Universidad Nacional del Centro de la Provincia De Buenos Aires
Facultad de Ciencias Exactas - Departamento de Computación y Sistemas
Ingeniería de Sistemas

Técnicas de aprendizaje para predecir atributos no funcionales en componentes de aplicaciones Android

Agüero, Silvana Ivonne

Minvielle, Martina

Director: Dr. Alejandro Zunino

Co-Director: Ing. Emiliano Sanchez

Trabajo final de carrera presentado como requisito parcial
para optar por el título de
Ingeniera de Sistemas

Tandil, Junio de 2017

Resumen

Durante los últimos años el uso y desarrollo de aplicaciones móviles ha ido creciendo constantemente debido a la proliferación de los dispositivos móviles y el aumento de su capacidad de cómputo. La re-utilización de software mediante la integración de componentes de terceros es una práctica muy común en el desarrollo de aplicaciones. La misma funcionalidad suele ser ofrecida por componentes alternativos que difieren en sus propiedades no-funcionales o atributos de calidad, como su tiempo de respuesta. Por lo tanto es importante elegir la alternativa más adecuada para ejecutar en un determinado contexto.

Por razones prácticas es costosa la ejecución y prueba de todos los servicios para determinar el más adecuado de acuerdo a su calidad de servicio. En este punto, es donde toma importancia definir un proceso para la medición y predicción de las propiedades no-funcionales.

A partir de la aplicación de técnicas de aprendizaje de máquina es posible construir modelos predictivos para estimar el desempeño futuro de un servicio en base a mediciones previas de las propiedades del mismo.

En este trabajo se presenta un enfoque para medir propiedades no-funcionales en tiempo de ejecución y construir modelos de predicción de estas propiedades. Se proponen dos herramientas para asistir la aplicación del enfoque. Una de ellas facilita la obtención de indicadores de propiedades de componentes Android, como tiempo de respuesta o precisión. La segunda herramienta permite utilizar diferentes técnicas de regresión sobre las mediciones obtenidas previamente. Así, se logra construir modelos predictivos sobre alguna propiedad de interés de algún componente específico. El enfoque se evaluó empíricamente sobre grupos de algoritmos y servicios de diferentes dominios.

Agradecimientos

Queremos agradecer a todas las personas que han posibilitado la realización de este trabajo, tanto directa como indirectamente. A nuestras familias, en especial a nuestros padres, Marta, Juan María, Graciela y Juan, quienes desde el comienzo nos apoyaron incondicionalmente y nos brindaron toda su ayuda y cariño en los buenos y malos momentos.

A todos nuestros amigos y grupos de estudio, que nos acompañaron durante el transcurso de la carrera y de quienes nos llevamos grandes aprendizajes.

Por último, pero no menos importante, a los directores de este proyecto, Alejandro y Emiliano, que nos han ayudado brindándonos su apoyo y conocimientos permitiéndonos realizar este trabajo de la mejor manera posible.

Índice general

Índice de figuras

Índice de cuadros

Glosario

ART Android Runtime

API Application Programmatic Interface

CC Correlation Coefficient

CPU Central Processing Unit

DVM Dalvik Virtual Machine

GPU Graphics Processor Unit

HTTP Hypertext Transfer Protocol

Java SE Java Standard Edition

Java ME Java Micro Edition

JVM Java Virtual Machine

MAE Mean Absolute Error

MLP MultiLayer Perceptron

REST Representational State Transfer

RMSE Root Mean Absolute Error

SMO Sequential Minimal Optimization

SGD Stochastic Gradient Descendent

SVM Support Vector Machine

TSP Travelling Salesman Problem

UML Unified Modeling Language

Capítulo 1

Introducción

1.1 Motivación

En los últimos años las aplicaciones móviles se han puesto en el centro de la escena debido a la proliferación de los dispositivos móviles y su creciente capacidad de cómputo y almacenamiento[14]. Estos dispositivos pasaron de ser terminales con capacidades limitadas, generalmente de propósito específico, como agendas electrónicas o teléfonos celulares, a ser pequeñas computadoras de propósito general con grandes capacidades de procesamiento, almacenamiento y acceso a Internet, como tablets y smartphones.

Los dispositivos móviles de hoy en día utilizan sistemas operativos similares a las computadoras personales, como Android y iOS. Por lo tanto, pueden ejecutar software al que hace unos años atrás solo se encontraba en dichas computadoras. Esto, sumado a su costo accesible, pequeño tamaño, movilidad y ubicuidad de las conexiones móviles de alta velocidad, ha impulsado el desarrollo de las aplicaciones móviles para una amplia variedad de fines, incluyendo entretenimiento, juegos, comunicaciones, redes sociales, comercio electrónico, turismo, educación, y mucho más.

Para reducir los costos y tiempos de desarrollo, es común la re-utilización de software mediante la integración de componentes de terceros. Un componente [11] es una entidad de software en tiempo de ejecución que en-

capsula un servicio, es decir, un conjunto de funciones y datos, a través de una interfaz específica Application Programmatic Interface (API). Los componentes se pueden clasificar de acuerdo al ambiente en el que residen durante su ejecución, distinguiéndose así aquellos que se ejecutan en nodos remotos, como los denominados Servicios Web [3], de aquellos que residen en el dispositivo, como procesos en segundo plano, bibliotecas de enlace dinámico, o simples objetos Java.

Las funcionalidades que implementan estos componentes buscan satisfacer las necesidades comunes de muchas aplicaciones, como el procesamiento de texto e imágenes, almacenamiento de datos en la nube, identificación de usuarios, algoritmos de optimización, etc. La misma funcionalidad suele ser ofrecida por componentes alternativos que difieren en sus propiedades no-funcionales o atributos de calidad [9]. Estas propiedades son los aspectos que utilizan los desarrolladores para juzgar su funcionamiento, tales como performance (por ej., tiempo de respuesta), disponibilidad (por ej., tasa de errores) o precisión de la respuesta.

Los dispositivos móviles tienen limitaciones en conflicto como la energía, el acceso a la red y la capacidad de cálculo que determinan el contexto de ejecución de estos componentes y que afecta los atributos de calidad de los mismos y de las aplicaciones que los invocan. Por lo tanto, es importante elegir los componentes adecuados de acuerdo con su calidad de servicio además de la funcionalidad requerida.

Sin embargo, estos componentes suelen ser *cajas negras* para los desarrolladores de aplicaciones móviles, que tienen acceso a la definición de sus APIs pero no así a su implementación interna, por lo que no cuentan con información de sus atributos dinámicos para elegir el componente adecuado en cada contexto de ejecución.

Teniendo en cuenta lo antes dicho, en este trabajo se propone estudiar y aplicar técnicas de aprendizaje de máquina [15] para construir modelos de predicción de las propiedades dinámicas de componentes de software en dispositivos móviles.

En las aplicaciones móviles, donde la disponibilidad de recursos puede variar rápidamente (cambio de red de acceso a Internet, capacidad limitada

de memoria y Central Processing Unit (CPU), etc) y teniendo en cuenta la forma en que los componentes consumen estos recursos, es interesante el uso de estos modelos como criterio de calidad para la selección de servicios y componentes candidatos, tanto en tiempo de desarrollo como en tiempo de ejecución.

1.2 Objetivos y Solución propuesta

El objetivo del trabajo consiste en desarrollar un enfoque para la elaboración de modelos de predicción de propiedades dinámicas, como tiempo de respuesta y precisión, de componentes ejecutados sobre dispositivos móviles. Basándonos en el hecho de que Android es el sistema operativo más difundido para dispositivos móviles, el enfoque se pondrá a prueba sobre diferentes casos de estudio en este sistema operativo.

El enfoque se basa en un proceso de aprendizaje de máquina. Dado un conjunto de componentes, como algoritmos o servicios Web, que implementan la misma funcionalidad y de los cuales queremos predecir alguna propiedad dinámica, el método propuesto es el siguiente:

1. Usar conocimiento del caso de uso para seleccionar características del contexto y de los datos de entrada del componente que puedan ser indicativos de esta propiedad.

2. Generar un conjunto de datos de entrada representativos del espacio de entrada para la evaluación de los componentes.

3. Ejecutar los componentes con las entradas generadas y tomar mediciones de las características identificadas en el punto 1 más la propiedad de interés a predecir: tiempo de respuesta, precisión de la respuesta, etc.

4. Usar estas mediciones con técnicas de aprendizaje de máquina para entrenar y evaluar modelos de predicción de la propiedad.

El enfoque propuesto se pondrá a prueba sobre grupos de algoritmos y servicios Web reales que implementan funcionalidades de interés para desarrolladores de aplicaciones móviles. Esta evaluación no sólo involucrará diferentes casos de estudio, sino también diferentes técnicas, como regresiones y redes neuronales, sobre diferentes propiedades de interés. Para llevar a cabo la medición de los componentes se implementó un framework que simplifica la ejecución de pruebas y mediciones de performance en la plataforma Android, y para el entrenamiento y evaluación de modelos se implementó una segunda herramienta que utiliza software de aprendizaje de máquina como Weka[5].

1.3 Organización

El resto del trabajo se organiza en 5 capítulos. A continuación se da un breve resumen de los temas que se abordan en cada uno de ellos.

En el capítulo2 se presenta el marco teórico, donde se definen los conceptos utilizados a lo largo de todo el informe, tales como: Android, desempeño, precisión, aprendizaje de máquina, regresión, modelos, componentes.

En el capítulo3 se presentan algunos trabajos relacionados desde diferentes perspectivas: herramientas de medición de performance para el sistema Android, y trabajos que involucran predicción de performance con técnicas de aprendizaje de máquina.

En el capítulo 4 se describe el enfoque y las herramientas propuestas. Se detalla la arquitectura e implementación de las mismas, ahondando en las decisiones de diseño que se consideran mas importantes.

En el capítulo 5 se presenta la evaluación del enfoque sobre tres casos de estudio. Se presentan las propiedades consideradas en los escenarios evaluados, los modelos que han sido generados y analizados, y los resultados alcanzados.

Finalmente, en el capítulo 6 se exponen las conclusiones del trabajo realizado, las limitaciones encontradas del enfoque y las herramientas, y posibles líneas de trabajo futuro.

Capítulo 2

Marco Teórico

En este capítulo se presentan los conceptos fundamentales del dominio, el cual está centrado en torno a la predicción de performance y precisión de componentes de software en dispositivos móviles.

2.1 Dispositivos Móviles

Los dispositivos móviles son artefactos electrónicos pequeños que se alimentan a través de una batería de litio. En este contexto, un smartphone o teléfono inteligente es un teléfono móvil con una mayor capacidad de cómputo y conectividad que un teléfono móvil convencional. Mientras el teléfono móvil es un dispositivo inalámbrico electrónico utilizado para acceder y utilizar los servicios de la red de telefonía celular, el término *inteligente* hace referencia a la capacidad de usarlo también como una computadora de bolsillo.

Una de las características más destacadas de los smartphones reside en la posibilidad que brindan de instalar aplicaciones mediante las cuales el usuario final logra ampliar las capacidades y funcionalidades del equipo, obteniendo así una personalización total del dispositivo. Otras características importantes son la capacidad multitarea, el acceso y conectividad a Internet vía WiFi o red móvil, el soporte de clientes de email, la eficaz ad-

ministración de datos y contactos, la posibilidad de lectura de archivos en diversos formatos como .pdf o .doc, y la posibilidad de obtener datos del ambiente a través de sensores especializados como el acelerómetro y el sistema de posicionamiento global conocido como GPS por sus siglas en inglés, entre otros.

Para poder ejecutar aplicaciones en los dispositivos móviles, los mismos poseen, al igual que las computadoras, sistemas operativos. Un sistema operativo es un intermediario entre el usuario de un dispositivo y el hardware del mismo. El objetivo de un sistema operativo es proveer un ambiente en el cual el usuario pueda ejecutar programas en un manera conveniente y eficiente. Un sistema operativo es software que administra el hardware del dispositivo. El hardware debe proveer mecanismos apropiados para asegurar la operación correcta de un sistema y evitar a los usuarios interferir con el funcionamiento apropiado del mismo.

Entre los sistemas operativos más populares en dispositivos móviles se encuentra Android. En las secciones siguientes se describe detalladamente este sistema y la framework que provee para el desarrollo de aplicaciones.

2.1.1 Android

Android es un sistema operativo de código abierto diseñado para dispositivos móviles tales como smartphones y tablets. Este sistema operativo está basado en un kernel Linux y es desarrollado por Google. El mismo cuenta con un middleware extensible y aplicaciones de usuario. Adicionalmente, posee una plataforma de distribución de aplicaciones disponible a partir de la versión 2.2 del sistema, denominada Google Play[1], que permite a los usuarios navegar y descargar aplicaciones que más se ajusten a sus necesidades y preferencias, personalizando de esta forma el dispositivo sencillamente. Por otro lado, también provee un framework para el desarrollo de aplicaciones[2] que utiliza Java como lenguaje de su interfaz (API).

La plataforma Android utiliza la máquina virtual Dalvik (Dalvik Vir-

[1] https://play.google.com
[2] https://developer.android.com

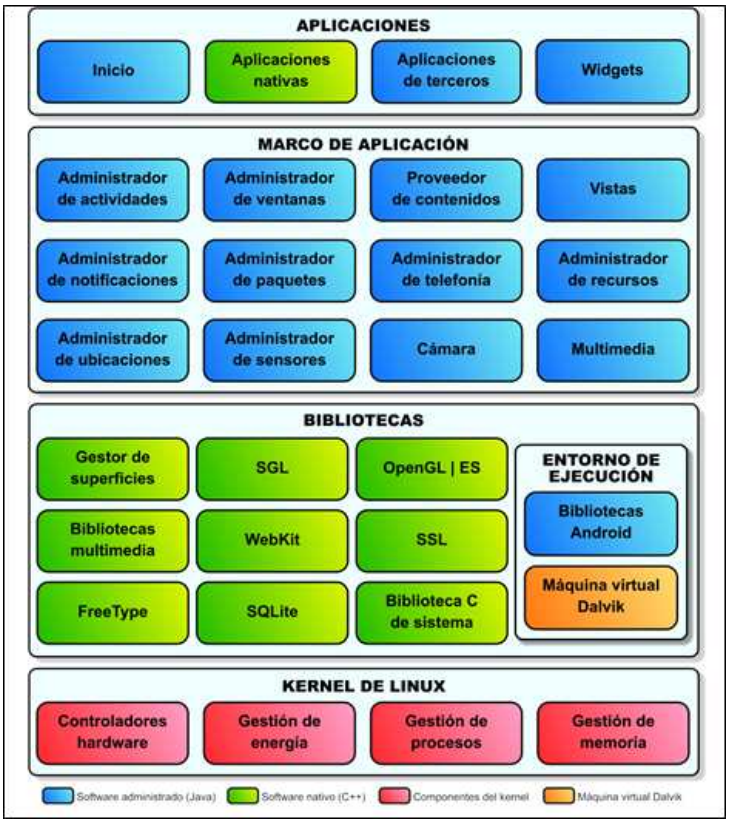

Figura 2.1: Diagrama de la arquitectura en capas empleada por Android

tual Machine (DVM)) para ejecutar aplicaciones programadas en Java a partir de la versión 5. Debido al escaso poder de procesamiento y memoria limitada de los dispositivos que ejecutan Android, no fue posible utilizar la máquina virtual Java estándar por lo que la compañía Google tomó la decisión de crear una nueva, la DVM, que fue optimizada para requerir poco uso de memoria y diseñada para ejecutar en simultáneo múltiples instancias de la máquina virtual, delegando en el sistema operativo Android subyacente el soporte para el aislamiento de procesos, la gestión de memoria e hilos de ejecución.

En la figura 2.1 se puede ver la arquitectura en capas empleada por el sistema Android. Las diferentes capas de la arquitectura son descritas a continuación:

- Kernel Linux: Android utiliza el núcleo de Linux como una capa de abstracción de hardware para los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente de hardware pueda ser utilizado mediante las llamadas correspondientes, sólo debe considerarse al momento de incluir un nuevo componente de hardware que los fabricantes hayan desarrollado los drivers correspondientes. Además del soporte de drivers, la capa es responsable de proporcionar otros servicios como la seguridad, el manejo de la memoria, procesos, etc.

- Entorno de ejecución de Android: como se ha adelantado previamente, cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik, la cual interpreta un lenguaje ligeramente diferente al tradicional bytecode de Java Virtual Machine (JVM). A partir de la versión 5.0 de Android, Dalvik es reemplazada por Android Runtime (ART) la cual logra reducir el tiempo de ejecución del código Java hasta un 33%. También se incluye en el entorno un módulo de librerías nativas con la mayoría de librerías disponibles en lenguaje Java. Estas bibliotecas, si bien resultan diferentes a las ofrecidas por Java Standard Edition (Java SE) y Java Micro Edition (Java ME), proveen prácticamente la misma funcionalidad.

- Bibliotecas: incluye un conjunto de bibliotecas nativas escritas en lenguaje C y C++ usadas en varios componentes de Android que proporcionan la mayor parte de las características de Android.

- Marco o framework de Aplicaciones: este es el framework que proporciona Android para el desarrollo de aplicaciones, servicios y otros componentes. Todo el conjunto de funciones del sistema operativo y bibliotecas nativas está disponible a través de este framework, cuya API esta escrita en el lenguaje Java. El framework permite que los desarrolladores tengan acceso a las mismas Application Programmatic Interface (API) utilizadas por las aplicaciones base del sistema. El foco principal del diseño de esta capa ha sido simplificar la

re-utilización de componentes: las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a restricciones de seguridad), un mecanismo que permite a los usuarios reemplazar fácilmente componentes.

- Aplicaciones: Este nivel contiene todas las aplicaciones de usuario, tanto las incluidas por defecto en Android así como como aquellas que el usuario vaya añadiendo posteriormente ya sean de terceros o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y bibliotecas de los niveles inferiores. Se brindará una descripción más detallada en la siguiente sección.

2.1.2 Aplicaciones Android

Además de las características técnicas, es importante resaltar que la popularidad de Android ha crecido muy rápidamente desde su lanzamiento. Esto se debe a la versatilidad que Android otorga a los dispositivos a través de las aplicaciones, que permiten adaptarlos según las necesidades de los usuarios. Android también permite que las aplicaciones se adapten a las características del dispositivo (pantalla, sensores, etc.), aprovechando las capacidades particulares de cada uno.

Un aspecto clave del diseño de las aplicaciones en Android es que éstas pueden reutilizar componentes de otras aplicaciones instaladas en el dispositivo. Por ejemplo, si una aplicación desea tomar una fotografía, es probable que ya exista una aplicación que cumpla esa funcionalidad, entonces, la nueva aplicación puede utilizar la existente sin necesidad de desarrollar una actividad propia para utilizar la cámara, esta invocación se realiza de modo tal que sea transparente para el usuario final.

El sistema Android provee cinco tipos de componentes básicos para el desarrollo de aplicaciones: Activity, Service, Content Provider, Broadcast Receiver e Intent. El componente *Activity* representa una pantalla simple que provee interfaz de usuario. Una aplicación puede estar formada por una o mas actividades que trabajan en conjunto y representan diferentes pantallas o vistas.

Los *Content Providers* son los encargados de administrar la información compartida por las aplicaciones. Las aplicaciones pueden almacenar sus datos en el sistema de archivos, en una base de datos SQLite, en la Web, o en cualquier otro lugar de acceso. A través del *content provider*, otras aplicaciones pueden consultar, o incluso modificar estos datos.

Las aplicaciones también pueden iniciar servicios que se ejecutan en segundo plano para realizar operaciones que requieran gran cantidad de tiempo, o interactúen con procesos remotos, por ejemplo la reproducción de música en segundo plano o la descarga de datos mientras el usuario interactúa con una aplicación diferente.

El componente *Intent* es un objeto de acción que facilita la comunicación entre componentes. Un intent puede verse como un mensaje entre componentes, por ejemplo, para iniciar una actividad o servicio, o solicitar una acción. Por último, el componente *Broadcast Receiver* funciona como puerta de enlace a otros componentes, respondiendo a los anuncios (Intents) originados por el sistema u otras aplicaciones, por ejemplo, cuando la pantalla se apaga, la batería es baja, o al capturar una fotografía.

2.2 Componentes de software

En el marco del desarrollo de software nos encontramos con la posibilidad de reutilizar código previamente desarrollado, testeado y deployado que cumple con una determinada funcionalidad, ahorrando tiempo y esfuerzo al desarrollador. Estas piezas de códigos ya implementadas se distribuyen como componentes de software que encapsulan a un conjunto de funciones y datos relacionados.

La re-utilización es uno de los objetivos principales al momento de diseñar un componente de software de calidad para ser usado en diferentes programas. La comunicación entre componentes se realiza a través de interfaces. Cuando un componente ofrece servicios al resto del sistema, el mismo proporciona una interfaz que especifica los servicios que otros componentes pueden utilizar y la manera en que pueden hacerlo. La interfaz puede verse como una firma del componente ya que el cliente no necesita

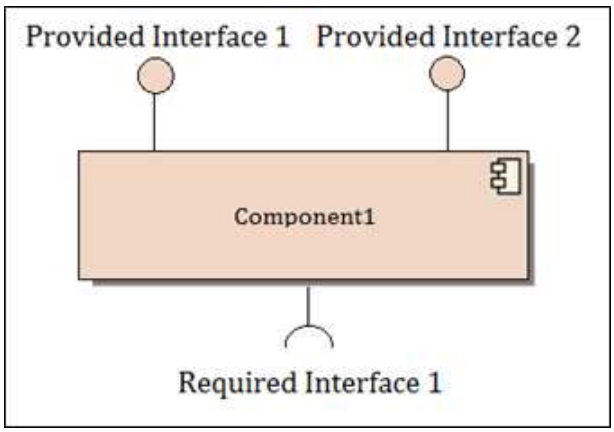

Figura 2.2: Componente UML con interfaces proveídas y requeridas.

conocer el procesamiento interno del componente para utilizarlo, condición que respeta el principio de encapsulamiento. Por otro lado, cuando un componente necesita de otro para su funcionamiento, el mismo establece las interfaces requeridas donde especifica los servicios que necesita. De acuerdo al lenguaje de modelado Unified Modeling Language (UML), las interfaces proporcionadas por componentes son representadas con símbolos de lollipop en el borde del componente y las interfaces requeridas por medio de sockets abiertos en el borde externo, como se ilustra en la figura 2.2.

Otra de las características fundamentales de los componentes es su capacidad de ser sustituidos tanto en tiempo de diseño como en tiempo de ejecución, pudiendo ser reemplazados por actualizaciones u otras alternativas sin romper el sistema en el que los componentes funcionan. El reemplazo es posible si el componente sucesor provee al menos la misma funcionalidad que el componente a reemplazar y si requiere a lo sumo las mismas funciones que el componente inicial.

2.3 Atributos de calidad

Además de su interfaz y funcionalidad, los componentes de software se caracterizan por un conjunto de propiedades que representan los aspectos no-funcionales o de calidad de servicio, llamados propiedades o atributos de calidad. Estas propiedades son características medibles que utilizan los usuarios para juzgar su funcionamiento [8]. Algunos ejemplos de estas propiedades son: el tiempo de respuesta, la disponibilidad, etc.

Al diseñar un sistema de software no sólo se espera cumplir con los objetivos de negocio sino también alcanzar un determinado grado de calidad de software capaz de satisfacer al grupo de usuarios y/o diseñadores. El diseño de la arquitectura de un sistema depende mayormente de los atributos de calidad demandados por los stakeholders. Largos tiempos de respuesta, caídas del sistema, interfaces confusas, no son características deseables en un sistema, por lo que toda decisión respecto al diseño de la arquitectura debe conducir al cumplimiento de ciertos atributos de calidad al mismo tiempo que cumple con la funcionalidad requerida.

Estos atributos de calidad se pueden dividir en dos grupos en base al momento en el cual son medidos. Un grupo incluye atributos cuantificados durante el tiempo de diseño como la escalabilidad, modificabilidad, etc. El segundo grupo incluye atributos cuantificables mientras el sistema se ejecuta como usabilidad, seguridad, etc. Este trabajo se enfoca en la predicción de propiedades dinámicas o en tiempo de ejecución.A continuación se describen las dos propiedades consideradas en la evaluación del enfoque.

Tiempo de respuesta

Se puede medir la calidad de un sistema a través de su desempeño (performance) evaluando la efectividad del uso de los recursos disponibles en tiempo de ejecución. Dependiendo el contexto, el desempeño puede medirse a través de varias propiedades, como el tiempo de respuesta o la latencia. El rendimiento de un sistema engloba, generalmente, el tiempo de los eventos que se producen y que el sistema debe responder a ellos. Estos

eventos pueden ser muy variados tales como alarmas, mensajes, peticiones a usuarios o procesamiento, pero básicamente se considera de todos ellos el tiempo que tarda el sistema para responder al evento. La complejidad para el manejo de estos eventos radica en su fuente, ya que pueden provenir desde una solicitud de usuario, de otros sistemas o desde el interior del propio sistema.

Precisión

Cabe destacar que no existe una definición estándar sobre el significado de precisión en un sistema, ya que se trata de una medida que evalúa qué tan exacta es la respuesta de una función (operación) de un componente, y cada función está ligada a resolver un problema o funcionalidad particular. Por ejemplo, en un problema de detección de rostros, la precisión puede medirse como la cantidad de rostros correctamente detectados sobre los rostros totales presentes, y en un problema de optimización puede significar el grado de cercanía del valor de la solución encontrada respecto a la solución óptima.

2.4 Aprendizaje de máquina

El aprendizaje de máquina o aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras *aprender* a partir de datos suministrados en forma de ejemplo. El aprendizaje a partir de datos es la base para comprender el proceso de aprendizaje de máquina ya que los datos son la única herramienta de la que se dispone y conoce a ciencia cierta sobre las características de un dominio cualquiera. El aprendizaje de máquina puede entenderse haciendo una analogía con el aprendizaje humano basado en la experiencia, en donde el hombre basa su conocimiento en tres partes: *i*) recuerdo, el hombre reconoce cuando ha sido la última vez que estuvo en una determinada situación (dataset), *ii*) adaptación, reconoce la última vez que se probó una acción (salida producida) y *iii*) generalización, reconoce si ha funcionado o

no esta acción (si fue correcta o no). El término generalización refiere a la similitud entre diferentes situaciones de manera tal que las opciones que han sido aplicadas en casos previos pueden ser usadas en nuevos casos.

El aprendizaje de máquina, entonces, es un proceso para que las computadoras modifiquen o adapten sus acciones (predictivas o de control) para que sus resultados sean más precisos, precisión que refleja la proximidad respecto a las acciones correctas. El aprendizaje de máquina reúne ideas de neurociencia, biología, estadística, matemática y física, para generar técnicas y hacer que la computadora aprenda. Un área importante relacionada con el aprendizaje de máquina es la minería de datos, el proceso de extraer información útil de un conjunto de datos masivos por medio de algoritmos eficientes.

Si se define el aprendizaje de máquina como la mejora de tareas a través de la experiencia, surge el cuestionamiento de como la computadora puede saber si está aprendiendo mejor o de qué forma podría mejorar ese aprendizaje. De aquí, surgen diferentes tipos de técnicas o algoritmos de aprendizaje. Por ejemplo, se le puede indicar a un algoritmo la respuesta correcta para un problema, así, la próxima vez que se aplique su desempeño será mejor. También, podría indicarse un conjunto de respuestas correctas para que el algoritmo *adivine* la forma de obtener estas respuestas para otros problemas (generalización). Alternativamente, se puede indicar si la respuesta obtenida es correcta o no sin señalar la respuesta real, que el algoritmo debería ser capaz de encontrar. Una variante podría ser asignarle un puntaje a la respuesta obtenida por el algoritmo que indique cuán correcta resulta ser.

Estas diferentes alternativas proveen una forma de clasificar las diferentes métodos de aprendizaje que será detallada en la siguiente sección. Cabe destacar que por más que existan distintos tipos de aprendizaje, todos los métodos comparten el mismo objetivo de generalización: la técnica debe producir salidas sensibles para datos de entrada que no fueron encontrados durante el aprendizaje, teniendo en cuenta también que el algoritmo debe lidiar con ruido en los datos, es decir, imprecisión en los valores que es inherente a la medición de cualquier proceso real.

2.4.1 Clasificación de las técnicas de aprendizaje

El modo de aprendizaje que una técnica particular puede realizar queda determinado por la naturaleza de los datos de entrada, es decir, los datos de entrenamiento (*dataset*). Básicamente las técnicas de aprendizaje se clasifican en tres grandes grupos: aprendizaje supervisado, no supervisado, y por refuerzo.

El aprendizaje supervisado utiliza un conjunto de datos basado en dos pares de objetos: los datos de entrada o conjunto de ejemplos del dominio y las respuestas correctas (*targets*) para una propiedad determinada . A través de las respuestas correctas provistas y basado en el conjunto de datos la técnica de aprendizaje generaliza el comportamiento para responder a todas las posibles entradas. Este modo de aprendizaje, entonces, es un proceso que se realiza mediante un entrenamiento controlado por un agente externo que determina la respuesta que debería generar la técnica a partir de una entrada determinada.

Dentro del aprendizaje supervisado, las técnicas pueden separarse en dos grupos de acuerdo a la naturaleza de la propiedad o respuesta.

Clasificación Consiste en asignar a cada ejemplo una etiqueta o clase a la que pertenece basado en el entrenamiento de ejemplares de cada clase. Los datos de entrenamiento son instancias que pertenecen a una única clase y el conjunto de clases cubre todas las salidas posibles, por eso se considera al proceso de clasificación como un proceso discreto. El algoritmo de clasificación tiene como objetivo encontrar umbrales de decisión que sirvan para identificar las diferentes clases.

Regresión El proceso de regresión predice valores numéricos de atributos a partir de funciones matemáticas polinomiales que describan o se ajusten lo más posible a todos los puntos del dominio, es decir, todos los valores del conjunto de entrenamiento correspondientes a la propiedad que se quiere predecir. Generalmente, se considera un problema de aproximación de función o interpolación al encontrar un valor numérico entre los valores conocidos. Por lo tanto, el eje primordial

del proceso de regresión es encontrar la función que mejor represente al conjunto de puntos, ya que funciones con distintos grados de polinomios producen diferentes efectos.

En el aprendizaje no supervisado, la máquina simplemente recibe los datos de entrada sin etiquetas o respuestas correctas como en el método supervisado, ni valores de recompensa desde el ambiente. Aun así, es posible desarrollar un framework formal para llevar a cabo aprendizaje no supervisado basado en la noción de que el objetivo es construir una representación de la entrada que puede ser usada para tomar decisiones, predecir futuras entradas, comunicar eficientemente entradas para otras máquinas, entre otras posibilidades.

El aprendizaje no supervisado puede entenderse como la búsqueda de patrones en los datos independientemente del ruido presente en los mismos. Por ejemplo, las técnicas de agrupamiento (clustering) son técnicas de aprendizaje no supervisado que agrupan un conjunto de objetos de modo tal que los objetos pertenecientes a un mismo grupo (*cluster*) comparten algún tipo de similitud entre ellos, de igual sentido que se diferencian con los objetos de otro grupo. A diferencia del proceso de clasificación, los grupos o clases no son conocidos fehacientemente antes del entrenamiento, un claro método de aprendizaje no supervisado.

Por ultimo, el aprendizaje por refuerzo se basa en la idea de no disponer de ejemplos completos del comportamiento deseado por el algoritmo, es decir, no indicar durante el entrenamiento exactamente la salida que se desea proporcione el clasificador ante una determinada entrada, sólo se le indica si la salida obtenida se ajusta a la deseada y en función a ello se re configuran los pasos.

2.4.2 Técnicas contempladas

El foco principal del trabajo es el desarrollo de un enfoque para predecir propiedades no-funcionales de componentes de software en ejecución, como el tiempo de respuesta, precisión de las respuestas, entre otros. Estas propiedades son valores continuos, motivo por el cual se entrenan y eva-

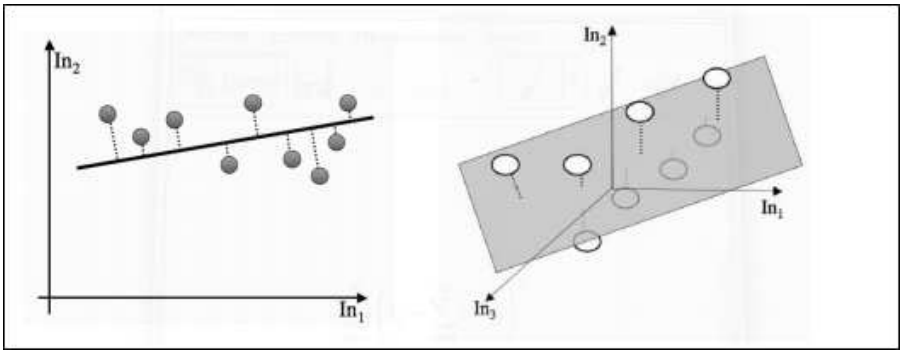

Figura 2.3: Regresiones lineales en dos y tres dimensiones..

lúan técnicas de regresión para su predicción. Las técnicas utilizadas se describen a continuación.

2.4.2.1 Regresión Lineal

La regresión es la predicción de un valor desconocido a través del cálculo de una función matemática a partir de los valores conocidos. Si se considera esta función como una función lineal, la salida será la suma de cada valor conocido multiplicado por una constante, lo cual define una línea recta (plano en 3D o hiperplano en dimensiones mayores) que circundan los puntos, como puede observarse en la Figura 2.3.

Para encontrar la recta (función lineal) que mejor se *ajusta* a los datos, se intenta minimizar la distancia entre cada punto y dicha recta. Esta distancia se mide a través de una línea auxiliar que atraviesa el punto y tope con la función. Luego, se intentará minimizar la función de error que que se calcula como la suma de las distancias. Si se minimiza la suma de los cuadrados de las distancias, se obtiene la minimización más común llamada optimización de mínimos cuadrados.

La minimización de este error, para ajustar la función lineal, puede realizarse con distintas técnicas de regresión lineal, como *ridge-regression* y *gradiente estocástico descendiente*. La primera aplica una penalización (*ridge*) a cada constante. La segunda, aplica un diferencial sobre la función obteniendo el gradiente el cual por definición, es la dirección en la que incrementa

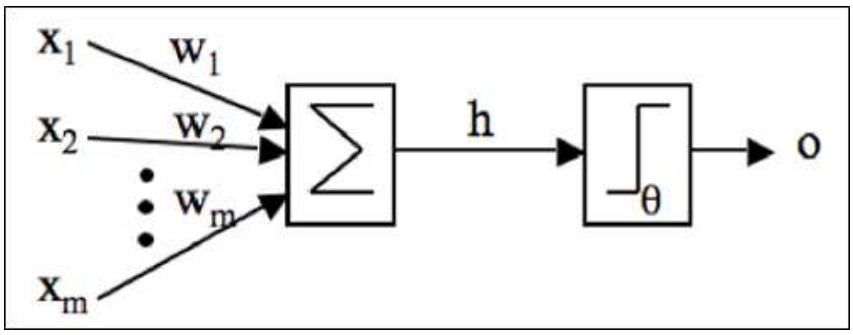

Figura 2.4: Modelo matemático neuronal .

o disminuye en mayor medida. Ya que el propósito del aprendizaje es minimizar el error de predicción, se debe seguir la función en dirección del gradiente negativo en la cual la función disminuye.

2.4.2.2 Red neuronal

La técnica de red neuronal presenta un modelo matemático sobre el comportamiento de una neurona.El mismo representa una célula nerviosa como *i*) un conjunto de entradas valoradas (w) que corresponde a las sinapsis, *ii*) un sumador que une las señales entrantes y *iii*) una función de activación (inicialmente una función umbral) que decide sobre la activación de la célula en base a las entradas actuales.

Como puede observarse en la figura 2.4, el modelo neuronales un dispositivo límite binario, las entradas son multiplicadas por los pesos y sumando sus valores; si la suma es mayor a un determinado umbral (produce salida 1) la célula se activa, de lo contrario (produce salida 0) se mantiene desactivada.

El perceptron es una colección de neuronas con un conjunto de entradas y pesos que unen las neuronas con dichas entradas. Las neuronas del Perceptron son completamente independientes entre sí, el estado de una neurona no infiere sobre las demás compartiendo sólo las entradas.

La esencia del aprendizaje de la red neuronal perceptron está centrada en los valores de pesos. La red debe ser entrenada para que los pesos se

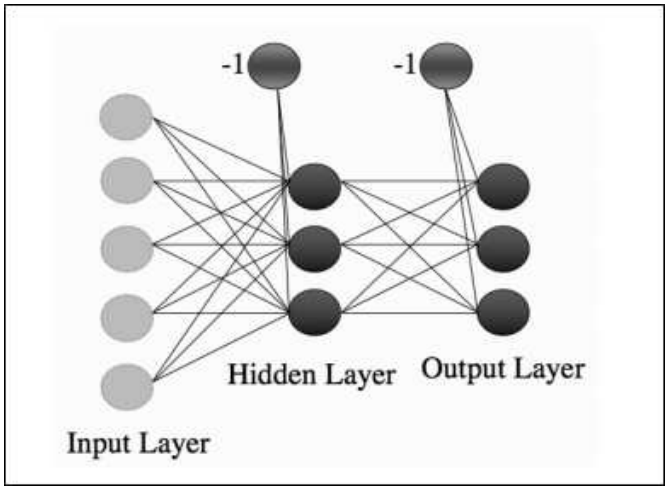

Figura 2.5: Red perceptrón multicapa

adapten y generen las respuestas correctas (*targets*).

El entrenamiento de MultiLayer Perceptron (MLP) consiste en dos partes, primero se obtienen las salidas con las entradas brindadas y los pesos actuales (*forwards*), y luego se actualizan los pesos considerando el error como la diferencia entre el valor obtenido y el real (propagación hacia atrás del error - *backwards*).

2.4.2.3 K-means clusterer

El algoritmo K - means aplica clustering sobre los datos de entrenamiento y recibe un parámetro *K* para dividir estos datos en *K* categorías. El algoritmo intenta localizar k centros en el espacio de entrada de modo tal que estos centros estén, como su nombre lo indica, en el centro de una categoría (*cluster*). La dificultad se presenta ya que al desconocer la categorización de estos grupos, resulta aún más difícil determinar la localización de cada centro.

El objetivo de determinar estos centros, en principio, a causa de incertidumbre total se posicionan los centros de forma aleatoria en el espacio de entrada. Una vez distinguidos los clusters, se determinan los puntos que pertenecen al mismo a través del cálculo de la distancia entre el punto y

todos los centros localizados, asignándose entonces, al cluster cuyo centro sea el más cercano. Finalmente, para cada centro se actualiza su ubicación utilizando la media antes definida. Estos pasos se realizan de forma incremental hasta que los centros dejan de modificar su ubicación.

Ya que este algoritmo sin duda es un método de clasificación y no de regresión, se considera importante resaltar la adaptación del mismo para utilizarlo con este fin. Así, una vez realizada la clasificación del punto que se quiere predecir, se realizará la predicción calculando el promedio de los valores de la propiedad a predecir de los otros puntos que están en el cluster.

2.4.2.4 Maquina de vector de soporte

La técnica de máquina de vector soporte Support Vector Machine (SVM). es un método propiamente relacionado con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (dataset) se pueden etiquetar las clases y entrenar un SVM para construir un modelo que prediga la propiedad de un nuevo dataset. Intuitivamente, un SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases en dos espacios lo más amplios posibles mediante un hiperplano de separación. Analíticamente, se toma la distancia existente entre la línea y el primer punto interceptado (en dirección perpendicular), si se ubica una 'zona desierta' alrededor de la línea, ningún punto ubicado en dicha zona puede ser clasificado ya que se encuentra demasiado cerca de la línea. El radio máximo que puede tener esta región es llamado margen, señalado como M y los puntos de cada clase más cercanos a la línea de clasificación se denominan vectores de soporte

2.4.3 Evaluación de modelos

Una vez entrenado un modelo de predicción con una técnica, la evaluación del modelo es importante para medir el nivel de acierto de las predicciones. Esta evaluación consiste en probar el modelo con un conjunto de datos de prueba y medir el error u otras métricas sobre los resultados. Estas métricas

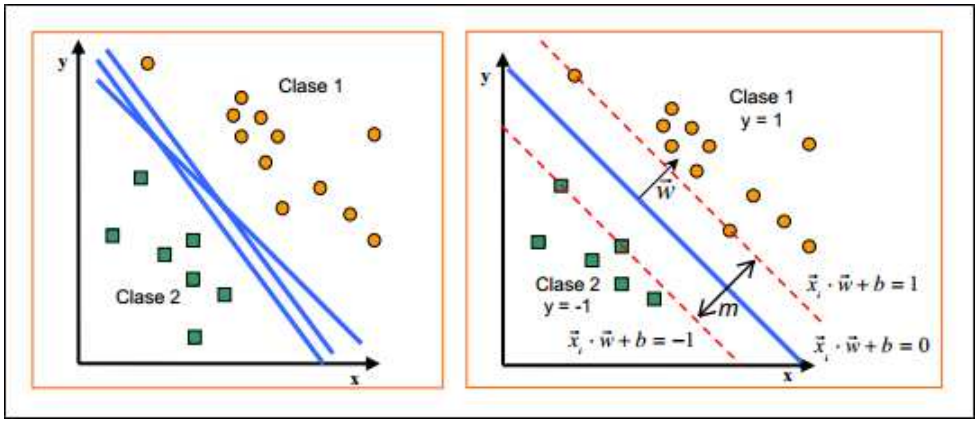

Figura 2.6: Metodología de operación del algoritmo SVM

de evaluación permite comparar el desempeño de modelos entrenados con diferentes técnicas.

Existen distintas formas para llevar a cabo esta evaluación. La mas simple consiste en usar como datos de prueba el mismo conjunto de datos utilizado para el entrenamiento de los modelos. Otro método consiste en separar los datos del problema entre datos de entrenamiento y datos de prueba. Por ultimo, también se puede validar el modelo de forma cruzada.

La validación cruzada (cross-validation) es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Por ejemplo, si consideramos diez subconjuntos para validación, los datos de entrada se dividen en diez partes, donde una se reserva para las pruebas y las otras nueve para el entrenamiento. Este proceso se repite diez veces y se calcula el promedio de las métricas de evaluación. Esto ayuda a determinar el nivel al que un modelo se podría generalizar para nuevos conjuntos de datos.

El presente trabajo contempla las siguientes métricas de evaluación para los modelos de regresión:

- CC (Coeficiente de correlación de Pearson): el coeficiente de correla-

ción de Pearson es un índice que puede utilizarse para medir el grado de relación de dos variables siempre y cuando ambas sean cuantitativas. En el presente trabajo se considera la correlación entre las variables del dataset con respecto a la propiedad a predecir.

- RMSE (Root Mean Absolute Error): el Root Mean Absolute Error (RMSE) representa la raiz cuadrática del promedio de la distancia euclídea entre el valor de la propiedad obtenida por la técnica y el valor real.

2.4.4 Ajuste del modelo: Overfitting y Underfitting

Cuando se genera (o entrena) un modelo de predicción, su desempeño es incierto hasta su evaluación o aplicación. En algunas ocasiones, la calidad del modelo es pobre generando respuestas imprecisas, de modo tal que se le deben aplicar acciones correctivas comprendiendo cómo se comporta y ajusta el modelo.

Los modelos pueden presentar dos problemas indeseables: *overfitting* y *underfitting*. El *overfitting* describe una función que se ajusta estrechamente a los datos de entrenamiento. El modelo aprendió los detalles y el ruido en los datos impactando negativamente en el desempeño del modelo. Este efecto es causado porque el ruido o las fluctuaciones aleatorias en los datos de entrada fueron usados para el aprendizaje.

Por otro lado, los modelos pueden presentar problemas de *underfitting*, cuando no interpretan bien los datos de entrenamiento, por lo que son incapaces de generalizar correctamente nuevos datos. Este efecto es causado porque la función o técnica elegida no es el indicada para representar el comportamiento de los datos. El efecto underfitting se caracteriza por sobre generalizar los datos. La incorporación de nuevos datos al conjunto de entrenamiento podría solucionar o apaciguar este efecto.

El modelo deseado, sin dudas, sería aquel que se encuentre en un punto de equilibrio entre un problema y otro, aunque este equilibrio es muy difícil de alcanzar en la práctica. La Figura 2.7 presenta tres modelos de regresión para un mismo grupo de datos que permiten interpretar gráficamente los problemas de underfitting y overfitting.

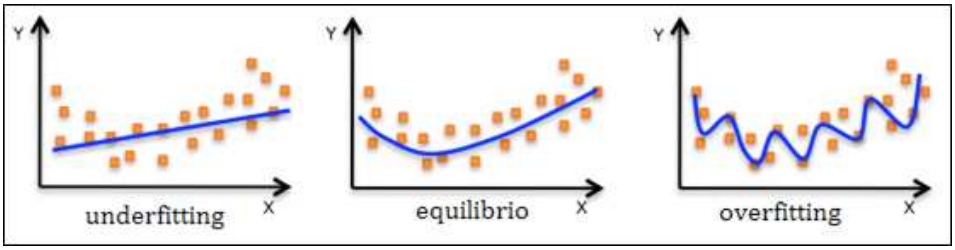

Figura 2.7: Contraste entre distintos efectos del modelo sobre los datos de entrenamiento.

Capítulo 3

Trabajos Relacionados

El enfoque propuesto es un proceso de aprendizaje de máquina que incluye una herramienta para recolectar mediciones de performance en Android, y otra herramienta para entrenar y evaluar modelos de predicción con diferentes técnicas de aprendizaje automático. En este capítulo se presentan trabajo relacionados, divididos en dos secciones. Primero, en la sección 3.1, se describe un conjunto de herramientas para monitorear y realizar pruebas de performance en Android. Luego, en la sección 3.2, se presentan algunos trabajos relacionados que también abordan el problema de predicción de atributos no-funcionales con técnicas de aprendizaje de máquina.

3.1 Herramientas para Android

Las herramientas de monitoreo y pruebas de performance para Android ofrecen a los desarrolladores la posibilidad de analizar los aspectos no-funcionales de una aplicación Android arrojando datos numéricos acerca de su desempeño y consumo de recursos. A continuación se describen algunas de estas herramientas.

Performance Monitors de Android

Es una herramienta integrada en el ambiente de desarrollo *Android Studio*[1] y cuenta con varias sub herramientas que monitorean y proveen información en tiempo real sobre la aplicación. Los datos capturados se almacenan en archivos para luego analizarlos en diferentes vistas. Pueden monitorearse tanto aplicaciones en dispositivos reales conectados o simulados a través de un emulador.

A través de cinco vistas diferentes se accede a la información sobre la aplicación evaluada. *LogCat* monitorea las excepciones y mensajes de log emitidos por la aplicación y el sistema operativo, útil para deputar la aplicación durante su desarrollo. Las restantes vistas proveen un monitoreo del consumo de memoria, CPU, GPU y red por parte de la aplicación.

Benchit

Benchit[2] es una biblioteca Open Source implementada en lenguaje Java para realizar mediciones de performance en Android. Benchit es rápido y sencillo de usar, ya que permite analizar áreas de código para determinar el tiempo de respuesta o latencia de la operación. Una forma sencilla de utilizar esta librería es a través de iteraciones sobre una misma sección de código. De esta forma, con cada iteración, la herramienta va almacenando el tiempo de ejecución del código (diferencia entre el tiempo de comienzo y tiempo de fin). Al término de las iteraciones, se podrá extraer el promedio, rango y desviación estándar de las mediciones. La herramienta, también provee la posibilidad de realizar comparaciones de código mostrando los resultados de manera ordenada. Esta opción es útil, por ejemplo, al momento de comparar el desempeño de distintos algoritmos o porciones de código que realicen la misma acción de forma diferente.

[1] *https://developer.android.com/studio/profile/android-monitor.html*
[2] *https://github.com/T-Spoon/Benchit*

Google Caliper

Google Caliper es un framework open source para implementar, ejecutar y visualizar resultados de microbenchmarks en aplicaciones Java, aunque brinda soporte para proyectos Android. Los microbenchmarks son mediciones realizadas a funciones simples como llamadas al kernel. Caliper permite obtener diferentes medidas del código Java, principalmente microbenchmarks, pero también tiene soporte para otros tipos de medidas incluyendo memoria disponible, u otras medidas arbitrarias de dominio específico como por ejemplo el radio de compresión.

JMeter

Apache JMeter es una herramienta Open Source implementada en lenguaje Java para realizar pruebas de carga y rendimiento de una variedad de servicios, con énfasis en aplicaciones y protocolos Web. Apache JMeter se puede utilizar para simular un gran volumen de carga en un servidor o grupo de servidores y probar su resistencia o analizar el rendimiento general bajo diferentes tipos de carga. En un principio, fue diseñada para realizar pruebas de rendimiento sobre aplicaciones Web pero luego ha sido extendida a otras funciones para cubrir diferentes categorías de testing, tal es el caso de los análisis de carga, de funcionalidad, desempeño, regresión, entre otros. JMeter es una aplicación de escritorio con una interfaz gráfica amigable para el usuario. Puede ser ejecutada bajo cualquier entorno o estación que acepte la máquina virtual de Java como es el caso de los sistemas operativos Windows, Linux y Mac.

A grandes rasgos, la herramienta simula un grupo de usuarios que envían peticiones a un servidor y retorna un conjunto de estadísticas sobre el desempeño y funcionalidad por medio de gráficos, tablas, etc, tanto sobre recursos estáticos y dinámicos.

El cuadro 3.1 presenta y compara las principales características de las herramientas antes mencionadas.

Herramienta	Tipo de herramienta	Máquina virtual	Propiedades medidas	Componentes medidos
Performance Monitor de Android	Herramienta integrada en el ambiente 'Android Studio'	Máquina virtual Dalvik y ART del dispositivo	Logcat	Código de Aplicación
			Memoria disponible y ocupada, incluyendo eventos de Garbage Collection	Memoria
			Tiempo de CPU incluyendo todos los núcleos	CPU
			Performance de la interfaz gráfica	GPU
			Transferencia de datos	Network
Benchit	Librería Open Source Java para Android	Máquina Virtual de Java	Latencia de operación	Áreas de código Java
			Promedio	
			Rango	
			Desviacion standard	
Google Caliper	Framework Open Source para Java	Máquina Virtual de Java	Microbenckmarks	Código Java
			Memoria disponible y ocupada	Memoria
			Medidas arbitrarias de dominio especifico	
Jmeter	Aplicación de escritorio	-	Pruebas de cargas y medidas de rendimiento	Servicios y lenguajes Web
				Protocolos de mensajería y conexiones en general
				Comandos y scripts de shell
				Base de datos

Cuadro 3.1: Información resumida de herramientas de pruebas de performance para aplicaciones Android y servicios Web.

3.2 Predicción de propiedades no-funcionales con aprendizaje de máquina

La predicción de propiedades no-funcionales ayuda a los arquitectos de software en la evaluación de sus sistemas durante la etapa de diseño, y a guiar las decisiones respecto a que componentes integrar a la arquitectura del sistema de acuerdo a sus requerimientos de calidad. Diferentes trabajos han recurrido al uso de técnicas de aprendizaje de máquina para construir modelos de predicción de performance y otros atributos de calidad dinámicos. En Frank Hutter and Leyton-Brown [4]se refieren a estos modelos como modelos empíricos de performance (*EPM* por sus siglas en inglés) ya que requieren la recolección de mediciones empíricas sobre los componentes.

La principal aplicación de estos modelos probablemente es el problema de selección de algoritmos, introducido en 1976 por John R. Rice [12]. Este

problema consiste en seleccionar el algoritmo, o configuración de algoritmo, de un portafolio de alternativas que minimice el tiempo de respuesta, según la instancia de datos de entrada. La predicción del tiempo de respuesta ha sido abordada con éxito usando técnicas de aprendizaje supervisado, principalmente de regresión [4]. En estos trabajos, los datos empíricos de entrenamiento son obtenidos en contextos de ejecución controlados, para enfocarse en la correlación entre la propiedad a predecir y las propiedades de los parámetros de entrada del algoritmo. Una limitación de estos modelos es que no generalizan la predicción de las propiedades de performance al contexto de ejecución, es decir, no consideran características del dispositivo y el ambiente de ejecución que influyen sobre el desempeño del componente, como su capacidad de cómputo y la disponibilidad de recursos.

Los modelos de predicción no sólo se utilizan para estimar el tiempo de respuesta del desempeño de los componentes, sino también para estimar otras propiedades. Por ejemplo, en Roberts and Howe [13] se proponen modelos para la predicción del tiempo de respuesta y para la probabilidad de éxito de diferentes algoritmos de planeamiento utilizando técnicas de aprendizaje de máquina incluidas en la librería Weka. Para el aprendizaje fueron utilizados los resultados de 4726 instancias de problemas de planning ejecutadas sobre 28 algoritmos de planeamiento conocidos. Por cada algoritmo y cada problema se registra si un plan fue encontrado exitosamente o no, y el tiempo (en segundos) requerido en completar la ejecución. A partir de esta información se entrenan y evalúan modelos con diferentes técnicas de aprendizaje supervisado. El trabajo presentado en J. R. Beveridge and Draper [6] también analiza la precisión de distintos algoritmos, en este caso, para el reconocimiento de rostros en imágenes, utilizando una técnica de regresión lineal denominada GLMM, por Generalized Linear Mixed Models.

Otros autores en Mersmann et al. [10] predicen la optimalidad o razón de aproximación de algoritmos para el problema del viajante utilizando una técnica de regresión no lineal llamada *MARS*, por sus siglas en inglés. El modelo *MARS* se aplicó con éxito para predecir la optimalidad del al-

goritmo de búsqueda local llamado 2-opt, independientemente del tamaño de la instancia del problema. Se argumenta que es sencillo aplicar la misma metodología a otros algoritmos y utilizar estos modelos para derivar una estrategia para el problema de selección de algoritmos en el contexto del problema del viajante.

La predicción de propiedades dinámicas de servicios Web, como su tiempo de respuesta, y probabilidad de fallos, es más compleja de abordar con respecto a algoritmos y componentes ejecutados localmente ya que depende de factores propios de la infraestructura de red y el proveedor del servicio, que no se puede monitorear desde el dispositivo cliente. Un enfoque ingenioso para abordar este problema fue propuesto en el articulo de Zheng[16]. En este trabajo, los autores se basan en la premisa de que las propiedades de performance de los servicios Web varían respecto a características del contexto, como la ubicación geográfica y el momento del día y la semana en el que se realiza una solicitud al servicio. De esta forma, recolectan la información del consumo de servicios de múltiples clientes alrededor del mundo para generalizar modelos de predicción. Este enfoque es conocido como predicción colaborativa ya que los datos empíricos de entrenamiento son brindados por múltiples nodos de manera distribuida. Los autores llevaron a cabo un experimento a gran escala que involucró más de 30 millones de invocaciones a servicios Web en más de 80 países, por usuarios distribuidos en más de 30 países. La observación experimental indica que diferentes usuarios pueden tener diferentes experiencias de uso sobre el mismo servicio, influenciados por la conexión de red y los ambientes heterogéneos entre usuarios y proveedores. Los datos se encuentran disponibles públicamente y han sido utilizados en varios trabajos para la construcción y comparación de modelos de performance con diferentes técnicas de aprendizaje de máquina [2][1].

Capítulo 4

Enfoque y Herramientas

En este capítulo se describe el enfoque propuesto para predecir atributos dinámicos de componentes de software, en base a propiedades medidas durante su ejecución. El enfoque plantea llevar a cabo la predicción mediante un proceso de aprendizaje de máquina que involucra dos pasos: medición y aprendizaje. El primero implica la recolección de indicadores o mediciones durante la ejecución del componente, lo que puede significar la ejecución de una pieza de código, la llamada a una función, o la invocación de un servicio. Lo segundo implica la construcción de modelos de predicción con técnicas de regresión. La construcción o entrenamiento de estos modelos se hace en un proceso interactivo con el usuario, que involucra una configuración inicial de los datos y la optimización automática de los parámetros de acuerdo a las tasas de error arrojadas por métricas de evaluación . Por último, los modelos generados pueden utilizarse en distintos casos prácticos, como por ejemplo, la selección automática de servicios y algoritmos, o soportar decisiones de diseño o implementación.

Como soporte para el enfoque, se desarrollaron dos herramientas independientes diseñadas para efectuar cada uno de los pasos mencionados. Por un lado, el framework *Android Meter* es una biblioteca Java para realizar pruebas de desempeños de código en Android y capturar mediciones durante la ejecución. Su diseño es genérico para usar sobre cualquier do-

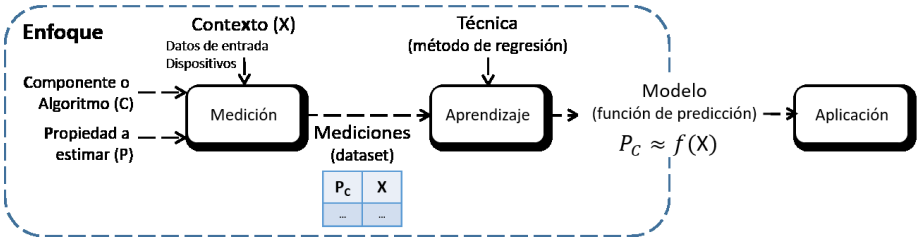

Figura 4.1: Esquema conceptual del enfoque en fases.

minio del que se quieran obtener mediciones de desempeño. Por otro lado, se desarrolló una herramienta standalone denominada *Nekonata* diseñada para integrar fácilmente técnicas de aprendizaje automático incluidas en librerías Java de terceros. La herramienta se usa para la construcción del modelo a partir del conjunto de datos capturados con el uso del framework Android Meter, mediante un proceso de automatización de los algoritmos en complemento de información gráfica para la colaboración interactiva del usuario.

Estas cuestiones se describen en detalle de la siguiente manera. En la sección 4.1 se profundiza sobre las distintas etapas del enfoque, incluyendo algunos usos prácticos de la propuesta. En las siguientes secciones se describen cada uno de las herramientas desarrolladas, presentando el framework para la medición y recolección de datos en la sección 4.2 y finalmente, en la sección 4.3 se presenta la herramienta para la construcción de modelos de predicción.

4.1 Etapas del método

El enfoque propuesto se conceptualiza como un proceso de dos fases complementarias, con las que se obtienen modelos de predicción de propiedades de componentes que pueden usarse en distintas aplicaciones o casos prácticos. La figura 4.1 muestra un esquema conceptual del enfoque, cuyas etapas se describen a continuación:

Medición y recolección de datos

El proceso comienza con la medición de un componente en ejecución y la recolección de estas mediciones en un dataset. Cada dataset pertenece a la ejecución de un componente diferente, del cual se extraen todas sus mediciones de desempeño (tiempo de respuesta, etc) mas las propiedades que podrían influir sobre este desempeño. Durante la ejecución de cada componente se van realizando las mediciones sobre distintos aspectos de la operación y registrando cada uno de estos resultados en un archivo para su posterior análisis. A su vez, el proceso de recolección de datos involucra la ejecución de sub etapas, pudiendo requerir el proceso algunas o todas las tareas que se detallan a continuación:

- Integración de componentes: A menudo se requiere la evaluación de componentes de terceros, tanto de servicios Web como algoritmos de bibliotecas instaladas, de manera que deben ser integrados correctamente a la herramienta para ser tratados como componentes.

- Selección de características del contexto: mientras se realiza la ejecución de los componentes, las características del dispositivo móvil donde se llevan a cabo las operaciones pueden ser capturadas como mediciones. Realizar las mediciones sobre distintos dispositivos móviles persigue la idea de que las capacidades de cada dispositivo (cantidad de memoria, CPU, etc) influye en el desempeño de los componentes, como por ejemplo, su tiempo de ejecución.

- Generación de datos de entrada: los componentes requieren un conjunto de parámetros o datos de entrada para su ejecución, por eso se debe proveer una manera para generar arbitrariamente instancias de datos de entrada del componente a medir, o la capacidad para obtener instancias externamente a través de enlaces o archivos.

- Selección de propiedades de la entrada: Es posible obtener propiedades de los datos de entrada, generalmente atributos que configuran o sirven para generar una instancia como su tamaño.

- Ejecución de componentes y obtención de mediciones: los componentes son ejecutados con las entradas generadas, mientras se toman y registran mediciones sobre las propiedades del contexto de ejecución (datos de entrada y dispositivo), que son los atributos independientes o predictores, además de las propiedades de interés a predecir. De esta forma, el dataset que se obtiene puede ser formado por una cantidad variable de propiedades en base a las características que hayan sido elegidas para ser medidas.

Aprendizaje

A partir del conjunto de datos obtenidos en la etapa anterior, se aplican técnicas de aprendizaje de máquina para la extracción de conocimiento. Tal conocimiento se refleja mediante la construcción de modelos predictivos que mejor se ajustan a la generalización de la información mediante un proceso de entrenamiento, optimización y evaluación de los mismos. El proceso de aprendizaje requiere indefectiblemente de un conjunto de datos de entrenamiento volcado en un archivo, una propiedad de ese conjunto a predecir y un conjunto de algoritmos que aplican técnicas de regresión sobre los datos. El aprendizaje está conformado por varias sub etapas o sub procesos.

- Configuración de las técnicas: las técnicas de regresión representan algoritmos que pueden involucrar diferentes parámetros de configuración. La variabilidad en estos valores permite ajustar las preferencias del algoritmo y consecuentemente, la obtención de modelos de calidad diferente. Por lo tanto, el desafío principal de esta parte de la etapa es encontrar los valores apropiados para cada uno de los parámetros, teniendo como base, el conocimiento sobre el efecto que el parámetro causa en el algoritmo. Al obtener la mejor configuración de una técnica, para un dataset y una propiedad particular, se facilita la comparación simultánea de todas las técnicas disponibles a través de las métricas de evaluación y en consecuencia seleccionar la más óptima.

- Construcción de modelos: la construcción o entrenamiento de modelos predictivos es un proceso que implica la ejecucion de una o mas técnicas de regresión sobre el conjunto de los datos y continúa opcionalmente con un proceso de evaluación.El entrenamiento utiliza las mediciones ya obtenidas en la etapa anterior y a partir de la propiedad que se desea predecir, el algoritmo va definiendo la función o modelo de regresión que mejor ajusta las relaciones entre los atributos independientes (o predictores) del dataset y el atributo a predecir. Esta etapa concentra el mayor porcentaje del tiempo computacional y uso de memoria, y varía de acuerdo a la complejidad o simplicidad de cada técnica en particular.

- Evaluación de modelos: una vez concluida la etapa de entrenamiento, el modelo obtenido puede ser analizado para determinar la medida en que este es capaz de generalizar cualquier entrada de dataset futura y estimar la propiedad de interés. La noción sobre el desempeño del modelo tiene lugar a partir de métricas existentes para modelos de regresión, que capturan el error de predicción o el coeficiente de correlación para conocer el grado de interdependencia de las propiedades.

Aplicaciones de los modelos

Finalmente, se pretende utilizar estos modelos de predicción en entornos de aplicación que permitan la selección del componente más adecuado en base a propiedades de los parámetros de entrada u otras propiedades del contexto de ejecución, a través de un proceso automatizado que determine al usuario la opción más favorable evitando la ejecución individual de cada componente. La construcción de modelos de predicción de performance ha ido creciendo debido a la utilidad que presentan en una gran variedad de contextos prácticos. A continuación, se detallan algunos:

Selección de algoritmos Como se ha tratado en algunos trabajos Frank Hutter and Leyton-Brown [4], los modelos de predicción son útiles para

la selección automática de algoritmos . A través de estos modelos se puede predecir con cierto error el rendimiento de cada uno de estos algoritmos candidatos y mediante una simple comparación seleccionar el más apropiado considerando la instancia del problema y las características del hardware.

Ajustes de parámetros y configuración automática del algoritmo Los modelos predictivos sirve a dos propósitos fundamentales. Por un lado, modelar el comportamiento o funcionalidad de un algoritmo parametrizado en base a la configuración de tales parámetros, en cuyo caso se puede alternar entre el aprendizaje del modelo y su uso para identificar configuraciones interesantes para evaluar posteriormente. Por otro lado, se puede modelar el rendimiento del algoritmo basado conjuntamente en las características de las instancias del problema y la configuración de sus parámetros. Tales modelos pueden utilizarse para ajustar los valores de tales parámetros y obtener una mejor predicción basada en la instancia particular.

Obtener una visión general de las instancias y el rendimiento de los algoritmos Los modelos se pueden utilizar para evaluar las características de la instancia y los valores de los parámetros del algoritmo que más impactan en el rendimiento. Algunos modelos son compatibles con este tipo de evaluaciones directamente. Para otros modelos, existen métodos de selección de atributos (características genéricas) para identificar un grupo más reducido de entradas del modelo que son claves, y describen el rendimiento del algoritmo casi tan bien como todo el conjunto de entradas.

Selección de servicio para composición de servicios Cuando varios servicios Web implementan la misma funcionalidad, los modelos de rendimiento resultan ser un buen criterio para escoger el mejor candidato entre ellos. Incluso en tiempo de ejecución, los proveedores de servicio pueden cambiarse si las condiciones del contexto y los parámetros

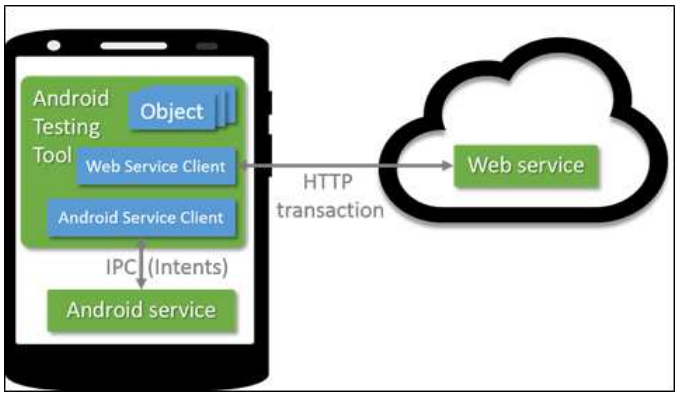

Figura 4.2: Esquema conceptual de componentes Android considerados.

de entrada se modifican.

Planificación de tareas en redes móviles Suponiendo un conjunto de tareas que deben asignarse entre un conjunto de dispositivos, los modelos de rendimiento podrían obtener una medida aproximada del tiempo de respuesta que cada tarea requerirá sobre cada dispositivo con el fin de minimizar el tiempo total de secuenciación de las tareas.

4.2 Framework de medición para Android

Android Meter es un framework diseñado para simplificar la medición de propiedades de performance de componentes ejecutados bajo el sistema Android. El framework esta implementado como una biblioteca Java que permite ejecutar diferentes piezas de software y capturar propiedades y mediciones durante su ejecución. La biblioteca ofrece el soporte necesario para adaptarla a distintos dominios de componentes y algoritmos, y recolectar las mediciones que sirven de fuente para construir modelos de predicción a través de técnicas de aprendizaje de máquina.

El framework se basa en el concepto central de *Componente*, que representa cualquier entidad de ejecución, como una funcion, algoritmo o servicio, que provee una funcionalidad a través de una interfaz específica.

En la figura 4.2 se ilustran distintos tipos de componentes que se pue-

den ejecutar y medir desde el framework, incluyendo servicios Web, servicios específicos de la plataforma Android, y cualquier algoritmo o pieza de código que se implemente como metodos de objetos Java. Las instancias de objetos hacen referencia a cualquier componente residente en el espacio de memoria de una aplicación, el componente específico para servicios web incluye cualquier componente remoto fuera del dispositivo y accedidos a través de protocolos de comunicación Web, como Hypertext Transfer Protocol (HTTP). Por último el componente específico para servicios Android incluye cualquier proceso ejecutado en segundo plano residente en el mismo dispositivo y accedidos a través de objetos Intent (mecanismo de comunicación entre procesos del sistema Android).

A continuación se expone en la figura 4.3 los principales conceptos que forman parte al framework *Android Meter*.

La obtención de diferentes conjuntos de mediciones se realiza a través de la creación de planes de pruebas, que se configuran y posteriormente se ejecutan para obtener los datasets. Al momento de configurar los planes de pruebas se tiene la posibilidad de especificar qué componentes se ejecutarán, con qué datos de entrada, y qué propiedades o métricas se desean capturar, ya sean propiedades del contexto, de los datos, o propiedades no funcionales de la ejecución de los componentes. Las propiedades a obtener pueden ser:

1. Propiedades del contexto: son las cualidades del entorno de ejecución, es decir, el dispositivo sobre el que se ejecuta la prueba de performance. Estas propiedades pueden ser estáticas, es decir, que se mantienen sin cambio durante la ejecución del plan de pruebas, como por ejemplo, el modelo del dispositivo, la arquitectura y cantidad de núcleos del CPU, tamaño de memoria, etc; y dinámicas, que son las propiedades del entorno que pueden variar durante la ejecución de los componentes, como el uso de CPU, el número de procesos en ejecución, tipo de conexión, etc.

2. Propiedades de los datos de entrada: son atributos que se pueden extraer de los parámetros con los que se invocan los componentes. Por

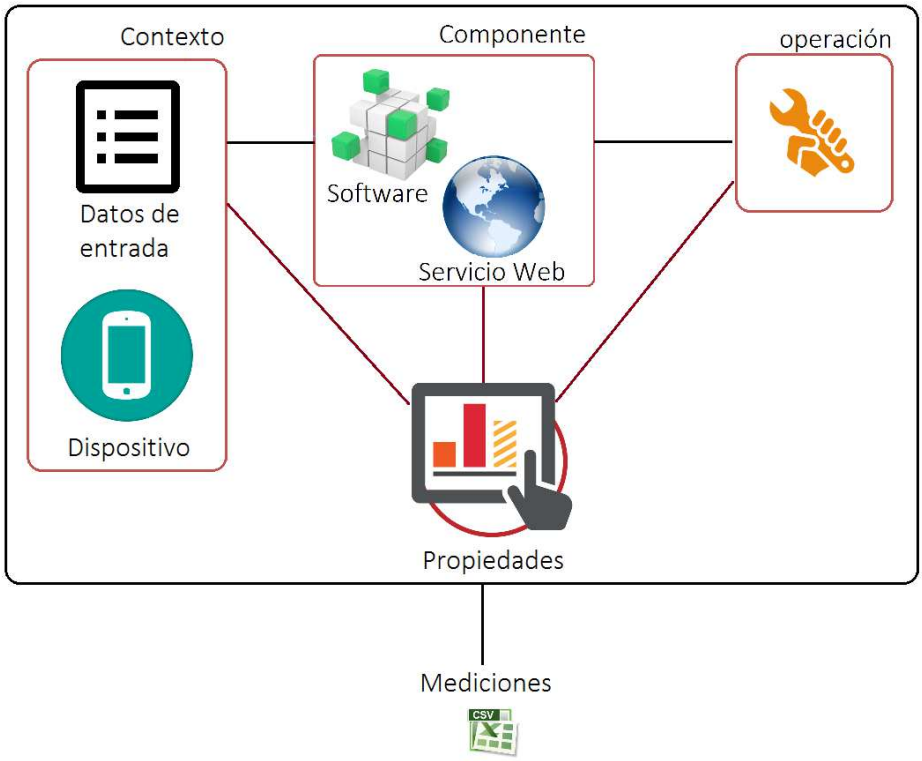

Figura 4.3: Conceptos principales del framework Android Meter.

ejemplo, dado un algoritmo de ordenamiento de vectores, el tamaño del vector de entrada seria una de estos atributos.

3. Propiedades del componente: atributos del componente como el nombre, ubicación, etc.

4. Propiedades no funcionales: son las propiedades de interés que se desean inferir a partir de las propiedades anteriores. Estas propiedades se capturan en cada operacion de componentes, es decir, la ejecucion de un componente con determinados datos de entrada. Por ejemplo, el tiempo de respuesta, el consumo de batería, si la operación fue ejecutada con éxito o con error, etc.

4.3 Herramienta de aprendizaje de modelos

Nekonata es una herramienta para la construcción de modelos predictivos a través de técnicas de aprendizaje de máquina. El eje que ha guiado el diseño de la herramienta ha sido el brindar soporte para el uso de cualquier librería que ofrezca aprendizaje de máquina a través de la implementación de todos los conceptos involucrados como objetos independientes a los cuales adaptar las funcionalidades de las librerías. La herramienta *Nekonata* lleva a cabo dos tipos de procesos en el desarrollo de sus funciones: procesos automáticos y procesos semi-automáticos que requieren la colaboración del usuario con el fin de mejorar los resultados a partir de vistas gráficas que visualizan el comportamiento y características de los modelos y además, para incluir en el proceso el interés y criterio del usuario.

El flujo de trabajo de la herramienta consta de dos etapas, una de configuración y entrenamiento de modelos y otra de evaluación de los mismos.

En la etapa de configuración, se indican la biblioteca y el dataset que se desea considerar, habilitando la elección de las técnicas de aprendizaje y de la propiedad dinámica a predecir. Una vez seleccionados todos los valores requeridos, se procede a ejecutar las técnicas de aprendizaje para generar los modelos de predicción. Un detalle importante a considerar es que la herramienta no sólo genera los modelos a partir de parámetros fijos,

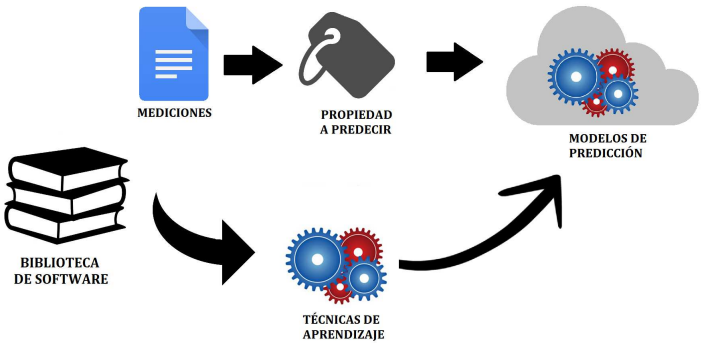

Figura 4.4: Diagrama de flujo del proceso de configuración y entrenamiento de modelos.

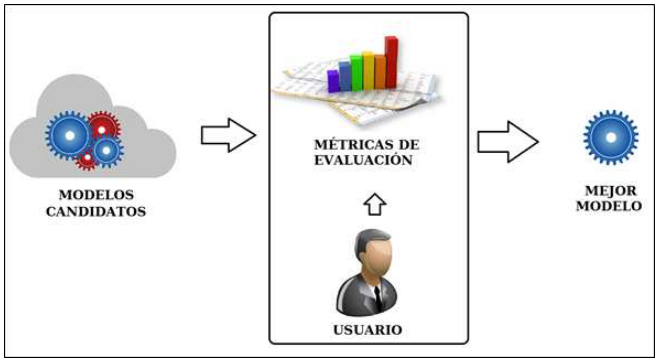

Figura 4.5: Diagrama de flujo de la fase de comparación de modelos.

sino que realiza una comparación interna entre modelos parciales para la obtención de una función de predicción más precisa y acertada.

Luego de la etapa de entrenamiento, los modelos generados por cada técnica son expuestos a una serie de métricas dispuestas de manera tal que permita visualizar las diferencias de desempeño entre técnicas con la misma métrica de evaluación considerada. Esta vista es un recurso ofrecido al usuario para decidir aquel modelo que a su criterio tenga mejor calidad a través de la comparación simultánea. Estas métricas de evaluación incluyen: el coeficiente de correlación de Pearson (CC) y la raíz del error cuadrático medio (RMSE).

Es conveniente trasladar el análisis a la mayor cantidad de métricas po-

sibles, ya que un mismo indicador podría arrojar valores cercanos entre un algoritmo y otro favoreciendo equivocadamente a uno de ellos, error que podría notarse al compararlos simultáneamente con otras métricas. La metodología de operación de esta fase se muestra en la figura 4.5.

Nekonata hace foco en este detalle y ofrece al usuario una vista de imágenes (4.6)con los indicadores medidos de cada algoritmo seleccionado por el usuario para aplicar el proceso de optimización y elegir, posteriormente, el que resulte más adecuado. La información gráfica complementaria que se brinda, detalla los algoritmos representados por medio de barras y agrupados en categorías separadas de acuerdo a cada métrica a modo de facilitar la comparación. Adicionalmente las barras se colorean en dos tonos diferentes para acentuar, por cada métrica, los clasificadores que significarían las mejores opciones para ese indicador.

La herramienta se desarrolló como una aplicación de escritorio pensada para desarrolladores e ingenieros de datos que necesitan un entorno usable y eficiente para construir modelos de predicción y analizar los datos . Esta implementación desvincula el framework de medición con la herramienta de aprendizaje, permitiendo la ejecución independiente entre ambas, y el puente de comunicación entre ellas es la correspondencia entre la salida de la primer herramienta con la entrada de la segunda.

El atributo principal que dirigió el diseño de la herramienta fue la extensibilidad de la misma. Esto significa que la herramienta puede aceptar diferentes tipos de bibliotecas de aprendizaje de máquina, datasets, modelos de regresión, parámetros y métricas. Esto se logró con una implementación orientado a objetos y basada en patrones de diseño.

Actualmente, la aplicación ofrece una implementación de las siguientes técnicas de regresión considerando la biblioteca de software Weka:

- Linear Regression: Es la implementación de la técnica Lineal Regression, considerando la variante de Ridge Regression.
- Neural Network Regression: Es la implementación de la técnica de Neural Network, considerando la variante de Multilayer Perceptron.
- Stochastic Gradient Descendent: Es la implementación de la técnica

Figura 4.6: Captura de pantalla de la herramienta

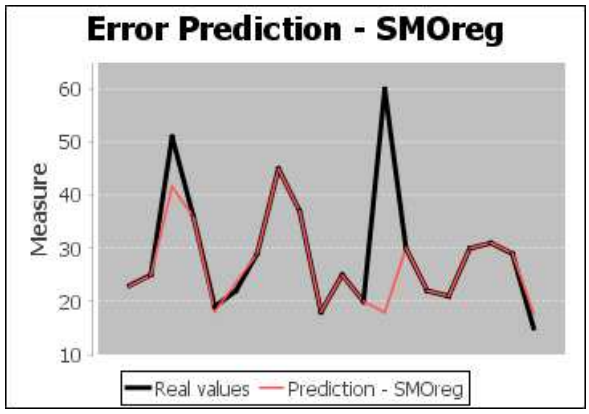

Figura 4.7: Captura de pantalla de la vista del error de predicción.

de Lineal Regression, considerando la variante de gradiente descendiente o SGD.

- Support Vector Machine Regression: Es la implementación de la técnica Vector Machine, considerando la variante de Sequential Minimal Optimization.

- Simple K Clusterer: Es una adaptación de la técnica de clustering K-Means Clusterer para regresión.

A través del uso de métricas se adquiere una idea estimativa del desempeño del clasificador frente al conjunto de datos de entrenamiento, sin embargo, podría resultar útil conocer el comportamiento general del algoritmo, contrastando cada uno de los valores reales del atributo clase con los valores predichos por el clasificador, y obtener así una vista exacta de la manera en que el clasificador se ajusta a los datos (Figura 4.7).

Este recurso es usado en la herramienta como un gráfico de dos líneas continuas de distinto color para representar el conjunto de datos de entrenamiento y el conjunto de datos predichos. Cada punto del dominio corresponde a cada instancia y la unión entre puntos sólo se realiza con fines ilustrativos.

Capítulo 5

Evaluación

En este capitulo se presenta la evaluación del enfoque propuesto con el objetivo de analizar el desempeño de las técnicas de aprendizaje de máquina utilizadas para predecir propiedades no-funcionales en dispositivos móviles. Con este fin, se crearon tres casos de estudio sobre los que se construyen y evalúan modelos de predicción.

El capítulo se organiza de la siguiente manera: en la sección 5.1 se detalla el proceso de evaluación propuesto. En las secciones 5.2, 5.3 y 5.4 se presentan los escenarios de estudio considerados, brindando una descripción de los componentes involucrados, las mediciones obtenidas y los modelos de predicción construidos y evaluados. Los resultados y discusión son expuestos en la sección 5.5 para analizar el desempeño de las técnicas de regresión sobre cada caso de estudio. Finalmente en la sección 5.6 se presentan las conclusiones del capítulo.

5.1 Proceso de Evaluación

El proceso de evaluación, ilustrado en la Figura 5.1, consiste en aplicar el enfoque sobre diferentes propiedades y grupos de componentes, y evaluar los modelos de predicción con diferentes métricas para analizar el desempeño de las distintas técnicas de aprendizaje incluidas en la herramienta.

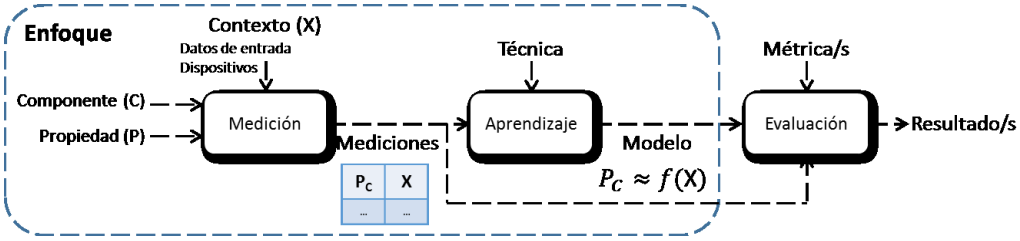

Figura 5.1: Proceso de evaluación

La calidad de los modelos será evaluada a través de dos métricas que computan de forma distinta el error de predicción. Cada modelo f es construido por una técnica T (regresión lineal, red neuronal, etc.) que se aplica sobre un dataset de mediciones para predecir una propiedad P (tiempo de respuesta, precisión) de un componente C. El proceso de evaluación consiste en tomar el conjunto de mediciones usado para el aprendizaje de modelos de predicción y utilizarlos para validar estos modelos a través las métricas de evaluación.

Para un análisis más profundo sobre las distintas técnicas y su desempeño, se usaron tres escenarios o casos de estudio de distintos dominios: algoritmos para multiplicación de matrices, algoritmos para el problema del viajante, y componentes para detección de rostros en imágenes. Cada escenario comprende un conjunto de componentes y datos de entrada a partir de lo cual se obtiene un dataset de mediciones. En todos los escenarios se considera el tiempo de respuesta como propiedad a medir y predecir. En el segundo y tercer caso de estudio, se considera adicionalmente como propiedad la optimalidad o precisión en el resultado arrojado por el componente.

El cuadro 5.1 presenta la lista de dispositivos sobre los que se han realizado las pruebas, con sus características mas relevantes.

Dispositivo	Frecuencia de CPU	Nucleos de CPU	Memoria RAM	Sistema Operativo
Lenovo K3 Note	1.7Ghz	Octa-Core	2GB	Android 5.0
Samsung Galaxy S3	1.4 Ghz,	Quad-Core	1GB	Android 4.3
Samsung Galaxy S4	1.9Ghz	Quad-Core	2GB	Android 4.2.2
Samsung Galaxy S5	2.5Ghz	Quad-Core	2GB	Android 4.4.2
Samsung Galaxy S6	2.1Ghz	Octa-Core	3GB	Android 5.0

Cuadro 5.1: Características de los dispositivos móviles utilizados.

5.1.1 Métricas de evaluación

Existe una gran cantidad de métricas por las cuales pueden evaluarse y compararse los modelos de regresión. Todos los indicadores comparan los valores reales con sus estimaciones, pero lo hacen de una manera ligeramente diferente.

Mean Absolute Error (MAE) es el promedio del error absoluto (diferencia entre los valores predichos y los observados), e indica cuán grande es el error que puede esperarse de la predicción. Al tratarse de una métrica basada en el error medio puede subestimar el impacto de errores grandes pero infrecuentes. Si el análisis se centra demasiado en la media arrojará conclusiones precipitadas, por lo tanto para ajustar errores grandes y raros, se calcula el error cuadrático medio (Root Mean Absolute Error (RMSE)). La ecuación de la métrica RMSE es:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N} f_i - {y_i}^2}$$

La misma nos permite llegar a una medida del tamaño del error que da más peso a los errores grandes e infrecuentes que la media.

El coeficiente de correlación de Pearson es una medida de la relación lineal entre dos variables aleatorias cuantitativas, independientemente de la escala de medida de las mismas. El coeficiente de correlación (Correlation Coefficient (CC)) es analizado en conjunto con el indicador *RMSE* ya que existe una estrecha relación entre ambos. Por ejemplo, si *CC* es 1, *RMSE*

debe ser 0, ya que todos los puntos se encuentran en la línea de la función de regresión; cuanto más cercano sea el valor de *CC* a 1 o -1, más próximos son los valores observados a la línea de predicción, y por tanto menor será el error absoluto reflejado en *RMSE*. La fórmula de CC entre dos variables x e y se define como:

$$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

5.1.2 Parámetros de configuración de las técnicas

La obtención de un modelo de calidad tiene lugar a partir de la optimización de los parámetros de configuración de la técnica de regresión. Estos parámetros pueden tomar distintos valores adaptándose a condiciones particulares de los datos, generando modelos mas precisos que otros. El Cuadro 5.2 presenta la lista completa de parámetros de configuración de cada técnica, el valor por defecto adoptado para cada una y el rango de valores con el cual se busca el modelo de mayor precisión.

Técnica	Parámetro		Valor por Defecto	Rango de Configuración	
Linear Regression	Ridge	R	1	-5	5
MultiLayer Perceptron	Learning Rate	LR	0.8	0.05	0.2
	Momentum	M	0.3	0.05	0.2
	Training Time	TT	200	-	-
	Validation setSize	VS	0	-	-
Stochastic Gradient Descent	Learning	L	0.01	0	0.01
	Lambda	R	0.05	0	0.1
Simple k-cluster	Cluster Numbers	N	30	-	-
Support Vector Machine	Complexy	C	1.0	0	25
	Exponent	E	3.0	2	-
	Kernel	K	Normalized PolyKernel	-	-

Kernel	Parámetro		Valor por defecto	Rango de configuración	
NormalizedPolyKernel	Exponent	E	2.0	-	-
PolyKernel	Exponent	E	2.0	-	-
Puk	Omega	O	1.0	-	-
	Sigma	S	1.0	-	-
RBFKernel	gamma	G	0.01	-	-

Cuadro 5.2: Parámetros de configuración de las técnicas de regresión

El proceso de optimización de las técnicas utiliza el rango de configuración establecido para iterar en pasos tomando valores intermedios y evaluando la calidad del modelo generado con tales configuraciones. Los resultados se comparan entre sí para determinar la mejor configuración para un dataset y propiedad a predecir particular.

Los rangos para cada técnica se establecieron tras un análisis de la influencia de estos valores sobre los datos. Los rangos de prueba se definieron de manera aleatoria, al mismo tiempo que el análisis de un rango ya propuesto servía como base para definir uno nuevo. Por otro lado, los valores elegidos por defecto fueron tomados de los valores propuestos por la herramienta Weka, a excepción del parámetro *Training Time* de la técnica *MultiLayer Perceptron* que fue reducido de 500 a 200 para mejorar el tiempo de ejecución del algoritmo.

En las siguientes secciones se presenta la aplicación del enfoque sobre distintos escenarios. Por cada escenario se describen los componentes involucrados, el proceso de medición de los mismos, los datos obtenidos con los que se entrenan las técnicas de predicción, y los resultados de la evaluación de los modelos obtenidos.

5.2 Caso de estudio 1: algoritmos para multiplicación de matrices

La multiplicación o producto de matrices es una operación matemática entre dos matrices $A := (a_{i,j})_{m \times n}$ y $B := (b_{i,j})_{n \times p}$ con la que se obtiene una nueva matriz $C := (c_{i,j})_{m \times p}$, donde cada número real $c_{i,j}$ de C esta definido por: $c_{i,j} = \sum_{r=1}^{n} a_{i,r} b_{r,j}$. El producto de matrices es un problemas de clase P, que a diferencia de los problemas NP tienen complejidad temporal polinómica. El producto entre matrices no es conmutativo, depende del orden de las matrices involucradas, y sólo es posible si el número de filas de la primera matriz es igual al número de columnas de la segunda.

Un dataset sobre este dominio ha sido formado tras la ejecución y medición del tiempo de ejecución de diferentes algoritmos de multiplicación de

matrices, con tamaños variados de matrices. Los componentes (algoritmos) considerados se describen a continuación:

Algoritmo Simple implementación simple del algoritmo desarrollada puramente en lenguaje Java.

Algoritmo Multi Thread corresponde a una versión paralelizada del componente anterior para aprovechar la capacidad multi núcleo de los dispositivos móviles.

Algoritmo RenderScript es una versión implementada con *RenderScript*[1], un framework para la paralelización de algoritmos en procesadores gráficos (GPU) sobre dispositivos Android, de modo que si el dispositivo tiene un Graphics Processor Unit (GPU) compatible con RenderScript, éste lo ejecutará.

Las instancias del problema utilizadas para la medición de los algoritmos, es decir, el conjunto de matrices A y B, fueron generadas aleatoriamente considerando diferentes dimensiones (tamaño) de las matrices involucradas. El dataset resultante de mediciones incluye entonces el tiempo de respuesta de la operación registrada en segundos, que es la propiedad a predecir, y el tamaño de las instancias del problema que son las propiedades a partir de las cuales se puede inferir el tiempo de respuesta. Estas propiedades son: el número de files de la matriz A (m), el número de columnas de A (n), que es igual al número de filas de B, y el número de columnas de B (p).

Las pruebas fueron ejecutadas sobre un dispositivo Samsung Galaxy S3. Lo interesante de este dataset, es que los algoritmos fueron implementados con diferentes técnicas para ser ejecutados por diferente elementos de hardware (Central Processing Unit (CPU), GPU). El dataset incluye un total de 450 entradas (mediciones), que corresponde a la ejecución de los 3 algoritmos con 150 diferentes instancias del problema (matrices de dimensión 1 hasta 150). La Figura 5.2 presenta gráficamente los resultados de estas mediciones.

[1]https://developer.android.com/guide/topics/renderscript/compute.html

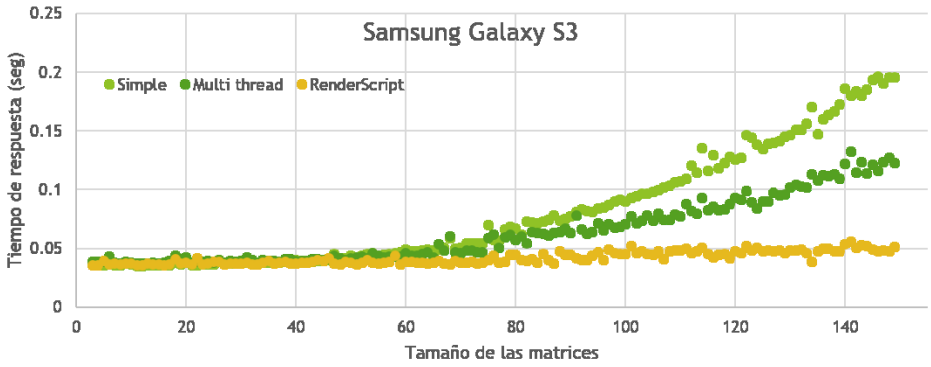

Figura 5.2: Mediciones del tiempo de respuesta de los algoritmos de multiplicacion de matrices

Respecto a los valores de tiempo registrados, puede notarse que el tiempo de respuesta de los algoritmos crece polinomialmente respecto al tamaño de las matrices, lo que coincide con la complejidad temporal de la multiplicación de matrices que es $O(m \times n \times p)$. También se observa que la versión del algoritmo multi-thread es mas eficiente que la versión simple, y la versión usando el framework de RenderScript es mucho más eficiente que las anteriores debido a la arquitectura de los procesadores gráficos (GPU), que están diseñados para realizar operaciones matriciales de coma flotante para el procesamiento de gráficos y otros calculas.

Lo interesante de este escenario es que permite la creación de modelos simples, por ejemplo, a partir de la técnica de regresión lineal, ya que existe una correlación clara entre el tamaño de las instancias del problema y el tiempo de ejecución que se puede aproximar con la siguiente expresión:

$$tiempo_{C,D} = constante1 + constante2 \times m \times n \times p$$

siendo $tiempo_{C,D}$ es el tiempo estimado de ejecución de un algoritmo C sobre un dispositivo D, para matrices de entrada $A_{m \times n}$ y $B_{n \times p}$.

En el cuadro 5.3 se contrastan cada una de las técnicas de regresión con cada uno de los componentes contemplados en el caso de estudio:

	Algoritmo simple		Algoritmo multi-thread		Algoritmo RenderScript	
	CC	RMSE	CC	RMSE	CC	RMSE
Linear Regression	0.9945	0.0042	0.9085	0.0156	0.5718	0.0002
MultiLayer Perceptron	0.9973	0.035	0.9187	0.0148	0.8578	0.013
SGD	0.9945	0.047	0.906	0.0209	0.5122	0.0061
Clusterer	0.996	0.012	0.9903	0.052	09929	0.0003

Cuadro 5.3: Resultados obtenidos del primer caso de estudio.

En el cuadro 5.3 se pueden apreciar las métricas obtenidas a partir de los modelos calculados. A simple vista, se puede apreciar que los modelos obtenidos para los dos primeros componentes presentan una gran correlación, significando que el atributo buscado (response time) tiene una gran dependencia de los atributos contemplados. Esto concluye que el tamaño de las matrices multiplicadas influye en el tiempo de respuesta para obtener el producto de dichas matrices. En contraste, los modelos obtenidos para el algoritmo RenderScript no presentan una correlación cercana a 1 como los demás algoritmos. El componente RenderScript optimiza la ejecución de la multiplicación, disminuyendo la relación entre el tamaño de la matriz y el tiempo de respuesta.

En todos los escenarios presentados, se ha optado por el caso intermedio de adaptación, considerando que el mejor modelo obtenido será aquel que evite casos de overfitting o underfitting, focalizando el análisis en los valores de CC y RMSE intermedios. Por lo tanto, en este caso, el modelo más eficiente es el *Multilayer Perceptron*.

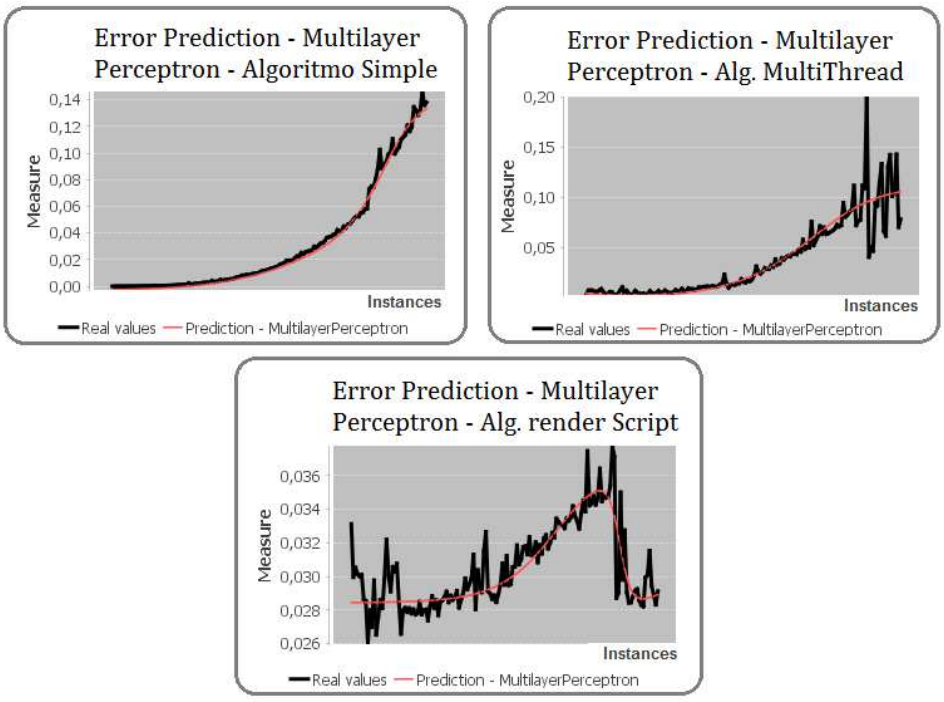

Figura 5.3: Lineas de predicción en la herramienta Nekonata

5.3 Caso de estudio 2: algoritmos para el problema del viajante

El problema del viajante o Travelling Salesman Problem (TSP) por sus siglas en inglés es un problema clásico de optimización combinatoria que consiste en encontrar la secuencia de caminos de menor distancia que debe recorrer un viajante para visitar n ciudades una y solo una vez cada una, empezando por una ciudad cualquiera de ellas y regresando al mismo lugar del que partió. En el campo de la teoría de complejidad computacional, el problema del viajante, es un problema NP-Completo y es modelado como un grafo de manera que las ciudades son sus vértices, los caminos son las aristas y las distancias entre caminos son los pesos de las aristas. Es un problema de minimización tras la búsqueda de un recorrido completo que comienza y finaliza en un vértice específico y visita el resto de los vértices

exactamente una vez con coste mínimo. Existen muchas variantes del problema, una de ellas se trata del TSP simétrico en el cual la distancia entre un par de ciudades es la misma en cada dirección formando un grafo ponderado no dirigido. Esta variante fue la versión considerada por la herramienta.

Los componentes o algoritmos considerados para este dominio se describen a continuación:

Best Fit algoritmo heurístico que selecciona las ciudades de acuerdo al costo de sus trayectos y son incorporadas al conjunto Solución en base al cálculo de la distancia marginal de las intersecciones..

Nearest neighbour partiendo de alguna ciudad arbitraria se analizan todos los vértices adyacentes y aún no visitados, y se añade a la solución aquel vértice cuya arista de costo sea la mínima.

Lineal Programming realiza una búsqueda exhaustiva simulando el algoritmo backtracking, considerando restricciones y funciones objetivo como fórmulas matemáticas.

Prim crea un árbol de recubrimiento sobre el grafo y crea un ciclo euleriano mínimo

Kruskal algoritmo similar a Prim en funcionamiento pero no en ejecución, variando su complejidad.

Local Transformations A través de un grafo hamiltoniano, se analizan las aristas cruzadas obteniendo un nuevo ciclo.

Con el fin de garantizar instancias variadas del problema (diferentes cantidades de ciudades, distintas representaciones y pesos) y para la medición de los componentes, se utiliza la librería TSPLib[2] que contiene archivos con múltiples instancias del problema.. La librería cuenta con 111 archivos de formato *TSP*. Los ejemplos oscilan desde 14 y hasta 18512 ciudades con un sólo ejemplo extremo de 85900, y utilizan cuatro funciones de cálculo de la

[2]https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

distancia entre ciudades: distancia euclidiana, distancia geométrica, distancia pseudo euclidiana y función techo. Para la obtención de las mediciones correspondientes, se realizó una pre selección de 32 archivos que comprenden ejemplos de entre 22 y 200 ciudades. Cabe destacar que fue necesaria la implementación de un parser para el uso de los archivos.

Por último, el dataset de mediciones obtenido para el entrenamiento de modelos incluye atributos de las instancias del problema y propiedades no-funcionales de la ejecución del componente. Los posibles atributos de instancia son el número de ciudades, la cantidad de caminos y la longitud de los mismo, sin embargo sólo se consideró la cantidad de ciudades; y las propiedades de ejecución del componente son el tiempo de respuesta, el valor solución, y la exactitud u optimalidad del resultado, que es la razón entre el valor de la solución encontrada y el valor óptimo de esa instancia del problema.

Las pruebas fueron llevadas a cabo sobre los dispositivos móviles Samsung Galaxy S4, S5 y S6. Los resultados presentados a continuación consideran el desempeño de los aldoritmos sobre el dispositovo S4 a modo de referencia ya que los mismos no presentaban diferencias significativas entre sí.

En la figura 5.4 se puede observar el tiempo de respuesta de los diferentes algoritmos considerando la cantidad de ciudades. A simple vista se puede observar que los algoritmos aproximados u óptimos tienen menor costo computacional debido que no exploran la totalidad de ciudades para obtener la respuesta. Mientras que los algoritmos exactos consumen más tiempo de respuesta ya que hacen un análisis exhaustivo e incluso pueden requerir de algoritmos auxiliares como el de ordenamiento. Gráficamente también se puede observar un umbral en 50 ciudades, dividiendo los tiempos de respuesta entre los cuales consideran la cantidad de ciudades como un factor influyente y aquellos en los que pasa desapercibido.

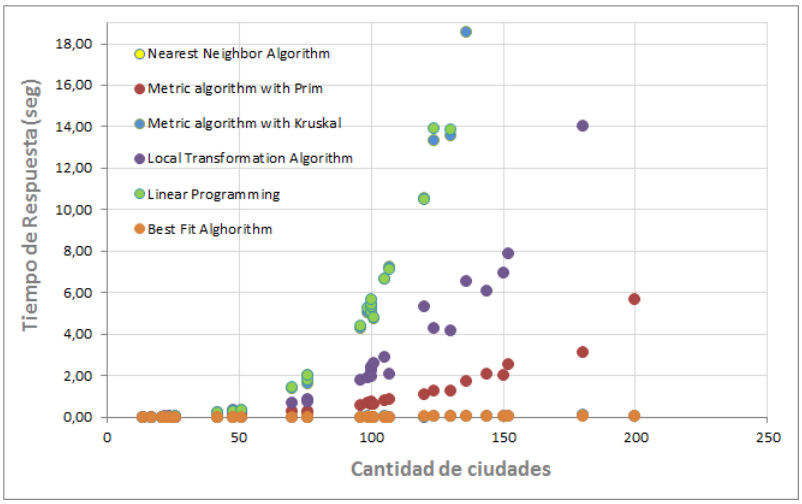

Figura 5.4: Mediciones de tiempo de respuesta de los algoritmos de resolución del problema del viajante.

A continuación se presentan los resultados de los modelos de regresión del tiempo de respuesta:

	Best Fit		Nearest Neighbour		Linear Programming	
	CC	RMSE	CC	RMSE	CC	RMSE
Linear Regression	0.98	0.0127	0.9281	0.0191	0.7535	17.8555
MultiLayer Perceptron	0.99	0.0034	0.9806	0.0050	0.9938	10.4876
Stochastic Gradient Descendent (SGD)	0.98	0.0045	0.9479	0.0090	0.8522	3.8502
Clusterer	0.98	0.0023	0.9920	0.0024	0.9961	1.5746
Sequential Minimal Optimization (SMO)	0.99	0.0001	0.9999	0.0001	0.9999	0.1214

Cuadro 5.4: Resultados obtenidos del caso de estudio dos

	Prim		Kruskal		Local Transformation	
	CC	RMSE	CC	RMSE	CC	RMSE
Linear Regression	0.9835	1.1767	0.7553	17.0618	0.8835	4.4046
MultiLayer Perceptron	0.9923	0.1618	0.8993	8.8769	0.9950	0.7954
SGD	0.9675	0.3545	0.9941	2.3924	0.9239	2.0863
Clusterer	0.9879	0.1823	0.9926	2.0626	0.9863	0.7248
SMO	0.9999	0.0071	0.9999	0.0983	0.9999	0.0242

Cuadro 5.5: Resultados obtenidos del caso de estudio dos

A partir de los cuadros 5.4 y 5.5se desprenden algunas conclusiones respecto al segundo caso de estudio. En primer lugar, tanto el modelo de *MultiLayer Perceptron* como el de SMO son, claramente, los mejores en cuanto a adaptación al modelo. Ambos presentan una correlación de los datos cercano a 1 (CC), lo que representa una buena precisión de los datos predichos en relación a los reales. Como se ha se explicado en la sección2.4.4, esto puede parecer lo más óptimo. Sin embargo, observando las métricas y las curvas de errores, ambos modelos presentan el problema de overfitting.

Teniendo esto en cuenta, se descartan las técnicas *MultiLayer Perceptron y* SMO como mejores alternativas para este tipo de problemas. A su vez, se descarta la técnica *Linear Regression* ya que presenta los niveles de correlación mas bajos y los errores más altos.

La técnica que mejor se ajusta para este caso de estudio es el SGD. A continuación, se presenta el comportamiento de este último con respecto a los valores de referencia.

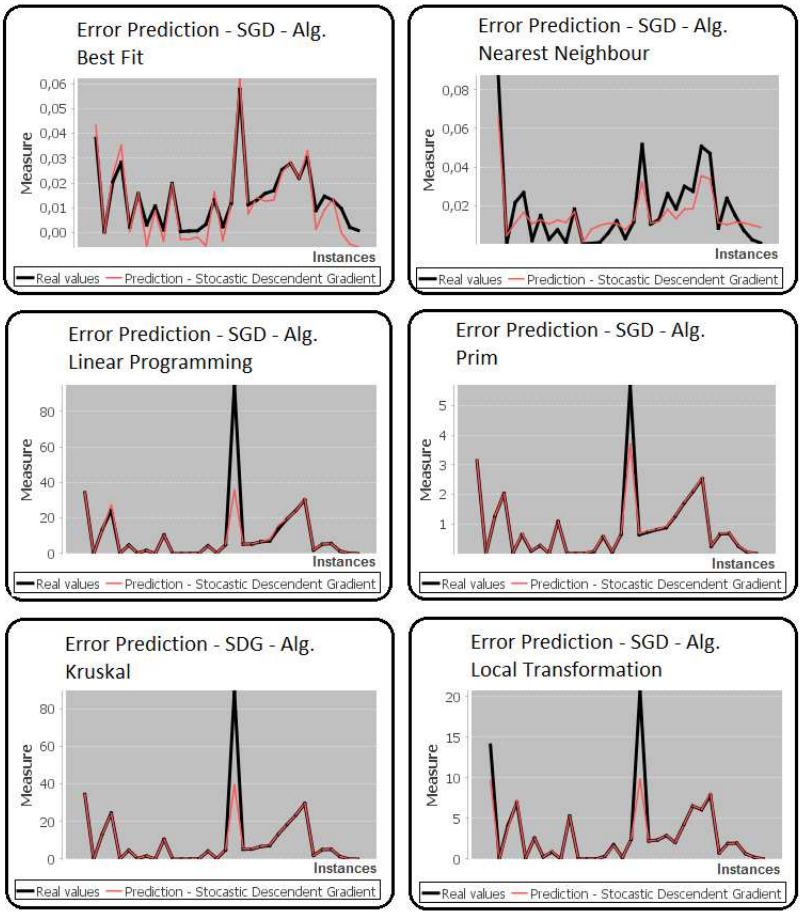

Figura 5.5: Lineas de predicción en la herramienta Nekonata

5.4 Caso de estudio 3: servicios para detección de rostros

El tercer escenario comprende un conjunto de servicios para la detección de rostros en imágenes. La función que ofrecen es identificar las regiones de una imagen que corresponden a rostros humanos. Hoy en día muchas soluciones para detección de rostro se proporcionan como servicios Web que, aunque no son tan eficientes y usables sin conexión a Internet como las bibliotecas de software, resultan generalmente más precisos y proporcio-

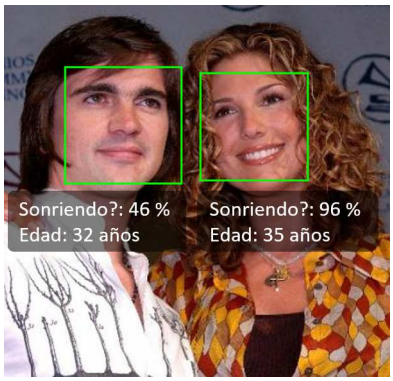

Figura 5.6: Ejemplo de detección de rostros en una imagen con el servicio de Microsoft Face API.

nan características adicionales del rostro como identificación de sonrisas, género e incluso estimaciones de edad. Sin embargo, el rendimiento y la precisión de los servicios varían según las propiedades de la imagen (tamaño, foco, oclusiones) y contexto de ejecución (capacidad de procesamiento, conexión de red). En la figura 5.6 se muestra un ejemplo de detección de rostros usando un servicio Web que incluye identificación de sonrisas y estimación de edad para cada rostro.

Los componentes considerados en el caso de estudio incluyen un servicio local para dispositivos Android (Google Play Services face detector[3]) y tres servicios Web (FaceRect API[4], Sky Biometry API[5] y Microsoft Face API[6]).

Google face API es una API para la detección de rostros en imágenes y vídeos en tiempo real que forma parte del paquete de servicios de Google Play para dispositivos Android. Estos servicios proporcionan una gran variedad de APIs de Google, como geolocalización, servicios de juegos y algoritmos de visión computacional como detección de rostros. Al ejecutarse de manera local en el dispositivo, la funcionalidad

[3]https://developers.google.com/vision/
[4]http://apicloud.me/apis/facerect
[5]https://skybiometry.com
[6]https://azure.microsoft.com/en-us/services/cognitive-services/

de detección de rostros no requiere conexión a Internet.

FaceRect API es un servicio Web simple y gratuito para detección de rostros en imágenes, consumido mediante interfaz Representational State Transfer (REST).

Sky Biometry API es un servicio Web mas completo que incluye características adicionales además de detección de rostros. También es consumido mediante una interfaz REST.

Microsoft Face API es un servicio Web en la nube que provee algoritmos de detección y reconocimiento de rostros muy avanzados. El servicio también se accede a través de una interfaz REST. El servicio forma parte de un grupo de APIs de inteligencia artificial llamado Servicios Cognitivos.

Estos cuatro componentes fueron evaluados a partir de un subconjunto de 290 imágenes pertenecientes al dataset llamado FDDB[7] con características muy variadas, incluyendo imágenes a color y escala de grises, rostros de diferentes tamaños y posiciones, distintas resoluciones de imágenes, etc. Como etapa de medición, cada servicio fue llamado especificando una imagen y un conjunto de parámetros booleanos con los que se le especifica al servicio que características se desean extraer de los rostros (posición, sonrisa, edad, etc). Por cada invocación se mide y registra el tiempo de respuesta en milisegundos. Las mediciones del servicio Google face API fueron realizadas sobre un dispositivo Lenovo K3 Note, mientras las mediciones de los servicios Web se realizaron con la herramienta JMeter[7].

A diferencia de los casos de estudio anteriores, la correlación entre el tiempo de respuesta y el tamaño de las instancias (medido, por ejemplo, como la cantidad de pixeles de la imagen) es muy baja. Esto se puede observar en la Figura 5.7, donde se ilustran las mediciones del tiempo de respuesta para el servicio de Google respecto al tamaño de las imágenes. Esta correlación es aun peor es las mediciones de servicios Web, por lo que las propiedades de las imágenes como su tamaño no fueron consideradas

[7] http://jmeter.apache.org/

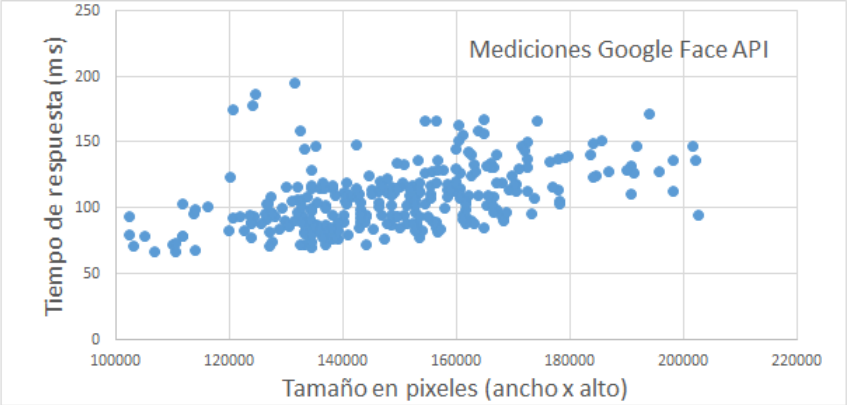

Figura 5.7: Mediciones del tiempo de respuesta de Google Face API

en el dataset como variables para la predicción. Los atributos del dataset son conformados por el valor de los parámetros booleanos con los que se invoca el servicio, lo que conlleva a que existan muchas instancias conformadas por atributos de igual valor que obtuvieron tiempos de respuestas distintos.. En consecuencia, dos instancias análogas en sus requerimientos podrían producir predicciones diferentes lo cual rompe el criterio de unicidad de toda función matemática.

En el cuadro5.6, se presentan los resultados de correlación y error de los modelos de tiempo de respuesta obtenidos para cada servicio:

	FaceRect API		SkyBiometry API		Microsoft Face API		Google Face API	
	CC	RMSE	CC	RMSE	CC	RMSE	CC	RMSE
Linear Regression	0.7000	153.6	0.08	637.54	0.41	71.78	0.6	59.17
MultiLayer Perceptron	0.7	153.52	0.09	637.83	0.58	64.04	0.66	55.54
SGD	0.7	704.11	0.06	676.21	0.41	73.3	0.59	59.31
Clusterer	0.991	27.92	0.13	633.84	0.56	65.28	0.78	45.83

Cuadro 5.6: Resultados obtenidos del caso de estudio tres.

Los datos exhibidos demuestran que el mejor modelo presentado para predecir el tiempo de respuesta para el problema de detección de rostros es el *MultiLayer Perceptron*. El algoritmo no sólo ha arrojado los valores más

bajos del error de predicción sino que en todos los casos, demuestra una buena adaptación a los dataset que por su naturaleza, difieren en volumen de instancias, cantidad y tipo de atributos.

A continuación, se presentan las lineas de predicción para demostrar el desempeño de la técnica.

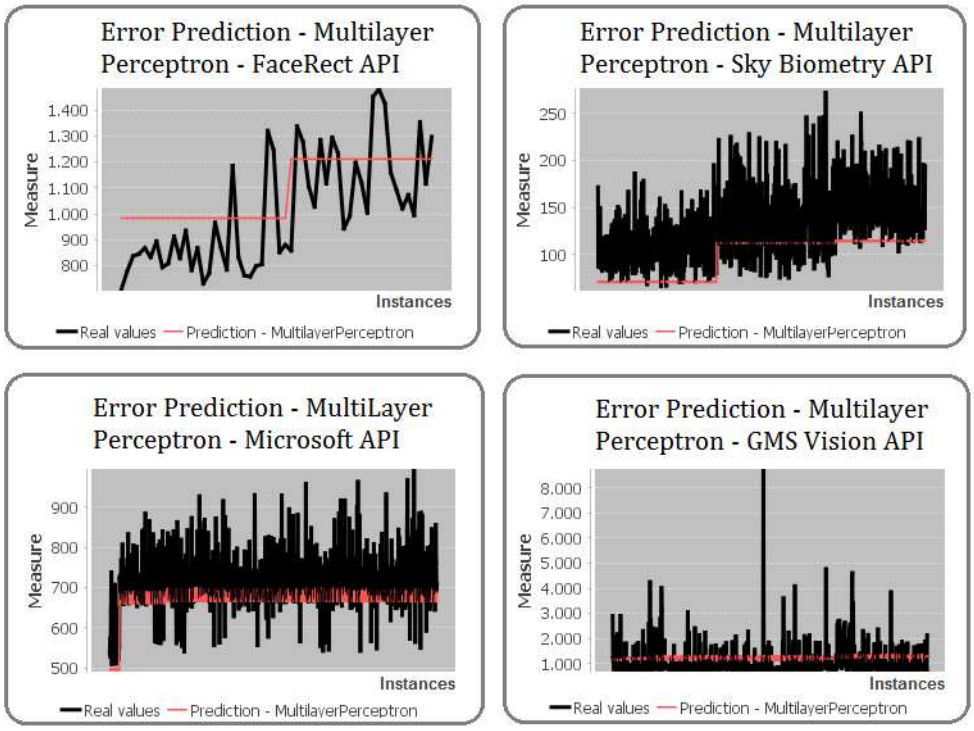

Figura 5.8: Lineas de predicción en la herramienta Nekonata

5.5 Análisis y discusión

En esta sección se analizan y discuten los resultados de las técnicas entre sí. Es importante mencionar algunas consideraciones sobre la técnica SMO por las cuales se excluyó su aplicación sobre el segundo y tercer caso de estudio. El desempeño de este algoritmo es considerablemente superior en la calidad de sus respuestas, sin embargo presenta algunas limitaciones para su uso; el desempeño del algoritmo no es apto para formatos de dataset

de cientos de instancias y múltiples atributos, ya que en algunos casos la técnica requirió más de un día de ejecución y en otros casos, sus resultados no pudieron ser visualizados.

En la evaluación del enfoque sobre los casos de estudio, solo se han ejecutado los componentes sobre un número reducido de dispositivos. Por lo tanto, no se consideraron características del dispositivo a la hora de capturar mediciones y atributos para el entrenamiento de modelos de predicción. Solo se consideraron atributos de los parámetros de entrada de los componentes, además de su tiempo de respuesta, por lo que los modelos solo generalizan la predicción sobre el dispositivo en los que fueron tomadas las mediciones.

Considerando las mediciones tomadas en la primer etapa, se puede analizar el comportamiento de los componentes respecto a un conjunto de entradas particulares. Así, en el primer caso de estudio se puede observar una gran dependencia entre el tamaño de la las instancias de la entrada y el tiempo de respuesta. En el segundo caso, esta dependencia se sigue observando pero mitigada mientras que en el último caso esta dependencia es baja debido a la complejidad del problema de detección de rostros.

Si el desarrollador desea conocer el tiempo de respuesta aproximado de un componente para un valor de entrada desconocido, utilizará el modelo de predicción obtenido del aprendizaje. Así, en el primer y último caso de estudio, el modelo de predicción más conveniente es el MultiLayer Perceptron, mientras que en el segundo es el SGD. En cuanto a la correlación, se destaca que en el caso de estudio de multiplicación de matrices, los algoritmos son sencillos y por lo tanto poseen una correlación muy alta. Esto significa, que la técnica determinó una buena relación entre las entradas de la función (entradas del problema) y el atributo a predecir (tiempo de respuesta). En el segundo caso, esta correlación es más baja debido a la mayor complejidad del problema del viajante, y por último, en el caso 3, el problema es más complejo no sólo por su objetivo sino porque no se han encontrado atributos de las entradas que tengan buena correlacion con el tiempo de respuesta. Esto significa una correlación mucho más baja y errores más altos.

Cuanto mayor sea la correlación entre los datos de entrada del problema y el tiempo de respuesta, menor es el grado de error en la predicción ya que es más factible que el modelo encuentre una formula matemática que describa la salida a partir de la entrada.

5.6 Conclusiones

En este capítulo se presentaron distintos casos de estudio y se analizaron los resultados producto de los mismos. En la primera sección se describió el procedimiento llevado a cabo para la evaluación de la calidad de los modelos predictivos construidos con técnicas de regresión. Luego, se describieron los escenarios que conforman cada uno de los casos de estudio donde se aplica el enfoque propuesto con las herramientas implementadas. Finalmente, se analizan los resultados de los modelos, de los que se extraen las siguientes consideraciones principales:

- En caso de dataset formados por pocas instancias, como el caso de estudio del problema del viajante, el comportamiento del SMO es similar al del Multilayer Perceptron. Sin embargo, es importante destacar el efecto de overfit de ambos algoritmos, y particularmente el tiempo de procesamiento que requiere el SMO.

- En el caso de un dataset conformado por pocas instancias y buena correlación entre el tiempo de respuesta y los datos de entrada, , como el caso de estudio dos, la técnica SGD presenta buenos resultados en el modelo obtenido en relación al tiempo de ejecución que conlleva.

- Analizando el desempeño de cada técnica y los resultados obtenidos, se concluye que la mejor técnica para predecir el tiempo de respuesta ha sido el Multilayer Perceptron. El mismo presenta buenos resultados en datasets complejos y simples, generalizando el comportamiento de los datos, como se puede apreciar en las figuras 5.8 y 5.3.

Capítulo 6

Conclusiones

En este trabajo se presentó un enfoque de aprendizaje automático para la estimación de atributos no funcionales en componentes de aplicaciones Android. Para darle soporte al enfoque, se implementaron dos herramientas especificas. La primera es una biblioteca para capturar mediciones de propiedades no funcionales durante la ejecución de componentes y algoritmos en dispositivos Android. La segunda es una herramienta de escritorio para entrenar modelos de predicción a partir de las mediciones capturadas. Esta herramienta incluye varias técnicas de regresión implementadas por la biblioteca Weka. Su diseño modular permite incorporar fácilmente las implementaciones de otras librerías de aprendizaje de máquina. La herramienta brinda la posibilidad de crear modelos predictivos óptimos, adecuados a un conjunto de datos particular y un atributo a predecir particular.

Ciertamente, el uso de modelos como criterio de calidad para la selección de servicios y componentes se está extendiendo cada vez más, promoviendo el desarrollo y optimización de técnicas de aprendizaje de máquina. En primer lugar, en este trabajo se presentaron las características principales de las técnicas que aplican aprendizaje sobre datos y se analizaron algunas propiedades no funcionales de componentes, como tiempo de respuesta. Luego, se estudiaron diferentes herramientas de recolección de datos en sistemas Android examinando los principales artefactos y mecanismos

involucrados en estas herramientas.

Como resultado del creciente desarrollo de aplicaciones, la integración de componentes de terceros se ha popularizado en diversas áreas, como el de las aplicaciones móviles. En este sentido, se introdujeron las nociones básicas de los dispositivos móviles, estudiando también, el sistema operativo más difundido en estos dispositivos, Android. Se analizaron las características más importantes del mismo, y se presentaron los conceptos necesarios para comprender el funcionamiento de las aplicaciones en este sistema.

La utilización de componentes y algoritmos de terceros en dispositivos móviles presenta ciertos desafíos. Estos dispositivos tienen limitaciones en conflicto como la energía, el acceso a la red y la capacidad de cálculo que determinan el contexto de ejecución de estos componentes y que afecta considerablemente los atributos de calidad y desempeño de los mismos y de las aplicaciones que los invocan. Por lo tanto, es importante elegir los componentes adecuados de acuerdo con su calidad de servicio además de la funcionalidad requerida. En este sentido se describieron algunas investigaciones recientes en el área que aplican aprendizaje de máquina y técnicas de regresión para estimar propiedades de desempeño de algoritmos y componentes en ejecución.

Por último, se llevo a cabo una serie de experimentos sobre 3 casos de estudio para validar el enfoque propuesto. Los mismos fueron evaluados con las métricas presentadas en secciones anteriores, señalando al modelo MultiLayer Perceptron como el mejor a nivel de predicciones bajas en error. Las estimaciones permiten predecir que en dominios de problemas clásicos como el caso del problema de la multiplicación de matrices y el problema del viajante métrico, el nivel de correlación alcanzado por el algoritmo es cercano a 1 y en cuanto al error en las predicciones a través de las métrica Root Mean Absolute Error (RMSE) , tendrán valores inmediatamente cercanos a cero. Para el caso de servicios, aunque las estimaciones arrojadas por el modelo presentan mas variabilidad, su comportamiento es considerablemente superior al resto de los modelos, en cuanto evita efectos de overfit (sobreajuste) y underfit (subajuste) sobre los datos. Esto indica

que una generalización del modelo propuesto puede resultar útil para determinar a priori qué componentes resultarán más eficientes, dadas ciertas características en las entradas, en los componentes y en la operación, para la predicción del tiempo de respuesta.

Respecto a los servicios para detección de rostros, se observa que las técnicas tiene dificultades para aprender modelos con buena precisión. Esto se debe principalmente a la complejidad del problema, ya que es difícil extraer características de las imágenes de entrada que estén correlacionadas con el tiempo de respuesta. En este caso de estudio, el modelo K means Clusterer logra superar al modelo MultiLayer Perceptron. Para el servicio Microsoft, el modelo alcanza un factor de correlación un 2% menor que el factor determinado por el modelo MultiLayer Perceptron, y para los servicios Google Play y SkyBiometry el modelo de cluster determina aún mejor la relación entre las variables y reduce el error en las predicciones respecto al modelo MultiLayer, que por sus resultados fue tomado como base para el análisis de las técnicas en general.

6.1 Limitaciones

La principal limitación que presenta el enfoque propuesto en este trabajo es que el entrenamiento de modelos de precisión requiere una gran cantidad de mediciones para generalizar mejor los datos. En la evaluación presentada, las mediciones fueron capturadas sobre unos pocos dispositivos y con un contexto de ejecución normal. Por lo tanto, los modelos construidos solo pueden predecir el tiempo de respuesta para esos dispositivos y contextos específicos. Idealmente la medición debería realizarse sobre múltiples dispositivos y con diferentes contextos de ejecución (por ejemplo, distinta disponibilidad de CPU, memoria, y otros recursos) de tal forma que los modelos puedan estimar con precisión el tiempo de respuesta de determinado componente en un contexto mas amplio. Sin embargo, esto es muy costoso porque requiere disponer de numerosos dispositivos y tiempo para capturar un conjunto de mediciones mas exhaustiva.

Una segunda limitación parte de la incapacidad de los dispositivos mó-

viles para llevar adelante ejecuciones mas complejas y costosas, que insumen gran cantidad de memoria y procesamiento. Este factor, limita la posibilidad de llevar a cabo la etapa de aprendizaje del enfoque sobre el mismo dispositivo móvil, obligando a desarrollar la herramienta de aprendizaje para su ejecución en computadoras de escritorio.

Por última, otra limitación que se presenta responde a la ausencia de un método automatizado para configurar dinámicamente los parámetros de las técnicas de regresión, debiendo determinar un rango estático y específico propio para cada técnica, para llevar a cabo el proceso de optimización, considerando de esta forma a todos los datos de entrenamiento por igual, sin distinción de formato, dominio, etc.

6.2 Trabajos Futuros

Cabe mencionar que existen algunas posibilidades de extensión interesantes para este trabajo. Como primer objetivo se pretende mejorar el tiempo de procesamiento requerido para la optimización y generación de los modelos predictivos. Para poder lograr esto, se puede otorgar a la aplicación la capacidad de ejecución paralela de los procesos de optimización y construcción de los diferentes modelos. Por otro lado, se debe dar un mejor soporte a la etapa de medición para considerar una mayor variedad de dispositivos y contextos de ejecución sobre los que capturar mediciones. Con este fin, se puede recurrir a algún proveedor de dispositivos en la nube (Device-as-a-Service), como Amazon WS Device Farm[1] o Samsung Remote Test Lab[2]. Estos servicios brindan acceso remoto a dispositivos móviles reales distribuidos geográficamente, lo que permite capturar mediciones sobre un grupo más amplio de dispositivos con características variadas.

Adicionalmente se pueden incorporar nuevos aspectos para mejorar y comparar las técnicas y modelos en la herramienta de aprendizaje:

- Independizar al módulo de optimización de técnicas, de diferentes

[1] https://aws.amazon.com/es/device-farm/
[2] http://developer.samsung.com/rtlLanding.do

rangos de configuración para cada uno de los parámetros que incluya la técnica. De esta forma, lograr más énfasis en los valores de cada parámetro y explotar al máximo el desempeño de las técnicas. Incluso, el usuario podría disponer de la posibilidad de personalizar estos valores.

- Informar y registrar el tiempo real consumido por cada operación de construcción de un modelo. Operación que incluye la configuración, optimización, entrenamiento y finalmente la evaluación del modelo con las métricas correspondientes.

- Extender el análisis comparativo a nivel de librerías, aprovechando la capacidad de la herramienta de integrar librerías de terceros. De esta manera, se logra comparar el desempeño de una técnica conocida en diferentes implementaciones, y seleccionar la mejor técnica independiente de la librería que la implemente.

- Considerar conjuntos de datos de la misma naturaleza que los usados durante la fase de entrenamiento, para evaluar los modelos creados. Brindar un mecanismo de evaluación de los modelos a partir de las métricas, y validar estos modelos creados frente a nuevos conjuntos de datos para esclarecer el desempeño y generalización lograda.

Bibliografía

[1] A comparative study for web services response time prediction. In *Comparative study on machine learning techniques in predicting the QoS-values for web-services recommendations*, 2015.

[2] R. D. Albu and F. Popentiu-Vladicescu. A comparative study for web services response time prediction. In *The 9th International Scientific Conference eLearning and software for Education*, April 2013.

[3] John Erickson and Keng Siau. Web service, service-oriented computing, and service-oriented architecture: Separating hype from reality. *Journal of Database Management*, pages 151 – 160, 2009.

[4] Holger H. Hoos Frank Hutter, Lin Xu and Kevin Leyton-Brown. Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[5] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[6] P. J. Phillips J. R. Beveridge, G. H. Givens and B.A. Draper. Factors that influence algorithm performance in the face recognition grand challenge. *Comput. Vis. Image Underst.*, 113:750–762, 2009.

[7] Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical report, University of Massachusetts Amherst, 2010.

[8] Paul Clements Len Bass and Rick Kazman. *Software Architect in Practice. Third Edition*. McGraw-Hill Science/Engineering/Math, 2012. ISBN 0013294278.

[9] Rick Kazman Len Bass, Paul C. Clements. *Software Architecture in Practice. 2nd edn*. Addison-Wesley Professional, Boston, USA, 2003. ISBN 0321154959.

[10] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, and Frank Neumann. A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence*, 2013.

[11] Helmut Petritsch. Service-oriented architecture (soa) vs. component based architecture. *International Journal of Web and Grid Services 2011*, 2016.

[12] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15: 65–118, 1976.

[13] Mark Roberts and Adele Howe. Learned models of performance for many planners. In *In ICAPS 2007, Workshop AI Planning and Learning*, 2007.

[14] Juan Manuel Rodriguez and Alejandro Zunino. M. c. introducing mobile devices into grid systems: a survey. *International Journal of Web and Grid Services 2011*, 7:1 – 40, 2011.

[15] T.M. Mitchell R.S. Michalski, J.G. Carbonell. *Machine Learning: An Artificial Intelligence Approach*. Springer Science and Business Media, 2013.

[16] M. R. Lyu Z. Zheng, H. Ma and I. King. Collaborative web service qos prediction via neighborhood integrated matrix factorization. *IEEE Transactions on Services Computing*, 6:289–299, 2013.