfuerzo para asegurar la convergencia a la solución óptima global del espacio de búsqueda definido por la combinación de atributos que se quiere probar. Además, a medida que aumenta la cantidad de atributos de las observaciones que se quieren predecir, aumenta también (y más rápido) el tamaño del espacio de búsqueda a una velocidad factorial.

Métodos de filtrado

Los métodos de esta categoría evalúan a los potenciales atributos antes de que se entrene el modelo, por lo que corresponde a un proceso de índole no supervisado. Estos métodos analizan información estadística de los atributos (como por ejemplo, la varianza) o a cada atributo de manera aislada. Esto provoca generalmente que en la práctica se seleccionen atributos importantes, pero que son redundantes. Además, si se utilizan hipótesis estadísticas para seleccionar a los atributos, ocurre el llamado problema de la multiplicidad, en donde pueden ocurrir falsos positivos en la selección de los parámetros [29].

El método más simple de ésta categoría consiste en eliminar aquellos atributos con una varianza baja o nula, ya que implica que el atributo en cuestión no aporta información significativa para diferenciar una observación de otra, tanto en problemas de regresión como de clasificación.

Métodos incrustados

Los métodos incrustados corresponden a métodos que utilizan modelos que en su implementación realizan selección de atributos de forma inherente. El caso más evidente de esto corresponde a la utilización de árboles de decisión (sección 3.5.2), en donde el modelo se va entrenando a medida que selecciona los atributos y sus valores específicos que mejor permiten particionar el conjunto de datos. Por tanto, durante el proceso de entrenamiento del

árbol, se van determinando aquellos atributos que son más relevantes para el proceso de partición del conjunto de datos en función a la cantidad de veces que el atributo es utilizado para realizar una partición en el conjunto. Una vez que el árbol ha sido entrenado, se puede acceder a la ponderación de importancia de cada uno de los atributos y bajo alguna heurística en particular, desechar aquellos cuya importancia sea relativamente baja para reducir así la dimensionalidad del conjunto completo [54].

La principal desventaja de estos métodos es que el modelo que es utilizado para realizar la selección de atributos genera un sesgo en los modelos subsiguientes que tomen las observaciones con los nuevos atributos acotados, por lo cual es ideal poder realizar la selección de parámetros del modelo selector en un conjunto de datos aislado del entrenamiento posterior, o en su defecto, a través de *cross-validation* que se explica en mejor detalle en la sección 3.2.1.

3.3.3. *Principal Component Analysis* (PCA)

PCA es una técnica usada ampliamente para reducir la dimensionalidad de los atributos, así como también para otros problemas como la compresión de datos sin pérdida de información, extracción de características en contextos de aprendizaje no supervisados y visualización de datos. Corresponde a proyectar datos que están en un espacio vectorial inicial de una dimensionalidad específica a una menor dimensión, de forma que la varianza en el nuevo espacio sea maximizada [9].

Matemáticamente hablando, si se tiene un conjunto de N observaciones definido por $\{x_n\}$, con $n = 1, \cdots, N$ y siendo x_n una variable euclidiana de dimensión D, el objetivo es buscar el valor óptimo de $M < D$ para maximizar la varianza del espacio vectorial proyectado. Sea la media de las observaciones del conjunto definidas por:

$$\overline{x} = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{3.8}$$

y sea entonces la varianza del espacio proyectado definida por:

$$\frac{1}{N}\sum_{n=1}^{N}\{u_1^T x_n - u_1^T \overline{x}\}^2 = u_1^T S\, u_1 \tag{3.9}$$

en donde S es la matriz de covarianza dada por:

$$S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \overline{x})(x_n - \overline{x})^T \tag{3.10}$$

y u_1 un vector unitario. El problema se reducen entonces a maximizar $u_1 S\, u_1$.

Una desventaja de PCA es que se pierde la interpretabilidad del nuevo espacio de atributos que es proyectado. Por ejemplo, si se describen observaciones de personas con los atributos de edad, renta promedio y quintil social, y se realiza PCA para reducir la dimensión a 2 componentes, no se podrá saber con certeza qué es lo que están representando los dos atributos que conforman el nuevo espacio.

Otra desventaja, por otro lado, es que PCA no funciona adecuadamente con variables categóricas, ya que no existe una noción correcta de distancia entre una categoría y otra para definir así la proyección de los atributos.

3.4. Modelos lineales de regresión

Como se mencionó en la sección 3.2, un modelo realiza regresión cuando necesita aprender a predecir un número real a partir de los atributos de entrada.

Los modelos de regresión lineales son aquellos que esperan que la función objetivo de una observación $\hat{y}(w, x)$ esté definida por alguna combinación lineal de los atributos que la componen. Esto se define matemáticamente por la fórmula:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p \tag{3.11}$$

donde w_i es el peso asociado al atributo x_i, con $i = 0, \cdots, p$.

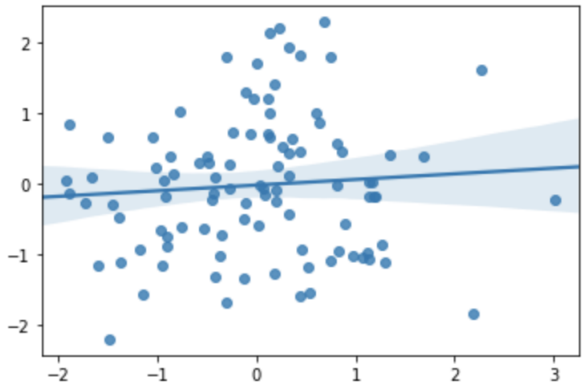


Figura 3.7: Ejemplo visual del objetivo de una regresión de índole lineal.

3.4.1. Mínimos cuadrados ordinarios

La técnica más sencilla para poder encontrar el vector de coeficientes $w = (w_1, \cdots, w_p)$, es minimizar el residuo de la suma de cuadrados entre las observaciones y la función objetivo. Esto resuelve matemáticamente el problema de la forma:

$$min_w(||Xw - y||_2^2) \tag{3.12}$$

donde X corresponde a la matriz de $n \times p$, donde n es el número de observaciones y p el número de atributos que tiene cada una. Gráficamente, lo que se está buscando es un hiperplano que logre pasar por todos los puntos del espacio vectorial de atributos.

Es posible añadir un parámetro de regularización (sección 3.2.5) al modelo anterior según la fórmula [43]:

$$min_w(||Xw - y||_2^2) + \alpha||w||_2^2 \tag{3.13}$$

que resuelve problemas inherentes a los mínimos cuadrados. Esta última es también conocida como regresión rígida, ya que limita los valores que pueden tomar los coeficientes, evitando

así que ciertos coeficientes tomen demasiada predominancia.

3.4.2. Lasso

Lasso es un método popular de regresión que utiliza la norma l^1 para obtener coeficientes del vector w que sean dispersos [20]. Esto hace que tienda a preferir soluciones con más pesos nulos (que valgan cero), que se traduce a reducir el número de características sobre las cuales la función $\hat{y}(w, x)$ es dependiente, lo cual puede ser beneficioso en ciertos contextos en donde se tienen muchos atributos que podrían no estar aportando valor a las predicciones. Matemáticamente, consiste en un modelo basado en la ecuación del modelo anterior 3.12, dada por:

$$min_w \frac{1}{2n} ||Xw - y||_2^2 + \alpha ||w||_1 \tag{3.14}$$

donde α es una constante y $||w||_1$ es la normal l^1 de los pesos.

3.5. Modelos no lineales de regresión

Los modelos lineales permiten entregar muy buenas predicciones cuando son utilizados con los hiperparámetros correctos y cuidando de tomar en cuenta los contextos en que cada uno funciona mejor. Sin embargo, si el supuesto original de linealidad entre la salida de y y las observaciones X no se cumple, el rendimiento decae considerablemente. Utilizar modelos no lineales, por otro lado, permite prescindir del supuesto de linealidad original y son recomendados cuando no se tiene noción de la relación entre los atributos y la salida final.

3.5.1. Máquinas de soporte

Las máquinas de soporte (SVM) corresponden a métodos no paramétricos para regresión y clasificación, y en donde su principal ventaja es que son efectivas en espacios con altas

dimensiones, aún cuando la cantidad de observaciones es menor que la cantidad de atributos de cada una.

Las SVM son más fáciles de entender inicialmente en un contexto con un problema de clasificación, que corresponde a asignar a una observación en particular una clase dependiendo del valor de sus atributos. Una SVM realiza clasificación al encontrar el hiperplano que maximiza el margen entre dos clases, que corresponde a la distancia entre los puntos más cercanos a una posible barrera de decisión. Dichos puntos se denominan los vectores de soporte.

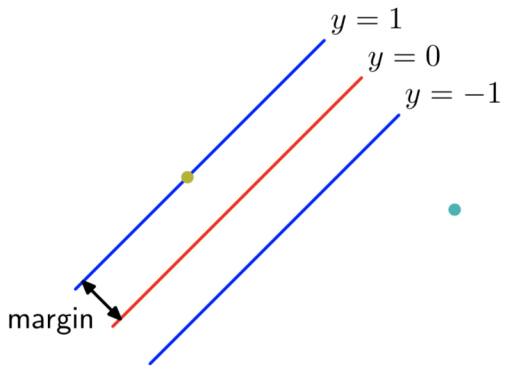


Figura 3.8: El margen corresponde a la distancia entre el hiperplano de decisión en color rojo el hiperplano azul que pasa por el vector de soporte de las clases [9].

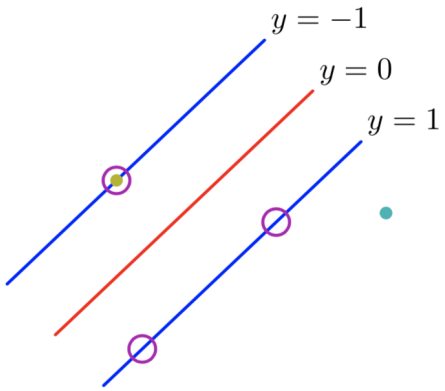


Figura 3.9: Los vectores de soporte encerrados por un círculo morado son utilizados para definir el margen [9].

Matemáticamente, una SVM está definida como:

$$max_{\alpha}\ L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)} \cdot x^{(j)} \rangle \tag{3.15}$$

Donde L corresponde al Langraniano y α a los valores de los vectores de soporte. En este

caso, los pesos no están presentes en la fórmula que se desea optimizar, ya que están definidos según:

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{3.16}$$

La ecuación 3.15 funciona cuando las variables son linealmente separables por un hiperplano, pero no es el caso en muchas aplicaciones de la vida real. Una forma de solventar esto es realizar una transformación del espacio original de los datos a una que es linealmente separable. Lo anterior se conoce como el método de kernel y es lo que hace las SVM las herramientas potentes que son. Para utilizar el método, se reescribe la ecuación 3.15 según:

$$max_\alpha \, L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \, K(x^{(i)}, x^{(j)}) \tag{3.17}$$

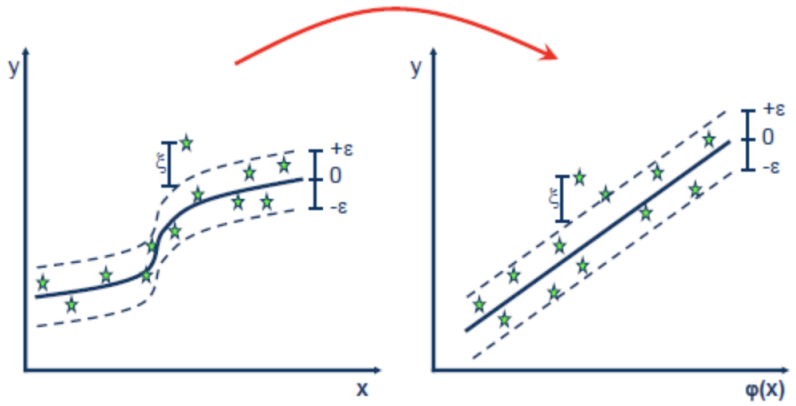


Figura 3.10: Noción de la transformación que realiza un método de kernel. Notar que en la izquierda se tienen datos que no son linealmente separables, y se llevan a un nuevo espacio en donde si lo son [47].

y se selecciona la transformación por kernel adecuada, dependiendo de las características de los datos que se desean trabajar. Entre los kernels más populares se encuentran el kernel

polinomial, que asume que la relación entre los atributos es polinomial y está dado por:

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d \tag{3.18}$$

Otro kernel bastantemente utilizado, es el kernel *Gaussian Radial Basis Function* (RBF).

$$k(x_i, x_j) = exp(-\gamma ||x_i - y_j||^2)) \tag{3.19}$$

Para regresión, se realiza una modificación a las máquinas de soporte originales, pasándose a llamar máquinas de soporte para regresión (SVR). En este tipo de máquinas, el margen se define como la distancia entre dos vectores de soporte específicos ϵ, que corresponden a observaciones puntuales del conjunto de datos.

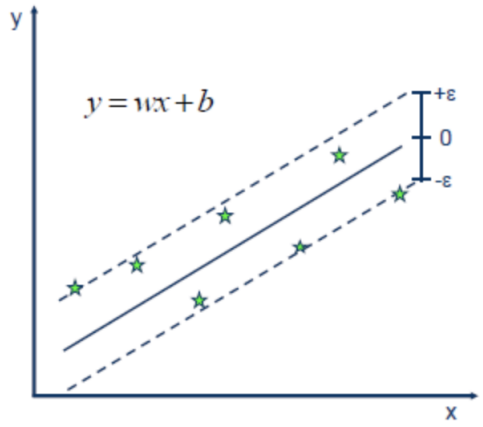


Figura 3.11: Definición del margen en SVR. Las líneas punteadas definen los hiperplanos paralelos al hiperplano de decisión sólido y son calculados según los vectores de soporte que se utilizan [47].

El hiperplano que se obtiene a través del cálculo de los márgenes es definido entonces por:

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i*) \cdot K(x_i, x) + b \tag{3.20}$$

3.5.2. Árboles de regresión

Los árboles de regresión son modelos que agrupan las observaciones en diversas áreas $R_1, \cdots, R_J$ dependiendo del valor de los atributos de cada una. Es fácil ver como actúa un árbol en un espacio de dos dimensiones tal y como se muestra en la figura 3.12.

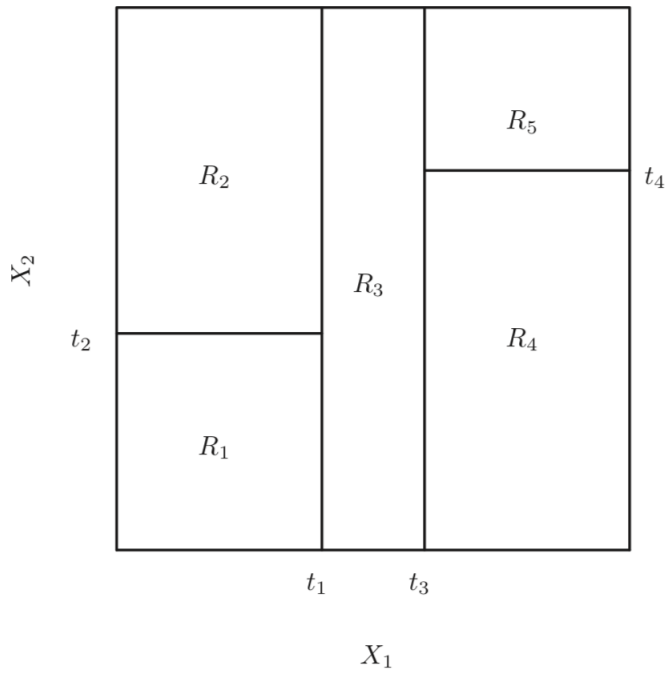


Figura 3.12: Visualización de las divisiones realizadas por un modelo de árboles de decisión en un conjunto de datos de dimensión 2 y con atributos X_1 y X_2. t_n corresponden a los valores pivote en donde se realiza una partición, y R_n a las regiones que son generadas [25].

El espacio definido por los atributos X_1 y X_2 es divido en 5 áreas de distinto tamaño definidas por los cortes $t_1, \cdots, t_4$. Estos cortes son las decisiones que se definen en el árbol cuya representación gráfica puede observarse en la figura 3.13.

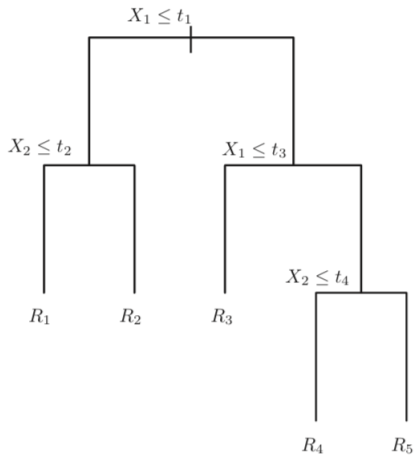


Figura 3.13: Visualización del árbol de decisión generando al particionar el conjunto de datos [25].

El valor que toma una observación con valores de atributos que la hacen pertenecer a un área R_j específica corresponde a la media de todos los valores que también caen en dicho cuadrante. El proceso de aprendizaje, por lo tanto, corresponde a realizar los cortes que mejor explican los distintos valores que pueden ir tomando las predicciones según el área en el que caen.

Matemáticamente hablando, se busca minimizar el error definido según:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{3.21}$$

en donde $\hat{y}_{R_j}$ es la media del valor de y para todas las observaciones del área R_j. En teoría, existen muchísimas formas de realizar los cortes para un espacio con observaciones, y explorarlas todas es computacionalmente infactible. Por tanto, para la construcción de dichas particiones se considera un enfoque *greedy* denominado partición binaria recursiva. En este procedimiento, se selecciona el atributo X_j y un corte s tal que se divida el espacio del atributo en dos regiones iguales $\{X|X_j \geq s\}$ y $\{X|X_j \leq s\}$ que lleve a la menor error posible.

El mayor problema con los modelos de árboles es que tienden a experimentar *overfitting*

debido a reglas que son demasiado estrictas en su construcción. Mientras más atributos utilice el árbol para construir un flujo de de decisión, más probable es que experimente el fenómeno. Para contrarrestar esto se realiza el proceso de poda de los árboles (*prunning*), en donde una vez que se tiene un árbol complejo y probablemente con *overfitting*, se realiza un proceso de búsqueda del subárbol que tenga el menor error en el conjunto de testing.

Es importante considerar que los árboles de decisión trabajan muy bien sobre datos con atributos que no siguen una relación lineal. En caso se que dicha relación lineal si exista, los árboles tienden a tener un performance menor que modelos más simple, como por ejemplo una regresión lineal rígida o con Lasso.

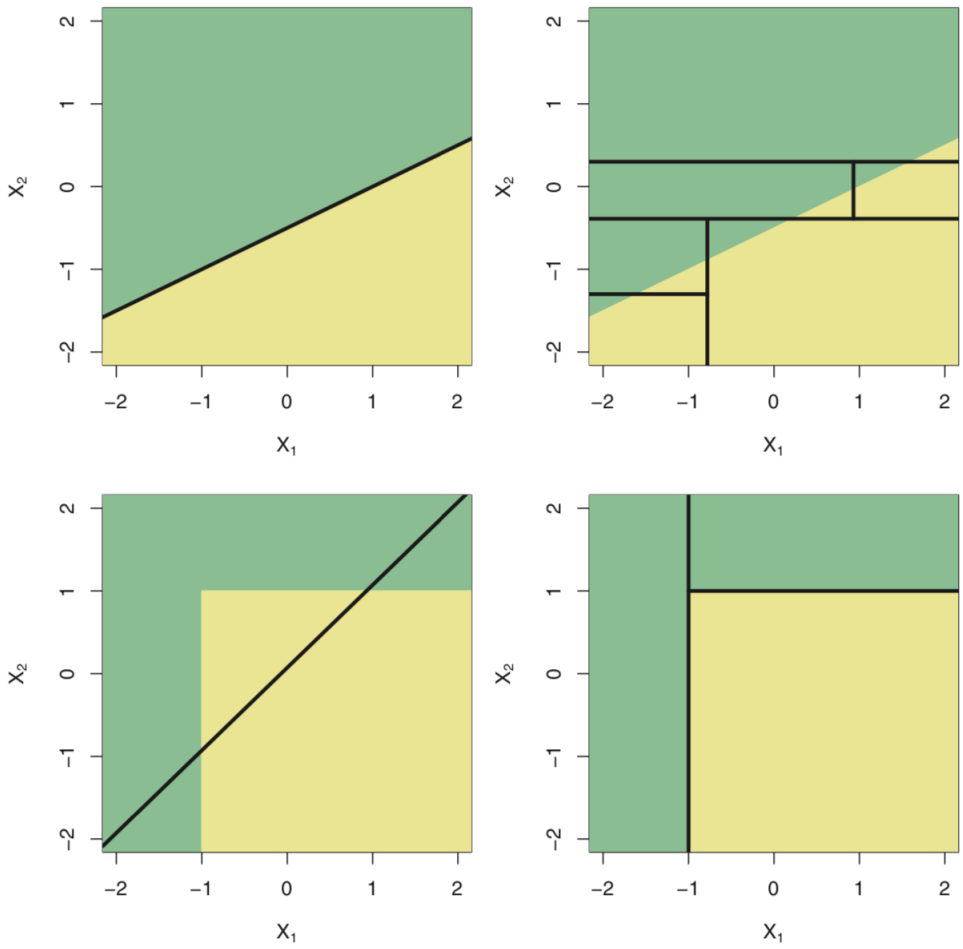


Figura 3.14: Comparación visual del rendimiento entre árboles de decisión y una regresión en un problema de clasificación. En las 2 figuras superiores se ejemplifica la clasificación en un dominio lineal, y en las inferiores en un dominio no lineal. A la izquierda se encuentra la división realizada por la regresión, y a la derecha la realizada por el árbol de decisión [25].

3.6. Optimización de modelos

Todos los modelos revisados anteriormente no funcionan correctamente si no se adaptan al problema que se quiere resolver. Partiendo primero por definir si la correlación entre los atributos y las predicciones son lineales o no, luego hay que refinar los hiperparámetros de cada

uno y utilizar diversas técnicas para evitar overfitting, underfitting y limitantes inherentes a su estructura a y los proceso que realizan.

3.6.1. Tuneo de hiperparámetros

Un **hiperparámetro** corresponde a alguna constante o heurística de un modelo que le da mayor flexibilidad para adaptarse a distintos conjuntos de datos y contextos distintos. Se les llama hiperparámetros debido a que en la literatura se suele llamar como **parámetro** a las variables libres (como los pesos en una regresión lineal) de cada uno de los modelos, cuyos valores son aprendidos durante el entrenamiento. Los hiperparámetros pueden ajustar regularización, inicialización de parámetros o modificar las heurísticas que se utilizan en el modelo, como por ejemplo, el tipo de kernel que se utilizará para realizar las transformaciones en una máquina de soporte.

La correcta selección de hiperparámetros es crucial para asegurar los mejores resultados posibles en cada uno de los modelos. Es imposible definir valores óptimos específicos para cada tipo de problema que se quiere trabajar en aprendizaje automático, por lo que estos tienen que ser calculados cada vez que se decide utilizar un modelo en un conjunto nuevo de datos. Los valores que son asignados provienen de heurísticas, desde la experiencia de los investigadores o desde la búsqueda empírica en el problema particular que se está abordando.

El proceso de tuneo de hiperparámetros corresponde a buscar el conjunto de hiperparámetros que minimiza el error de validación para un modelo y conjunto de datos específico. Esta búsqueda puede hacerse a mano a través de varios experimentos que se realizan en un subconjunto de observaciones limitada y aleatoria, o bien, a través de técnicas de validación como *cross-validation* (sección 3.2.1).

La selección de valores de un hiperparámetro se puede realizar con experimentación empírica tal y como se muestra en la figura 3.15. Aquí, un investigador debe observar los resultados del comportamiento del hiperparámetro y decidir su valor según lo que se puede interpretar de éstos.

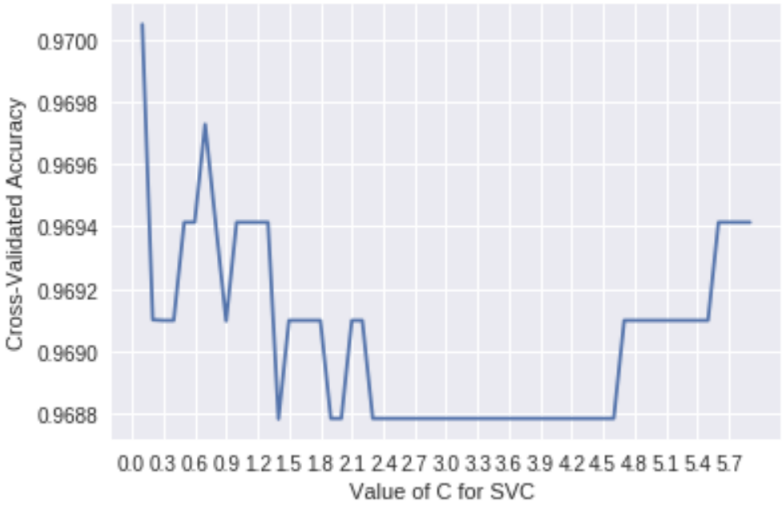


Figura 3.15: Ejemplo de investigación del comportamiento del error de una máquina de soporte de clasificación a medida que cambia el valor del hiperparámetro *C*.

Dado que el problema de encontrar el conjunto óptimo de hiperparámetros corresponde a un problema clásico de búsqueda de un espacio vectorial, es posible utilizar algoritmos de inteligencia artificial, como por ejemplo, algoritmos genéticos de búsqueda, *simulated annealing* y otros más que en conjunto escapan del alcance del presente trabajo. La forma más fácil de realizar una búsqueda más automatizada es recorrer la combinación de los distintos posibles valores que pueden tomar los hiperparámetros y seleccionar aquel conjunto que maximiza la precisión de las predicciones.

3.6.2. Métodos de ensamblaje

Los métodos de ensamblaje combinan las predicciones de varios modelos base con algún algoritmo específico para mejorar la generalidad y robustez de ellos. Esto es especialmente útil para modelos que tienden a experimentar *overfitting* en su implementación, tal y cómo sucede en árboles de decisión. Existen dos grandes familias de métodos de ensamblaje: métodos que promedian las predicciones de cada uno de los modelos base, o métodos que buscan iterativamente reducir el sesgo de ellos. En la primera familia se encuentra *Bagging* y *Random forests*, y en la segunda los métodos de *AdaBoost* y *Gradient Tree Boosting*. Estas familias se

conocen también en la literatura por *averaging methods* y *boosting methods* respectivamente.

Bagging

Los métodos de *Bagging* pertenecen a una clase de algoritmos que generan varias instancias de un modelo en varias versiones del dataset original de los datos originales utilizados para realizar el entrenamiento. El objetivo de realizar esto es reducir la varianza del estimador original (por ejemplo, en árboles de decisiones). Generalmente, los métodos de bagging funcionan mejor con modelos complejos, como por ejemplo, árboles de decisión de mayor profundidad. Matemáticamente, si se tiene un conjunto de entrenamiento D con n observaciones, el proceso de *bagging* genera m nuevos conjuntos de datos D_i, cada uno con un tamaño $n\prime$. El número $n\prime$ depende del tipo de algoritmo que se desea utilizar.

Cuando se utilizan subconjuntos de datos tal que $n\prime < n$, entonces se habla de *Pasting*. Aquí, cada uno de los subconjuntos no tiene elementos repetidos y se asegura que cada subconjunto tiene un tamaño menor al original. Si se permite que los elementos se repitan, tal que ya no se pueda afirmar que $n\prime = n$, se habla de *Bootstrap* en donde durante el muestreo se permite repetición.

La principal desventaja de los métodos de *Bagging* es que cada instancia del modelo original no tiene correcta información con respecto al rendimiento de los demás modelos. Esto es especialmente complejo cuando se tiene que por las características del modelo base seleccionado, este tiende a predecir un área en conjunto en cada instancia distinta de él, haciendo que el rendimiento no mejore (figura 3.16).

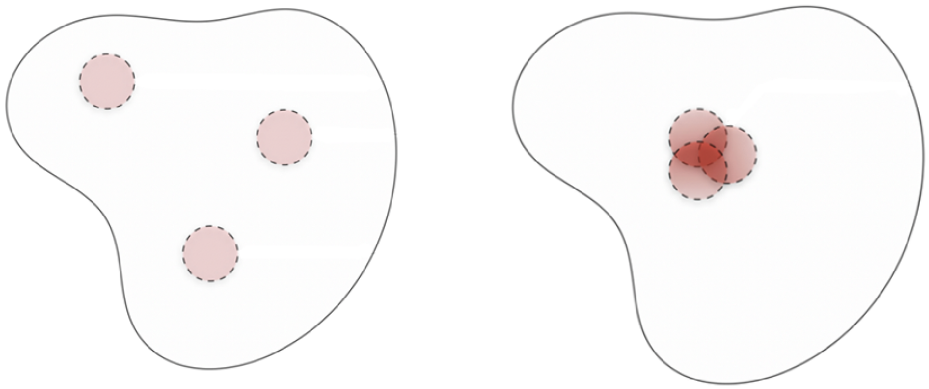


Figura 3.16: Demostración gráfica de los problemas comunes que tiene bagging. En rojo se pueden apreciar las áreas en las que cada instancia del modelo no realiza buenas predicciones en el espacio de atributos. En este caso, y para ejemplificar, se tienen 3 instancias distintas. En la figura de la izquierda, si el espacio de atributos cae en la zona roja de la instancia *A*, las instancias *B* y *C* podrán de igual manera estimar correctamente y mitigar el mal resultado de *A*. En otro espacio de atributos con una zona de mala predicción más densa, se tiene que bagging no logrará mejorar el error, ya que todos las instancias estarán igual de equivocadas [17].

Random forest

Random forest, o bosques aleatorios, son un método inspirado en bagging en donde se utilizan árboles de decisión como modelos base y utilizan técnicas de combinación y perturbación especialmente diseñadas para ellos. Cada instancia de árbol es entrenada en un subconjunto de atributos obtenido por bootstrapping. Además, el proceso de división del conjunto original es realizado con un subconjunto de los atributos restantes, sin considerarlos todos. Esto añade aleatoriedad a cada una de las instancias del árbol, permitiendo así disminuir el *overfitting*.

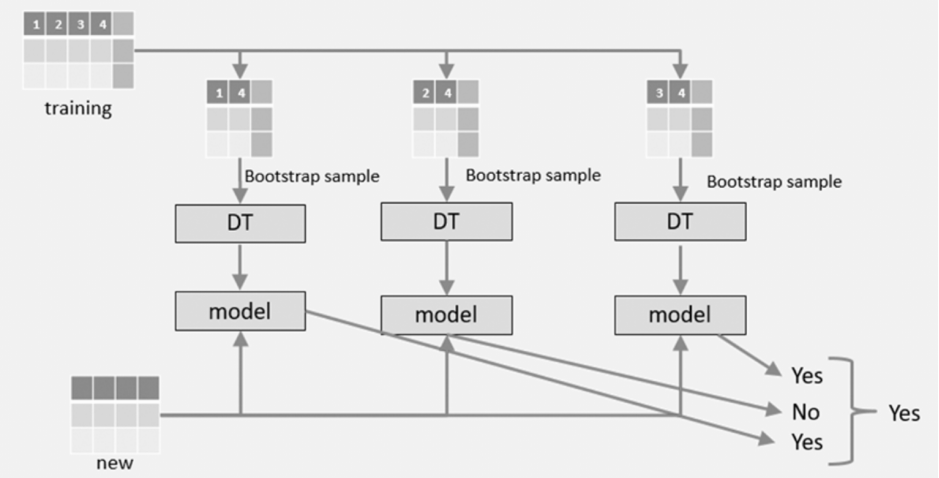


Figura 3.17: Estructura general de random forest ejemplificado en un problema de clasificación binario [48].

Existe una variación del algoritmo de *Random forest* llamado *Extreme Random Forests*, en donde la aleatoriedad va un paso más allá y en lugar de computar el mejor split en base al subconjunto de atributos previamente seleccionado, el split es asignado de manera aleatoria. Esto logra, en ciertos contextos, reducir aún más la varianza.

AdaBoost

El principio fundamental de *AdaBoost* es entrenar una serie de modelos débiles (que por si solos tienen un error importante) de manera repetitiva en versiones modificadas del conjunto original de los datos. En cada una de las iteraciones del algoritmo, se le asignan pesos $w_1, w_2, \cdots, w_n$ a cada una de las n observaciones. Inicialmente a los pesos se les asigna un peso igual de $w_i = \frac{1}{N}$, haciendo que cada una de las observaciones tenga la misma relevancia. A medida que las iteraciones pasan, aquellas observaciones que han sido incorrectamente predecidas aumentan el valor de su pesos, mientras que las que si fueron predecidas correctamente, disminuyen el suyo. Esto permite que cada instancia del modelo débil pueda obtener información del modelo que pudo aprender de las observaciones, haciendo que la nueva instancia se concentre en aquellos atributos que hayan tenido el peor rendimiento.

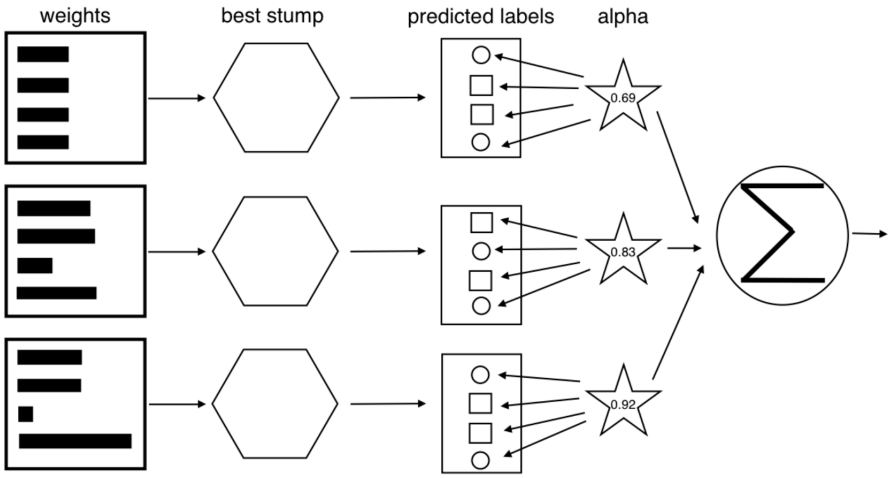


Figura 3.18: Esquema gráfico que contiene el proceso de AdaBoost.

Gradient Boosting

El algoritmo de *Gradient Boosting* es bastante similar a *AdaBoost*, con la excepción de que en lugar de utilizar pesos para darle relevancia a ciertos atributos en cada iteración, se utiliza el concepto de gradiente, que pare efectos explicativos de definirá de manera cuadrática. La predicción de una instancia $F_{m+1}(x)$ en la iteración $m + 1$ está dada por:

$$F_{m+1}(x) = F_m(x) + h(x) \tag{3.22}$$

Para encontrar el valor de $H(x)$, la solución comienza asumiendo que una función h perfecta implica que:

$$h(x) = y - F_m(x) \tag{3.23}$$

Con esto, F_{m+1} intenta corregir el error obtenido por el estimador de la iteración anterior.

En el caso de *Gradient Tree Boosting* se utilizan árboles como estimadores base.

3.7. Estimación de distribución por método de kernel

La estimación de densidad por método de kernel (*kernel density estimation*) corresponde a un método basado en vecinos y es sucesor del método clásico de obtención de distribuciones, que corresponde a la generación de histogramas [24].

Los histogramas son visualizaciones simples de la distribución de un conjunto de datos específico, donde se definen intervalos (o bins) y se van generando rectángulos en cada uno de ellos para percibir la cantidad de observaciones en él.

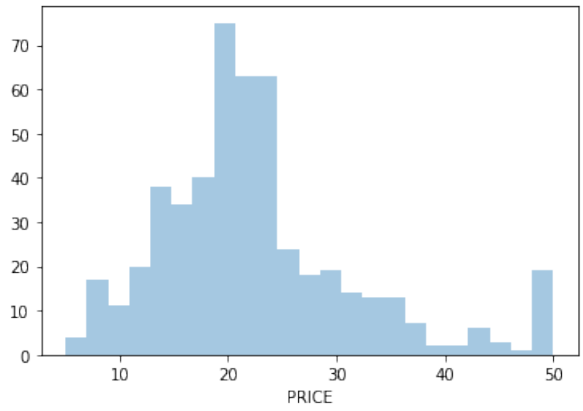


Figura 3.19: Histograma del precio de las casas en Boston, proveniente del dataset *Boston Housing-Price*.

Los histogramas definen una distribución de probabilidad según:

$$\hat{p}(x) = \frac{1}{nw} \sum_{i=1}^{n} B(x - \hat{x}_i; W) \tag{3.24}$$

donde $\hat{p}(x)$ es la probabilidad de obtener el valor x, $\hat{x}$ es el centro del intervalo, w es el ancho del intervalo y n la cantidad de intervalos. La función B se describe generalmente por:

$$B(x; w) = \begin{cases} 1 & x \in (-w/2, w/2) \\ 0 & \text{de otro modo} \end{cases} \tag{3.25}$$

El principal problema con utilizar histogramas es que primeramente generan una visualización discontinua de la distribución de los datos y que varía demasiado dependiendo del tamaño del intervalo y el valor de $\overline{x}_i$ en cada uno. Esto provoca que la distribución real no pueda ser capturada correctamente, haciendo necesario utilizar métodos más sofisticados.

El **método del kernel** propone iterar sobre la definición del histograma, definiendo la distribución por:

$$\hat{p}(x) = \frac{1}{nw} \sum_{i=1}^{n} K\left(\frac{x - x_i}{w}\right) \tag{3.26}$$

Donde K es la función kernel que debe ser seleccionado de acorde al problema que se quiere abordar y en donde generalmente se utiliza el kernel gausiano definido por:

$$\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}u^2) \tag{3.27}$$

La selección de un kernel gausiano no implica realizar un mayor supuesto respecto a la distribución original de los datos cuya distribución está siendo estimada.

El parámetro de largo del intervalo, o *bandwidth*, pasa a ser un parámetro que controla el trade-off entre la varianza y el sesgo. Es necesario definir el valor óptimo de *bandwidth*, lo cual es realizado a través de alguna regla o heurística determinada, y donde generalmente que se utiliza la regla de Scott definida según:

$$h = \frac{3{,}5\hat{\sigma}}{\sqrt[3]{n}} \tag{3.28}$$

donde $\hat{\sigma}$ es la desviación estándar de los datos. Existen maneras más sofisticadas de poder determinar el *bandwidth* óptimo para un conjunto particular de datos, las cuales son exhaustivamente exploradas en Heidenreich et al. 2013 [24].

Capítulo 4

Estado del Arte

A continuación, se presenta el estado del arte referente a los avances en ingeniería de software para contextualizar el estado actual de la industria y del avance actual sobre el tópico de estimación de esfuerzo de proyectos de desarrollo de software.

4.1. Métodos de estimación de esfuerzo

La estimación de esfuerzo de software es una de las áreas más complejas e importantes del desarrollo y la administración de proyectos de software. La dificultad recae en realizar pronósticos de parámetros al inicio del ciclo de vida del proyecto, en donde el alcance del mismo aún no está completamente claro y en donde la incerteza respecto a las funcionalidades que lo compondrán es muy alta [12]. A menudo, la falta de conocimiento sobre factores externos que pueden influenciar al proyecto y riesgos que pueden ocurrir producen que las estimaciones por juicio experto sean imprecisas y terminen siendo demasiado optimistas. Dichas estimaciones tiene un gran impacto en la entrega del software en una ventana de tiempo definida [2].

La propagación de metodologías ágiles ha logrado disminuir la tasa de fallo de los proyectos de software [23], pero esta sigue siendo substancial y observable en el día a día por profesionales del rubro [14].

Un pronóstico mas preciso al inicio de los proyectos tiene un impacto significativo en la probabilidad de que le proyecto se pueda completar [2], por lo que se ha hecho un mayor énfasis en este proceso en los últimos años. En las últimas décadas, diferentes modelos de estimación de esfuerzo han sido desarrollados. Las técnicas que se han desarrollado se pueden agrupar en 4 grupos principales, que corresponden a métodos de estimación por evaluación experta, por analogía, con modelos paramétricos y con aprendizaje automático.

4.1.1. Estimación por evaluación experta

En este grupo de métodos de estimación, el predictor corresponde a un ente humano que utiliza su experiencia en el rubro para pronosticar el esfuerzo de los proyectos de software [33], que puede consistir de uno o más actores humanos que utilizan su conocimiento para segmentar potenciales proyectos de software en unidades más pequeñas y atómicas, que generalmente corresponden a casos de uso, requerimientos (funcionales y no funcionales) o historias de usuario. Estos actores estiman la duración total del proyecto en base a las características generales del proyecto, la cantidad de unidades segmentadas, las características de dichas unidades y otros atributos externos o internos a la organización como por ejemplo, las habilidades de los desarrolladores del equipo, la disponibilidad de conocimiento para desarrollar el proyecto dentro y fuera de la empresa, entre otros.

Existen diversas formas de implementar los juicios expertos. La técnica más sencilla que se puede implementar es designar a un solo actor humano con suficiente conocimiento y experiencia para que realice las estimaciones de las funcionalidades del proyecto o del proyecto completo. Esta técnica tiene una gran cantidad de error y sesgo en su implementación debido a que la estimación siempre tendrá el sesgo de la persona que realiza las estimaciones y será muy sensible al error humano [31]. Por tanto, es más común utilizar técnicas que permitan la participación de más entes humanos en donde la estimación final es normalmente consensuada.

Las técnicas de estimación por evaluación experta más conocidas corresponde al método de Delphi, PERT y Planning Poker.

El método de Delphi es un procedimiento de pronosticación muy antiguo basado en la comunicación directa entre miembros de un panel de expertos. Los expertos responden cuestionarios sistemáticos sobre los cuales se llega eventualmente a consensos [44].

PERT (*Program Evaluation and Review Technique*) es una técnica que se originó inicialmente como una herramienta para planificar el desarrollo un sistema completo de armamento en EE.UU. La técnica considera un proyecto como una red acíclica de eventos y actividades, en dónde la duración del mismo se define por un plan de flujo de sistema en donde cada funcionalidad tiene un valor esperado de su duración y una varianza asociada. Estos valores se obtienen a partir de un proceso en donde participan uno o más entes humanos, que en base a su experiencia asignan pesos al flujo que se traducen en tiempo de trabajo [46].

Planning Poker es la técnica más recomendada para utilizar junto a metodologías ágiles [8] y corresponde a definir que la estimación de los proyectos sea realizada por un grupo de personas expertas, a través de un protocolo específico. Este protocolo consiste en ir por cada historia de usuario individual que compone al proyecto, contextualizarla y permitir que todos los miembros participantes de la instancia realicen una estimación en secreto. Luego, la estimación individual es revelada a todos los demás miembros, y en caso de que todos hayan estimado por igual, se estime con dicho valor. En caso de que no haya consenso en la estimación individual, se abre el debate para justificar el valor que cada uno de los miembros asigno y se repite le proceso hasta lograr consenso. [36]

Las estimaciones en la metodología de *Planning Poker* se hacen generalmente con puntos que son interpretados por el equipo, aunque existen también versiones que utilizan la metáfora de tallas S, M, L y XL para definir el esfuerzo de una historia de usuario [36]. En aquellas sesiones en dónde se utilizan puntos, cada uno tiene una representación consensuada de unidad de esfuerzo. Existe varias técnicas para determinar la equivalencia entre un punto de funcionalidad y su equivalencia, en dónde generalmente el comité de expertos que participará utiliza como referencia alguna tarea o funcionalidad base que puede realizar en el plazo de medio día, u otras técnicas más específicas en donde existe una correlación matemática entre puntos y esfuerzo. La escala de dichos puntos es generalmente una que refleje la complejidad creciente de las funcionalidades, en donde generalmente se utiliza el número de Fibonacci para el incremento entre valores.

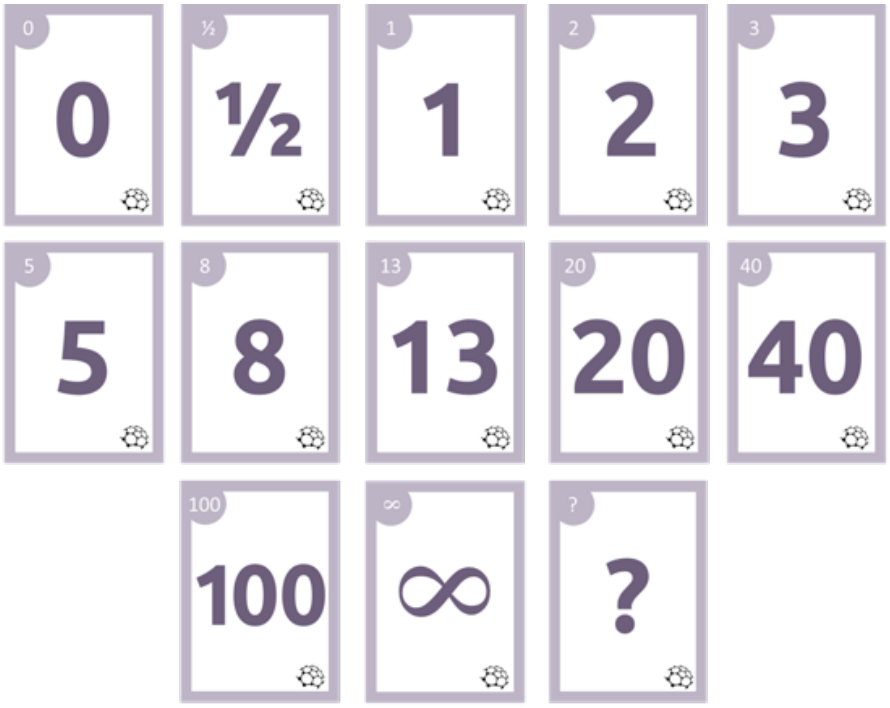


Figura 4.1: Ejemplo visual de cartas utilizadas en una sesión de Planning Poker.

Las técnicas de estimación de esfuerzo basadas en conocimiento experto son mayormente utilizadas por su simplicidad y fácil aplicación [53]. Sin embargo, estas técnicas tienden a estar sesgadas principalmente por tres razones que causan estimaciones sobre-optimistas e impactan la probabilidad de éxito del proyecto: técnica, psicológica y política. La primera se refiere al sesgo producido por la desinformación de los entes expertos, tanto a riesgos como a factores externos que afectan a las funcionalidades del proyecto. Las otras dos se enfocan en el factor humano, haciendo énfasis en la imposibilidad de planificar factores como la presión ejercida por los clientes, necesidades políticas de demostrar conocimiento y presión por ventanas de tiempo holgadas. Este sesgo puede ser controlado a medida que el equipo de la organización madura a través del tiempo [36]. Por otro lado, no se es posible cuantificar y justificar todos los factores y variables que son consideradas durante el proceso de pronóstico de manera consistente.

4.1.2. Estimación basada en analogías

Este tipo de estimación se basa en la hipótesis de que proyectos que son fundamentalmente similares entre sí, ya sea por tipo de tecnología usada, tipo de funcionalidades, contextos en los que se construyen o el equipo que los componen, deberían ser relativamente similares en la cantidad de esfuerzo necesario que se debe invertir para completarlos. Lo anterior es una conclusión obtenida a través de la técnica de *Case-Based Reasoning* [28]. En estas técnicas, se infiere una estimación del esfuerzo requerido para completar un nuevo proyecto de software a través de la identificación de un conjunto relevante de atributos que lo describen y que pueden se comparados con otros proyectos. Estos atributos son luego utilizados para encontrar proyectos similares y promediar el esfuerzo promedio de cada uno de ellos.

Existen otras variaciones de técnicas por analogía en donde se busca estimar las unidades fundamentales del proyecto por analogía y similitud según ciertos atributos, para luego calcular el esfuerzo del proyecto con la sumatoria de las predicciones individuales.

La principal ventaja de este tipo de modelos es que son mucho más simples de explicar a usuarios externos, como clientes, stakeholders y unidades internas de administración, en comparación a métodos más cerrados como el uso de redes neuronales en estimación con aprendizaje automático. La otra ventaja, es que estas técnicas pueden ser usadas para modelar relaciones complejas y no lineales entre variables dependientes e independientes [16]. Sin embargo, la principal desventaja de estas técnicas es que requieren de un gran número de proyectos o unidades fundamentales similares para poder entregar pronósticos certeros. Lo anterior es algo que ocurre raramente en la práctica [42].

4.1.3. Modelos paramétricos

En un esfuerzo de poder eliminar los sesgos humanos que existen en las estimaciones basadas en juicios expertos, se han propuesto una serie de modelos paramétricos para lograr estimaciones más precisas. Estos modelos se caracterizan por traducir características del desarrollo de software a parámetros numéricos, que son luego utilizados para calcular el esfuerzo según alguna función matemática.

COCOMO

Uno de los primeros modelos en el área de estimación de esfuerzo de proyectos de software corresponde a *COCOMO* del inglés *Constructive Cost Model* [12] que utiliza el tamaño de programas de proyectos anteriores para determinar el esfuerzo requerido. El tamaño de los programas es medido en función a la cantidad de líneas de código escritas para su completación. *COCOMO* se puede aplicar a tres clases de software:

1. *Proyectos orgánicos*: corresponden a proyectos desarrollados por equipos pequeños con buena experiencia trabajando con requerimientos poco rígidos, que implica requerimientos levemente ágiles en el sentido de que pueden cambiar en el transcurso del proyecto de forma muy leve.
2. *Proyectos semi-separados*: equipos medianos con experiencia mixta trabajando con requerimientos poco rígidos.
3. *Proyectos incrustados*: proyectos desarrollados con un conjunto de restricciones fuertes.

El *COCOMO* básico utiliza 3 ecuaciones fundamentales. La primera de ellas es:

$$E = a_b \cdot (KLOC)^{b_b} \tag{4.1}$$

En donde E corresponde al esfuerzo total necesario que se debe aplicar para completar el programa o proyecto medido en *meses humanos*, $KLOC$ la cantidad de miles de líneas de código necesarias estimadas, a_b y b_b son constantes que dependen del tipo de proyecto y tienen valores pre-calculados según la experiencia transferida desde otros proyectos de software (Tabla 4.1). La determinación de $KLOC$ tiende a ser bastante difícil y en general no se tiene un proceso formal definido para hacerlo, siendo entonces más común realizarlo desde la experiencia de un ente humano o alguna heurística consensuada. Dada la estimación del

esfuerzo, se puede obtener la duración total del proyecto según la siguiente formula:

$$D = c_b \cdot (E)^{d_b} \tag{4.2}$$

Tipo de proyecto	a_i	b_i
Orgánico	3.2	1.05
Semi-desacoplado	3.0	1.12
Incrustado	2.8	1.20

Cuadro 4.1: Valores para las constantes de COCOMO según el tipo de proyecto [12].

En dónde D es la duración total. Por último, *COCOMO* provee una última métrica para determinar la cantidad de personas requeridas para completar el proyecto dada por:

$$P = \frac{E}{D} \tag{4.3}$$

Una variación más reciente de la técnica anterior corresponde a COCOMO II y utiliza atributos adicionales de los proyectos para calcular el denominado *effort adjustment factor* (EAF), que se añade a la fórmula de esfuerzo de la siguiente manera:

$$E = a_b \cdot (KLOC)^{b_b} \cdot EAF \tag{4.4}$$

El factor anterior se calcula utilizando atributos del producto que se quiere construir, del hardware que se utilizará, del equipo que desarrollará la solución y del proyecto en sí. Cada atributo en particular es asignado a un valor según una tabla de ratings (Tabla 4.2), cuyo producto determina el valor de EAF:

$$EAF = \prod_{1}^{K} k_i \tag{4.5}$$

Donde k_i es el valor asignado al atributo i. La gran ventaja de esta nueva versión de COCOMO es que logra considerar atributos importantes dentro de la estimación que no se ven reflejadas con la cantidad de líneas de código del proyecto. Si bien este método tiene mejores resultados en la predicción que la primera versión de COCOMO, éstos siguen siendo muy altos y fuera de márgenes de error tolerables [49].

Atributos	Ratings					
	Muy bajo	Bajo	Nominal	Alto	Muy Alto	Extra Alto
Producto						
Confiabilidad requerida	0.75	0.88	1.00	1.15	1.40	
Tamaño de la base de datos		0.94	1.00	1.08	1.16	
Complejidad	0.7	0.85	1.0	1.15	1.30	1.65
Hardware						
Limitantes de rendimiento		1.0	1.11	1.30	1.66	
Limitantes de memoria		1.00	1.06	1.21	1.56	
Volatibilidad		0.87	1.00	1.15	1.30	
Tiempo de *turnabout*		0.87	1.00	1.07	1.15	
Equipo						
Capacidad analítica	1.46	1.19	1.0	0.86	0.71	
Experiencia en aplicaciones	1.29	1.13	1.00	0.91	0.82	
Capacidad ingenieril	1.42	1.17	1.00	0.86	0.70	
Experiencia con VM	1.21	1.10	1.00	0.9		
Experiencia en el lenguaje	1.14	1.07	1.00	0.95		
Proyecto						
Aplicación de metodologías de ingeniería de software	1.24	1.10	1.00	0.91	0.82	
Uso de herramientas	1.24	1.10	1.0	0.91	0.83	
Tiempo de desarrollo	1.23	1.08	1.0	1.04	1.10	

Cuadro 4.2: Valores de los atributos que afectan a COCOMO II. Estos valores han sido calibrados según observaciones de la industria al momento de proponer al iteración del método [10]. Aquellas áreas que no tienen valores no poseen una calibración testeada.

SLIM

Otro modelo paramétrico popular consiste en *Putnam's Software Life-cycle Model* (SLIM). Putnam observó que los perfiles de contratación de desarrolladores seguían una relación con la distribución de Rayleigh (Figura 4.2).

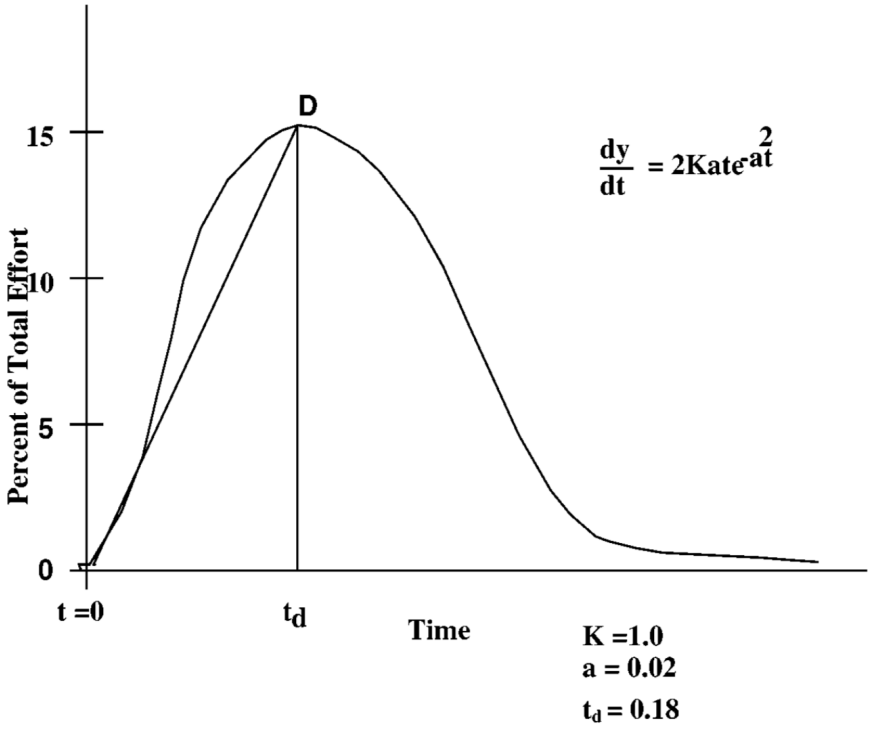


Figura 4.2: Distribución de Rayleigh obtenido ajustando los parámetros $K = 1{,}0$, $a = 0{,}02$ y $t_d = 0{,}18$ [10].

La curva de Rayleigh se define por la siguiente ecuación diferencial:

$$\frac{dy}{dt} = 2Kate^{-at^2} \tag{4.6}$$

Donde las variables K, a, t y e representan parámetros ajustables de la distribución y dependen de las características que se están utilizando para la predicción.

Este método soporta atributos de entrada como la función de puntos, líneas de código y otros

más. En este modelo, se utiliza la productividad para obtener la distribución de la capacidad del equipo de desarrollo (o *Manpower*) a partir de características de los proyectos. Putnam definió la siguientes relación:

$$\frac{B^{\frac{1}{3}} \cdot \text{Size}}{\text{Productivity}} = \text{Effort}^{\frac{1}{3}} \cdot \text{Time}^{\frac{4}{3}} \tag{4.7}$$

Donde Size es el tamaño del proyecto medido en líneas de código, puntos de función u otra heurística, B una constante de escalamiento que se utiliza para calibrar el atributo elegido para Size, Productivity corresponde al factor de productividad de la empresa (también manejado a partir de heurísticas), Time es el marco temporal en el que se mueve el proyecto (generalmente en años) y Effort el esfuerzo en horas requerido para realizar el proyecto. Realizando un despeje, se define el esfuerzo en este modelo como:

$$\text{Effort} = \left[\frac{\text{Size}}{\text{Productivity} \cdot \text{Time}^{\frac{4}{3}}}\right]^3 \cdot B \tag{4.8}$$

SEER-SEM

El último método relevante corresponde a SEER-SEM, que utiliza una serie de modelos estadísticos para atributos como tamaño, tecnologías, esfuerzo, costo, defectos, mantención y otros más para modelar el esfuerzo total del proyecto. En este modelo, el esfuerzo

$$K = D^{0,4} \cdot \frac{S_e}{C_{te}}^E \tag{4.9}$$

Donde S_e es el tamaño efectivo del proyecto que se calcula dependiendo del atributo de entrada que se utiliza para la predicción, C_{te} la tecnología efectiva que se calcula capturando una serie de factores relacionados con la eficiencia del equipo o su productividad, D la complejidad del tamaño del equipo y E una variable de entropía que corresponde a una constante (empíricamente se ha demostrado que el valor óptimo es 1,2).

Los modelos paramétricos tienen varias ventajas con respecto a modelos anteriormente vistos: en general son buenos para realizar estimaciones rápidas de esfuerzo, no tienen el mismo sesgo que tienen los modelos basados en juicio experto y permiten transparentar aquellas características de los proyectos que están siendo consideradas para la estimación . Sin embargo, tienen como desventaja principal que son muy dependientes del lenguaje de programación en el cual se desarrollará el proyecto [42] y no consideran atributos externos a la programación misma de funcionalidades, como por ejemplo el tiempo invertido en el diseño visual, de interacción y de servicios de la misma. Otra desventaja importante que es necesario considerar a la hora de utilizar este tipo de soluciones es que generalmente estos toman las líneas de código producidas en los proyectos como característica principal. Lo anterior no es muy factible en la industria actual como medida de estimación, ya que existe un gran porcentaje de retrabajo y cambios en la estructura de las funcionalidades de manera mucho más rápida gracias a la incorporación de métodos ágiles [37].

4.1.4. Aprendizaje automático

Dadas las desventajas y deficiencias que se han podido observar en los métodos anteriores, que persisten incluso combinándolos con método de lógica difusa [16], se han realizado durante las últimas décadas múltiples investigaciones con el objetivo de realizar estimaciones utilizando algoritmos de máquinas de aprendizaje modernos [51]. Estos algoritmos han resultado ser muy eficientes para poder manejar el problema de la incertidumbre en la estimación y sus resultados demuestran su potente capacidad predictiva para el esfuerzo de los proyectos, tanto al inicio del ciclo de vida, como en etapas intermedias de este [32]. Dado que estos algoritmos se caracterizan por realizar estimaciones automáticas que no requieren de intervención humana, más allá de configurarlos correctamente, se disminuye el sesgo producido en otros tipos de estimación de índole humana, psicológica o política.

Con modelos de máquinas de aprendizaje, se tiene mayor dinamismo respecto a las características que se utilizan para las estimaciones. Esto se debe a que en estos modelos los parámetros no están fijos como en modelos paramétricos y pueden cambiar según los datos que se van observando.

Incluso siendo tan prometedores, estos algoritmos son utilizados nada o muy poco en la industria. La razón detrás de esto corresponde a que los algoritmos estaban siendo entrenados con bases de datos demasiado antiguas, con tecnologías, prácticas y metodologías que no son homologables en los proyectos modernos, cuya metodología principal de ciclo de vida es la de tipo Ágil [42].

En el año 2012, Wen et al. [51] realizó el estudio más completo acerca del estado de la utilización de los algoritmos de máquinas de aprendizaje para realizar estimaciones de esfuerzo. En él, se revisaron 84 estudios cuyos resultados arrojaron que los algoritmos más efectivos corresponden a aquellos en la categoría de redes neuronales, máquinas de soporte y árboles de decisión. En particular, las redes neuronales demostraron tener la ventaja de que no son sensibles a la distribución de los datos, pero si a la calidad de los atributos utilizados para la predicción y al número de observaciones disponibles [5]. Esto no las hace factibles para datasets muy pequeños, en empresas que tienen pocos proyectos en su portafolio.

De toda la investigación que se ha realizado en las últimas década, es posible extraer recomendaciones transversales a todos los modelos que estén en la categoría. El énfasis debe realizarse en el pre-procesamiento de los datos, especialmente en planificar estrategias para manejar outliers, datos incompletos y su correlación. Si bien es necesario analizar los datos y el tipo de contexto en el que se está trabajando, existen una serie de recomendaciones que han sido obtenidas de investigaciones empíricas. Respecto a los outliers, eliminar aquellos que estén fuera de 3 veces la desviación estándar del conjunto de datos es lo más recomendable para evitar que estos desenfoquen los resultados predictivos del modelo que vayamos a seleccionar. Es recomendado además eliminar observaciones con atributos sin información ya, que utilizarlas podría reducir significativamente la variabilidad de los datos [42]. Con respecto a la correlación de los datos, es en general buena práctica eliminar aquellos atributos que tengan un grado de correlación muy alto entre sí, para así disminuir la cantidad de parámetros de entrada de los modelos.

Incluso con la eliminación de outliers, datos incompletos y atributos fuertemente correlacionados, existe el hecho de que los algoritmos de máquinas de aprendizaje son muy sensibles al ruido en los conjuntos de entrada, por lo que es necesario utilizar métodos de ensamblado de modelos [39]. Estos métodos combinan la predicción de varios modelos individuales

y generan una única predicción que suele ser más certera. Sin embargo, estos métodos son bastante costosos computacionalmente hablando y se recomienda que de ser utilizados, sean los más simples posible [4].

Si bien los modelos de máquinas aprendizajes mencionados anteriormente son los que muestran mejores resultados, se han investigado una gran variedad de ellos a medida que el área del aprendizaje automático iba progresando en su investigación propia. Se han estudiado una gran cantidad de modelos para *SDEE*. Estos incluyen una gran cantidad de técnicas de regresión (lineales y no lineales), el uso de redes neuronales, aprendizaje por instanciación (*instance-based learning*), modelos basado en árboles de reglas, razonadores basados en casos específicos, aprendizaje lento (*lazy learning*), clasificación Bayesiana utilizando árboles de clasificación, SPM (*Support vector machines)*, entre otros [34] [35]. La mayoría de estos estudios evalúa solo una cantidad limitada de técnicas de modelamiento en un conjunto de datos, lo cual limita la generalización de sus resultados. Además, los resultados de dichos estudios son difíciles de comparar debido a las diversas características de los escenarios en los que son empleados. Por tanto, el problema de qué modelo utilizar sigue aún sin ser respondida [15].

En el año 2018, se realizó un estudio que buscaba estandarizar los resultados de distintas investigaciones de aprendizaje automático aplicado a predicción de esfuerzo de proyectos para poder realizar una comparación justa entre ellos [42]. Este estudio era necesario considerando que todos los estudios publicaban resultados en distintas métricas, o realizaban procesos de pre-procesamiento y ajuste de algoritmos que eran cuestionables si se hacía referencia al estado del arte respecto de los algoritmos de máquinas de aprendizaje. De este estudio se pudo determinar que el algoritmo que mejor se comportó fueron las máquinas de soporte vectorial, en inglés *Support Vector Machines* (SVM). Este resultado no asegura que dicho algoritmo se comporte de igual manera en todos los casos y datasets de proyectos debido a la naturaleza de los algoritmos de aprendizaje. Lo más relevante, es que el uso de métodos ensambladores produce un mejor resultado que los estimadores por sí solos.

	SVM	MLP	GLM	Ensemble	SVM	MLP	GLM	Ensemble
	Effort				*Duration*			
ME	0.02	0.02	0.01	0.02	0.01	0.01	0.01	0.01
MAE	0.19	0.26	0.27	0.23	0.14	0.19	0,19	0.17
MSE	0.07	0.12	0.13	0.1	0.04	0.06	0.07	0.05
RMSE	0.27	0.34	0.35	0.31	0.2	0.25	0.26	0.23
MMRE	0.13	0.21	0.18	0.17	0.15	0.24	0.26	0.21
PRED (0.25)	76.91%	64.65%	61.96%	69.44%	75.65%	64.82%	62.55%	69.02%
PRED (0.3)	81.19%	71.37%	67.93%	74.73%	81.61%	72.96%	71.12%	76.49%
MMER	0.47	0.45	0.57	0.42	0.26	0.31	0.34	0.29
MBRE	0.16	0.22	0.23	0.2	0.17	0.15	0.25	0.22

Figura 4.3: Cuadro comparativo entre distintos modelos de aprendizaje automático [42].

Si se lleva esto a un paso más y se realiza un estudio comparativo con modelos estándar, es claro que el aprendizaje automático es la opción preferente a la hora de querer realmente minimizar la incerteza de los pronósticos de duración de los proyectos.

4.2. Características utilizadas para la estimación

Si bien es importante el tipo de modelo a utilizar para la estimación del esfuerzo de un proyecto de software, es más relevante aún definir los atributos del proyecto que deberían ser considerados a la hora de estimar. Los modelos paramétricos y de juicio experto tienen por lo general un tipo de atributo para realizar la estimación, el cual es rara vez posible de cambiar. Sin embargo, estos atributos deberían ser por lo menos analizados considerando que existen varias investigaciones que respaldan que la consideración de ellos afectan las duraciones de los proyectos informáticos. A continuación se describen los atributos más relevantes utilizados para las predicciones a lo largo del tiempo.

Los primeros intentos de estimación utilizaron el tamaño de los proyectos desarrollados en función de la cantidad de líneas de código que los componían. Esta unidad como parámetro de entrada para modelos de aprendizaje automático no es factible en la industria moderna, puesto que no considera el retrabajo de funcionalidades debido a calidad insuficiente o por requerimientos que cambian a mitad de camino, tal y como sucede en metodologías del

modelo Ágil [37]. Además, esta métrica depende demasiado de la tecnología con la que se construirá el software. Lo anterior es incompatible en la industria moderna por modelos nuevos como la arquitectura por micro-servicios, en donde una funcionalidad puede estar compuesta de múltiples sub-servicios con diversos lenguajes de programación [41].

Posteriormente, se comenzaron a generar modelos que intentaban desacoplar la estimación del esfuerzo con la tecnología utilizada. Para esto, se comenzó a diseccionar los proyectos en unidades mínimas de valor que tomaron el nombre de puntos de funcionalidad. Un enfoque más moderno y compatible con las metodologías ágiles es la utilización del concepto de puntos de historia de usuario, que si bien tiene una discrepancia técnica con el anterior, son en esencia lo mismo. Estos puntos han permitido a modelos de juicio experto poder realizar estimaciones subjetivas según la experiencia de las personas participantes en las sesiones de estimación. El mismo concepto fue utilizado también para modelos paramétricos modernos, como SLIM.

Los enfoques más modernos implican realizar una selección y extracción de atributos inherentes al desarrollo de software para estimar. Los primeros aprontes provienen del modelo COCOMO II [10], en donde se utilizan atributos del producto, del hardware y del equipo de desarrollo a utilizar.

La mayoría de las investigaciones que han utilizado modelos de aprendizaje automático o por analogía como solución para realizar los pronósticos de esfuerzo, son realizadas en la base de datos ISBSG [18], que consiste en un repositorio de información sobre proyectos de distinta índole y que almacena más de 8.000 proyectos. Estos atributos hacen referencia a distintas características de proyectos, los cuales se pueden agrupar por la descripción del proyecto en detalle, el tamaño del proyecto y atributos relacionados con el tamaño, el esfuerzo real ejercido para completar el proyecto, los defectos detectados en el proyecto, la planificación del proyecto, la arquitectura utilizada, las técnicas utilizadas, la documentación utilizada y detalles de la plataforma en sí. Estos atributos han sido utilizados por diversos estudios, de los cuales existe una conclusión en común, que se puede resumir en que se debe hacer énfasis en el procesamiento de los datos iniciales para que se adecúen lo mejor posible a los modelos utilizados y a las suposiciones que se hagan [42] [5] [51].

Los modelos modernos intentan favorecer datasets con un mayor volúmen de datos para mejorar el poder predictivo de sus estimaciones, sin considerar que datasets como el de ISBSG no incluyen factores humanos importantes, como por ejemplo, la madurez técnica del equipo que desarrolla el software [18]. Tampoco se tiene una forma de medir correctamente el esfuerzo requerido en términos de diseño visual, de interacción y de servicios que podría llegar a componer una funcionalidad.

4.3. Métricas de rendimiento

En el área de la predicción de esfuerzo, se han estandarizado una serie de métricas para determinar el rendimiento de un método en particular. Estas métricas corresponden al error relativo medio (MRE) de las observaciones predecidas, que en el caso del presente trabajo corresponde al error relativo en cada una de las historias de usuario pronosticadas y se define por:

$$MRE(t) = \frac{|y_t - y_t^p|}{y_t} \tag{4.10}$$

donde y_t es el valor real de la observación y y_t^p es la predecida. La segunda métrica corresponde al error relativo promedio de todos los proyectos ($MMRE$) definido como:

$$MMRE = \frac{1}{n} \sum_{t=1}^{n} MRE(t) \tag{4.11}$$

donde n es la cantidad total de proyectos. Y por último, a $PRED(25)$, en donde $PRED(x)$ se define por:

$$PRED(x) = \frac{1}{n} \sum_{t=1}^{n} \begin{cases} 1 & \text{si } MRE(t) \leq x \\ 0 & \text{de otro modo} \end{cases} \tag{4.12}$$

y que se traduce en el porcentaje de proyectos pronosticados que tienen un $MMRE$ bajo el

25 %. Se dice que un modelo predictivo del esfuerzo de desarrollo de software es de alta calidad cuando se cumple que $PRED(25)$ es mayor a un 80 % [27].

Capítulo 5

Solución Propuesta

Para poder abordar la problemática planteada, se propone una solución que consiste en la definición de un *framework* de pronóstico de proyectos que se basa en atributos de las historias de usuario que los componen. El flujo de pronóstico se puede resumir en el siguiente diagrama:

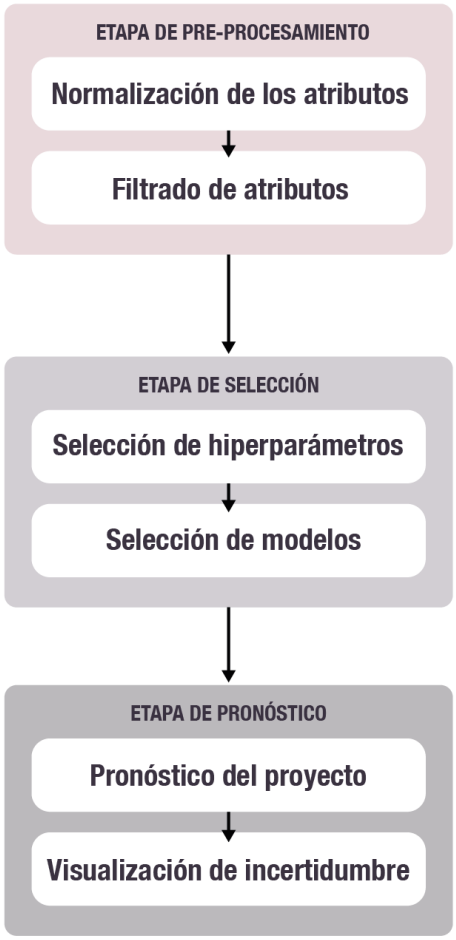


Figura 5.1: Definición general del *framework* propuesto.

Este framework tiene como objetivo brindar un pronóstico de un proyecto en particular en base a los datos históricos de otros proyectos disponibles en el momento. Está compuesto por 3 etapas fundamentales, que corresponden a una etapa de preprocesamiento, etapa de selección y una etapa de pronóstico. En cada una de estas etapas se proponen una serie de particularidades que se detallan más adelante y que se ven argumentadas a través de ciertos procesos y actividades de soporte. Estas actividades se pueden visualizar de mejor manera en el siguiente diagrama:

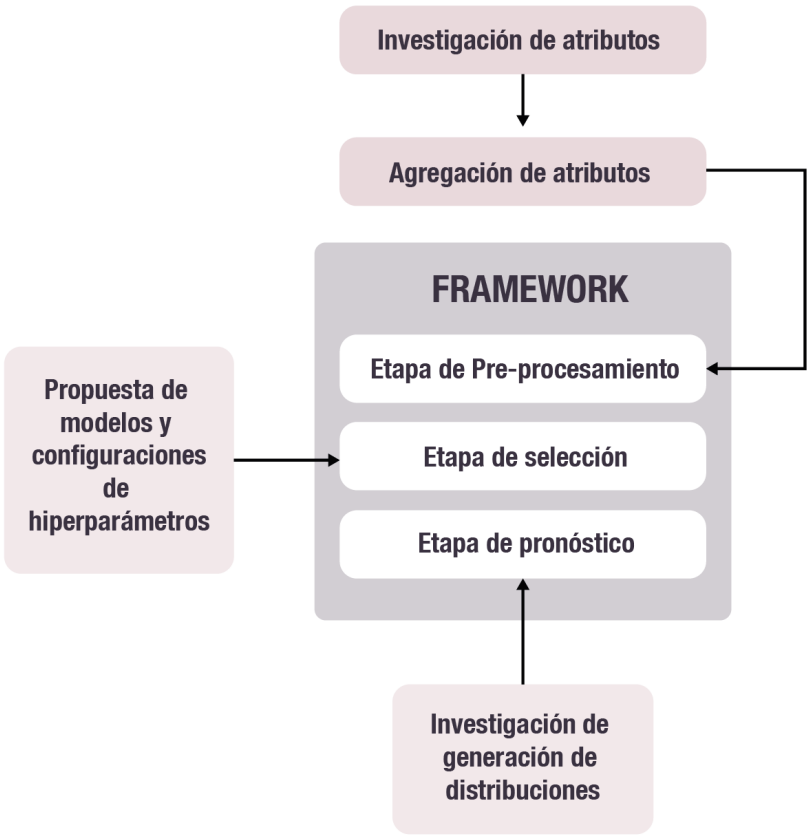


Figura 5.2: Actividades de soporte principales realizadas para la definición del *framework*.

A continuación, se define cada una de las etapas en detalle, asociando además las actividades de soporte realizadas.

5.1. Etapa de pre-procesamiento

En la etapa de pre-procesamiento del *framework*, se toman las historias de usuario finalizadas hasta el momento en que se quiere predecir un proyecto específico, que de ahora en adelante denotaremos por P_i, sobre las cuales se realiza un proceso de normalización y filtrado de sus atributos. Para poder definir los procesos concretos que suceden en esta etapa, es necesario

definir primero qué información de las historias de usuario es relevante para pronosticar el esfuerzo del proyecto que componen. Para esto, se realiza un análisis de la información original con la que vienen dichas historias y un análisis de posible información adicional que podría ser interesante explorar. Luego, se define un método para poder almacenar la nueva información de las historias y una forma de poder acceder a ella, para así combinarla con la original y utilizarla para los pronósticos.

5.1.1. Análisis de atributos

Entre las fuentes de información que se utilizan para el análisis de atributos se considera la documentación de requerimientos de los proyectos, modelos de datos de sistemas, modelos de arquitectura, *changelogs*, mensajes de *commits* y archivos de configuración generales de servicios. Otra fuente, por otro lado, corresponde a la información provista por las personas que trabajaron históricamente en los proyectos que se utilizarán para realizar la validación de la propuesta.

Atributos en bruto

Las historias de usuario disponibles en la plataforma de gestión de proyectos *Taiga* y que en general están disponibles en las plataformas modernas de gestión (tales como *JIRA* [1]) vienen con los atributos de nombre, estimación inicial del comité de *planning poker*, esfuerzo real, distribución del esfuerzo por cargo, descripción, *sprint* del proyecto a la que pertenece y categoría de historia de usuario definidas por proyecto. A continuación se analiza cada uno de los atributos y su posible utilidad a la hora de realizar predicciones de esfuerzo.

- **Nombre**: Si bien es posible argumentar que un algoritmo de predicción puede aprender ciertos patrones de nombres de historias de usuario que impliquen más o menos esfuerzo requerido por el equipo, no existe un lenguaje común entre cada proyecto respecto a cómo deben ser redactados dichos nombres. Esto hace que el atributo no sea útil para las predicciones considerando que cada proyecto tiene un contexto demasiado distinto.

- **Estimación inicial del comité**: La estimación inicial del comité corresponde a un número decimal llamado punto de historia de usuario en donde cada uno equivale a aproximadamente 4 horas de trabajo. Este atributo es fundamental para las predicciones del esfuerzo puesto a que durante la asignación de los puntos ocurre un intercambio de conocimiento y opinión entre los miembros del comité evaluador, en donde se llega a un consenso a través del *Planning Poker*. Durante este proceso de interacción humana, se considera el contexto del proyecto, el modelo de negocio del cliente, los requerimientos no funcionales de la historia, las herramientas y tecnologías que se utilizarán durante el desarrollo, los miembros que conformarán el equipo de desarrollo, la experiencia de cada uno de los miembros del comité, información referente al cargo de cada uno de los miembros del comité y otro tipo de información que no es posible rescatar sin que se necesite grabar o transcribir la conversación entera debido a la naturaleza de los diálogos humanos. Todo esto se ve últimamente reflejado en un número decimal cuyo valor es muy alto para los modelos predictivos, puesto que esconde el consenso del proceso de estimación inicial y no refleja toda la conversación ocurrida en el proceso.

- **Esfuerzo real**: Este atributo corresponde al número total de horas humanas que tomó completar la historia de usuario. Esto incluye la realización inicial de la historia, como también todo el tiempo invertido en diseño y aseguramiento de calidad. Este atributo es fundamental puesto que será usado para calcular el error de las predicciones de los distintos modelos a utilizar.

- **Distribución por cargo**: Corresponde al desgloce de la estimación inicial del comité por cada cargo.

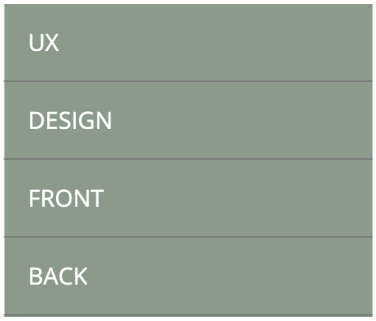


Figura 5.3: Listado de roles entre los cuales es posible distribuir esfuerzo.

La distribución por cargo pueden llegar a ser útil en el proceso de predicción, ya que existen tareas cuya duración y esfuerzo requerido varían según el mismo. Por ejemplo, La distribución en un único cargo puede fácilmente implicar que para una historia de usuario en particular no fue necesario realizar un proceso de integración entre sistemas, lo cual se puede traducir en un aumento de la velocidad de desarrollo.

- **Sprint del proyecto**: El principal problema con este atributo es que las etapas no siguen una estructura lógica o estándar y responden solamente a las necesidades y priorizaciones que tiene y realiza el cliente. Es incluso deseable que las historias de usuario puedan no estar asociadas a una etapa aún para poder ser predichas. Por tanto, el atributo no es relevante para las predicciones.

- **Categoría**: La categoría de una historia de usuario depende completamente del proyecto, puesto a que se construyen dentro del dominio del mismo proyecto. Pueden incluso existir proyectos sin categorías definidas en donde las historias de usuario solo están asociadas a una sola. Por tanto, se decide no utilizar el atributo puesto a que entorpece el aprendizaje de los distintos modelos a utilizar.

- **Descripción**: A partir de la descripción de una historia de usuario es posible obtener gran cantidad de información relevante acerca de ella, incluyendo lógica del modelo de negocio, herramientas y prácticas utilizadas en la realización de la historia de usuario, requerimientos no funcionales y/o sub-requerimientos que pueden estar contenidos en la historia de usuario. Sin embargo, en la plataforma *Taiga* existe una gran cantidad de historias sin descripción. Además, aquellas que sí poseen descripción no siguen un

formato único y estándar, por lo que en ellas falta información que en otras sí puede estar.

Figura 5.4: Diferencia entre la calidad de la descripción de las historias de usuario. A la izquierda se puede observar una descripción pequeña y sin mucha cantidad de información, y a la derecha una descripción más extensa que contiene muchos más detalles gracias a la utilización del lenguaje de redacción *markdown*.

Luego, todas las descripciones entregan información distinta acerca del proyecto, lo podría hacer que los modelos predictivos interpreten incorrectamente que la falta de una palabra clave específica en una historia de usuario implique la inexistencia del uso de una librería, framework o práctica de desarrollo. De forma menos abstracta, se puede ejemplificar lo anterior cuando se tienen historias de usuario que se realizaron con el mismo framework, pero que en donde una tiene el framework en cuestión en su descripción, mientras que la otra tiene una descripción vacía. Esto podría hacer que la información adicional de la descripción forme un sesgo erróneo en los modelos.

Una posibilidad para reparar este atributo y utilizarlo en las predicciones es estandarizar la descripción de las historias de usuario pasadas y llenar la descripción de aquellas que están vacías junto a las personas que las realizaron. El problema de lo anterior es que no están disponibles todas las personas que históricamente participaron en dichas US, y es muy poco probable además poder completar con el mismo nivel de detalle una historia de usuario que tiene más de 2 años de antigüedad.

Por todo lo anterior, parece mejor no utilizar la descripción y añadir de forma manual la información acerca de prácticas, requerimientos no funcionales y librerías a través del método propuesto posteriormente.

Atributos propuestos

Luego del análisis de los atributos anteriores, se puede observar que debido a que no se utilizará la descripción de las historias de usuario para las predicciones, existe una gran cantidad de información que se está perdiendo. Por otro lado, se tiene que en la literatura se nombran una serie atributos denominados como claves en los atrasos de los proyectos de software. En búsqueda de aumentar la calidad de las predicciones realizadas, se revisan una serie de atributos relevantes y factibles de obtener para proyectos completamente nuevos.

A nivel de proyecto, se definen una serie de atributos que serán propagados a todas las historias de usuario:

- **Equipo**: *Cantidad de personas asignadas al proyecto*. La cantidad de personas que son parte de un equipo es un atributo relevante, considerando por sobre todo que mientras más grande es el equipo, más compleja es la dinámica interna que ocurre en ellos. Para poder transparentar esta dinámica no trivial, es pertinente utilizar este atributo.
- **Equipo**: *Horas semanales planificadas*. Teóricamente, si se tiene una mayor cantidad de horas semanales asignadas al proyecto por parte del equipo, este debería requerir más esfuerzo. Como no se tiene un atributo referente a la duración máxima del proyecto, es necesario mostrarle a los modelos predictivos de alguna forma esta relación.
- **Planificación y control**: *Metodología de desarrollo*. Si bien la metodología de desarrollo en la empresa es ágil, existen ciertos proyectos en donde se prefirió realizar una planificación en cascada considerando que eran proyectos inamovibles o pequeños.
- **Planificación y control**: *Prácticas planificadas*. La utilización de ciertas prácticas como *Test-Driven development* hacen que el proceso de desarrollo de software tenga un costo de esfuerzo mucho menor que si no se utilizan. Otra práctica muy común en la industria y que reduce bastante el esfuerzo requerido es la utilización de integración continua.
- **Planificación y control**: *¿Es un refactory de un sistema previo?* Trabajar sobre un sistema previo puede significar un aumento en el esfuerzo requerido para completar el

proyecto en situaciones en que dicho sistema tiene varios componentes que no pueden ser reutilizados [19].

- **Cliente**: *Cantidad de proyectos previos con el cliente*. Uno de los aspectos más importantes en el desarrollo de software es poder educar adecuadamente al cliente sobre los procesos que giran entorno a la implementación de los sistemas. Que el cliente sea alguien que ya ha hecho proyectos con la empresa implica una disminución del esfuerzo requerido en dicho proceso educativo.

- **Arquitectura**: *Arquitectura utilizada en el proyecto*. La arquitectura utilizada en un proyecto es uno de los atributos que más impacta en el esfuerzo requerido para finalizar los proyectos. Por ejemplo, una arquitectura de micro-servicios requiere la implementación y unión de múltiples sistemas pequeños, mientras que en una arquitectura monolítica el sistema es uno solamente.

Por otro lado, se proponen además los siguientes atributos que están a nivel de historias de usuario:

- **Requerimientos**: *Documentación con la que cuenta la funcionalidad*. La documentación referente a diseño de interfaces, procesos, modelos de datos, de arquitectura, de conexiones y mucha más es relevante en cada historia de usuario. Empíricamente, se tiene que cuando no existe documentación clave a la hora de desarrollar las historias de usuario, estas pueden quedar incompletas y por tanto, el esfuerzo total aumenta al tener que realizar retrabajo.

- **Requerimientos:** *Requerimientos no funcionales*. Los requerimiento no funcionales son relevantes, ya que generalmente están asociados a una cantidad de esfuerzo mayor. A modo de ejemplo, el requerimiento no funcional de seguridad implica aumentar el esfuerzo requerido para una funcionalidad en particular, al tener que invertir tiempo en investigar cómo mejorar la seguridad, probarla y realizar decisiones a nivel de implementación.

- **Usuario**: *Tipo de usuario objetivo que afecta la funcionalidad*. Cuando los sistemas tienen más de un tipo de usuario, el esfuerzo para una funcionalidad en particular

tiende a implicar más esfuerzo que el inicialmente previsto. Se propone como sistema de categorización de usuario las categorías de *end-users*, *Administrative entity* y *Super-admin* que engloban en su mayoría los usuarios tipo que interactúan con los sistemas en la industria.

- **Complejidad**: *Escala de complejidad*. La escala de complejidad es la definida en la investigación realiza por Shahid et al. en [55]

- **Complejidad**: *Tecnologías relevantes utilizadas*. Dependiendo de la tecnología que se utiliza, se pueden tener distintos resultados de esfuerzo, ya que ciertas funcionalidades son más rápidas y fáciles de utilizar en una tecnología y otra. Este atributo solo considera tecnologías relevantes, y no revisa tecnologías más específicas como librerías utilizadas.

- **Complejidad**: *Estilos, patrones o tácticas arquitectónicas relevantes utilizadas*. Corresponden a patrones arquitectónicos que pueden ser detectados antes de que inicie la implementación de las funcionalidades. Por ejemplo, la utilización de algún patrón de estilo de código específico para la funcionalidad, la utilización de *websockets*, etc.

- **Complejidad**: *¿Existen tecnologías relevantes nuevas (para el equipo) a utilizar?* Este atributo mide si el equipo de desarrollo se enfrentará a tecnologías con las cuales nunca ha trabajado anteriormente. Particularmente para este atributo es fácil ver que cuando una contraparte humana se enfrenta a un proceso completamente nuevo, existe una diferencia de esfuerzo adicional que se debe considerar.

- **Complejidad**: *¿Existe vinculación a otros sistemas?* La vinculación con otros sistema tiende a requerir la interacción con proveedores, la cual puede o no ser eficiente dependiendo del tipo de problema, del proveedor y de la calidad de los canales de comunicación. En general, se aprecia un esfuerzo adicional al presentarse este contexto.

- **Complejidad**: *¿Es una sección nueva en términos de front-end?* Se define como una sección a un módulo de interfaz gráfica o con implementación mayoritariamente en el frontend. Está relacionado con la distribución de trabajo en frontend de los atributos originales.

- **Complejidad**: *¿Es una sección nueva en términos de back-end?* Se define como una sección en backend como un módulo de lógica o procedimiento en donde la mayoría de la implementación es realizada en el backend. Está relacionado con la distribución de trabajo en backend de los atributos originales

- **Complejidad**: *Listado de actividades de alto nivel que desarrolla la funcionalidad*. Con el objetivo de poder traer la información de las descripciones inconsistentes al *framework*, se define este atributo cuyos valores categóricos corresponden a actividades genéricas que realizan los usuarios. Por ejemplo, el llenar un formulario, ver un video o eliminar entidades.

5.1.2. Agregación de atributos

Para asegurar una independencia entre los atributos propuestos para las predicciones y las plataformas de gestión que se utilizan en la industria (que en el caso particular de la empresa en donde se está validando el método corresponde a *Taiga*), se propone un sistema de agregación de los datos que utiliza la metáfora de *tags* en su interfaz gráfica. Cada *tag* corresponde al valor de una variable categórica específica, en donde los *tags* de proyectos se propagan a las historias de usuario, siendo estas últimas el output del sistema.

La plataforma utiliza el patrón de arquitecura *Model-View-Viewmodel* o *MVVM* cuyas ventajas con respecto a otros modelos más antiguos como *Model-View-Controller* se pueden evidenciar en [21], en donde se destaca la ventaja de que se tiene la tecnología de la capa de presentación y la tecnología de la capa lógica totalmente desacopladas. Lo anterior permite un mayor grado de escalabilidad y de modificabilidad de la arquitectura que permite realizar una migración a arquitecturas más complejas de manera más fácil. De ahora en adelante, entiendase por *frontend* a la capa de presentación de la plataforma en donde está la interfaz visual, y a *backend* a la capa de lógica en donde se encuentra la lógica de negocio, de almacenamiento y la conexión a la base de datos no relacional. La arquitectura de la plataforma se puede visualizar en el siguiente esquema:

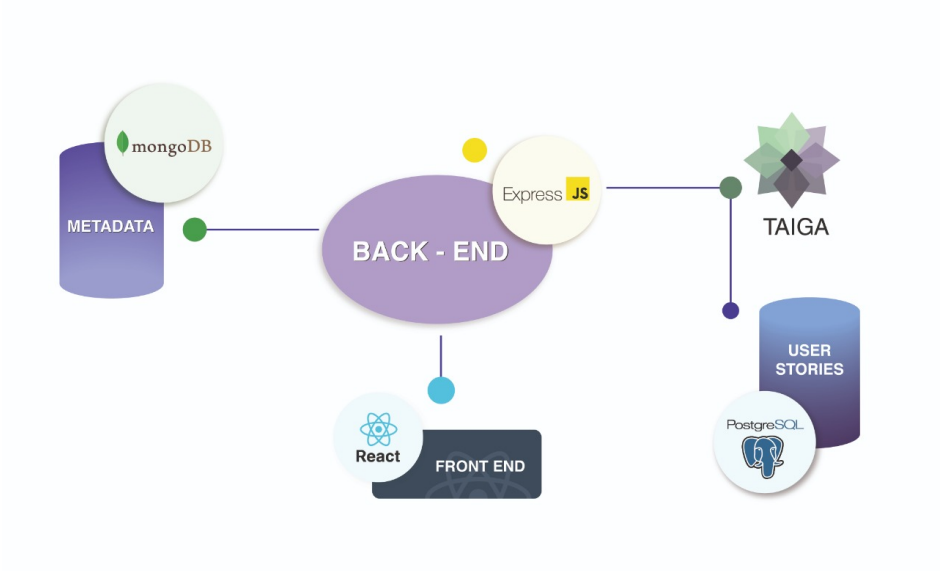


Figura 5.5: Arquitectura utilizada para la implementación de la plataforma de agregación de atributos (MVVM).

Para poder añadir la información extra y combinarla eventualmente con información proveniente de las plataformas de gestión que se usen, se debe realizar un proceso investigativo para recopilar los valores de los atributos definidos anteriormente y colocarlos en la plataforma, proceso al cual se le referenciará como *Tagging* de ahora en adelante. Este proceso puede llegar a ser muy engorroso si la plataforma no es lo suficientemente simple, por lo que se le dió énfasis a la interfaz en ser lo más eficaz posible y que entregue herramientas de ayuda para quien realiza el proceso.

La plataforma contiene la lista de todos los proyectos de la empresa, que se encuentra resumida en un dashboard inicial de bienvenida. Para cada proyecto se muestra el porcentaje de historias de usuario que han sido etiquetadas, e información específica referente a los mismos tags.

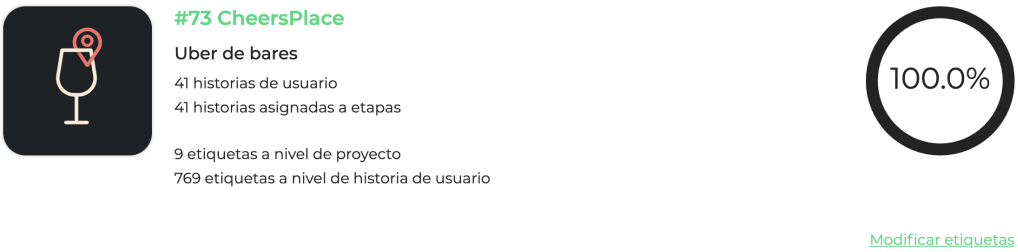


Figura 5.6: Información resumida de cada proyecto en el dashboard inicial.

Al acceder a un proyecto en particular, se muestra un formulario en el cual se pueden añadir, modificar o eliminar tags a nivel del proyecto completo. Estos tags son eventualmente propagados a las historias de usuarios durante las predicciones.

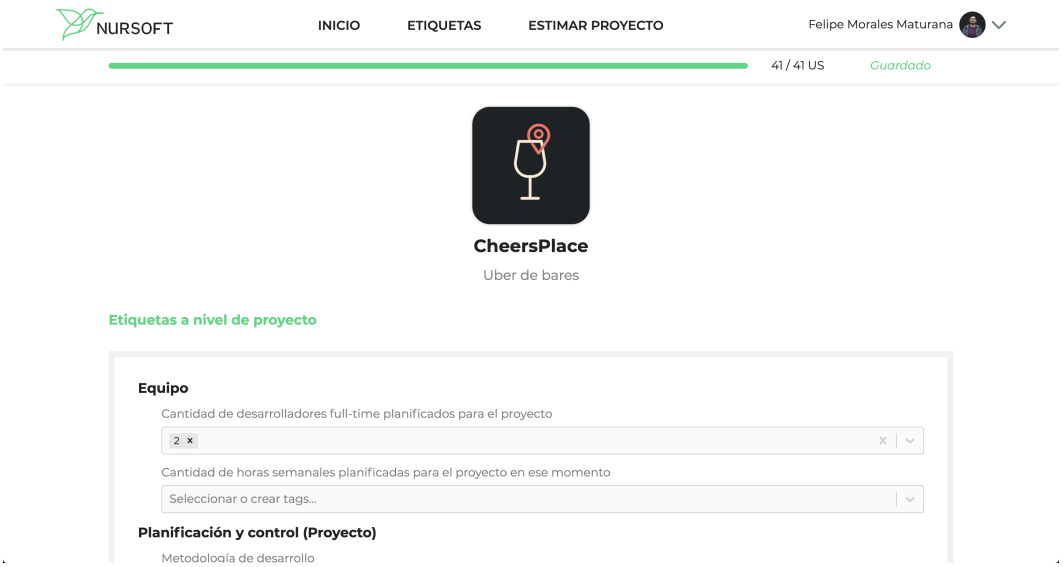


Figura 5.7: Vista de un proyecto particular en la plataforma de agregación.

Abajo del formulario de etiquetas del proyecto completo, se muestra una lista con todas las historias de usuario del proyecto que hayan sido asociadas a etapas específicas del mismo. Esto es de gran importancia, ya que pueden existir historias de usuario creadas que siempre estuvieron en el *backlog* del proyecto y que nunca fueron parte de una etapa o versión específica que implicara esfuerzo por parte del equipo.

Figura 5.8: Listado de historias de usuario de un proyecto en donde se registra esfuerzo.

Cada una de las historias de usuario contiene un formulario en donde se pueden modificar los atributos de historias de usuario que fueron definidos en la sección 5.1.1.

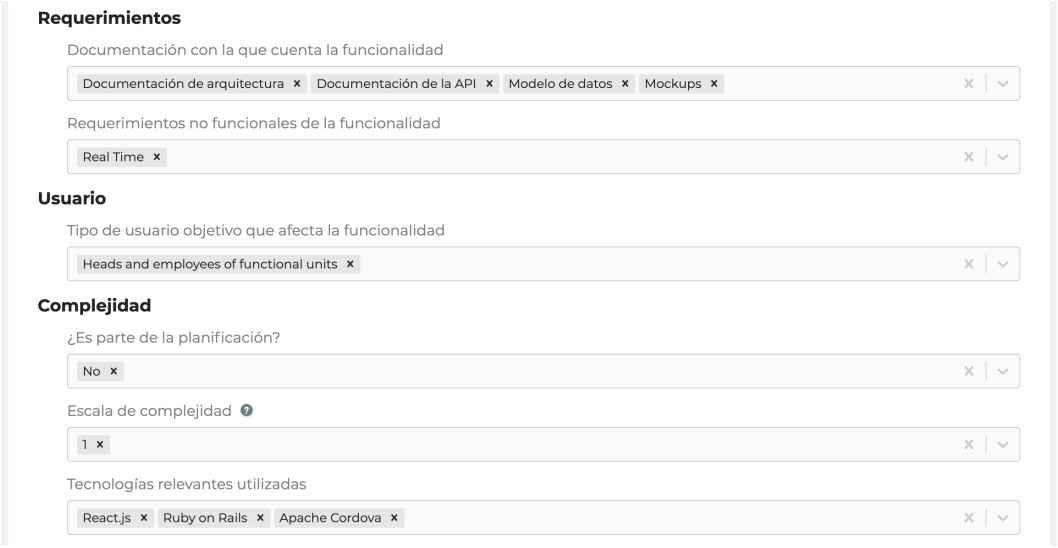


Figura 5.9: Formulario de atributos de una historias de usuario en particular.

La plataforma de agregación actúa luego como una fuente de información que es concatenada con la información original de las historias de usuario documentadas en las distintas plataformas de gestión. Esto se realiza a través de la disponibilización del servicio por medio

de un *endpoint* REST interno.

5.1.3. Definición de conjuntos

Es muy importante tener en claro que los proyectos que se realizan en empresas de desarrollo de software tienen una componente temporal asociada, lo que hace que la obtención de los conjuntos de entrenamiento y de validación no puede realizarse de manera arbitraria.

Dado un conjunto de proyectos P que se conoce han finalizado y un proyecto particular al cual se le desea realizar una estimación de esfuerzo P_i, es necesario definir el conjunto de entrenamiento como todas aquellas historias de usuario cerradas hasta el momento en que comienza dicho proyecto. Esto debido a que si no se considera dicha restricción, se estaría estimando el proyecto P_i con proyectos P_j con $j > i$ que están en el futuro. Lo anterior se podría definir como una forma de hacer trampa en la etapa de entrenamiento, puesto que cuando se lleve el algoritmo a la práctica y se desee estimar un proyecto nuevo, este no tendrá forma de ver los proyectos que se realizarán en el futuro en la empresa.

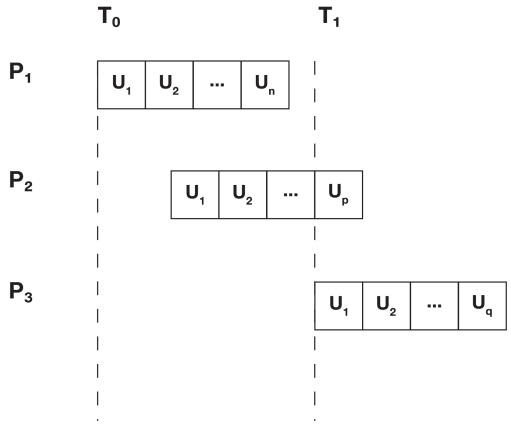


Figura 5.10: Ejemplo de la definición del conjunto de entrenamiento y de pruebas. Para estimar el proyecto P_3, se deben considerar todas las historias que esten completas antes del tiempo t_1, siendo este conjunto $U_{train} = \{U_1, \cdots, U_n, U_1, U_2, \cdots, U_{p-1}\}$, mientras que el conjunto de pruebas corresponde a la historias de usuario del proyecto P_3, correspondiente a $U_{test} = \{U_1, \cdots, U_q\}$.

Para efectos del presente trabajo, denotaremos al conjunto de entrenamiento como el conjunto de historias de usuario definidas por el criterio anteriormente descrito como X_{train}, y el conjunto de pruebas como X_{test}.

5.1.4. Normalización y filtrado

El conjunto de datos en bruto que se utiliza como entrada al framework es el listado de historias de usuario provista por el *backend* anteriormente definido, y por el listado de historias provenientes de la plataforma de gestión particular que se esté utilizando. El conjunto completo de historias de usuario es entonces el listado de todas las historias, cuyos atributos corresponden a una concatenación de los atributos originales y los añadidos en la plataforma de *tags*. El formato inicial de los atributos no es compatible con algoritmos de aprendizaje automático, por lo que es necesario realizar un pre-procesamiento definido a continuación:

1. En bruto, todos los tags corresponden a valores de comportamiento categórico, independiente del tipo de dato que están representando. Por ejemplo, la categoría de 'Complejidad' de una historia de usuario tiene inicialmente valores categóricos $\{1, 2, 3, 4, 5\}$, siendo que en la práctica es un atributo numérico en donde la magnitud de las variables tiene influencia en el valor de la estimación. Este tipo de atributos es convertido desde categórico a numérico, quedando entonces su dominio como $\mathbb{R}$.

2. Siguiendo lo mismo que en el paso anterior, aquellos tags de índole booleana son transformados a atributos que viven en el dominio $\{0, 1\} \subset \mathbb{R}$.

3. A continuación, los tags que son realmente categóricos como por ejemplo las tecnologías relevantes utilizadas en el proyecto en donde puede haber más de una activa, se transforman a vectores *One-Hot* en donde el atributo original se transforma en N atributos nuevos, donde N es el número de valores distintos posibles para el atributo original y donde el atributo nuevo A_i tiene su dominio en $\mathbb{B}$.

4. Por último, se aplica un método de filtrado simple en donde se eliminan aquellos atributos cuya varianza es 0. Una varianza con este valor se traduce en que el atributo

en cuestión tiene el mismo valor en todas las observaciones. Esto hace que el atributo no otorgue información a los modelos y se vuelva completamente irrelevante para la predicción. Realizar este filtrado inicial es importante para asegurar de no caer en problemas debido a la maldición de la dimensionalidad.

5.2. Etapa de selección

Dado que el conjunto de historias de usuario de entrenamiento cambia y aumenta a medida que se van pronosticando nuevos proyectos en el tiempo, cada pronóstico individual se convierte en un problema único de aprendizaje automático por sí solo. Por otro lado, se tiene como objetivo aumentar la precisión del proceso propuesto lo más posible para cualquier tipo de proyecto. Ambos antecedentes hacen necesario que el *framework* se adapte al conjunto de entrenamiento que se le presenta y elija el modelo que entregue los mejores resultados para cada uno de los proyectos a pronosticar.

Antes de definir el conjunto de modelos tentativos con sus respectivos potenciales hiperparámetros sobre los cuales el *framework* decidirá realizar el pronóstico, es necesario definir cómo se realiza la selección de ellos. Esta selección está definida de forma genérica y funcionará para cualquier conjunto de modelos que se deseen explorar.

Para efectos explicativos, sean $M_1, \cdots, M_n$ el conjunto de n modelos distintos que se desean explorar en la selección, en donde por ejemplo M_1 podría corresponder a una regresión logística. Sea además $C^i_1, \cdots, C^i_{p_i}$ el conjunto de p_i configuraciones de hiperparámetros para el modelo M_i que se desean explorar.

5.2.1. Selección de hiperparámetros

Cada uno de los modelos del conjunto que se desea explorar es sometido a un proceso automático de tuneo de parámetros, el cual consiste en la generación de un espacio de búsqueda definido por la combinación de todos los valores de los hiperparámetros que se desean

explorar y la ejecución de una heurística *greedy* para la obtención de la configuración de hiperparámetros óptimo. Cabe notar que este proceso es computacionalmente costoso debido a que la cantidad de configuraciones distintas sigue un aumento factorial.

El rendimiento de una configuración C_j^i, con $j = 1, \cdots, p_i$ está definida según el error cuadrático promedio (MSE) que se obtiene al ponderar el error en un proceso de *3-Fold cross validation* realizado sobre el conjunto de entrenamiento total X_{train}. Se selecciona entonces aquella configuración que tenga el menor error de entre todas las configuraciones. El proceso se puede ver resumido en el siguiente diagrama:

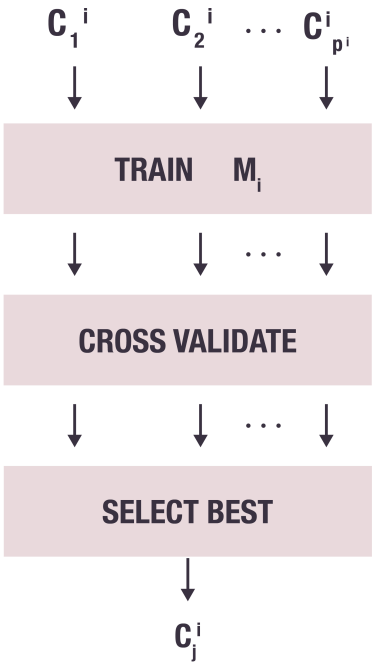


Figura 5.11: Diagrama que describe el proceso de selección de atributos.

5.2.2. Selección de modelos

El proceso de selección de modelos es muy similar al descrito para la selección de hiperparámetros. En este caso, se tienen todos los modelos que se desean explorar con sus configuraciones de hiperparámetros óptimas respectivas. Se obtiene el error de cada uno de los

modelos haciendo otro *3-Fold cross validation* sobre el mismo conjunto de entrenamiento X_{train}, seleccionando aquel que presenta el error más bajo. Se podría argumentar otro camino factible que consiste en dividir el conjunto de entrenamientos X_{train} definido anteriormente en dos nuevos subconjuntos: $X_{model} \subset X_{train}$, con el cual se podrían entrenar todos los modelos predictivos candidatos, y $X_{val} \subset X_{train}$ en donde se validaran los candidatos para encontrar aquel con mejor rendimiento. Cabe notar que en este método hipotético se tendría que $X_{model} \neq X_{val}$ y que X_{model} contendría el 85 % de los datos y X_{val} el 15 %. El problema de esta forma de realizar la validación, es que generalmente el conjunto de validación es demasiado pequeño cuando se tiene un número de observaciones más pequeño y además genera resultados muy distintos cuando se tomen historias significativamente más largas que las demás dentro de cada conjunto. Por otro lado, *cross-validation* es especialmente más efectivo a la hora de validar conjunto de datos con pocas observaciones [11].

5.2.3. Modelos propuestos

Es posible realizar generalizaciones respecto a los modelos que pueden ser más efectivos según las características de las historias de usuario. Si se considera la combinación de los atributos originales y los atributos propuestos, se tiene que un gran pocentaje de ellos corresponden a atributos discretos entre 0 y 1, a los cuales se les puede referir también como atributos booleanos. Este tipo de atributos genera problemas para modelos lineales que utilizan algún tipo de distancia dentro de su implementación, debido a que se intenta compatibilizar distancias reales con discretas. Por tanto, la primera condición que deben satisfacer los modelos candidatos es el manejo efectivo de atributos categóricos convertidos por una transformación *One-hot* a atributos booleanos, sin que estos produzcan ruido durante la predicción.

Para poder validar los nuevos atributos de las historias de usuario propuestos, es necesario además utilizar modelos que entreguen información respecto al uso de los atributos de los datos de entrenamiento. Modelos lineales como una SVR que realizan transformaciones de los datos originales o de su espacio vectorial original no permite transparentar el manejo de los atributos de entrada. Por tanto, y considerando la primera condición establecida, son los

modelos basados en árboles de decisión los que definen a los modelos candidatos.

Uno de los principales problemas conocidos de los árboles de decisión y sus variantes es que tienden a experimentar *overfitting* fácilmente. Para poder reducir este sobreajuste, se utilizaran árboles de decisión simples como base para distintos métodos de ensamblado, lo que permite reducir el sobreajuste sin perder mucha precisión. Finalmente, los modelos propuestos corresponden a ensamblados de árboles, donde se tienen como modelos candidatos a *Random Forest*, *AdaBoost* y *Extreme Random Forest*. El espacio de búsqueda de hiperparámetros de cada uno de los modelos se define a continuación:

Modelos	Hiperparámetro	Explicación	Valores
Extreme Random Forest	n_estimators	Numero de árboles en ensamblado	300
	max_depth	Máxima profundidad del árbol	2, 3, $\cdots$, 15
	min_samples_leaf	Numero de observaciones en nodo hoja	2, 3, $\cdots$, 20
Random Forest	n_estimators	Numero de árboles en ensamblado	300
	max_depth	Máxima profundidad del árbol	2, 3, $\cdots$, 15
	min_samples_leaf	Numero de observaciones en nodo hoja	2, 3, $\cdots$, 20
AdaBoost	n_estimators	Número de árboles en el ensamblado	50, 100, $\cdots$, 200
	learn_rate	Controla cuán rápido incrementan o disminuyen los pesos de las observaciones en el ensamblado	0.05, 0.10, $\cdots$, 0.95
	loss	Función de perdida calculada	lineal, cuadrática, exponencial

Cuadro 5.1: Distintas configuraciones de hiperparámetros utilizadas.

5.3. Etapa de pronóstico

El modelo M seleccionado en la etapa anterior, corresponde a un predictor del esfuerzo de una historia de usuario particular. Para poder predecir el proyecto entero, es necesario recordar que el esfuerzo de toma desarrollar un proyecto P corresponde a la suma del esfuerzo individual de cada historia de usuario que lo compone. Esto se traduce a la ecuación

$$E_P = \sum_{1}^{i} E_{U_i} \tag{5.1}$$

Donde E_p es el esfuerzo requerido para completar el proyecto P y E_{U_i} es el esfuerzo requerido para completar la i-ésima historia de usuario $U_i \in P$. Por tanto, basta obtener las predicciones de todas las historias de usuario de un proyecto para estimar el proyecto mismo.

5.3.1. Distribución del valor esperado del estimador

Para poder transparentar la varianza del estimador, se calcula la distribución del valor esperado del estimador. Es importante hacer la distinción entre esta distribución y la distribución de los datos originales, donde la última no es trivial de predecir. Para esto, se utilizando el método de *Kernel density estimation* (Sección 3.7), en donde el ancho de banda se determina con el método de la regla de Scott [24]. Es importante notar que aunque el método asume distribuciones gausianas al sobreponer las curvas, el resultado en general puede adaptarse correctamente a los datos del estimador. Por otro lado, si bien no es posible realizar suposiciones respecto a la distribución del esfuerzo en un proyecto, si es posible hacerlo cuando se está revisando la distribución del valor esperado de un predictor.

Capítulo 6

Validación y Resultados

Para validar el *framework* definido en el capítulo anterior, se realizará el pronóstico de todos los proyectos existentes en la empresa de desarrollo *Nursoft* un total de 20 veces, a excepción de 2 proyectos en particular que se dejarán como base para poder realizar los pronósticos iniciales. Dichos proyectos corresponden a los primeros en un marco ordenado temporalmente y a los cuales es necesario aislar para que el primer proyecto que se quiere pronosticar pueda tener un conjunto de entrenamiento T_{train}. En cada iteración, se realizarán 3 pronósticos distintos para cada proyecto, en donde en uno se considerarán solamente los atributos propuestos, en otro solamente los atributos en bruto de las historias de usuario y en el último la combinación de todo.

Una vez obtenidos los resultados de cada iteración sobre los 10 proyectos restantes, luego de no considerar los proyectos base previamente definidos, se realizará un análisis respecto a la importancia que tiene cada uno de los atributos de las historias de usuario en el pronóstico del esfuerzo de los proyectos, se contrastarán los resultados obtenidos al pronosticar con y sin los atributos propuestos, y por último, se utilizarán las métricas de $PRED(25)$ y $MMRE$ para comparar el rendimiento del *framework* con el estado del arte actual y con el proceso de pronóstico basado completamente en *Planning Poker* que se utiliza actualmente en la empresa.

6.1. Importancia de los atributos

Los atributos propuestos obtenidos luego del proceso de agregación son en total 159, de los cuales 2 % son booleanos, 3 % numéricos y el restante 95 % son atributos categóricos transformados a través de un proceso *One-hot*. Sus características se encuentran descritos en el anexo B.

Para poder obtener la importancia de los atributos durante el pronóstico, se utilizó el promedio de pesos asignados a cada uno de ellos en todos los árboles utilizados por los métodos de ensamblado seleccionado en cada proyecto. La importancia de aquellos atributos con un valor significativo de importancia pueden ser visualizados en el siguiente mapa de calor:

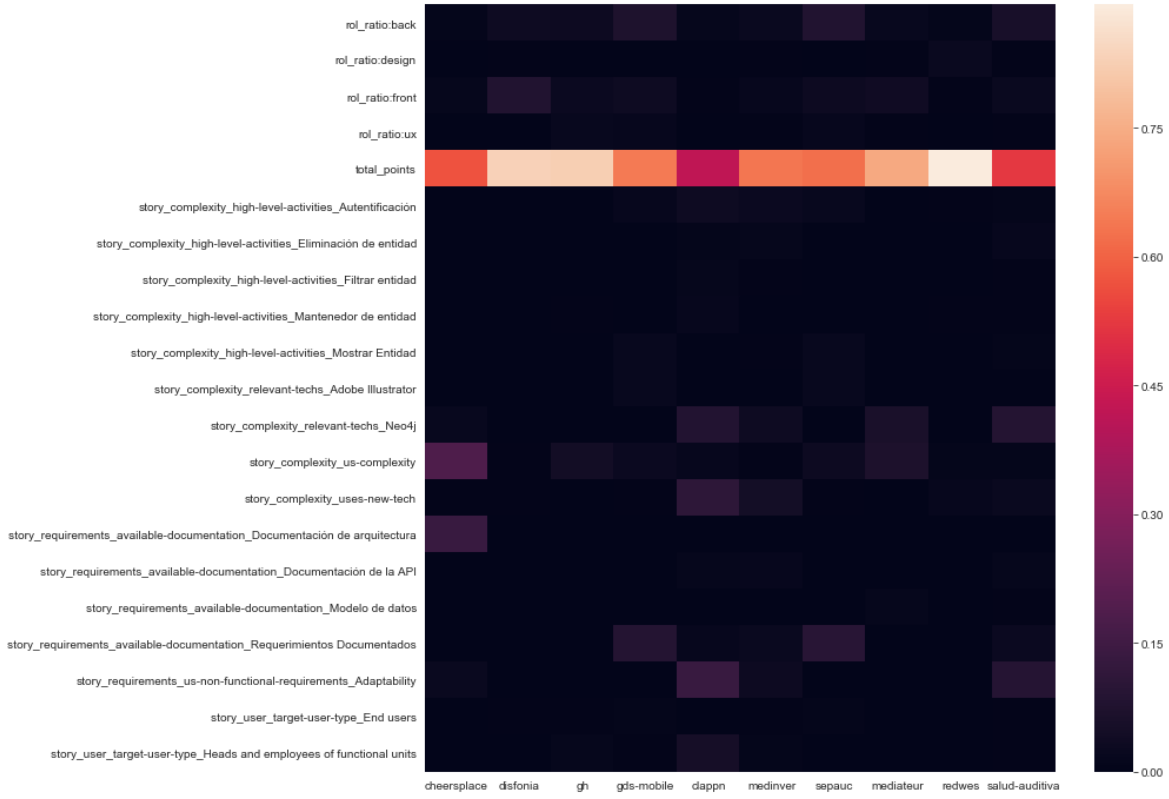


Figura 6.1: Importancia de atributos por proyecto

Esta información representa la importancia promedio de los atributos en los árboles del ensamblado en cada una de las 20 iteraciones. Es evidente notar que el atributo que presenta

el mayor grado de importancia en las decisiones de cada árbol del ensamblado es el de *total_points* en todos los proyectos. Este atributo corresponde al puntaje asignado por el comité evaluador durante el proceso de *Planning Poker*.

La importancia general de los atributos promedio de todos los proyectos se puede visualizar en la siguiente figura:

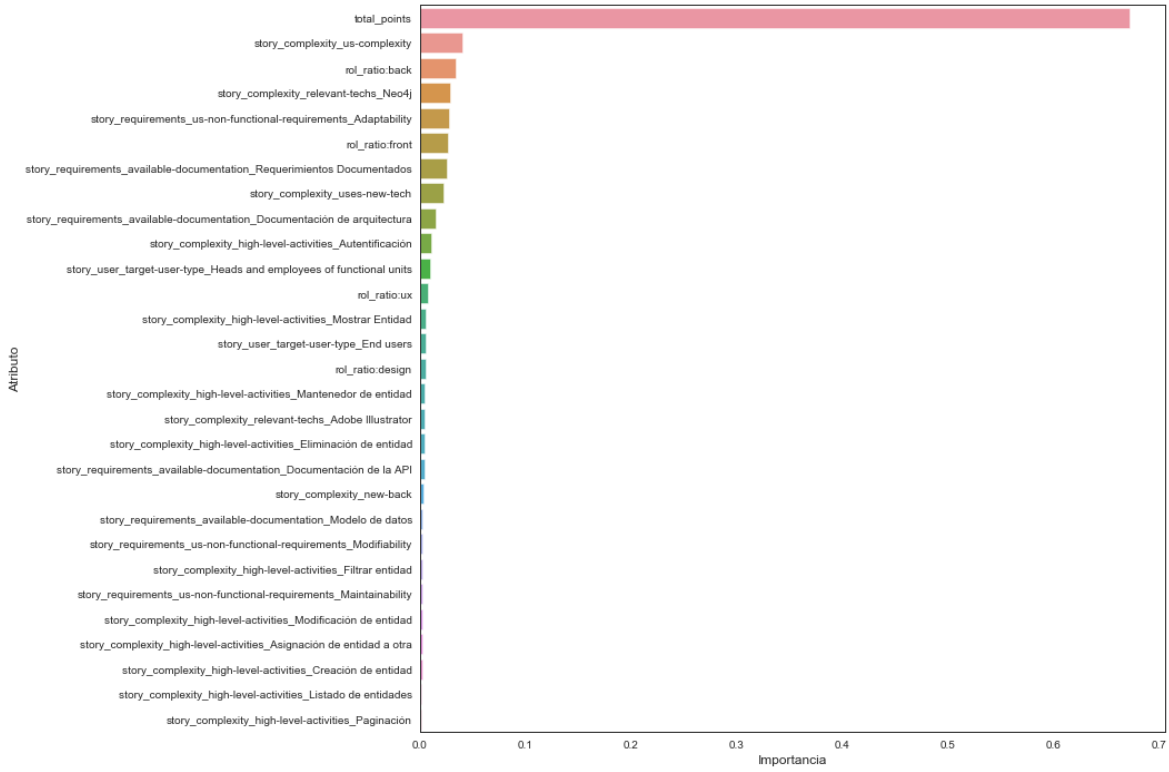


Figura 6.2: Importancia general de los atributos

En donde se hace aún más evidente que es *total_points* el atributo más importante para realizar el pronóstico y que en contraste con el resto de los atributos tiene un valor de importancia porcentual al menos 2 órdenes de magnitud mayor.

Para poder visualizar el resto de los atributos de mejor manera, se elimina *total_points* del mapa de calor pro proyecto, obteniéndose lo siguiente:

Figura 6.3: Importancia de atributos por proyecto, sin considerar el puntaje asignado por el comité

En esta visualización es posible detectar la importancia de los atributos propuestos en las predicciones, en donde se tiene que no existe un atributo que sea muy importante en todos los proyectos de desarrollo. A nivel general, se tiene lo siguiente:

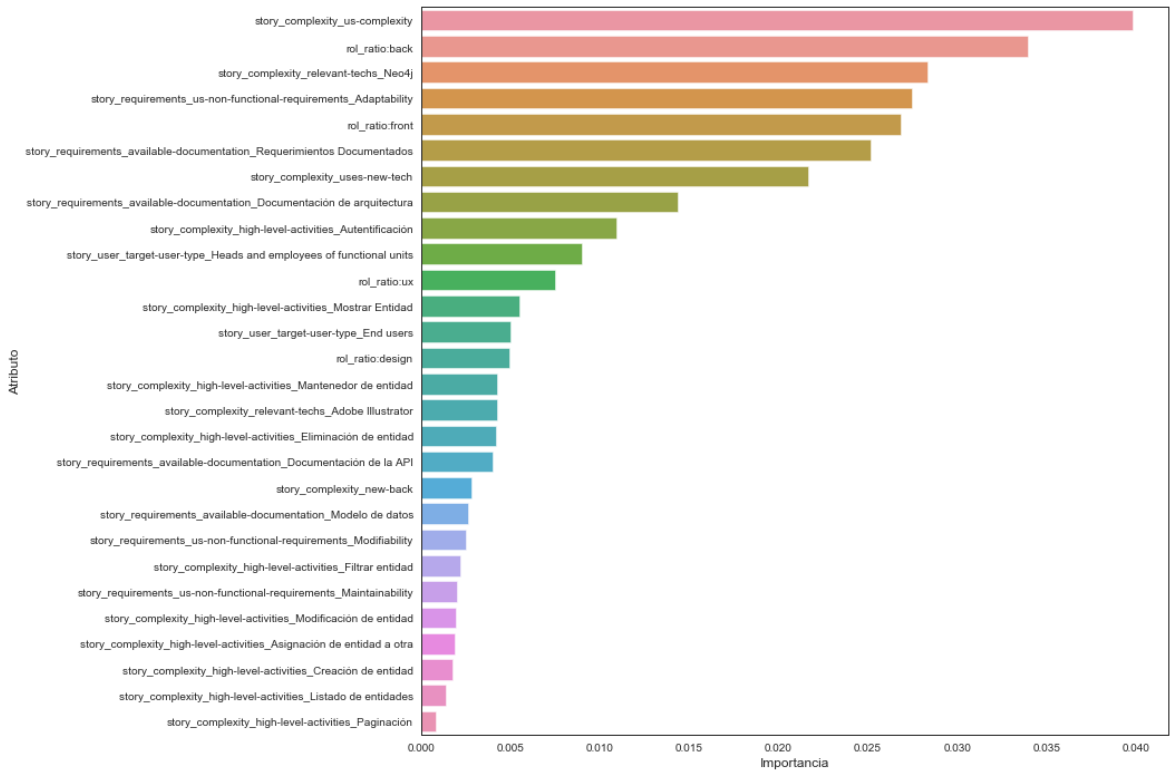


Figura 6.4: Importancia general de los atributos, sin considerar el puntaje asignado por el comité

Se puede notar que en promedio el atributo de complejidad es el que tiene mayor importancia con un valor cercano a 4 %, lo cual sigue siendo marginal en comparación a la importancia de *total_points* que tiene un valor aproximado de 66 % de importancia.

Si se consideran pronósticos realizados utilizando solo los atributos propuestos, se obtiene lo siguiente:

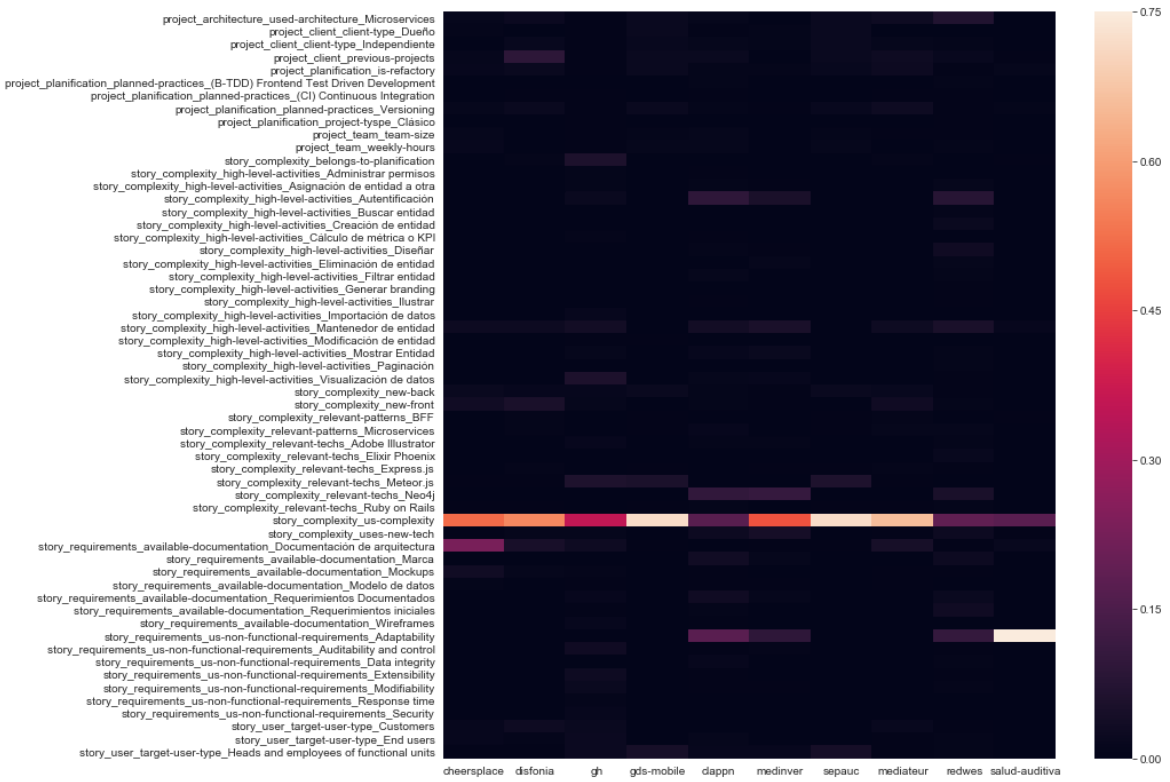


Figura 6.5: Importancia de atributos al pronosticar solo con los atributos propuestos, desagregado por proyecto

Se observa nuevamente que el atributo predominante para realizar las estimaciones corresponde a la complejidad. Dicho atributo es el que mejor representa de manera proporcional cuán grande será una historia de usuario o no con una baja precisión, debido a que corresponde a un número incremental pero subjetivo. En promedio, la importancia se puede ver reflejada en el siguiente diagrama:

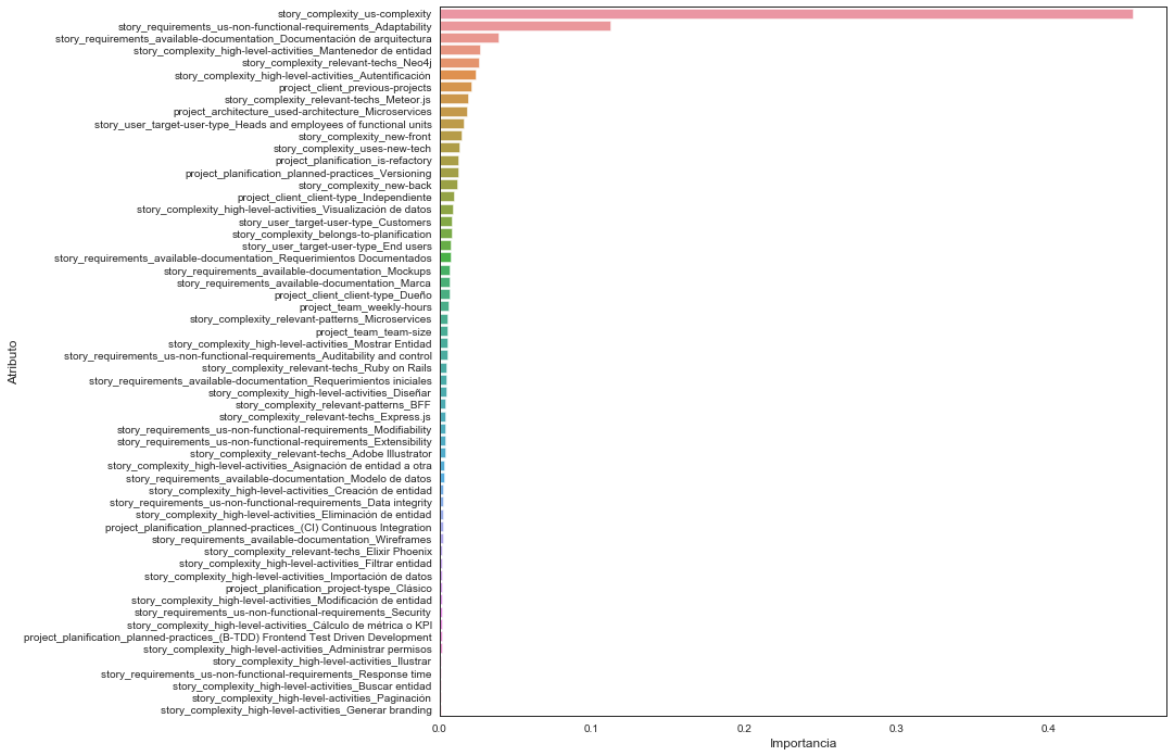


Figura 6.6: Importancia promedio de atributos al pronosticar solo con los atributos propuestos

En donde se puede observar que el atributo relevante que le sigue a la complejidad, es la demanda de Adaptabilidad en los proyectos. Cuando se tiene adaptabilidad como requerimiento no funcional, es necesario generalizar aún más las soluciones de software que se generan y dedicar tiempo a decisiones de bajo nivel que permitan responder al cambio fácilmente. Lo anterior, hace que las historias de usuario tiendan a demorar un poco más que los demás.

6.2. Rendimiento de la solución

A continuación se presentan los resultados obtenidos en torno al comportamiento general del *framework*, su rendimiento en comparación con el método actual que se utiliza para pronosticar el esfuerzo de proyectos en la empresa y su rendimiento general en contraste con

resultados contemporáneos definidos en el estado del arte.

6.2.1. Comportamiento de selección de modelos

Los modelos candidatos propuestos en la sección anterior tienen una frecuencia de selección reflejado en el siguiente mapa de calor:

Figura 6.7: Frecuencia de selección de los modelos pro proyecto

Se puede observar que la selección de modelos es relativamente consistente en la mayoría de los proyectos, a excepción del proyecto *mediateur* en donde se selecciona de forma casi aleatoria entre *Random Forest* y *Extreme Random Forest*.

En promedio, los modelos seleccionados corresponden a los siguientes:

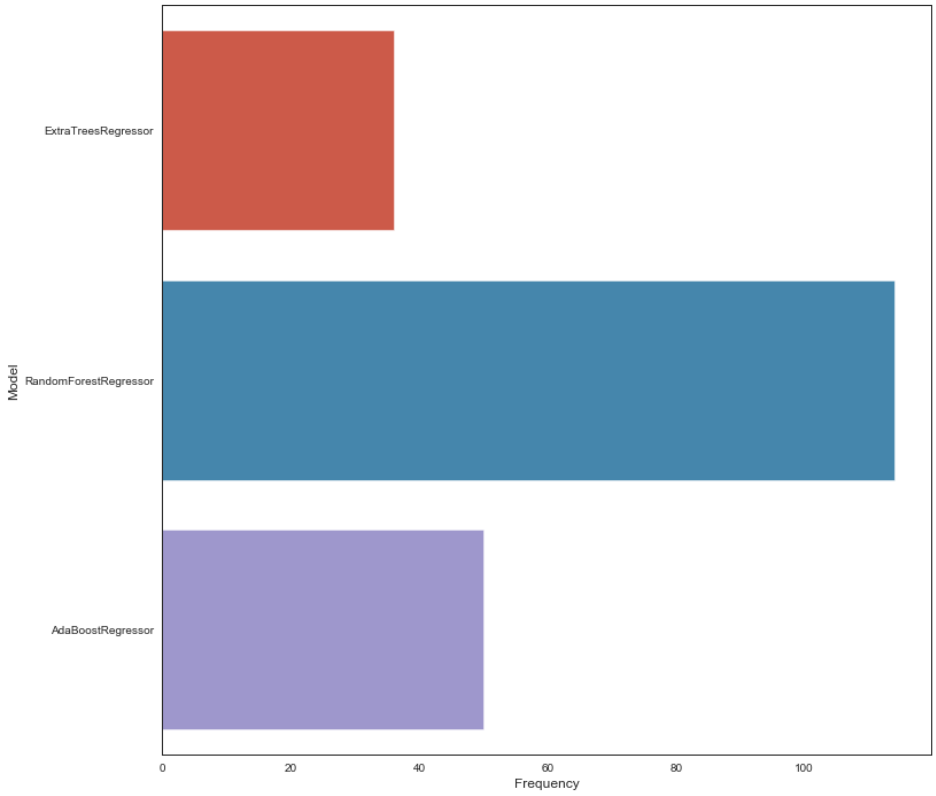


Figura 6.8: Frecuencia de selección general de los modelos

Se puede notar que el modelo de ensamblado seleccionado de mayor frecuencia es *Random Forest*, siguiéndole *AdaBoost*.

6.2.2. Rendimiento del framework según atributos utilizados

Con ánimo de ver el efecto de los atributos propuestos en el rendimiento general del *framework*, se tiene el siguiente diagrama comparativo del error relativo (*MRE*):

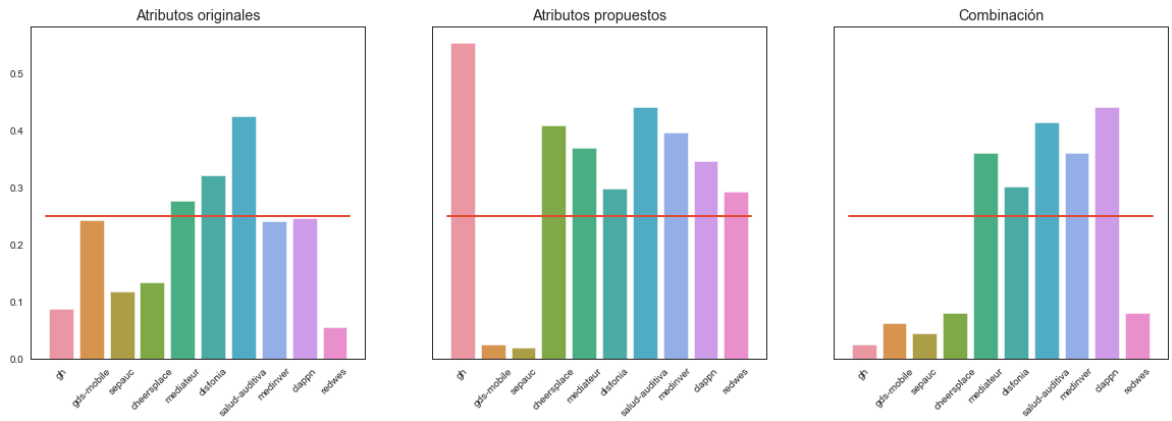


Figura 6.9: Comparación de rendimiento entre conjunto de atributos utilizados para el pronóstico. A la izquierda está el rendimiento al utilizar solo los atributos base, al centro solo los atributos propuestos y a la derecha la combinación. El eje Y corresponde al MRE promedio entre todas las iteraciones de los proyectos pronosticados, por tanto, mientras más alto es peor. La línea roja que se puede observar es el limite de $MRE = 25\,\%$ que es utilizado por la métrica de $PRED(25)$

Se puede observar en la figura 6.9 que en términos de la métrica de $PRED(25)$, el rendimiento de los pronósticos es mejor cuando no se incorporan los atributos propuestos. Esto se puede visualizar aún más considerando el siguiente diagrama:

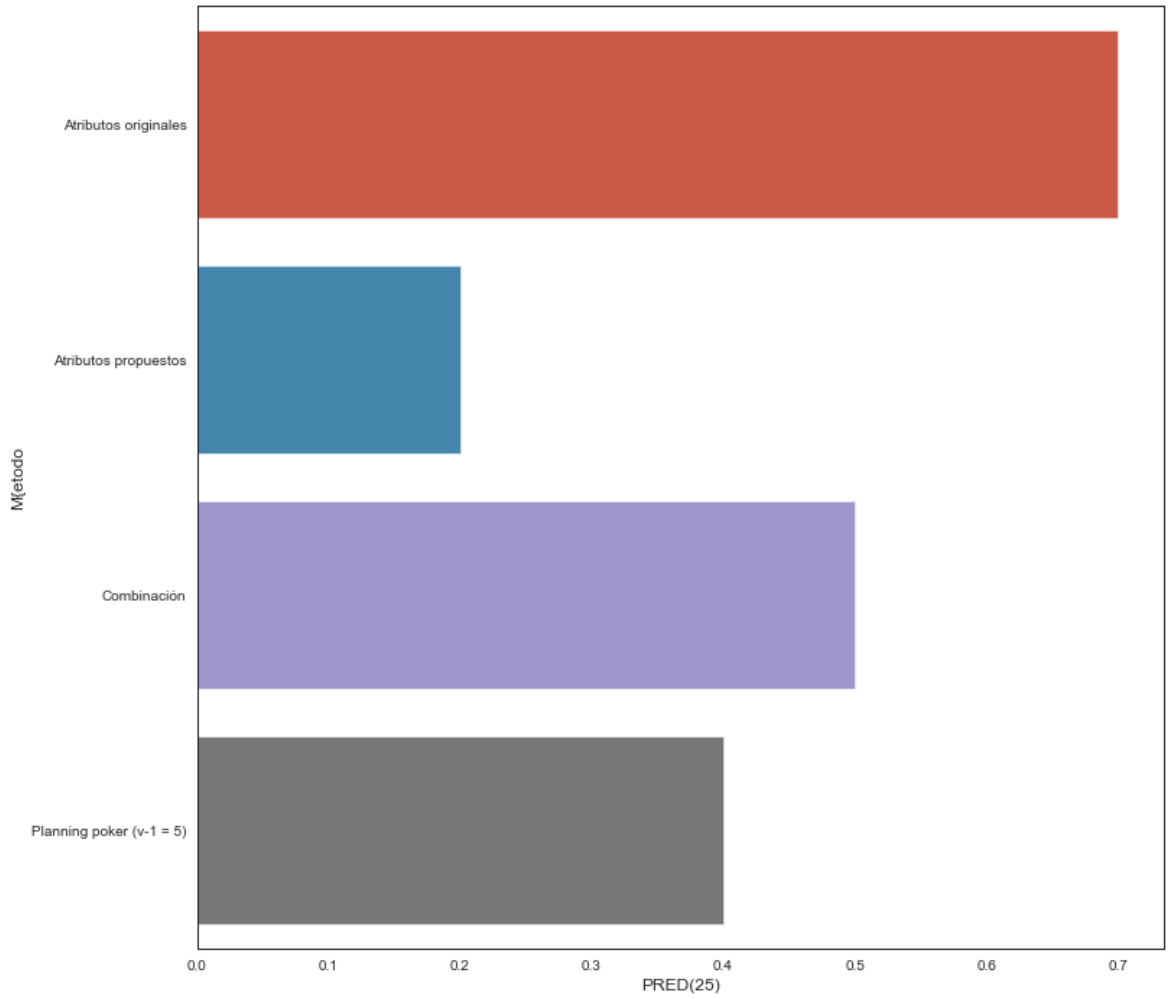


Figura 6.10: Comparación de PRED(25). En rojo se encuentra el valor obtenido con atributos base, en azul con los atributos propuestos, en morado con la combinación y en gris utilizando la metodología actual de la empresa (*Planning poker*)

Se puede observar en la figura 6.10 que el rendimiento del framework utilizando solo los atributos propuestos tiene un rendimiento inferior al *Planning poker* y un aumento de este al incorporar los atributos base, siendo el rendimiento aún mejor cuando no se utilizan los atributos propuestos. Es importante notar que el cálculo del PRED(25) en un dataset de proyectos acotado es muy punitivo, haciendo que se perciba una gran disminución de rendimiento cuando un proyecto en particular no es pronosticado correctamente.

El error relativo promedio de los pronósticos se encuentra resumido en el siguiente diagrama:

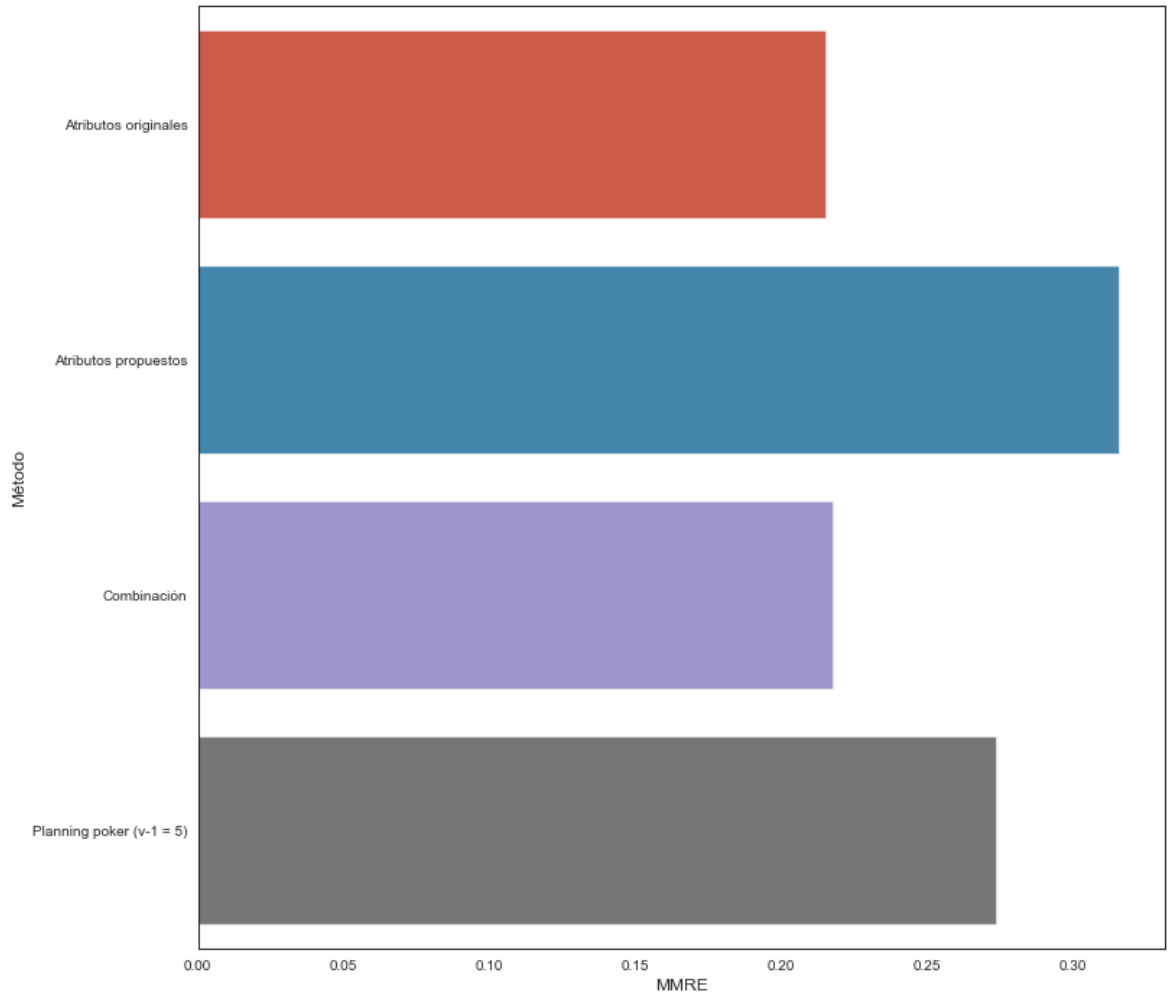


Figura 6.11: Comparación de MMRE. En rojo se encuentra el valor obtenido con atributos base, en azul con los atributos propuestos, en morado con la combinación y en gris utilizando la metodología actual de la empresa (*Planning poker*)

Es muy relevante hacer énfasis en que el error relativo promedio es muy similar entre la utilización de los atributos base y la combinación con los propuestos. Esto da nociones de que la incorporación de los atributos propuestos añade inestabilidad al framework. Para explorar esto, se tiene lo siguiente:

	Base	Combinación
Varianza de MMRE general	0.1095	0.1629
Promedio de la varianza de MRE en cada iteración	0.1199	0.1778

Estos resultados indican que la incorporación de los atributos propuestos aumentan la varianza del *MRE* de los pronósticos, lo que confirma la inestabilidad adicional.

6.2.3. Relación entre error y cantidad de datos

Uno de los desafíos más importantes a la hora de realizar los pronósticos es la poca cantidad de datos que se tiene en el momento. Por tanto, una pregunta natural que surge es si existe alguna relación entre el error relativo y la cantidad de datos disponibles, en donde se esperaría que este disminuyera cuando se tienen más datos. El comportamiento del tamaño de los conjuntos de entrenamiento en cada proyecto puede observarse en el anexo C. A continuación, se muestra la relación entre el tamaño del conjunto de entrenamiento y el error relativo para el pronóstico que tiene la combinación de los atributos:

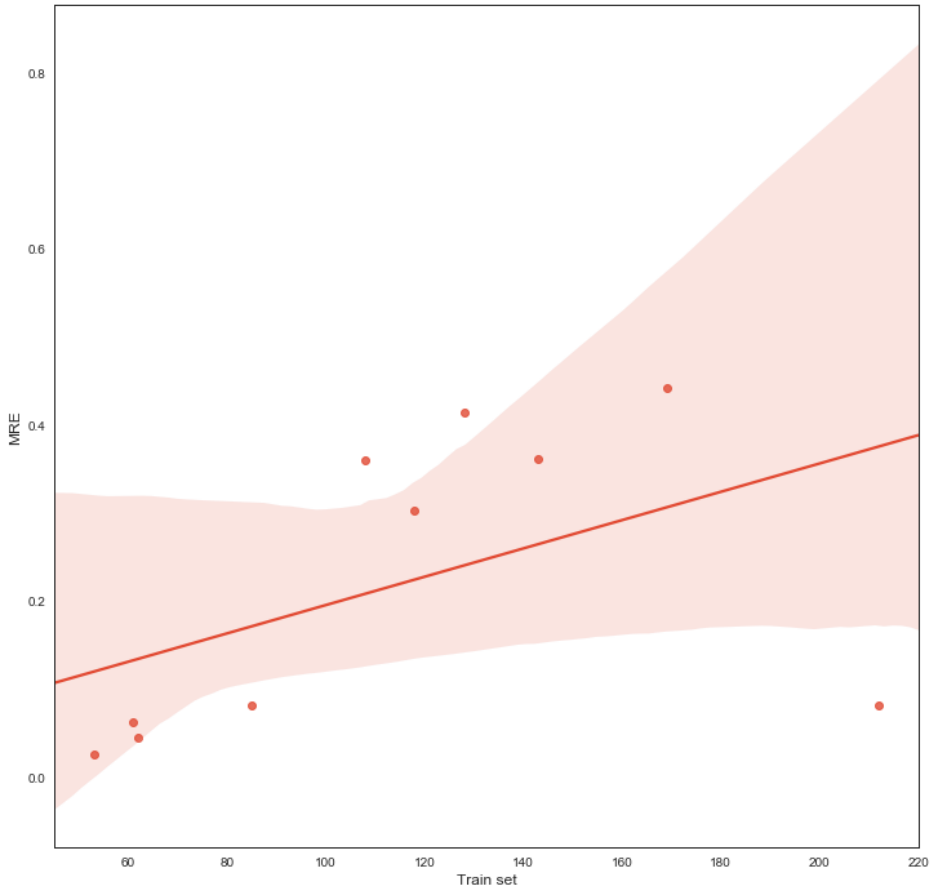


Figura 6.12: Correlación entre tamaño de dataset y el error

Se puede observar una relación directamente proporcional entre el error y el tamaño del dataset, lo cual va en contra de lo que inicialmente se esperaría. Esto es debido a que una de las limitantes más importantes en la estimación de cada proyecto es la baja cantidad de historias de usuario históricas disponibles que conforman el conjunto de entrenamiento completo. Es lógico esperar que a medida que aumente el tamaño del conjunto de entrenamiento, disminuya el error asociado a la falta de datos. No se tiene una explicación muy clara respecto a este fenómeno, pero si se puede entrar a sospechar que podría estar ocurriendo un caso de *negative transfer*.

6.2.4. Distribuciones obtenidas

El framework genera distribuciones cuando se realiza en pronóstico de un proyecto en particular. A continuación, se presentan las distribuciones de los proyectos pronosticados con los atributos base en la iteración 6:

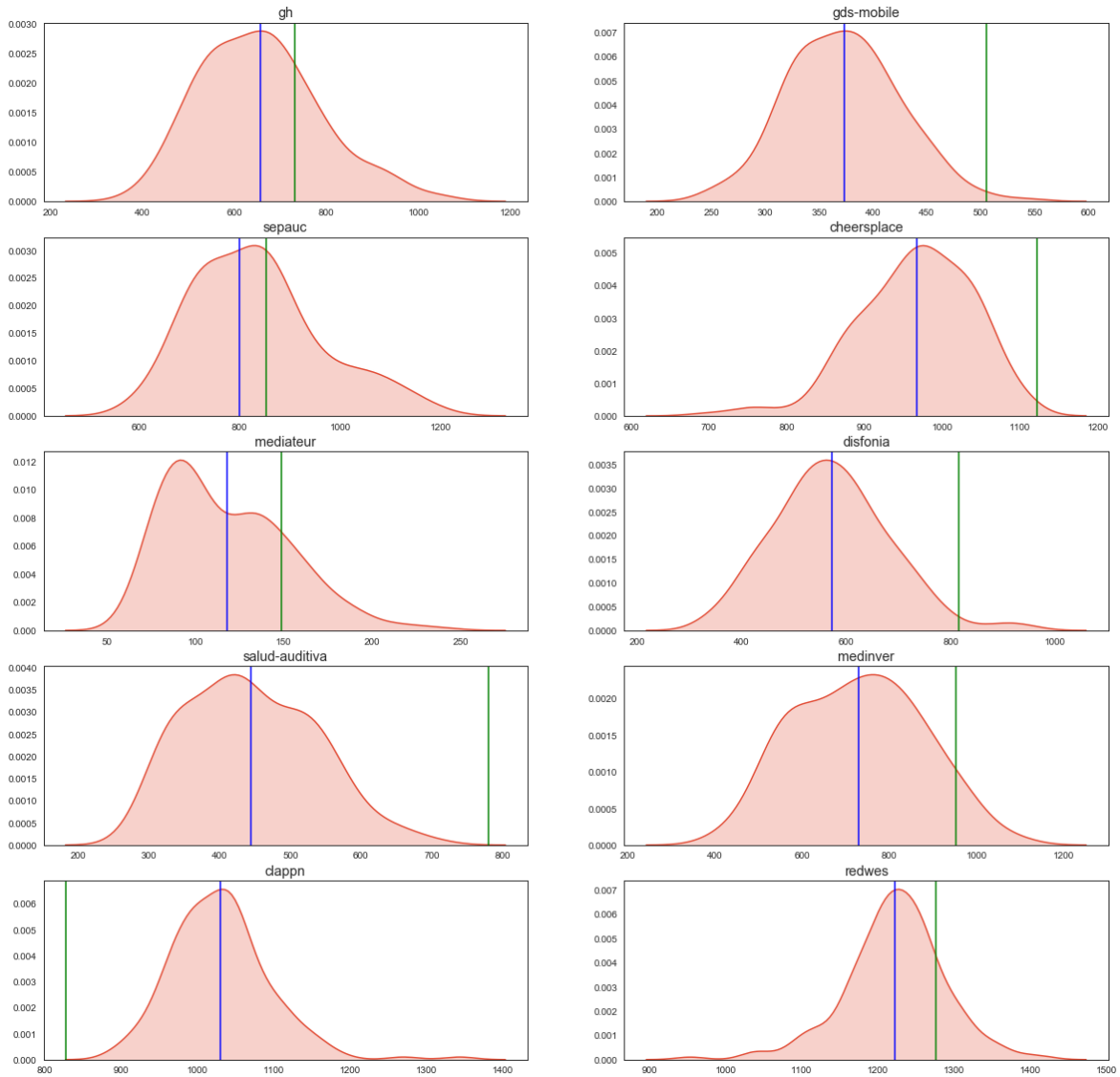


Figura 6.13: Distribuciones del valor esperado del estimador para los proyectos de la iteración 6 en particular utilizando solo los atributos base. La línea vertical azul corresponde al valor entregado por el estimador al considerar el voto de todos los predictores base. La linea verde, por otro lado, corresponde al valor del esfuerzo real

El rendimiento de estas distribuciones no es fácil de medir, debido a que no se conoce la distribución real del valor esperado del estimador. Por lo tanto, al evaluación es más cualitativa que cuantitativa.

Se puede observar que en general las distribuciones obtenidas son representativas del valor promedio obtenido por los métodos de ensamblado, representando adecuadamente también el error del proceso de pronóstico en la visualización presentada. Además, es posible observar que, en general, el *framework* tiende a entregar pronósticos que subestiman el esfuerzo real necesario para terminar los proyectos. Esto tiene sentido, considerando que es la estimación del comité (*total_points*) la que tiene mayor predominancia por sobre los demás atributos, ya que *Planning Poker* tiende a subestimar historias de usuario en la práctica.

Conclusiones

En este trabajo se definió, implementó y validó un *framework* de pronóstico del esfuerzo requerido para realizar proyectos de software a partir de las historias de usuario que los componen. Este *framework* permite agilizar futuras investigaciones al automatizar los procesos de selección de modelos, de selección hiperparámetros y de visualización de la incertidumbre de los pronósticos, haciendolo flexible y fácilmente parametrizable. Además, el framework permite añadir fácilmente cualquier atributo que se desee explorar en el futuro a través de una plataforma de agregación construida para apoyar el proceso de análisis. Para el presente trabajo en particular, se propusieron y validaron un conjunto de atributos candidatos a utilizar en el proceso predictivo. Dentro de este framework se propuso utilizar ensamblados de árboles de decisión para realizar las predicciones, para poder validar la importancia final de los atributos propuestos y manejar de modo adecuado la presencia mayoritaria de variables categóricas. Todos los modelos candidatos utilizados (*Random Forest*, *AdaBoost* y *Extra Random Forest*) mostraron tener una frecuencia de selección lo suficientemente significativa para mantenerse dentro del framework. En particular, fue *Random Forest* el que tuvo una mayor frecuencia de selección, que no fue lo suficientemente superior a la frecuencia de los otros dos modelos como para descartar estos últimos.

De los experimentos realizados, se tiene que entre los atributos propuestos para los pronósticos, los más relevantes en las decisiones que realizaron los árboles de decisión de los ensamblados fueron la complejidad numérica de las historias de usuario, la presencia del requerimiento no funcional de adaptabilidad y la presencia de tecnologías nuevas de naturaleza compleja, en donde en particular, fue *Neo4J* la que tuvo la mayor importancia con respecto a los demás. De los atributos base, es el puntaje asignado por el comité evaluador el atributo

con mayor importancia, mostrando una relevancia de alrededor el 66 %. Entre los atributos base y los atributos propuestos se tiene una diferencia, favorable para los base, de al menos 2 órdenes de magnitud. Esto implica que ante la presencia de nuevos atributos, es predominante la utilización del puntaje del comité evaluador para realizar el pronóstico. Por otro lado, se tiene que el rendimiento del *framework* es superior cuando se utilizan solamente los atributos base que combinándolos con los propuestos. Además, la utilización de los atributos propuestos aumenta la varianza de los pronósticos de los proyectos, influyendo negativamente en la precisión del proceso. Todo esto indica que el puntaje del comité evaluador es un atributo suficientemente robusto por sí solo para poder realizar los pronósticos de los proyectos y es suficiente para tener errores relativos promedio más bajos.

Se definió una estrategia para estimar y visualizar la incertidumbre del pronóstico entregado, la cual se basó en el método de *Kernel density estimation* utilizando un kernel gausiano y un ancho de banda definido por la regla Scott [24]. Esta estrategia es suficiente para satisfacer la problemática de visualización de incertidumbre y transparencia de esta, debido a que se entregan distribuciones de probabilidad representativas de la estimación inicial.

En general, el rendimiento del *framework* depende de los atributos que se consideran a la hora de realizar los pronósticos, teniendo este al realizar pronósticos con la estimación inicial del comité un rendimiento mejor que el proceso actual de estimación basada en *Planning Poker*. La ganancia en rendimiento se ve reflejada por el hecho de que la métrica de $PRED(25)$ aumenta desde un 40 % a un 70 % en la empresa, que es inferior al 80 % óptimo definido según en estado del arte. Sin embargo, hay que tener en consideración que la métrica de $PRED(25)$ en un conjunto de proyectos acotado como el utilizado varía enormemente cuando un proyecto en particular se torna difícil de pronosticar, por lo que hay que tener cuidado también en analizar el $MMRE$ obtenido en los proyectos, que presenta también una mejora en comparación con el estado del arte.

Realizando una comparación con las métricas del estado del arte de modelos de aprendizaje automático 4.3, se tiene que el *framework* tiene un rendimiento comparable y mejor que métodos propuestos en el estado del arte que también utilizan métodos de ensamblado [42], siendo opacado por el rendimiento reportado por métodos de máquinas de soporte. Por lo tanto, el *framework* como proceso es válido para realizar la predicción de las historias de

usuario cuando se utiliza el puntaje asignado durante el *Planning Poker* y significativamente mejor que los pronósticos que se hacen actualmente en la empresa.

Finalmente, como trabajo futuro, existen varias líneas de trabajo que se pueden desarrollar para mejorar el framework. En primer lugar, se puede explorar la estimación de la distribución del valor esperado utilizando métodos más sofisticados, como por ejemplo, la regresión quantílica realizada a través de redes neuronales o algún modelo relevante que permita obtener distribuciones más realistas y que no dependan del supuesto gaussiano en su implementación. En segundo lugar, se podría realizar un estudio que compare modelos de ensamblado utilizando modelos base distintos a los árboles de decisión y contrastar los resultados con los obtenidos en el presente trabajo. Es de especial interés realizar alguna comparación con máquinas de soporte, que son las que tienen el mejor rendimiento reportado en el estado del arte. En tercer lugar, sería interesante buscar maneras de automatizar el proceso de etiquetado de los atributos base para extender al framework la validación de nuevos posibles atributos que puedan ser estudiados. Lo anterior sería de gran ayuda para agilizar el proceso de etiquetado (agregación de atributos) y poder probar nuevos atributos de manera simple y eficiente. Por último, una de las líneas de desarrollo corresponde a incorporar el *framework* al software de gestión actual *BlackSheep* que se utiliza en la empresa para realizar reportes de esfuerzo.

Bibliografía

[1] Jira software. `https://www.atlassian.com/software/jira`. Último acceso 2019-08-01.

[2] Adam Alami. Why do information technology projects fail? *Procedia Computer Science*, 100:62 – 71, 2016. International Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016.

[3] MA Awad. A comparison between agile and traditional software development methodologies. *University of Western Australia*, page 30, 2005.

[4] Damir Azhar, Patricia Riddle, Emilia Mendes, Nikolaos Mittas, and Lefteris Angelis. Using ensembles for web effort estimation. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 173–182. IEEE, 2013.

[5] Nassif A. B., Azzeh M., Capretz L. F., and Ho. D. Neural network models for software development effort estimation: a comparative study. *Neural computation and applications*, (23):2369–2381, 2016.

[6] Zornitza Bakalova and Maya Daneva. A comparative case study on clients participation in a'traditional'and in an agile software company. In *Proceedings of the 12th International Conference on product focused software development and process improvement*, pages 74–80. ACM, 2011.

[7] S Balaji and M Sundararajan Murugaiyan. Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, 2(1):26–30, 2012.

[8] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001.

[9] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[10] Barry Boehm. Cost estimation with cocomo ii. 11 2002.

[11] Rasmus Bro, Karin Kjeldahl, AK Smilde, and HAL Kiers. Cross-validation of component models: a critical look at current methods. *Analytical and bioanalytical chemistry*, 390(5):1241–1251, 2008.

[12] Boehm BW. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[13] Grupo CHAOS. Chaos manifesto 2013. `https://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf`. Último acceso: 2017-05-21.

[14] Marly Monteiro de Carvalho, Leandro Alves Patah, and Diógenes de Souza Bido. Project management and its effects on project success: Cross-country and cross-industry comparisons. *International Journal of Project Management*, 33(7):1509–1522, 2015.

[15] Murillo M. J. et al. A genetic algorithm based framework for software effort prediction. *Journal of software engineering research and development*, 5(4):1–3, 2017.

[16] Soufiane Ezghari and Azeddine Zahi. Uncertainty management in software effort estimation using a consistent fuzzy analogy-based method. *Applied Soft Computing*, 67:540–557, 2018.

[17] Maël Fabien. Boosting and adaboost clearly explained. `https://towardsdatascience.com/boosting-and-adaboost-clearly-explained-856e21152d3e`. Último acceso: 2019-08-02.

[18] Marta Fernández-Diego and Fernando González-Ladrón-De-Guevara. Potential and limitations of the isbsg dataset in enhancing software engineering research: A mapping review. *Information and Software Technology*, 56(6):527–544, 2014.

[19] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[20] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[21] Raffaele Garofalo. *Building enterprise applications with Windows Presentation Foundation and the model view ViewModel Pattern*. Microsoft Press, 2011.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[23] Chaos Group. Chaos report 2015. https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf. Último acceso: 2019-06-18.

[24] Nils-Bastian Heidenreich, Anja Schindler, and Stefan Sperlich. Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *AStA Advances in Statistical Analysis*, 97(4):403–433, 2013.

[25] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[26] John Johnston and John DiNardo. *Econometric methods*, volume 2. New York, 1972.

[27] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd. What accuracy statistics really measure [software estimation]. *IEE Proceedings - Software*, 148(3):81–85, June 2001.

[28] Janet Kolodner. *Case-based reasoning*. Morgan Kaufmann, 2014.

[29] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

[30] Petronio L and Silvio R. L. Software effort estimation using machine learning techniques with robust confidence intervals. *19th IEEE International Conference on Tools with Artificial Intelligence*, pages 181–185, 2007.

[31] Dragan Lazarević and Bojan Prlinčević. Project management: Cost, time and quality. *Center for Quality*, 2014.

[32] Cuauhtemoc Lopez-Martin, Claudia Isaza, and Arturo Chavoya. Software development effort prediction of industrial projects applying a general regression neural network. *Empirical Software Engineering*, 17(6):738–756, 2012.

[33] Jøorgensen M and Shepperd M. A systematic review of software development cost estimation studies. *IEE Trans Soft Eng*, 33(1):33–53, 2007.

[34] Shepperd M. Software project economics: a roadmap. *Future of software engineering*, pages 304–315, 2007.

[35] Shepperd M. Evaluating prediction systems in software project estimation. *Inf Softw Technol*, 54(8):820–827, 2012.

[36] Viljan M and Tomaž H. On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9):2086 – 2095, 2012. Selected papers from the 2011 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA 2011).

[37] Gurpreet Singh Matharu, Anju Mishra, Harmeet Singh, and Priyanka Upadhyay. Empirical study of agile software development methodologies: A comparative analysis. *SIGSOFT Softw. Eng. Notes*, 40(1):1–6, February 2015.

[38] M Douglas McIlroy, J Buxton, Peter Naur, and Brian Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98, 1968.

[39] Leandro L Minku and Xin Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, 55(8):1512–1528, 2013.

[40] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[41] Sam Newman. *Building microservices: designing fine-grained systems*. .ºReilly Media, Inc.", 2015.

[42] Przemyslaw Pospieszny, Beata Czarnacka-Chrobot, and Andrzej Kobylinski. An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software*, 137:184 – 196, 2018.

[43] Ryan M Rifkin and Ross A Lippert. Notes on regularized least squares. 2007.

[44] Gene Rowe and George Wright. The delphi technique as a forecasting tool: issues and analysis. *International Journal of Forecasting*, 15(4):353 – 375, 1999.

[45] Nayan B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010.

[46] Nayan B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010.

[47] Saed Sayad. Support vector machine - regression (svr). `https://www.saedsayad.com/support_vector_machine_reg.htm`. Último acceso 2019-08-01.

[48] Harshdeep Singh. Understanding random forests. `https://medium.com/@harshdeepsingh_35448/understanding-random-forests-aa0ccecdbbbb`. Último acceso: 2019-08-02.

[49] Derya Toka and Oktay Turetken. Accuracy of contemporary parametric software estimation models: A comparative analysis. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pages 313–316. IEEE, 2013.

[50] Adam Trendowicz and Ross Jeffery. *Estimation Under Uncertainty*, pages 81–124. Springer International Publishing, Cham, 2014.

[51] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41 – 59, 2012.

[52] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Citeseer, 2000.

[53] Robert K Wysocki. *Effective project management: traditional, agile, extreme*. John Wiley & Sons, 2011.

[54] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. .ºReilly Media, Inc.", 2018.

[55] Shahid Kamal Tipu Ziauddin and Shahrukh Zia. An effort estimation model for agile software development. *Advances in computer science and its applications (ACSA)*, 2(1):314–324, 2012.

Anexos

A. Proceso de reporte de esfuerzo actual en la empresa Nursoft

El reporte de esfuerzo es un de los procesos fundamentales sobre los cuales el presente trabajo está basado, ya que permiten entregar los valores esperados reales de esfuerzo que se utilizan en el proceso de aprendizaje automático supervisado. Medir la cantidad de horas humanas exactas es un proceso complejo y difícil de monitorear en las organizaciones tradicionales y que generalmente no se realiza. En *Nursoft*, en cambio, se tiene una cultura orientada a la excelencia, en donde el correcto reporte del trabajo realizado es parte de los procesos estándares que realizan todos los trabajadores. Todo esto permite saber con un error reducido el esfuerzo real que toma realizar cada una de las historias.

El proceso de reporte de esfuerzo se realiza a través de la plataforma *BlackSheep* en donde se tiene una sección en su implementación específica para realizar reportes de esfuerzo eficientes (Figura A.1). El reporte es diario, y el esfuerzo final de una historia de usuario se define como la suma de todas las horas reportadas por todos los miembros del equipo, incluyendo horas complementarias provenientes de actividades auxiliares.

Figura A.1: Formulario utilizado en *Nursoft* para el reporte de esfuerzo

B. Características generales de los atributos propuestos luego del proceso de agregación

Para verificar la consistencia de los atributos, se realiza un esquema de atributos nulos que se presenta a continuación:

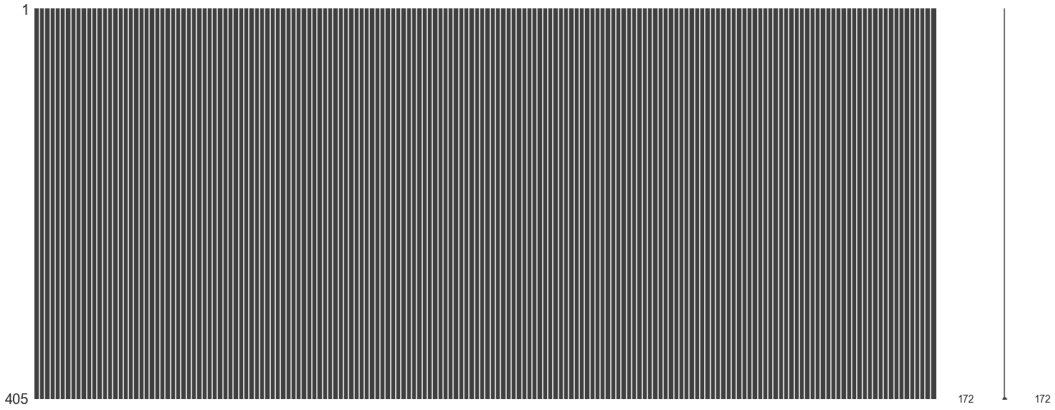


Figura B.2: Análisis de atributos nulos luego del proceso de agregación

En este esquema se puede apreciar que no existen atributos que jamás hayan sido usados. En caso de ser así, existirían agujeros blancos en la visualización anterior. La frecuencia de utilización de atributos está dada por el siguiente esquema:

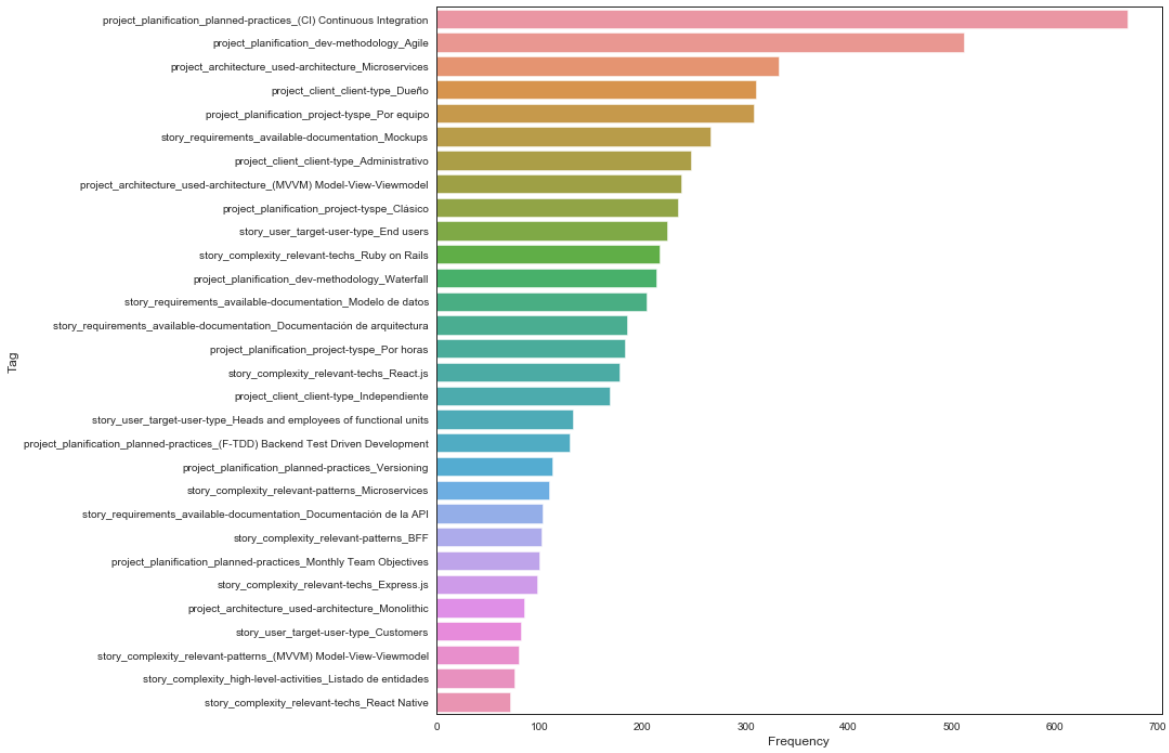


Figura B.3: Frecuencia de aparición de los atributos propuestos. Por simplicidad, no se muestra la frecuencia de todos ellos

En donde se puede notar que el atributo que está más presente en las historias de usuario es la utilización de integración continua y metodologías efectivamente ágiles, continuando además con la adopción de micro-servicios en su arquitectura.

C. Comportamiento del tamaño del conjunto de entrenamiento a través del tiempo

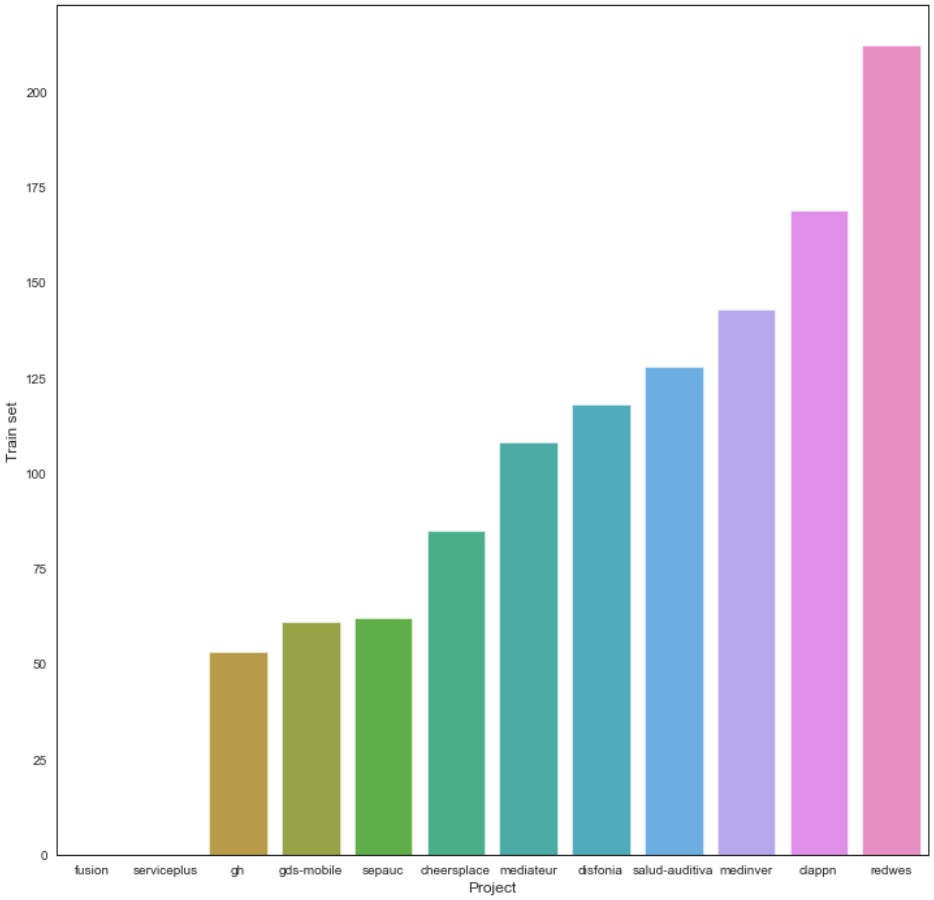


Figura C.4: Evolución del tamaño del training set

En el diagrama anterior, los dos primeros proyectos no tienen un valor de conjunto de entrenamiento debido a que son utilizados como base para los pronósticos. A medida que para el tiempo en cada uno de los proyectos, más grande es el conjunto de datos con el que puede trabajar el *framework*