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

por
Agüero, Silvana
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, Marzo de 2017

Resumen

Durante los últimos años el desarrollo de aplicaciones en sistemas Android ha ido creciendo exponencialmente debido a la gran capacidad de cómputo que han adquirido los dispositivos móviles, haciendo que la reutilización de software mediante la integración de componentes de terceros se torne una práctica muy común. De este modo la misma funcionalidad suele ser ofrecida por componentes alternativos que difieren en sus propiedades no-funcionales o atributos de calidad, componentes con sus propias características, limitaciones y prestaciones, simplemente *cajas negras* a la vista del desarrollador. Por lo tanto, la elección del servicio más adecuado para ejecutarse en un determinado contexto, no es tarea sencilla.

Las limitaciones de hardware de los dispositivos móviles hace incapaz la ejecución o prueba de todos los servicios para determinar, luego, el más adecuado de acuerdo a su calidad de servicio y a la funcionalidad requerida. En este punto, es donde toma importancia el proceso de aprendizaje automático.

A través de la aplicación de modelos predictivos basados en la experiencia, es posible determinar el servicio que más satisface las condiciones del entorno, en base a un conjunto de restricciones del contexto de ejecución, a un conjunto de características de entrada y un conjunto de servicios que ofrecen la misma funcionalidad. Ya que estas condiciones o necesidades varían, el objetivo del aprendizaje de máquina es el desarrollo de sistemas que puedan cambiar su comportamiento (decisión) de manera autónoma basados en su experiencia (información generalizada). El aprendizaje de máquina está principalmente centrado en el estudio de la complejidad computacional y enfocado en el diseño de soluciones factibles a esos problemas.

El presente trabajo presenta un enfoque basado en tres fases complementarias y se proponen dos herramientas diseñadas para cumplir con los objetivos de las dos primeras fases. De este modo, se expone una herramienta de medición que facilita la obtención de indicadores de propiedades de componentes Android y otra herramienta que propone diferentes técnicas de regresión sobre las mediciones obtenidas previamente. Así, se logra construir modelos predictivos sobre alguna propiedad de interés de algún componente específico.

Agradecimientos

Índice general

Índice de figuras

Índice de cuadros

Glosario

ART Android Runtime

API Application Programmatic Interface

CC Correlation Coefficient

CPU Central Processing Unit

CSV Comma-Separated Values

DVM Dalvik Virtual Machine

EPM Empirical Performance Model

FN False Negative

FP False Positive

GPU Graphics Processor Unit

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

Java SE Java Standard Edition

Java ME Java Micro Edition

JDK Java Development Kit

JNI Java Native Interface

JRE Java Runtime Environment

JVM Java Virtual Machine

MAE Mean Absolute Error

MLP MultiLayer Perceptron

REST Representational State Transfer

RMSE Root Mean Absolute Error

SMO Sequential Minimal Optimization

RAE Relative Absolute Error

RRSE Root Relative Squared Error

SGD Stochastic Gradient Descendent

SQL Structured Query Language

SVM Support Vector Machine

TP True Positive

TSP Travelling Salesman Problem

UML Unified Modeling Language

VM Virtual Machine

XML eXtensible Markup Language

Capítulo 1

Introducción

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 reutilización de software mediante la integración de componentes de terceros. Un componente[11] es una entidad de software en tiempo de ejecución que encapsula 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 (para el caso de aquellos componentes que procesan datos de entrada para obtener un resultado de salida).

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 considerablemente 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.

1.1 Motivación

Para predecir la performance de un sistema, algunos enfoques definen un modelo del mismo como función de agregación que considera el desempeño individual de sus componentes. Así, por ejemplo, el tiempo de respuesta de un *mashup* de servicios Web puede determinarse como la suma de los tiempos de respuesta de los servicios invocados [15]. Lo mismo se puede aplicar sobre otras propiedades de arquitecturas y aplicaciones que involucran el ensamblado de diferentes componentes [2][17].

Muchos estudios se enfocan en el análisis y predicción de los aspectos dinámicos de componentes individuales, siendo las técnicas de aprendizaje de máquina, y en particular las de regresión, las más utilizadas. La mayor parte de estos estudios se enfocan en algoritmos de optimización e inteligencia artificial [4], donde se destaca el trade-off entre los tiempos de ejecución y la precisión o calidad de las respuestas obtenidas. En estos trabajos, se obtienen mediciones a partir de sucesivas ejecuciones de los algoritmos y se entrenan modelos con estas mediciones. Los modelos son funciones que dado determinado algoritmo e instancia de un problema permiten estimar su desempeño. Generalmente, diferentes algoritmos se desempeñan mejor que otros para distintas instancias, por lo que los modelos resultan útiles como criterio de decisión en la selección y parametrización automática de algoritmos, la asignación óptima de tareas en contextos Grid, entre otros escenarios.

La predicción de performance también se ha extendido a servicios Web [19] en una modalidad conocida como predicción colaborativa, en donde las mediciones son recolectadas y compartidas por múltiples nodos distribuidos en todo el mundo. En este caso, los modelos permiten determinar latencia, disponibilidad, y otras propiedades de un servicio a partir de la ubicación geográfica del cliente que lo invoca y otras características del contexto.

En las aplicaciones móviles, donde el contexto de ejecución y disponibilidad de los recursos puede variar rápidamente (cambio de red de acceso a internet, reducción de la batería, uso limitado de memoria y Central Processing Unit (CPU), etc) y diferentes componentes consumen estos recursos de diferente forma, es interesante el uso de estos modelos como criterio de calidad para la selección de servicios y componentes, tanto en tiempo de desarrollo como en tiempo de ejecución. Por esta razón, se plantea la posibilidad de desarrollar un enfoque para construir y evaluar modelos de predicción en el contexto de aplicaciones móviles.

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 de componentes accedidos por aplicaciones 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 que implementan la misma funcionalidad y de los cuales queremos predecir propiedades dinámicas, el método propuesto es el siguiente:

1. Usar conocimiento del dominio para seleccionar características del contexto y de los datos de entrada del componente que puedan ser indicativos de su desempeño.
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 las propiedades de interés a predecir: tiempo de respuesta, calidad de la respuesta, etc.
4. Usar estas mediciones con técnicas de aprendizaje de máquina para entrenar y evaluar modelos de predicción.

El enfoque propuesto se pondrá a prueba sobre grupos de componentes y servicios reales que implementan funcionalidades de interés para desarrolladores de aplicaciones móviles. Esta evaluación no sólo involucrará diferentes dominios, sino también diferentes técnicas, como regresiones y redes neuronales, sobre diferentes propiedades de interés. Para llevar a cabo la medición de los componentes se implementará una herramienta de test de performance para la plataforma Android, y se utilizará software de aprendizaje de máquina como Weka [6].

En conclusión, se espera poder proveer un análisis de los dominios y técnicas consideradas, como así también compararlas tanto en términos de precisión como también en cuanto a su generalización a diferentes contextos de ejecución (dispositivos) y datos de entrada

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 herramientas de benchmarks tanto para el sistema Android como para Java y se exponen algunos trabajos relacionados desde diferentes perspectivas, predicciones que incluyen modelos de performance y precisión como aproximación de éxito de los resultados, no sólo en algoritmos de optimización sino también en servicios Web.

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

En el capítulo 5 se presenta la evaluación de los modelos obtenidos a partir de todo el framework presentado. Se presentan las variables consideradas en los escenarios evaluados, los modelos que han sido analizados y las conclusiones alcanzadas, presentando justificativo para la selección de los mejores modelos predictivos alcanzados.

Finalmente, en el capítulo 6 se exponen las conclusiones del trabajo realizado, las limitaciones actuales del enfoque y la herramienta, 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 (funciones, algoritmos, servicios, etc.) en dispositivos móviles.

2.1 Dispositivos Móviles

Los dispositivos móviles son artefactos electrónicos pequeños con capacidad de trasladarse, 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 3G, el soporte de clientes de email, la eficaz administración 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 establecer 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 plataforma 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 Android denominada Google Play 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 una plataforma para el desarrollo de aplicaciones, utilizando Java como el lenguaje predeterminado.

La plataforma Android utiliza la máquina virtual Dalvik 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 Dalvik Virtual Machine (DVM) entonces, 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 figura2.1 se puede ver la arquitectura en capas empleada por el sistema Android y se brinda seguidamente una breve descripción de los componentes.

Las capas interactúan entre sí respetando el estilo arquitectónico tradicional, donde cada una de las capas utiliza los servicios ofrecidos por las anteriores, y ofrece a su vez sus propios servicios a las capas de niveles superiores.

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, la gestión de procesos,

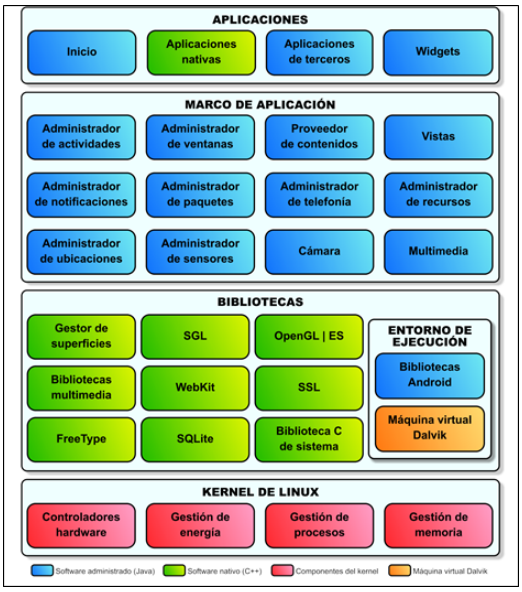


Figura 2.1: Diagrama de la arquitectura en capas empleada por Android

la pila de protocolos.

- 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 tradiciones de Java Virtual Machine (JVM) bajo el formato .dex. 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 librerías en lenguaje C o C++ usadas en varios componentes de Android proporcionando la mayor parte de las capacidades más características de Android, algunas de ellas fueron expuestas en la figura anterior. En el caso de las bibliotecas SQLite se provee acceso a la utilización y administración de bases de datos Structured Query Language (SQL). Estas librerías están compiladas en código nativo del procesador y muchas de ellas utilizan proyectos de código abierto.

- Marco o entorno de Aplicaciones: Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones de herramientas como sensores, localización, servicios, barra de notificaciones,

entre otros. y además 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 reutilización 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 sección siguiente

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 éstos dispositivos a través de las aplicaciones, que permiten adaptarlos según las necesidades de los usuarios. Además, el hecho de que Android sea de código abierto da lugar a la creación de diferentes versiones del sistema operativo que permiten personalizar cada aspecto del mismo. Android también permite que las aplicaciones se adapten a las características del dispositivo (pantalla, sensores, etc.), aprovechando las capacidades de cada dispositivo particular.

Respecto a las herramientas de desarrollo, Android ofrece un paquete que combina el Integrated Development Environment (IDE) Eclipse con un conjunto de herramientas que simplifican el desarrollo, permitiendo incluso, crear dispositivos virtuales que emulan cualquier configuración de hardware soportada.

Por otra parte, en el contexto de los servicios Web, Android provee soporte para clientes Hypertext Transfer Protocol (HTTP) mediante la clase *AndroidHttpClient*, y por tanto, permite la utilización de servicios Web Representational State Transfer (REST) a través de ella.

Un aspecto único 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. El componente *Activity* representa una pantalla simple y es el único que provee interfaz de usuario. A través de estas actividades se logra el acceso a la información almacenada en los componentes *Content Pro-*

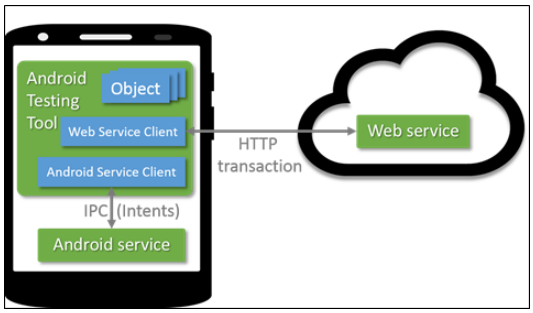


Figura 2.2: Diagrama de la arquitectura en capas empleada por Android

vider que 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 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 actividades 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.

Respecto a los anuncios o mensajes de difusión masiva, el componente *Broadcast Receiver* funciona como puerta de enlace a otros componentes, respondiendo mayormente a los anuncios originados por el sistema, por ejemplo, cuando la pantalla se apaga, la batería es baja, o al capturar una fotografía.

El último componente denominado *Intent* representa acciones a realizar, específicamente, tienen por finalidad iniciar actividades, servicios o *Broadcast Receivers* realizando el binding en tiempo de ejecución entre los componentes de las aplicaciones (tanto entre componentes de una misma aplicación como de diferentes). Cada Intent cuenta con una estructura de datos donde se realiza una descripción abstracta de una operación a realizar, luego, la entrega de estos objetos se realiza de diferente manera de acuerdo el tipo de componente que se desee activar.

La manera en que los componentes interactúan entre sí son resumidos en la figura 2.2 expuesta a continuación.

2.2 Componentes de software

En el marco del desarrollo de software para diferentes plataformas y para móviles más específicamente, nos encontramos con la posibilidad de reutilizar código previamente desarrollado, testeado y deployado que cumpla con una determinada funcionalidad ahorrando tiempo y esfuerzo al desarrollador. Estas piezas de códigos ya implementadas y disponibles se conocen bajo el nombre

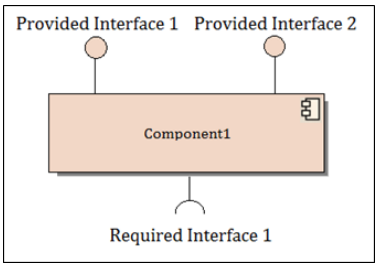


Figura 2.3: Componente UML con interfaces proveídas y requeridas.

de componentes de software e incluyen en su definición paquetes de software, servicios o recursos web, o módulos que encapsulan a un conjunto de funciones o datos relacionados; sin embargo, cualquiera sea la pieza que represente, frecuentemente los componentes son vistos y tratados como objetos o colección de objetos.

La reutilización es uno de los objetivos principales al momento de diseñar un nuevo componente de software de gran calidad para ser usados en diferentes programas. Por lo tanto la implementación y diseño de un componente conlleva un gran esfuerzo e incluye procesos complejos como la documentación, testing robusto a través de entradas comprensibles y la posibilidad de retornar mensajes de error y códigos, y el diseño enfocado a la utilización en formas imprevistas. Si bien los componentes actúan sin modificar su código, el comportamiento de los códigos fuentes del componente puede cambiar basado en la extensibilidad de la aplicación.

Todos los procesos de sistemas son estructurados bajo componentes separados de modo tal que todos los datos y funciones de un componente mantengan una misma relación semántica, principio que se mantiene en el contenido de las clases. Por esta razón, se le atribuye a los componentes las características de ser modulares y cohesivos.

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 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 modelo Unified Modeling Language (UML) y lo reflejado en la figura 2.3, las interfaces proporcionadas son representadas con símbolos de lollipop en el borde del componente y las interfaces requeridas por medio de sockets abiertos en el borde externo.

Otra de las características fundamentales de los componentes es su capaci-

dad de ser sustituibles 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 por lo 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

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, muchos factores determinan las cualidades que deben proporcionarse en la arquitectura de un sistema. Estas cualidades [5] van más allá de la funcionalidad (capacidades), servicios y comportamientos del sistema El diseño de la arquitectura de un sistema depende mayormente de los atributos de calidad (también denominados propiedades no funcionales) demandados por los stakeholders que de la funcionalidad en sí misma que especifica el comportamiento que debería tener el producto, para considerar un software de utilidad; 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.

2.3.1 Performance

Se puede medir la calidad de un sistema a través de su desempeño evaluando la efectividad del uso de los recursos disponibles en tiempo de ejecución. Dependiendo el contexto, la eficiencia puede medirse a través de varios parámetros, incluyendo tiempo de respuesta, ancho de banda y disponibilidad.

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 simplemente tiempo de procesamiento, pero básicamente se considera de todos ellos el tiempo que tarda el sistema para responder ante el 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.

Sin importar el contexto de desarrollo del sistema, el patrón de los eventos que llegan y el patrón de respuestas se pueden caracterizar, y esta caracterización hace posible la construcción de escenarios generales de rendimiento empleados en diferentes benchmarks, por ejemplo el tiempo de respuesta puede significar el tiempo máximo y mínimo de procesamiento para cierto elemento del sistema o la cantidad máxima de eventos que un determinado servicio puede atender por unidad de tiempo.

La respuesta del sistema a un estímulo puede ser caracterizado por la latencia (el tiempo entre la llegada del estímulo y la respuesta del sistema a la misma), los plazos de procesamiento, la fluctuación de la respuesta (la variación de latencia), el número de eventos no procesados debido a que el sistema era demasiado ocupado para responder, y los datos que se perdió debido a que el sistema era demasiado ocupado.

2.3.2 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 un componente, y cada componente está fuertemente ligado al dominio en el que se presenta, por ejemplo, en un problema de detección de rostros, la precisión puede significar la cantidad de rostros correctamente detectados sobre los rostros totales presentes, y en un problema de clasificación puede significar la correctitud en la predicción de la clase para un cierto atributo.

En términos generales, se toma la precisión como el grado de cercanía (o dispersión) de las mediciones respecto al verdadero valor, es el grado en que si las mediciones se repitieran bajo las mismas condiciones los resultados serían los mismos. A menudo, la medida de precisión es confundida con el término exactitud que indica la distancia del valor medido respecto al valor verdadero. Por ejemplo, si un experimento contiene un error sistemático, el aumento del tamaño de la muestra en general aumenta la precisión, pero no mejora la exactitud. El resultado sería una cadena consistente (aunque incorrecto) de los resultados del experimento defectuoso. La eliminación del error sistemático mejora la exactitud, pero no cambia la precisión.

2.4 Aprendizaje de máquina

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. La representación de los datos reales puede alcanzar el tamaño de terabytes por lo que dificulta hacer una analogía respecto al aprendizaje humano basado en la experiencia.

El hombre basa su conocimiento en tres partes: *i*) recuerdo, el hombre reconoce cuando ha sido la última vez que se estuvo en una determinada situación (dataset), *ii*) adaptación, reconoce la última vez que se probó una acción (salida producida) y *iii*) generalización, si ha funcionado o no (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. Estas fueron las bases del primer Aprendizaje Artificial y es conocido como procesamiento de símbolos ya que las computadoras manipulan símbolos que reflejan el ambiente. En contraste, los métodos de

aprendizaje de máquinas son llamados sub-simbólicos ya que no se utilizan símbolos.

El aprendizaje de máquina, entonces, es un proceso para que las computadoras modifiquen o adapten sus acciones (predictivas o de control de robot) 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 y biología, estadística, matemática y física, para generar técnicas y hacer que la computadora aprenda. Otro concepto importante en este ámbito es la minería de datos, el proceso de extraer información útil de un conjunto de datos masivos llamado comúnmente dataset por medio de algoritmos de alto grado de eficiencia y enfatizados nuevamente en la ciencia computacional.

Quizás, el punto de inflexión en este campo de estudio es la complejidad computacional de la aplicación de técnicas de aprendizaje de máquina sobre un gran volumen de datos, de manera que los algoritmos de complejidad polinomial muy grande pueden resultar un problema. Generalmente la complejidad se divide en dos partes: la complejidad del entrenamiento y la complejidad de aplicar el algoritmo entrenado. El proceso de entrenamiento usualmente, se realiza escasas veces, incluso tan sólo una, y el tiempo que requiere no resulta tan crítico, por lo que se enfatiza sobre la segunda parte donde se pretende que la decisión sea lo más rápida posible con un bajo costo computacional.

Si se define en forma imprecisa el concepto de aprendizaje como la mejora de tareas a través de la práctica surgen algunos cuestionamientos asociados, de qué forma la computadora puede saber si está aprendiendo mejor o de qué forma podría mejorar ese aprendizaje. De aquí, toman lugar diferentes tipos de aprendizaje de máquina, 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 la máquina *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 ya que la máquina debería ser capaz de encontrarla; una variante podría ser asignarle un puntaje a la respuesta obtenida por el algoritmo acordando cuán correcta resulta ser y no sólo indicarlo mediante los valores verdadero o falso.

Estas diferentes respuestas proveen una forma fácil de clasificar los diferentes métodos de aprendizaje, los cuales serán detallados en las secciones siguientes en base a la naturaleza de la entrada y la naturaleza de la salida; sin embargo todos los métodos comparten el mismo objetivo de generalización: el algoritmo debe producir salidas sensibles para datos de entrada que no fueron encontrados durante el aprendizaje teniendo en cuenta también que el algoritmo puede lidiar con ruido en los datos, una pequeña imprecisión en los valores que es inherente a la medición de cualquier proceso real.

2.4.1 Clasificación por la naturaleza de la entrada

El modo de aprendizaje que un algoritmo particular puede realizar queda determinado por la naturaleza de los datos de entrada, es decir, puede variar en

base a la información contenida en los datos de entrenamiento (*training set*).

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 determinada característica; a través de las respuestas correctas provistas y basado en el conjunto de datos el algoritmo 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 el algoritmo a partir de una entrada determinada.

Contrariamente, 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.

Por último, en el aprendizaje [5] 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, dando lugar a una percepción misteriosa sobre la forma que se espera que el método aprenda sin recibir ningún tipo de devolución. Sin embargo, 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.

Nótese que los valores de entrada y etiqueta describen vectores ya que cada ejemplo (*instance*) del conjunto de entrenamiento posee varias características (*features*); si se tuvieran todos los posibles ejemplos de un problema, no habría necesidad alguna de aprendizaje.

2.4.2 Clasificación por la naturaleza de la salida

2.4.2.1 Clasificación

La clasificación también se conoce como un proceso de categorización que 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. Estas dos características claves no se asemejan a la realidad, algunos ejemplos podrían pertenecer parcialmente a dos clases diferentes, incluso podría haber situaciones en las que fuere imposible categorizar la entrada, por ejemplo, si se considerara una máquina vendedora que utiliza un algoritmo para aprender a distinguir monedas nacionales, al introducir una

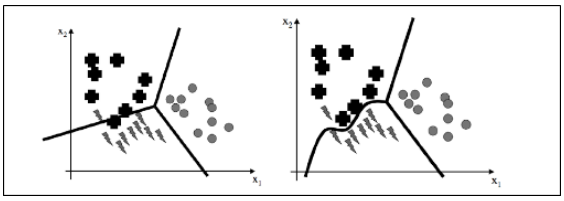


Figura 2.4: Conjuntos de umbrales de decisión para problemas de clasificación.

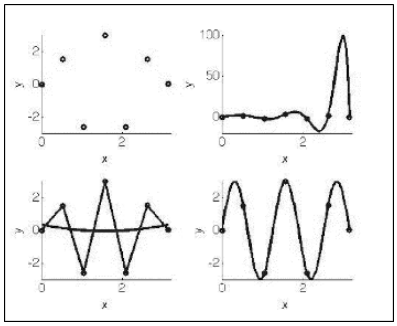


Figura 2.5: Comparación de distintas funciones polinomiales de regresión.

moneda extranjera, el clasificador será incapaz de responder correctamente, e identificará la moneda como aquella que más se asemeje. Este fenómeno se conoce como *novelty detection*. El algoritmo de clasificación tiene como objetivo encontrar umbrales de decisión que sirvan para identificar las diferentes clases, en la figura 2.4 puede observarse a la izquierda un conjunto de umbrales de decisión lineales y a la derecha un conjunto alternativo que separan más idóneamente las diferentes clases pero requiere líneas que no son rectas.

2.4.2.2 Regresión

A diferencia de la clasificación, 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 al atributo 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 causan diferentes efectos. La figura 2.5 refleja la situación expresada anteriormente, donde se muestra el conjunto de valores del dominio y tres curvas alternativas de representación. El gráfico inferior izquierdo combina, además, una función cúbica y línea recta.

La forma más correcta de saber cuál solución presentada es la mejor, se evalúa el grado de generalización que permite cada una; se toma algún punto

incluido entre los puntos ya representados y se utiliza la curva para predecir el valor, luego se compara entre los valores para saber cual resultó más próximo. En este caso, el gráfico inferior derecho.

En síntesis, un algoritmo de regresión puede interpolar entre dos puntos de datos, dando la apariencia de ser una práctica poco inteligente y no muy compleja en un espacio bi dimensional, pero su complejidad es exponencialmente mayor en dimensiones más grandes.

2.4.2.3 Clustering

El proceso de clustering es la tarea de agrupar 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.

Clustering es una de las tareas principales de la minería de datos y una técnica común en el análisis estadístico de los datos, usado en diversos campos, entre los que se destaca el aprendizaje de máquina, el reconocimiento de patrones, el análisis de imágenes, recuperación de información, bioinformación, compresión de datos y gráficos computacionales.

2.4.2.4 Estimación de densidad

La función de densidad de probabilidad [18] es un concepto fundamental de la estadística y caracteriza el comportamiento probable de una población en tanto especifica la posibilidad relativa de que una variable aleatoria continua X tome un valor cercano a x; especificar la función de densidad brinda una descripción natural de la distribución de X, y permite encontrar probabilidad asociadas con X de la relación:

$$P(a < X < b) = \left(\overline{+}\right) f(d)dx \forall a < b$$

El conjunto de datos de entrenamiento puede considerarse como una función de densidad desconocida, donde la estimación de la densidad es, entonces, la estimación de una función de probabilidad desconocida de los datos observados; un uso natural de esta estimación es la investigación informal de propiedades del conjunto de datos, obtener indicadores sobre estas propiedades como la asimetría y multimodalidad de los datos. En algunos casos, los indicadores pueden proteger conclusiones consideradas verdades autoevidentes, mientras que otros marcan el camino para un análisis posterior y/o recolección de datos.

2.4.2.5 Reducción de dimensionalidad

Una de las principales motivaciones para la reducción dimensional es la dificultad de interpretación de datos de tres dimensiones o más, teniendo en cuen-

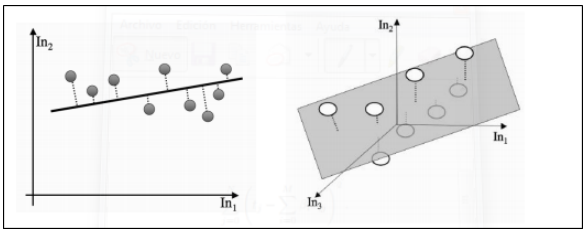


Figura 2.6: Algoritmo Linear regression en dos y tres dimensiones..

ta que cuanto mayor sea el número de dimensiones, más datos son requeridos infiriendo explícitamente en el costo computacional de muchos algoritmos. Por otro lado, reducir el espacio de dimensiones permite eliminar ruido sobre los datos, mejorar significativamente los resultados de los algoritmos de aprendizaje, simplificar el dataset y lograr resultados más claros y entendibles, entre otros.

Existen tres maneras de llevar a cabo la reducción: *i*) a través de la selección de atributos, observar las características disponibles y descubrir si las mismas son útiles, por ejemplo, el factor de correlación de las mismas con la salida, *ii*) la derivación de nuevas características a partir de las existentes, aplicando transformaciones al dataset cambiando los valores del dominio del gráfico mediante el movimiento y la rotación de los mismos, los cuales pueden ser descritos como una matriz aplicada a los datos con el fin de combinar características y detectar cuáles son útiles y cuáles deberían ser descartadas (reducción) y *iii*) a través del uso de clustering para agrupar puntos similares y permitir la identificación de un conjunto reducido a ser utilizadas.

2.4.3 Funciones contempladas

El foco principal del trabajo desarrollado ha sido guiado por la búsqueda de predicción de valores numéricos sobre componentes de ejecución, como el uso de CPU, consumo de red, precisión y desempeño de las respuestas, entre otros. Estos indicadores son valores continuos, motivo por el cual se utilizaron modelos de regresión, los mismos se describirán en las subsecciones siguientes.

2.4.3.1 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 línea recta, la salida será la suma de cada valor conocido multiplicado por una constante la cual define la línea recta (plano en 3D o hiperplano en dimensiones mayores) que circundan los puntos como puede observarse en la figura 2.6.

Para encontrar la recta que mejor se adapte a los datos se intenta minimizar la distancia entre cada punto y dicha línea, esta distancia se mide a través de

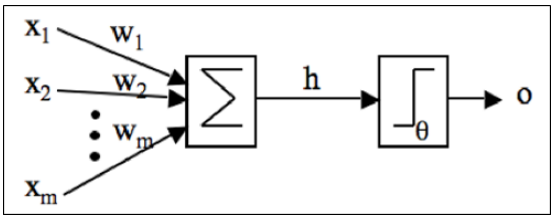


Figura 2.7: Modelo matemático neuronal McCulloch - Pitts.

una línea auxiliar que atraviese el punto y tope con la función, por ejemplo mediante el Teorema de Pitágoras. Luego, se intentará minimizar la función de error que mide la suma de cada distancia, si se ignora las raíces y sólo se minimiza la suma de los cuadrados de los errores, se obtiene la minimización más común llamada optimización de mínimos cuadrados.

La minimización de esta función da lugar a la implementación de diversas alternativas de la función lineal. En el presente trabajo se desarrollan dos variantes, la función denominada *ridge-regression* y la función *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 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.

2.4.3.2 Red neuronal

Se presenta un modelo matemático sobre el comportamiento de una neurona, el modelo neuronal McCulloch - Pitts denominado así debido a sus creadores Warren McCulloch y Walter Pitts, ambos produjeron un ejemplo perfecto al modelar una célula nerviosa como *i*) un conjunto de entradas valoradas (w_i) que corresponde a las sinapsis, *ii*) un sumador que une las señales entrantes (equivalente a la membrana de la célula que recolecta la carga eléctrica) 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 2.7, el modelo McCulloch - Pitts es un dispositivo límite binario, las entradas son multiplicadas por los pesos o fuerzas sinápticas y luego se suman 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. Teniendo en cuenta que una red neuronal puede llevar a cabo cualquier cálculo que realiza una computadora normal, los pesos o valores de la fuerza sináptica deben ser elegidos correctamente, considerando en consecuencia, un método capaz de configurar dichos pesos. Observando la neurona McCulloch y Pitts, se puede notar fácilmente que las entradas son independientes al cerebro, por lo que sólo cambia el valor de los pesos y el umbral. Considerando que el aprendizaje sucede entre las neuro-

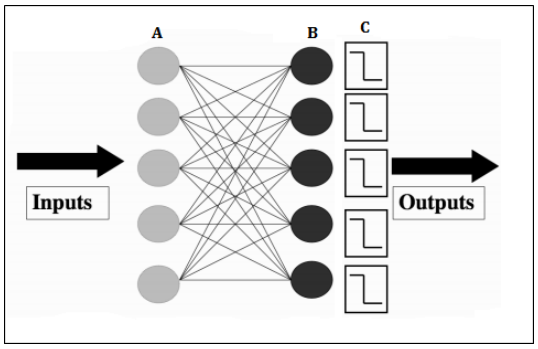


Figura 2.8: Red neuronal perceptron

nas y éstas están conectadas mutuamente, se debe tener en cuenta la manera en que en cambian los pesos y los umbrales de las neuronas para que la red pueda obtener la respuesta correcta más frecuentemente.

Teniendo en cuenta que una red neuronal puede llevar a cabo cualquier cálculo que realiza una computadora normal, los pesos o valores de la fuerza sináptica deben ser elegidos correctamente, considerando en consecuencia, un método capaz de configurar dichos pesos.

Observando la neurona McCulloch y Pitts, se puede notar fácilmente que las entradas son independientes al cerebro, por lo que sólo cambia el valor de los pesos y el umbral. Considerando que el aprendizaje sucede entre las neuronas y éstas están conectadas mutuamente, se debe tener en cuenta la manera en que en cambian los pesos y los umbrales de las neuronas para que la red pueda obtener la respuesta correcta más frecuentemente.

Red Neuronal Perceptron

El perceptrón es una colección de neuronas McCulloch y Pitts con un conjunto de entradas y pesos que unen las neuronas con dichas entradas.

Como puede observarse en el gráfico de la figura 2.8, las neuronas del Perceptrón son completamente independientes entre sí, el estado de una neurona no infiere sobre las demás, sólo comparten las entradas. Un punto a destacar es que el número de entradas y de neuronas no necesariamente debe corresponderse, en general hay m entradas y n neuronas.

El mayor inconveniente es conocer con certeza los valores que deben tener los pesos para que las salidas sean las correctas, esta es la finalidad de la red neuronal, aprender si determinada neurona debe o no activarse de forma correcta.

Un proceso de análisis sobre los pesos podría arrojar que los pesos toman valores muy grandes en la activación de una neurona (cuando no debería hacerlo) o en caso contrario, valores muy pequeños, por lo que se podría computar una función de error como la diferencia entre la salida que produjo la neurona y la salida de la red. En el caso particular de entradas en cero, se modifica

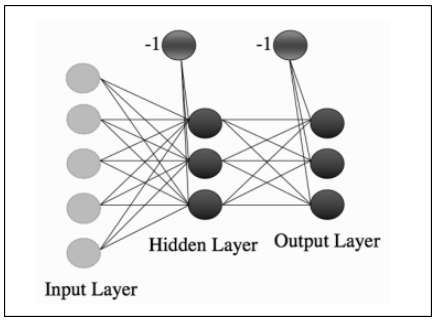


Figura 2.9: Red perceptrón multicapa

el valor de umbral añadiendo a la red Perceptron un parámetro extra denominado bias, los pesos de esta entrada son usualmente asignados al índice cero.

Red neuronal Perceptrón multicapa

La esencia del aprendizaje de la red neuronal perceptrón está centrada en los valores de pesos, una buena práctica entonces, sería la incorporación de nuevos pesos. Como alternativa para ello podrían agregarse conecciones hacia atrás para conectar las neuronas de salida con las entradas nuevamente, o podrían agregarse nuevas neuronas creando redes mucho más complejas, como el caso reflejado en la figura 2.9 donde se representa un modelo perceptrón de tres capas con cinco entradas, tres salidas y tres neuronas en la capa oculta.

La nueva red, ahora, debe ser entrenada para que los pesos nuevos se adapten y generen las respuestas correctas (*targets*). Estas respuestas son conocidas y se puede computar la diferencia entre estas y las salidas pero se desconoce si el peso incorrecto pertenece a la primera capa o a la segunda y cuáles activaciones de las neuronas de la capa intermedia son correctas, razón por la cual esta capa se denomina capa oculta ya que es imposible examinar y corregir sus valores de forma inmediata.

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*).

Por lo tanto, con la función de error y de activación, puede calcularse el diferencial para modificar los pesos en dirección del gradiente negativo, mejorando así la función de error. El objetivo es obtener los gradientes de esos errores y usarlos para decidir cuánto deben actualizarse dichos pesos en la red sobre la capa de salida y, luego de actualizarla, se opera hacia atrás (*backwards*) a través de la red hasta llegar a la entrada nuevamente. Sólo existen dos problemas, para las neuronas de salida se desconocen las entradas que le corresponden y para las neuronas escondidas, se desconocen las salidas esperadas (*targets*) y en caso de existir más capas incluso podrían no conocerse las entradas. Para

solucionar estas cuestiones se aplica la regla de la cadena la cual afirma que para conocer la forma en que varía el error en base a la variación de los pesos, se debe analizar la variación del error en función de las entradas de los pesos y multiplicarlo por el valor de la variación de los pesos según la variación de entradas.

Esta forma de resolución es muy útil ya que permite conocer todas las derivadas que se necesitan. Puede escribirse la función de activación de los nodos de salida en términos de los nodos ocultos y de los pesos de salida para luego enviarlo hacia las capas ocultas de la red (hacia atrás) para decidir cuáles eran los salidas esperadas para esas neuronas.

2.4.3.3 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.

Los algoritmos de aprendizaje generalmente se basan en minimizar alguna clase de error, en consecuencia la primera acción que se realiza es definir un criterio que lo describa teniendo en cuenta: *i*) una medida de la distancia entre puntos, definir una medida para cuantificar estas distancias, por lo general se usa la distancia euclídea y *ii*) determinar el punto central de un conjunto de puntos (*mean average*) considerando el espacio euclídeo ya que en espacios curvos es otra la interpretación. Teniendo en cuenta estos conceptos, el algoritmo K - means calcula el punto medio (centro) de cada cluster (μ_c(i)). lo que resulta equivalente a minimizar la distancia euclídea de cada punto del cluster al centro del mismo.

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 determinar 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 dejar 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 del atributo a predecir de los otros puntos que están en el cluster.

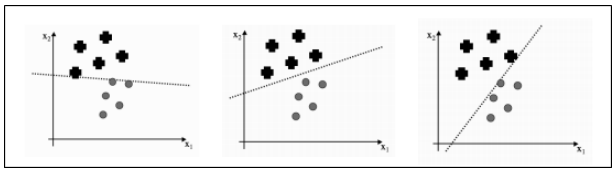


Figura 2.10: Comparación de tres clasificadores lineales distintos.

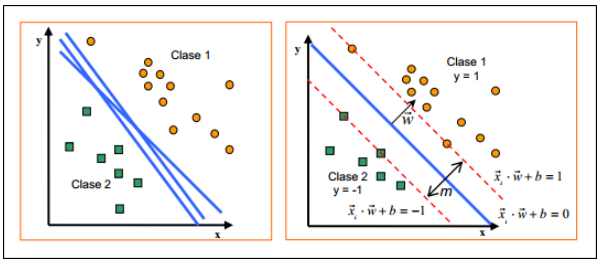


Figura 2.11: Metodología de operación del algoritmo SVM

2.4.3.4 Maquina de vector de soporte

Al tratar con problemas de clasificación existen varios clasificadores lineales que ofrecen respuestas correctas pero diferentes, como se expone en la figura 2.10. Si bien las soluciones laterales son acertadas, las líneas divisorias son muy próximas a algunos puntos del conjunto de datos de entrenamiento dando paso a una posible imprecisión futura en la predicción, situación que no se observa en el ejemplo central de la figura.

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. Esta región conforma un cilindro simétrico alrededor de la función en 3D y un hiper - cilindro en dimensiones mayores. 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. Si se considera el mejor clasificador como aquel que atraviesa la zona desierta, se debe tener en cuenta que el margen debe ser lo más grande posible y que si los vectores de soporte son los puntos más importantes de los datos, luego del entrenamiento pueden descartarse los puntos que no pertenecen al vector y utilizarlos para clasificar, lo cual conlleva una gran eficiencia en el almacenamiento de datos. El algoritmo puede encontrar varios hiperplanos de separación posibles y detectar aquel que sea más óptimo. En la figura 2.11 se reflejan estas dos situaciones y se expone, en el gráfico derecho las notaciones matemáticas formales del algoritmo.

A través de la función $g(x) = w^t x + b$ se definen los dos hiperplanos (clasificador). Para obtener el punto más cercano desde la línea de clasificación considerando la clase 2 (ver figura 2.11) se recorre en dirección perpendicular

partiendo del límite de la clase 1 hasta llegar al límite de la clase 2, el primer punto interceptado se denomina x. Para obtener un clasificador de buena calidad, se pueden definir un conjunto de restricciones que indiquen la manera en que el clasificador podría obtener la respuesta correcta. Por ejemplo, si en lugar de tomar las dos clases como $g(x) = 1$ y $g(x) = 0$, se podrían igualar a los valores 1 y -1 y así obtener una respuesta correcta diferente. Estos tipos de problemas pueden ser resueltos de forma directa y eficiente (por ejemplo, en tiempo polinomial), gracias a que existen algoritmos de programación cuadrática que brindan soluciones efectivas.

Todas las consideraciones expuestas hasta el momento se realizaron tras la asunción de que los datos podían ser divididos con funciones lineales, pero no siempre es posible, en este caso la solución es introducir variables. Estas variables indican que, al comparar clasificadores, si uno clasifica un punto en un lado incorrecto y el otro lo ubica mucho más lejos (también en un lado incorrecto) el primero es mejor que el segundo, el error no fue tan extremo como en el segundo y esta información debe ser incluida en los criterios de minimización modificando las restricciones, incluyendo un parámetro C. C es el parámetro que indica el trade off que existe entre ambos parámetros: cuanto más pequeño sea el valor de C más importancia toma el margen sobre algunos errores, cuanto más grande se vuelve C ocurre lo contrario. Esto transforma el problema en un clasificador de margen suave ya que se permiten algunos errores. La forma de evitarlos es modificando las características para que los datos sean linealmente separables, los datos no pueden inventarse de modo que las características deben ser derivadas a partir de las existentes. Para ello se describe una nueva función llamada kernel denominada $\varphi(x)$ que se aplica sobre las diferentes variables de entrada, su función principal es transformar los datos para obtener un espacio de dimensión mayor. Como función kernel puede utilizarse cualquier función simétrica que está definida positivamente (positividad de la integral de funciones arbitrarias). Incluso, la conjunción de dos funciones kernel significa una nueva función kernel. Elegir la función kernel más conveniente es un problema complejo. A pesar que existen algunas teorías, el proceso más común es la experimentación con distintos valores para determinar la función más adecuada utilizando el conjunto de validación. El algoritmo que aplica vectores de soporte se rige bajo todos los conceptos expresados a lo largo del apartado para realizar clasificación, pero también aplica para realizar regresión sobre los datos, variante que fue utilizada en el presente trabajo. La figura 2.12 muestra un ejemplo de la aplicación del algoritmo SVM para regresión.

2.4.4 Evaluación de modelos

La evaluación del rendimiento de un modelo es una de las fases principales en el proceso de ciencia de datos. Indica el nivel de acierto de las predicciones del conjunto de datos mediante un modelo entrenado. Existen dos formas para evaluar: evaluar el modelo y validar el modelo de forma cruzada. Estos métodos permiten conocer el rendimiento del modelo como un número de

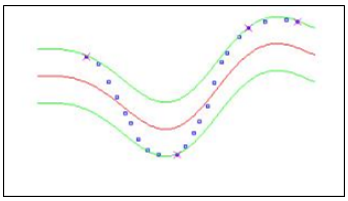


Figura 2.12: Algoritmo SVM para regresión con función kernel de base radial.

métricas que se usan habitualmente en estadísticas y aprendizaje automático. La evaluación y la validación cruzada son métodos estándares para medir el rendimiento de un modelo. Ambos generan métricas de evaluación que sirven para inspeccionar y comparar con las de otros modelos.

La evaluación se basa en los valores predictivos junto con las etiquetas o valores verdaderos. De forma alternativa, es posible usar la validación cruzada para realizar automáticamente varias operaciones de entrenamiento, puntuación y evaluación (10 subconjuntos) en distintos subconjuntos de los datos de entrada. Los datos de entrada se dividen en 10 partes, donde una se reserva para las pruebas y las otras 9 para el entrenamiento. Este proceso se repite 10 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 implementa las siguientes métricas de evaluación para los modelos de regresión:

- CC (Coeficiente de correlación de Pearson)
- MAE (Mean Absolute Error)

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

- RMSE (Root Mean Absolute Error)

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

- RAE (Relative Absolute Error)

$$\frac{\sum_{i=1}^{N} |f_i - y_i|}{\sum_{i=1}^{N} |\overline{f_i} - y_i|}$$

- RRSE (Root Relative Squared Error)

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

- COMB

$$(1 - CC) + RRSE + RAE$$

- SIMPLE ERROR

$$\sum_{i=1}^{N}(f_i - y_i)^2$$

El término *error* representa la diferencia entre el valor predicho y el valor verdadero. Normalmente, se calcula el valor absoluto o el cuadrado de esta diferencia para capturar la magnitud total de errores en todas las instancias, dado que la diferencia entre el valor verdadero y el predicho puede ser negativa en algunos casos. Las métricas de error miden el rendimiento de predicción de un modelo de regresión en cuanto a la desviación media de sus predicciones a partir de los valores reales. Los valores de error más bajos implican que el modelo es más preciso a la hora de realizar predicciones. Una métrica de error general de 0 significa que el modelo se ajusta a los datos perfectamente.

2.4.5 Ajuste del modelo: Overfitting y Underfitting

Cuando un clasificador es entrenado se genera un modelo de predicción cuya calidad es incierta hasta su 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 efectos indeseables . El efecto de *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, ha sido incapaz de generalizar. Este efecto es causado porque el ruido o las fluctuaciones aleatorias en los datos de entrada fueron usados para el aprendizaje. Los problemas de overfitting son más probables en modelos no parametrizados y no lineales, los cuales tienen mayor flexibilidad al aprender funciones. Por lo tanto, muchos algoritmos de aprendizaje de máquina no parametrizados incluyen parámetros o técnicas para limitar y restringir los detalles que el modelo aprende. Por ejemplo, los árboles de decisión son algoritmos de aprendizaje no parametrizados muy flexibles y pueden estar sujetos al efecto de overfitting, en cuyo caso se procede a podar el árbol una vez que el aprendizaje sea suficiente, eliminando así algunos detalles innecesarios. Para solucionar este efecto se debe modificar el grado del polinomio de la función utilizada por el modelo.

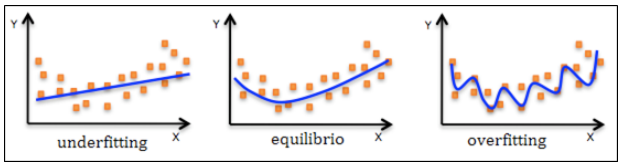


Figura 2.13: Contraste entre distintos efectos del modelo sobre los datos de entrenamiento.

Por otro lado, los modelos pueden presentar efectos de *underfitting*, funciones que no interpretan o definen los datos de entrenamiento, por lo que son incapaces de generalizar correctamente nuevos datos, este efecto es causado porque la función o modelo elegido no es el indicado 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 modelos con underfitting y overfitting, aunque esta eficiencia es muy difícil de alcanzar en la práctica. La figura 2.13 refleja el contraste entre los tres posibles estados de un modelo.

El análisis de ambos efectos, se realiza gráficamente describiendo la performance del algoritmo y la forma en que va aprendiendo a través del tiempo. Al graficar la habilidad basada en los datos de entrenamiento y en los datos de validación (testing), puede notarse que el error basado en en el último gráfico de la figura 2.13 comienza a disminuir por efectos de overfitting ya que comienza a aprender datos irrelevantes y ruidos del conjunto. En cuanto al gráfico central (modelo esperado), se puede notar que el error aumenta mientras el modelo aprender a generalizar.

Capítulo 3

Trabajos Relacionados

El enfoque propuesto básicamente es un proceso de aprendizaje de máquina que incluye una herramienta para recolectar mediciones de performance en Android (y Java), y otra herramienta para entrenar y evaluar modelos de predicción con diferentes técnicas de aprendizaje automático. Por lo tanto, este capítulo se dividirá en dos secciones para describir en la sección 3.1 un conjunto de herramientas sobre benchmarks en Android y en la sección 3.2se presentarán algunos trabajos ya realizados que llevan a cabo la predicción de atributos a través de modelos.

3.1 Herramientas de benchmarks para Android

Las herramientas de benchmarking han ido incrementando su popularidad ya que ofrecen a los usuarios la posibilidad de analizar a fondo todos los aspectos de un dispositivo android arrojando datos numéricos acerca de la potencia, eficiencia y capacidad de los mismos. La recolección y conocimiento de estas propiedades no funcionales son relevantes y necesarios al momento de determinar cuál dispositivo será el más adecuado para satisfacer la carga de trabajo esperada. A continuación se describen algunas herramientas de benchmarking.

3.1.1 Performance Monitors de Android

Es una herramienta integrada en el ambiente de desarrollo *Android Studio* y cuenta con varias sub herramientas que proveen información en tiempo real sobre la aplicación, estos datos capturados se almacenan en archivos para luego analizarlos en diferentes vistas. Pueden realizarse pruebas de rendimiento sobre dispositivos Android conectados directamente a la aplicación o simulados a través de un emulador, en ambos casos se debe tener en cuenta que *Android Device Monitor* no puede ser utilizada. *Android Monitor* utiliza la Vir-

tual Machine (VM) del dispositivo o emulador dependiendo de la versión del sistema Android.

A través de cinco vistas diferentes se accede a la información sobre la aplicación evaluada. *LogCat* muestra en detalle las excepciones emitidas por la aplicación, útil para detectar y eliminar errores que mejoren el funcionamiento de la aplicación.

Durante la ejecución de la aplicación también hay seguimiento gráfico del consumo de memoria junto con los eventos de garbage collection (*GC*) para detectar relaciones entre éstos y los puntos de latencia, el porcentaje total de uso de Central Processing Unit (CPU) (incluyendo todos los núcleos) que se utiliza en modo de usuario y modo kernel, una visión general de la performance de la interfaz incluyendo la cantidad de tiempo que le lleva al thread preparar, procesar y ejecutar los comandos gráficos, y finalmente, el seguimiento de cada solicitud de red que es realizada, al mismo tiempo que permite controlar la manera y el momento en que se llevan a cabo las transferencias de datos en la aplicación para optimizar el código subyacente de manera apropiada.

Adicionalmente, la herramienta *Android Monitor* provee funciones para examinar información relevante sobre el estado de los servicios del sistema, y permite realizar capturas y videos de la pantalla.

3.1.2 Benchit

Benchit es una librería Open Source implementada en lenguaje Java de Benchmarking para Android. El uso de esta herramienta se hace mediante un repositorio maven llamado *JitPack* (accedido a través de la URL *https://jitpack.io*) el cual contiene el proyecto *T-Spoon/Benchit*. JitPack puede utilizarse tanto en proyectos Android como en la VM de Java y provee artefactos listos para su uso como archivos jar y aar. El proyecto ha sido configurado para ejecutarse a partir de la versión 8 de Android.

Benchit es un framework rápido y sencillo ya que permite añadir benchmarks en áreas de código deseadas para determinar el tiempo 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 observar el resultado en la herramienta LogCat mostrando tres propiedades: promedio, rango y desviación estándar. La herramienta, también provee la posibilidad de realizar comparaciones entre todos o algunos benchmarks del código mostrando los resultados de manera ordenada. Esta opción es útil, por ejemplo, al momento de comparar el desempeño de varios algoritmos o sentencias que realicen la misma acción de forma diferente.

3.1.3 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 accediendo a través de la rama de la versión 0.5 ya que la rama 1.0 no funciona correctamente en Android. Existen dos alternativas de acceso al proyecto a través del repositorio maven o el repositorio git. Caliper permite obtener diferentes medidas del código Java, principalmente microbenchmarks, pero también tiene soporte para otros tipos de medidas incluyendo memoria disponible, ocupada, u otras medidas arbitrarias de dominio específico como por ejemplo el radio de compresión.

3.1.4 Java Microbenchmark Harness

Java Microbenchmark Harness (*JMH*), es una herramienta que facilita la implementación de benchmarks en forma correcta. El término correcto hace alusión a las optimizaciones que tanto la máquina virtual como el hardware subyacente aplican sobre el código durante la ejecución de los benchmarks y que no se aplican en sistemas de producción real, infiriendo en conclusiones erróneas sobre el rendimiento. *JMH* fue diseñada por los mismos desarrolladores de la máquina virtual de Java posibilitando la construcción, ejecución y análisis de benchmarks no sólo escritos en lenguaje Java sino en otros lenguajes soportados por la máquina virtual.

La librería *JMH* ofrece cinco tipos de medidas sobre el código (modos) especificados por medio de anotaciones Java.

Throughput mide el número de operaciones por segundo, un estimativo sobre la cantidad de veces por segundo que el componente podría ser ejecutado.

Average time mide el tiempo promedio que el componente requiere para ejecutarse.

Sample time mide el tiempo efectivo que el componente requirió en ejecutarse, incluyendo tiempo máximo y mínimo.

Single Shot mide el tiempo de ejecución de un simple método benchmark.

All computa y retorna todas las mediciones anteriores.

3.1.5 JMeter

JMeter es una herramienta Open Source diseñada por Apache e implementada completamente en lenguaje Java para realizar pruebas de carga y medidas de rendimiento. Apache JMeter puede utilizar para simular un gran volumen de carga en un servidor o grupo de servidores y en la red 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

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	
Java Microbenchmark Harness	Proyecto en código Java	Máquina Virtual de Java para sistemas Windows, Linux y Mac	Throughput	Benchmarks escritos en cualquier lenguaje soportado por la máquina virtual Java
			Average time	
			Sample time	
			Single Shot	
			All	
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 las herramientas de benchmarking para Android.

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. Al tratarse de un framework basado en java, el único requerimiento es la instalación de la herramienta Java Development Kit (JDK) a partir de la versión 6.

El cuadro comparativo 3.1 resalta las propiedades o medidas que pueden realizar las herramientas antes mencionadas y los componentes sobre los cuales realizan las mediciones.

3.2 Predicción de propiedades no funcionales con aprendizaje de máquina

La predicción de propiedades no funcionales ha contribuido a los arquitectos de software en la evaluación de sus sistemas durante la etapa de diseño, y guiar las decisiones respecto a los componentes que deben integrarse a la arquitectura del sistema de acuerdo a los requerimientos de calidad esperados. 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. Hutter, Xu y Hoos [4]se han referido 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 componente o 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.

Este capítulo describe trabajos que inducen al desarrollo de una herramienta capaz de generalizar cualquier característica propia del componente de ejecución como así también del entorno donde se ejecuta para la aplicación de la técnica de aprendizaje de máquina que resulte la más adecuada para la predicción. Está organizado de la siguiente manera: la sección 3.2.1 presenta tres estudios sobre algoritmos de optimización que abarcan la predicción de otro de los indicadores más importantes, como lo es la precisión y en la sección 3.2.2 presenta dos estudios de propiedades dinámicas sobre Servicios Web.

3.2.1 Predicción de precisión sobre algoritmos de optimización.

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. Mark Roberts, Adele Howe y Landon Flom [13] han construido dos 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 4726 benchmarks, problemas o instancias para ejecutarse sobre

28 algoritmos de planeamiento conocidos. Por cada algoritmo y cada problema se registra si un plan fue encontrado (éxito) a través de los valores verdadero o falso y se registra el tiempo (en segundos) requerido en completar la ejecución. Ya que cada algoritmo define su propia forma en que un resultado es exitoso o no, de forma automática obtienen las métricas de precisión porcentuales sobre la salida.

Otros autores (Mersmann; Bischl; Trautmann; Wagner; Bossek y Neumann [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 calidad de aproximación del algoritmo de búsqueda local llamado 2-opt independiente del tamaño de la instancia sobre la base de las características de los casos generados con una precisión muy alta. Se cree firmemente que debiera ser 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 de los problemas Travelling Salesman Problem (TSP).

Finalmente, el trabajo desarrollado por los autores Beveridge, Givens, Phillips y Draper [7] se analiza la precisión de tres componentes diferentes para el reconocimiento de rostros en imágenes, utilizando una técnica de regresión lineal denominada GLMM, por Generalized Linear Mixed Models.

3.2.2 Predicción sobre Servicios Web

La predicción de propiedades dinámicas de servicios Web, como su tiempo de respuesta, throughput y probabilidad de fallos, es más compleja de abordar con respecto a componentes locales ya que depende del estado 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[19]. 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 [1][16].

Capítulo 4

Enfoque y Herramientas

En este capítulo se describe el enfoque propuesto para extraer información y conocimiento de un conjunto de características inherentes a componentes de software y propiedades estáticas y dinámicas del dispositivo android de ejecución para analizar las relaciones y dependencias y predecir atributos no funcionales en base a dichas propiedades. Para el diseño se hizo énfasis en la optimización combinada de los parámetros de los algoritmos de aprendizaje automático.

El enfoque plantea llevar a cabo la predicción de propiedades no-funcionales mediante un proceso de aprendizaje de máquina a través de *i*) la recolección de indicadores o mediciones tomados a partir de la información provista de la ejecución de piezas de software, considerando atributos de componentes, atributos inherentes al problema de entrada, y atributos de la operación y resultados de la ejecución, y *ii*) la construcción de modelos como un proceso interactivo con el usuario a partir de la configuración inicial de los datos, la optimización automática de los parámetros de acuerdo a las tasas de error arrojadas por métricas de evaluación y el ajuste final del modelo a través del análisis de las curvas de aprendizaje.

Como soporte para el enfoque, se desarrollaron dos herramientas independientes entre sí y diseñadas para efectuar el objetivo y conexión de las dos fases propuestas. Por un lado, se desarrolló una herramienta denominada *Android Performance Testing and Prediction* cuyo diseño se adapta fácilmente a la implementación de cualquier dominio computacional del que se quiera obtener indicadores de desempeño. Al tratarse de un framework implementado para el sistema Android, permite obtener de manera directa los benchmarks del dispositivo de interés para el análisis. Por otro lado, se desarrolló una herramienta standalone denominada *Nekonata* diseñada para brindar soporte al uso de las funciones de cualquier librería que realice aprendizaje automático y minería de datos escritas en lenguaje Java y que consiste en dos fases, una primer etapa para la construcción del modelo a partir del conjunto fuente de

benchmarks mediante un proceso de automatización de los algoritmos en complemento de información gráfica para la colaboración interactiva del usuario y finalmente una segunda etapa de ajustes al modelo teniendo en cuenta los efectos de overfitting y underfitting consecuentes del entrenamiento.

Estas cuestiones se describen en detalle de la siguiente manera. En la sección 4.1 se enumeran algunos de los usos prácticos de los modelos incluyendo aplicaciones de la propuesta. Luego, en la sección 4.2 se profundiza sobre las distintas etapas del enfoque y flujo de trabajo. En la sección 4.3 se describen cada uno de los frameworks desarrollados, presentando la herramienta para la recolección de datos en la subsección 4.3.1 y finalmente, en la sección 4.3.2 se presenta la herramienta para la construcción de modelos evaluativos.

4.1 Aplicaciones de la propuesta

Los problemas de clase NP - Completos están presentes en la mayoría de ámbitos computacionales. Afortunadamente, en la medida que estos problemas resultan difíciles de resolver frente a los peores casos de entrada, se hace más factible resolverlos aún considerando problemas de grandes instancias.

Desafortunadamente, estos algoritmos pueden exhibir variaciones extremas de ejecución a través de las instancias con distribuciones reales, incluso, aunque la dimensión del problema se mantuviera constante, la misma instancia puede requerir dramáticamente diferentes tiempos de ejecución en función del algoritmo utilizado. Existe una escasa comprensión teórica de las causas de esta variación. Durante la última década, una cantidad considerable de trabajo ha intentado demostrar cómo utilizar las técnicas de aprendizaje automático supervisado para la construcción de modelos de regresión que proporcionen respuestas aproximadas a esta pregunta en base a los datos de rendimiento del algoritmo analizado, en otras palabras, podría creerse que es posible predecir el tiempo que requerirá un determinado algoritmo para ejecutarse bajo una entrada en particular construyendo un modelo de tiempo de ejecución del algoritmo como una función de las características específicas de cada instancia del problema.

La construcción de tales modelos conocidos como *modelos de actuación empírica* (EPM por sus siglas en inglés) ha ido creciendo y motivada debido a la utilidad que presentan frente a 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 y la configuración en una variedad de formas (un problema clásico de selección del mejor algoritmo entre un determinado conjunto); a través de Empirical Performance Model (EPM) se logra predecir el rendimiento de cada uno de estos algoritmos candidatos y mediante un análisis comparativo, seleccionar el más apropiado considerando la instancia del problema y las características del hardware.

Ajustes de parámetros y configuración automática del algoritmo EPM 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.

Generación de benchmarks fuertes Un modelo predictivo para uno o más algoritmos se puede utilizar para establecer los parámetros de los generadores de benchmarks existentes con el fin de crear instancias asociadas al algoritmo particular.

Obtener una visión general de las instancias y el rendimiento de los algoritmos EPM se puede 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 y composición 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 de entrada se modifican.

Programación 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 exacta del tiempo de respuesta que cada tarea requerirá sobre cada dispositivo con el fin de minimizar el tiempo total de secuenciación de las tareas.

Otros.

4.2 Etapas del método

El enfoque propuesto se conceptualiza como un proceso de tres fases complementarias. Este ciclo o flujo de trabajo da lugar a tres etapas bien definidas por cada dominio o escenario de estudio, desde la obtención de indicadores hasta la predicción de propiedades no funcionales en entornos de aplicación. La figura 4.1 muestra un esquema conceptual del enfoque, cuyas etapas se describen a continuación:

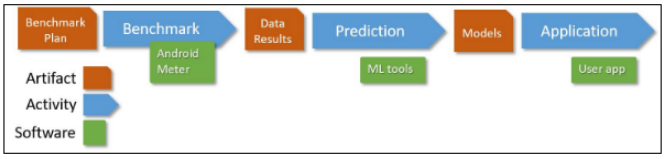


Figura 4.1: Esquema conceptual del enfoque en fases.

Testing El proceso comienza con la creación de datasets. Cada dataset pertenece a un escenario o dominio diferente del cual se extraen todas las características que podrían influir sobre el desempeño del componente. Durante la ejecución de cada servicio o pieza de software se van realizando las mediciones o métricas sobre distintos aspectos de la operación y registrando cada uno de estos benchmarks en un archivo para su posterior análisis.

Learning A partir del conjunto de benchmarks obtenidos, se aplican técnicas de aprendizaje de máquina para la extracción de conocimiento de estos datos y construir, consecuentemente, modelos predictivos que mejor se ajusten a la generalización de la información mediante un proceso de entrenamiento y evaluación de los mismos.

Predict Finalmente, se pretende utilizar estos modelos de predicción en entornos de aplicación que permitan la selección del componente más adecuado en base a un conjunto de propiedades del problema de entrada, las propiedades internas del dispositivo en el cual se llevará a cabo la ejecución, y un conjunto de restricciones que deben satisfacerse, a través de un proceso automatizado que determine al usuario la opción más favorable evitando la ejecución de cada componente.

4.3 Herramientas

El trabajo presentado conforma dos de las tres fases propuestas para el enfoque global del desarrollo. La primer fase se lleva a cabo en un framework particular para la medición de propiedades de componentes Android que será detallada en la sección 4.3.1. La segunda fase para la construcción de modelos predictivos a través de técnicas de aprendizaje de máquina se desarrolla en una segunda herramienta la cual será detallada en la sección 4.3.2.

4.3.1 Framework de medición para Android

Enfoque general

"Android Performance Testing and Prediction" es un framework diseñado para realizar mediciones de performance de componentes ejecutados bajo el sistema Android. Es una herramienta de testing que permite "correr" diferentes

piezas de software y evaluar propiedades influyentes y características del entorno de ejecución.

La herramienta cuenta con el soporte necesario para adaptarse a cualquier dominio informático y exportar las mediciones correspondientes en archivos de formato CSV, mediciones que servirán de fuente para herramientas de predicción a través de técnicas de aprendizaje de máquina.

Objetivos

El rendimiento de los componentes de ejecución (algoritmos, servicios Web, procesos ejecutándose en segundo plano, etc.) dependen de varios factores: “el contexto de ejecución” donde el componente está funcionando, los “parámetros de entrada” que requiere el componente de la operación, y su “implementación interna”. Sin embargo, al utilizar componentes de terceros, generalmente se desconoce su implementación actuando como "cajas negras" a los desarrolladores móviles, que sólo conocen sus interfaces de aplicaciones, pero no su trabajo interno.

Para elaborar un análisis del rendimiento, técnicas de aprendizaje automático sobre los datos recogidos empíricamente pueden ser utilizados para construir modelos de predicción del tiempo de ejecución del componente, como una función de los parámetros de entrada y las características específicas del contexto de ejecución: configuración de algoritmos, selección de servicios, planificación de trabajos, por citar algunos ejemplos.

Enfoque

El enfoque principal de esta herramienta está centrado en dos propiedades: el tiempo de respuesta y la precisión de los resultados.

El tiempo de respuesta refiere al tiempo total que demanda un componente en ejecutar una operación o tarea, es decir, el tiempo para responder a una solicitud con una entrada determinada. Por otro lado, la precisión es una medida que evalúa la calidad de los resultados o salida de un componente y tiene diferentes significados semánticos dependiendo de la funcionalidad requerida, por ejemplo, en problemas de clasificación, se toma el concepto de precisión como la medida estadística de la eficacia de un clasificador, la precisión está relacionada a la identificación o exclusión correctamente de una condición. En problemas de optimización, también conocido como optimalidad, la precisión es una relación entre la solución de salida obtenida y la solución óptima conocida.

Para construir un modelo de predicción del tiempo de respuesta sobre un componente particular, se deben considerar dos variables al momento de evaluar, un conjunto acerca de las características del contexto de ejecución (E,) por ejemplo, núcleos de CPU, tipo de red, etc. y otro conjunto acerca de las características de la entrada en particular (I), por ejemplo, en tamaño expresado en bytes. Por lo tanto, el modelo resulta una función R(c) que depende tanto de E como de I.

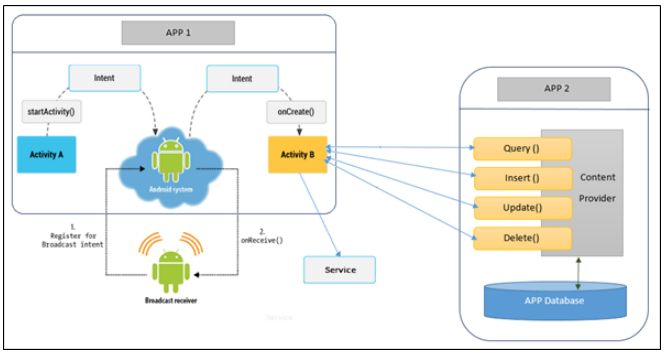


Figura 4.2: Esquema conceptual de componentes Android considerados.

Del mismo modo se construye un modelo de predicción de precisión sobre un componente, como en la mayoría de los casos esta medida no depende de las características del contexto en el que se ejecuta, el modelo respondería a una función como la siguiente:

Componentes

El análisis de rendimiento que se propone en esta herramienta se basa en entidades de software individuales de ejecución que proveen servicios a través de una interfaz específica. De aquí en más, estas entidades se denotarán componentes.

En la figura 4.2 se pueden observar los tres tipos de componentes generalizados por la herramienta para obtener los resultados de mediciones adecuados, incluyendo servicios Web y servicios específicos de la plataforma Android, para toda pieza de software remanente los componentes son tratados simplemente como 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 (como único mecanismo de comunicación entre procesos del sistema Android).

Diseño e implementación

El framework "Android-Performance-Testing-and-Prediction" está conformado por tres proyectos individuales. El proyecto "Android-Testing-Tool" constituye el modelo base para llevar a cabo el proceso de medición de propiedades de cualquier tipo de componente. Los proyectos restantes han sido implementados para desarrollar diferentes dominios de aplicación, los cuales incluyen la

Figura 4.3: Diagrama estructural de la herramienta template 'Android Testing Tool'.

dependencia al proyecto modelo. El proyecto "Evaluation-of-Face-Detection-Services" fue diseñado para obtener mediciones sobre servicios que ofrecen reconocimiento facial y el proyecto "Examples-Android-Testing-Tool" fue diseñado para dominios de problemas NP, incluyendo a problemas de la clase P y NP - Completos.

A continuación se expone en la figura 4.3 los principales factores que han servido de guía para el diseño del proyecto base "Android-Testing-Tool".

La forma de organizar diferentes modelos de evaluación se realiza a través de la creación de planes de prueba, que se configuran y posteriormente se ejecutan para obtener un archivo de formato CSV con el resultado de todas las mediciones incorporadas al plan de pruebas. La herramienta implementa por defecto un objeto llamado "TestPlan" que define una ejecución sistemática de las operaciones y métricas sobre ellos. Básicamente, un modelo de plan de pruebas se compone de un conjunto posible de componentes, un conjunto posible de objetos de entrada, y un conjunto posible de métricas para realizar las mediciones, dando la posibilidad de configurar estas propiedades a los requerimientos deseados. Una vez configurado y ejecutado el plan de pruebas, los resultados con las mediciones son almacenados en un objeto denominado ´Results' y los datos son exportados a un archivo de formato CSV para su posterior procesamiento.

La herramienta utiliza una representación simplificada de los componentes, pudiendo ser éstos servicios Web o simplemente piezas de software implementadas por el programador. Los parámetros que recibe son dos objetos del tipo Input y Output, siendo éstos, instancias de objetos que representan la entrada y salida del problema respectivamente. Cada uno de los componentes asociados a un problema específico ejecutan una operación o tarea a través de la llamada al método execute del componente. Esta operación es responsable de evaluar la ejecución de la instancia de entrada en el componente y retornar un objeto de salida o respuesta del problema. Tanto ´Input´ como ´Output´ son dos conceptos abstractos que representan instancias reales del problema. Una instancia de entrada puede encapsular no sólo los parámetros requeridos para realizar la ejecución del componente, sino la configuración de los mismos. Por otro lado, una instancia de salida encapsula la respuesta o resultado de esa operación. Durante la ejecución del plan de pruebas (ejecución de cada uno de los componentes asociados), se evalúan los indicadores que fueron configurados como métricas en el plan. Se distinguieron cuatro tipos de métricas en función del elemento de medición, determinando el parámetro que recibe:

1. Métricas globales sobre características estáticas del contexto: cualidades del entorno de ejecución que se mantienen estáticas (sin cambio) durante el

plan de pruebas, por ejemplo, modelo del dispositivo, arquitectura de la CPU, cantidad de núcleos de CPU, tamaño de memoria, etc.

2. Métricas generales sobre características de la entrada: atributos inherentes al dominio del problema, por ejemplo, en el problema de detección de rostros algunas características podrían ser el nombre de imagen, tamaño o contraste, formato de archivo, etc.

3. Métricas generales del componente: propiedades del componente como el nombre, su ubicación, etc.

4. Métricas de operación: Son medidas que actúan sobre la ejecución del componente, por lo cual distinguen en tres tipos diferentes: • Características de la salida: características del resultado de la operación como su tamaño, calidad, etc. al igual que las entidades de entrada, son inherentes al dominio del problema. En la detección de rostros, por ejemplo, la salida podría ser un vector con la ubicación de los puntos detectados y una característica de salida interesante podría ser, el tamaño de ese vector, es decir, el número de rostros detectados. • Características dinámicas del contexto: características del entorno de ejecución que pueden variar de una operación a otra, como el uso de CPU, el número de procesos en ejecución, tipo de conexión, ubicación del dispositivo, etc. • Características de desempeño: medidas de interés sobre el rendimiento que varían de una operación a otra, como el tiempo de respuesta, el consumo de batería, operaciones ejecutadas con éxito o con error, etc.

El framework de medición ha sido diseñado para soportar distintos dominios sobre los cuáles tomar las medidas deseadas. Por el momento sólo implementa dos tipos de dominios: por un lado, problemas clásicos del tipo NP para el análisis del desempeño de diferentes algoritmos de resolución y por otro lado, aplicaciones de propósito general, en este caso, aplicaciones que ofrecen reconocimiento facial para la evaluación de diferentes servicios que ofrecen esta misma funcionalidad, como ya ha sido anticipado. Los problemas de complejidad NP se contemplan bajo un proyecto individual llamado "Examples-Android-Testing-Tool" para la evaluación de desempeño de distintos algoritmos considerando dos aspectos fundamentales: tiempo (aproximación del número de pasos de ejecución que emplea el algoritmo) y espacio (aproximación de la cantidad de memoria utilizada). Los problemas implementados pertenecen tanto a la clase P como a la clase NP-Completos. Por otro lado, el objetivo de la incorporación de dominios que implican el uso de servicios remotos se basa en la idea de automatizar el testeo continuo de uno o varios servicios, en este caso, aplicaciones que ofrecen reconocimiento facial, teniendo en cuenta las características que estos pueden proveer y aquellas que el usuario de la aplicación desee considerar, de esta manera determinar el servicio más adecuado (según el contexto) para la ejecución de una tarea. Este dominio ha sido diseñado a través de un proyecto individual bajo el nombre de "Evaluation-of-Faces-Detection-Services". Este proyecto se implementó como un modelo básico y genérico sobre el proceso de detección de rostros. Tal proyecto conforma una estructura general para almacenar y acceder a todos los atributos posibles que cualquier servicio que brinde la funcionalidad de detección de rostros pudiera ofrecer. Es importante destacar algunos detalles sobre los componentes, entra-

das y métricas que fueron tomados en cuenta en el diseño del proyecto. Cada componente representa un servicio particular, cuya performance es evaluada y analizada; en base a los parámetros de configuración que acepta cada uno de estos servicios, se generan diferentes instancias de componentes posibles, un servicio que admite dos variables de configuración, por ejemplo, da lugar a cuatro tipos de componentes como producto de la combinación de opciones configurables posibles. En cuanto a las entradas del dominio, pueden añadirse al plan de pruebas a través de archivos o de manera estática especificando la ruta de cada imagen. Finalmente, fueron consideradas métricas respecto al proceso u operación (rostros correctamente detectados, rostros detectados y Tiempo de respuesta) y respecto a la imagen de entrada (contraste, entropía, formato, intensidad promedio, cantidad de rostros en la imagen, cantidad de píxeles de la imagen, tamaño y nombre). Respecto al servicio, sólo se registra el nombre del mismo.

4.3.2 Herramienta de entrenamiento y evaluación de modelos

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 automatizados y procesos que requieren la colaboración del usuario con el fin de mejorar los resultados a partir de vistas gráficas sobre el comportamiento y características de los modelos y además, para incluir en el proceso el interés y criterio del usuario.

El enfoque y objetivo de la herramienta se concluyen en dos etapas, una fase de entrenamiento de modelos y una fase de evaluación y mejora de los mismos.

En la figura 4.4 se muestran las vistas iniciales de la herramienta , presentación y configuración de datos.

4.3.2.1 Entrenamiento de modelos

En esta sección se brindará un enfoque global y expresarán las consideraciones de los conceptos relacionados al proceso de entrenamiento. Posteriormente, se describe en detalle el flujo de desarrollo del proceso. Estos conceptos han sido el eje del diseño de la herramienta. A continuación, se presenta un listado de los objetos tratados:

Librerias

Al día de hoy se han implementado bibliotecas que realizan aprendizaje automático en múltiples lenguajes de programación. A pesar de la gran variación que existe entre unos y otros, el uso de técnicas de aprendizaje de máquina se

Figura 4.4: Captura de pantalla de la herramienta: (A) Presentación, (B) Configuración de datos, (C) Menú de selección de opciones.

rige bajo los mismos conceptos: durante la etapa de entrenamiento, un conjunto de datos que servirán como base de datos del proceso y una lista de algoritmos categorizados según realicen una función de regresión o clasificación (distinción que lleva a cada biblioteca definir los tipos de atributos numéricos y categóricos), finalmente durante la etapa de evaluación, es necesario el uso de métricas o indicadores matemáticos para evaluar la calidad del clasificador. Estos principios permiten generalizar el término *librería* independizando la implementación específica de cada biblioteca añadida al sistema.

La herramienta Nekonata utiliza la biblioteca Weka, una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato y popularmente conocida ya que contiene una extensa colección de técnicas para preprocesamiento de datos y modelado.

Base de datos

Los archivos dataset usados para el entrenamiento, a menudo son vistos como una grilla de valores cuyas columnas representan cada una de las características del dominio comúnmente denominadas clases o atributos y cuyas filas denominadas instancias representan cada uno de los ejemplos del escenario de estudio. Bajo estas consideraciones, a nivel conceptual existe un objeto único como dataset que obtiene los datos a través de un proceso de parseo del archivo fuente a los objetos correspondientes de la librería utilizada, generalizando toda funcionalidad requerida mediante el acceso a la representación de los atributos e instancias.

Instancias

Tal como se adelantó anteriormente, cada dataset está formado por un conjunto de ejemplos tomados de un dominio en particular. Cada ejemplo constituye una instancia individual del problema representada por un conjunto de dos valores, el nombre del atributo y su respectivo indicador.

Modelos

La herramienta Nekonata está orientada al aprendizaje supervisado de predicción mediante funciones de regresión e incluye la técnica adaptada de agrupamiento a través de cluster como alternativa. Ambos casos fueron diseñados para incorporar los algoritmos que se deseen asegurando el correcto uso de sus funciones.

Parámetros

Los algoritmos de aprendizaje automático se rigen bajo fórmulas matemáticas que a menudo incluyen constantes o coeficientes cuyo valor incide directamente en la calidad y desempeño del clasificador. Algunos algoritmos no tienen parámetros adicionales más que los atributos del dominio, otros en cambio, tienen parámetros simples o complejos. Un parámetro simple es aquel conformado por un único valor y un parámetro complejo aquel constituido por una serie de valores, en la mayoría de casos se trata de algoritmos internos del algoritmo principal, tal es el caso de la función Kernel que utilizan los algoritmos de vectores de soporte.

Optimización

Los algoritmos pueden aplicarse utilizando los valores por defecto de los parámetros sin embargo pueden variarse conjuntamente para adaptarse mejor a los datos de entrenamiento y producir modelos predictivos más adecuados. Un proceso de optimización define rangos de valores válidos para cada uno de los parámetros admitidos y ejecuta a través de un proceso evaluativo, la combinación cruzada de cada opción, evaluando cada alternativa posible. Como resultado, se obtiene la mejor de todas las configuraciones para el algoritmo en cuestión.

La herramienta Nekonata considera la optimización de algoritmos de regresión, clusterers y funciones kernel.

Las consideraciones mencionadas anteriormente son el punto de partida en el mecanismo de aprendizaje. El proceso de entrenamiento de modelos (Figura 4.5) básicamente lleva a cabo la configuración de todos los datos requeridos para ejecutar los algoritmos clasificadores y obtener, posteriormente los modelos. Estos datos están directamente afectados por la biblioteca de aprendizaje automático que se use por lo que se dispondrá de diferentes opciones de selección variando entre una librería y otra. Esta disposición dio lugar a un flujo determinado en el orden de las actividades.

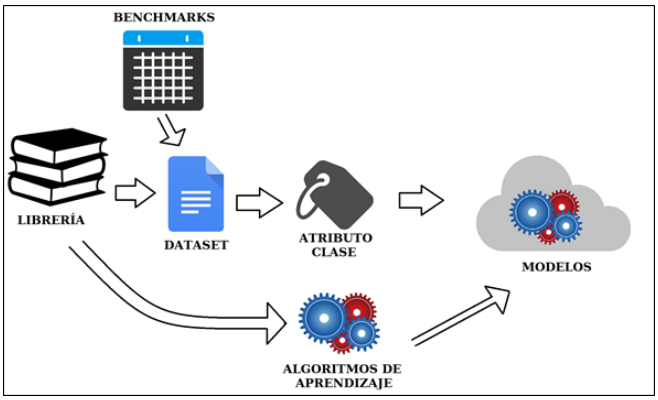


Figura 4.5: Diagrama de flujo del proceso de entrenamiento de modelos.

La selección de la librería de aprendizaje automático es el punto de partida condicionando el resto de los datos configurables. Luego, al seleccionar el archivo fuente con los benchmarks medidos en formato CSV, a través de un proceso de parseo se obtiene el archivo comúnmente llamado dataset adaptado al formato de instancias de la librería. Con este nuevo formato se extraen los atributos para seleccionar aquel que se pretenda predecir y junto con la selección de algoritmos se inicia la optimización de los mismos.

Los algoritmos de aprendizaje se habilitan al momento de elegir la librería a utilizar ofreciendo sólo aquellos que la librería dispone. La multi selección de clasificadores brinda la posibilidad al usuario de elegir los algoritmos de mayor preferencia o interés para optimizar en ese momento evitando el proceso de optimización de todos los clasificadores lo cual significaría un ahorro en el uso de los recursos computacionales.

Finalmente, como resultado de esta etapa se obtiene el conjunto de clasificadores seleccionados cuyas configuraciones son las más favorables para los datos de entrenamiento

4.3.2.2 Evaluación de modelos

En esta sección se detallarán las consideraciones y los enfoques generales del proceso de evaluación y métricas de error. También, se especificará el flujo de las actividades llevadas a cabo para calificar el desempeño del clasificador y realizar ajustes de ser necesario.

Evaluación

El dataset de origen, también denominado conjunto de datos de aprendizaje o entrenamiento, es utilizado para obtener el modelo predictivo que generalice esos datos adecuadamente. El término generalizar hace referencia a obtener una función que se ajuste a los datos en cuanto minimice el *error empírico* que

es el error producido por el algoritmo. Se asume que los datos de entrenamiento constituyen una muestra lo suficientemente representativa para aplicar un clasificador.

El proceso de evaluación, entonces, debe utilizar nuevos datos de entrada (preferentemente distintos al conjunto de entrenamiento) y predecir la salida a partir de éstos arrojando ciertamente una tasa de error en la predicción hecha por el clasificador. Existen dos métodos clásicos de evaluación en base al origen de los datos usados para la validación de modelos. El más sencillo resulta de utilizar los mismos datos que fueron usados para la fase de entrenamiento. El método restante conocido bajo el nombre de validación cruzada ('Cross validation' por su nombre en inglés) tiene una metodología más compleja.

El método de validación cruzada utiliza un coeficiente *K* de repetición y división; los datos de entrada se dividen en *K* subconjuntos, utilizando uno de ellos como dato de prueba y el resto (*K-1*) como datos de entrenamiento. El proceso es repetido durante *K* iteraciones, con cada uno de los posibles subconjunto de datos de prueba. Finalmente se calcula la media aritmética de los resultados de cada iteración para obtener un único resultado. Los resultados de la evaluación son contemplan mediante un conjunto de indicadores que se describirán más adelante. Nekonata implementa ambos métodos e incluye la funcionalidad necesaria para acceder a todos los valores predictivos.

Métricas

Las métricas usadas para la evaluación de los algoritmos simplemente son fórmulas matemáticas o composición de ellas que involucran únicamente a dos variables, los valores reales del dominio y los valores predictivos. En la herramienta se ha implementado el conjunto de indicadores más popular brindando, conjuntamente, la posibilidad de utilizar las métricas ofrecidas por cada biblioteca incorporada. También, se han definido e incorporado cuatro características globales asociadas a las métricas para brindar soporte a otras funcionalidades. Toda métrica *i*) brinda algún tipo de información, acerca del error de predicción o estadísticas o atributos de los datos; *ii*) tiene una representación particular de los indicadores, los valores pueden estar normalizados en una escala del 0 al 1, expresados en porcentajes o simplemente adaptados a la escala de los datos; *iii*) definen por su naturaleza un requerimiento minimo o maximo de su indicador y *iv*) son aplicables a un tipo de función ya sea regresión o clasificación.

Con el fin de extremar las mejoras y asegurar que el modelo resultante sea el más favorable para el conjunto de datos usados en el entrenamiento el proceso de evaluación de modelos fue dividido en dos fases. Luego de la etapa de entrenamiento, los algoritmos clasificadores optimizados son expuestos a una serie de métricas dispuestos de manera tal que permita visualizar las diferencias de desempeño entre clasificadores con la misma métrica 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 y continuar mejorando el modelo elegido a partir del análisis de los efectos de underfit y

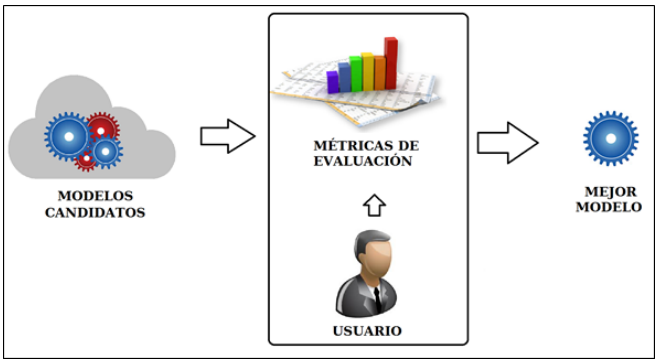


Figura 4.6: Diagrama de flujo de la fase de comparación de modelos.

overfit.

Fase 1: Comparación de modelos

La optimización de un algoritmo clasificador para obtener la configuración más apropiada para el conjunto de datos es conveniente,sin embargo, resulta más útil aplicar un proceso de optimización a un conjunto de algoritmos candidatos para luego analizar cuál de ellos es el que mejor generaliza los datos. La comparación entre clasificadores se realiza por medio de métricas que arrojan una estimación acerca del error de predicción, es decir, una medida que refleja la diferencia entre los datos reales del dominio y los valores predichos por el algoritmo. Es conveniente trasladar el análisis a la mayor cantidad de métricas posibles, 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 4.6.

Nekonata hace foco en este detalle y ofrece al usuario una vista de imágenes 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 disposición de todas las métricas de evaluación se han distribuido en imágenes dispares para agruparlas según la clase de información que representan, ya sean indicadores del error de predicción o características sobre los datos. En el primer caso también se distinguen entre aquellas métricas interpretadas a valores normalizados entre cero y uno y métricas a valores de escala del atributo. De esta forma se agrupan en conjuntos los indicadores proclives a

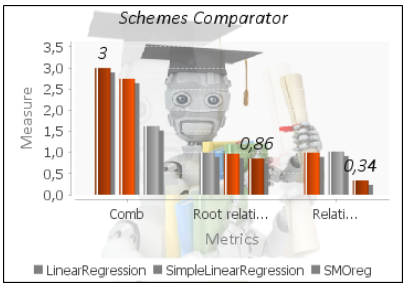


Figura 4.7: Captura de pantalla de la vista de indicadores sobre el error de predicción normalizados.

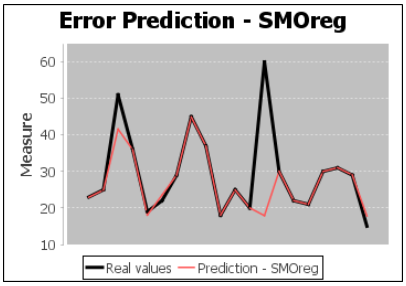


Figura 4.8: Captura de pantalla de la vista del error de predicción.

compararse mutuamente como puede observarse en la figura 4.7.

El aporte del usuario se incorporó para personificar el interés y criterio para determinar los indicadores más significativos para basar la elección del modelo más favorable. De esta manera, cada usuario puede basar su elección analizando y comparando los indicadores que a su criterio son más relevantes.

Fase 2: Ajustes al modelo

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 4.8).

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.

El análisis del error de predicción es la base para comprender la calidad del modelo construido hasta el momento aunque no es el único objetivo, ya que la vista permite extraer conocimiento de cómo se comporta el conjunto de datos

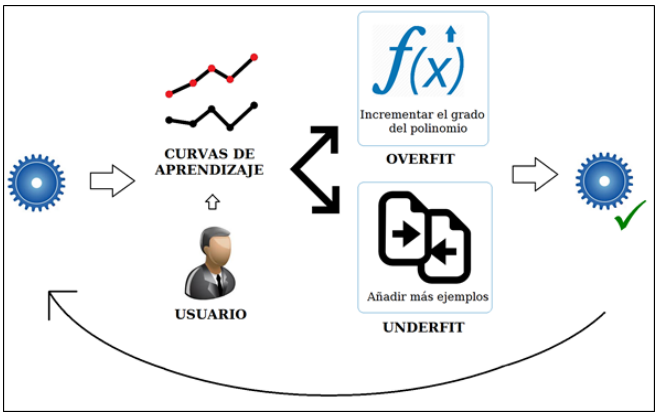


Figura 4.9: Diagrama de flujo de la fase de ajuste.

de entrenamiento, existencia de valores extremos, la variación de los valores tomados, entre otros. Es un recurso gráfico para complementar la información que se le brinda al usuario y encaminarlo a un correcto proceso de ajuste del modelo.

El flujo de desarrollo de esta fase se planteó como un proceso de carácter opcional e iterativo para el usuario. Se puede optar simplemente por almacenar el modelo elegido durante la etapa de comparación anterior o repetir las veces que se desee un procedimiento de análisis de las curvas de aprendizaje del modelo y las acciones consecuentes para reparar los posibles efectos, como se muestra en la figura 4.9.

A continuación se explicará el funcionamiento de la fase brindando una visión general del planteamiento de las curvas de aprendizaje con las respectivas consideraciones que se tuvieron en cuenta y la manera en que se ha incluido la participación del usuario para la transformación del conjunto de datos.

Curvas de aprendizaje Las curvas de aprendizaje se componen por dos líneas de puntos representadas en el mismo plano cartesiano. Ambas líneas grafican el error de predicción utilizando los dos métodos de evaluación conocidos, aplicando el conjunto de entrenamiento o el método de validación cruzada.Las curvas de aprendizaje implementadas por la herramienta son mostradas en la figura 4.15.

El procedimiento considera una cantidad de instancias determinadas, cantidad que se incrementa en un factor constante para aplicar el modelo y calcular el error cuadrático medio, es decir, la diferencia cuadrática entre el valor real del atributo y el predicho por el modelo.

En un primer paso, la herramienta toma en cuenta las primeras cinco instancias, luego las primeras diez, las primeras quince y así sucesivamente para conformar cada punto del dominio cuyo valor de ordenada es el error cuadrático, error añadido en la herramienta como parte del conjunto de métricas

consideradas. Por cuestiones de facilitar la vista se impuso un límite en los valores del dominio reducido en trescientas (300) instancias en caso de que el conjunto de entrenamiento supere dicha cantidad.

Opciones para el usuario En Nekonata se han incluido dos acciones posibles para reparar, en caso de evidenciar, los efectos de overfit o underfit del modelo. Estas dos acciones no son mutuamente excluyentes de manera que el usuario puede elegir libremente alguna o ambas acciones a la vez, sin embargo, la herramienta le recomienda la acción que debería tomar en caso de un efecto u otro. Ambas acciones realizan una transformación en la base de datos, cuando el usuario decide incorporar nuevas instancias o ejemplos del dominio introduce un archivo el cuál es parseado al formato conocido por la librería de uso y es unido al conjunto original siendo ahora, el nuevo conjunto de datos. Por otro lado, cuando el usuario decide aumentar el grado del polinomio que emplea el modelo, elige un número entre uno y cuatro para modificar este polinomio y así, añadir a la base de datos los nuevos atributos originarios del nuevo polinomio completo para ese grado.

4.3.2.3 Diseño e implementación

A pesar que la herramienta fue pensada para la predicción de componentes de android,se desarrolló como una aplicación de escritorio debido a la limitación del hardware de los dispositivos móviles para ejecutar los procesos de optimización que consumen un gran porcentaje de recursos computacionales. Esta implementación desvincula la herramienta de medición con la herramienta de entrenamiento y evaluación de modelos, permitiendo la ejecución independiente entre ambas por lo que el puente de comunicación entre ellas es la correspondencia entre la salida de la primer herramienta con la entrada de la segunda; la herramienta de medición crea archivos de formato Comma-Separated Values (CSV), los cuales serán usados en la herramienta como el conjunto de datos fuente para el entrenamiento.

Entorno y tecnologías

Para la implementación del framework Nekonata se utilizó el entorno de desarrollo Eclipse cuyo lenguaje de programación es Java. El proyecto ha sido configurado con la versión Java Development Kit (JDK) 1.7 y Java Runtime Environment (JRE) 1.8 y además, se han incorporado tecnologías de terceros para el entorno las cuáles cumplen distintos roles en la herramienta.

Para el diseño de la interfaz gráfica se utilizó mayormente la librería SWING de java incorporada en el JDK aunque también se incluyeron componentes de la librería nativa AWT. La principal ventaja de la librería SWING es brindar una interfaz adaptada a cada sistema operativo sin necesidad de cambio de código, es decir independiente a la plataforma. Complementariamente se incorporaron las dos siguientes librerías: *jgoodies-forms-1.8.0.jar* y *miglayout15-swing.jar*

Por otro lado, para la creación de vistas para el usuario se incorporó la biblioteca gráfica de Java JFREECHART que facilita la creación de varios tipos de gráficos profesionales, en el caso de la herramienta se han utilizado los gráficos de barras y lineales para representar la información requerida por el usuario. Esta biblioteca ha sido elegida por brindar objetos de alta calidad y ofrecer una extensa gama de funciones para reforzar y mejorar la información mostrada en los gráficos. Se utilizó la versión 1.0.19 de la biblioteca en complemento con la librería JCOMMON en la versión 1.0.23.

Respecto al uso de técnicas de aprendizaje de máquina, actualmente existen muchas tecnologías y frameworks que proveen esta funcionalidad para utilizar en entornos Java, sin embargo, como se ha adelantado anteriormente en la herramienta sólo se incluyó la biblioteca *Weka* en la versión 3.8 y se añadió también un paquete para la optimización de parámetros perteneciente a la misma librería denominado MULTISEARCH en su versión más actual del mes de agosto del 2016.

Diseño

Considerando todo lo anteriormente descrito, el concepto principal de la herramienta fue la extensibilidad de la misma desde todos los enfoques posibles. Esto significa que la herramienta pudiera aceptar diferentes tipos de librerías de aprendizaje de máquina, datasets, modelos de regresión, parámetros y métricas; lo que conlleva a un gran grado de abstracción de las clases que conforman la aplicación y, por lo tanto, se consideró importante la asociación de mismas, para un entendimiento mayor de parte de un nuevo desarrollador.

La finalidad de la etapa de implementación fue un buen framework de trabajo para el futuro desarrollo de modelos, librerías y métricas. Si bien la herramienta sólo trabaja con Weka actualmente y la mayoría de su funcionalidad es enteramente parser , se considera que el grado de abstracción es lo bastante elevado para poder soportar la finalidad deseada.

Aplicando los conceptos teóricos antes presentados, se exponen a continuación los paquetes y las clases principales que componen el desarrollo de Nekonata.

Databases Como ya se mencionó anteriormente se desea poder administrar, leer y escribir datasets de forma dinámica y segura. Para esto, se creó una estructura para almacenar cada individuo (instancia) con una estructura *Hashtable*, con los nombres de los atributos como claves de los valores numéricos que representan a cada uno de ellos.

Asimismo, se desarrolló un conjunto para representar los datasets ya que son estos las bases de los modelos, del aprendizaje y del framework, tomando como entrada un csv y creando la base de datos necesaria para el correcto funcionamiento de la aplicación.

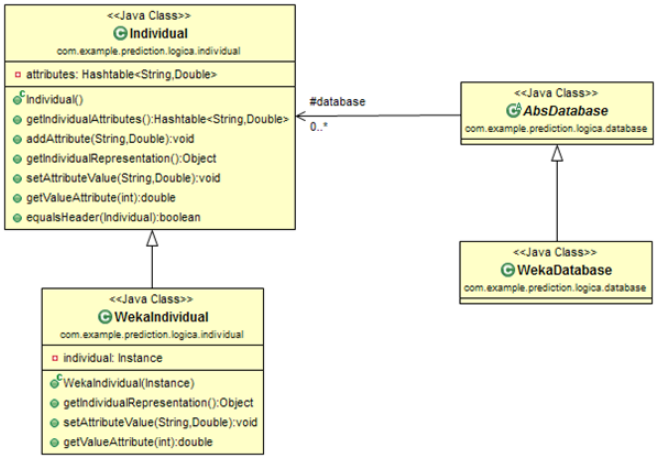


Figura 4.10: Diagrama de clases de las bases de datos implementadas

Modelos Los modelos que la herramienta presenta son de tipo regresivo pero no necesariamente son modelos intrínsecamente predictivos y por eso el framework presenta una división clara entre modelos regresivos y clasificadores (siempre considerando que estos últimos se utilizan para la regresión).

A partir de esto, implementando los métodos abstractos presentados, se pueden incluir cualquier método de clasificación requerido, siempre considerando importante que retorne un valor denso. Actualmente, los modelos implementados son los presentados en la sección 2.4.3.

Como se puede apreciar en la figura 4.12 , se realizó una abstracción intermedia considerando la librería Weka. Esto se debe principalmente al comportamiento en común que tenían dichos modelos, pero no es mandatoria dicha abstracción al agregar una nueva librería o al agregar nuevos modelos.

Ya que el diseño fue impulsado por la necesidad de abstracción e independencia de las librerías subyacentes, se considera importante la posibilidad de que en un futuro cualquier desarrollador pueda incorporar modelos implementados de forma particular.

Parámetros Para implementar los modelos, la iniciativa fue modelar el concepto teórico que los rige: funciones. Como ya fue explicado, las funciones están moldeados por variables y parámetros. Las variables son aquellos atributos modelados por la Hashtable en la clase individuo. Debido a los modelos considerados en el apartado anterior, los parámetros (o las constantes de una función) fueron modeladas en dos partes:

1. Parámetros simples: Son aquellos que sólo tienen un valor numérico y que tiene un valor único en la función. Rigen en funcionamiento del método de aprendizaje y determinan la calidad y finalidad que tendrán los mismos. Estos tipos de parametros se utilizan en todo el core de modelos

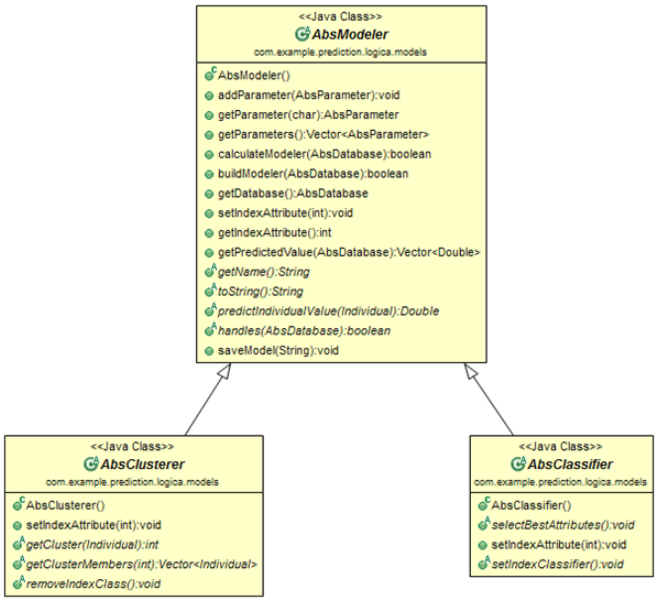


Figura 4.11: Diagrama de clases de los modelos base implementados.

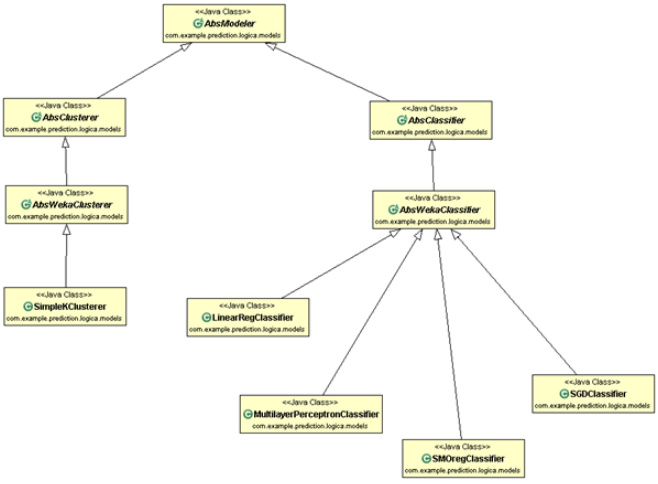


Figura 4.12: Diagrama de la relación entre los modelos implementados.

Figura 4.13: Diagrama de clases de los parametros implementados

ya que son la idea fundamental de cualquier función matemática.

2. Parámetros kernel: Son funciones matemáticas que se emplean en las Support Vector Machine (SVM). Estas funciones son las que le permiten convertir lo que sería un problema de clasificación no lineal en el espacio dimensional original, a un sencillo problema de clasificación lineal en un espacio dimensional mayor. Debido a que estas son funciones dentro de los modelos, es conveniente el modelado de las mismas de forma particular.

Observando ambos puntos, se puede apreciar una composición de parámetros ya que las funciones se conforman por parámetros simples y los parámetros kernel son funciones internas. Esto se puede observar en la figura 4.13 .

Métricas Otra parte importante de la implementación de la herramienta es la capacidad de la misma de cuantificar y valorizar los modelos obtenidos. Las mismas ya vienen implementadas por la librería Weka, pero considerando la finalidad de abstraer comportamiento, las mismas fueron parseadas y se crearon nuevas clases.

La implementación presentada surge a partir de la forma de implementación que tiene la librería weka. La misma presenta una clase que representa la relación entre los valores reales que presenta la base de datos y los valores calculados por el modelo. Sin embargo, la librería presenta la particularidad de solo calcular métricas de los modelos que la misma toma como de regresión. Así, el Simple K clusterer quedaría excluido de este grupo y, por lo tanto, no podría ser valorado. La implementación planteada permite que este modelo quede a la altura de los otros presentados y que, si se desea, se pueden crear nuevas métricas sin tener la clase *AbsEvaluation* que represente esta relación.

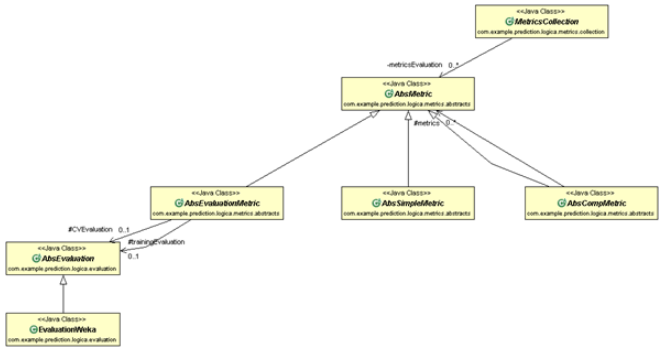


Figura 4.14: Diagrama de clases de las métricas implementadas.

Sólo sería necesario crear la métrica heredando de *AbsSimpleMetric* e implementar la forma de cálculo requerida.

Optimización El paquete que se procede a explicar es implementado principalmente considerando la función que proviene de Weka, permitiendo la prueba de varios valores para los parámetros sin necesidad de pruebas constantes, apuntando a una mejor y más rápida optimización del modelo. Puede omitirse dicha implementación si se desea agregar un nuevo modelo, pero es conveniente la explicación del mismo para futuras adaptaciones de modelos de weka que se quieran agregar o de librerías que tengan esta posibilidad también.

La idea del paquete de optimización es que, de forma transparente para el usuario de la aplicación, el modelo consiga adaptarse a la base de datos analizada. Esto permite que la aplicación Nekonata pueda adaptarse a grandes rangos de valores objetivo. Como ya se dijo anteriormente, no es un paquete necesario en la aplicación ya que los parámetros pueden ser puestos de forma fija en un modelo, pero esto restringe el rango y las posibilidades del mismo.

Los optimizadores cumplen la función de probar valores para los parámetros y se quedan con aquellos que minimizan el grado de error del modelo. Asimismo, cabe destacar que Weka proporciona dos tipos de optimizadores. Los primeros son los simples, que consideran cada parámetro del modelo desde un valor al otro. Los segundos son los kernels y prueban valores no solo para los parámetros propios de los modelos sino para los parámetros que conforman los kernels.

Cómo agregar clases nuevas Todo lo anteriormente planteado se mantiene coherente y de forma congruente gracias a la clase librería y lo que conlleva agregarla a la aplicación.

Al crear la librería, y como se ve en el diagrama de clases, se deben implementar dos métodos que devuelven dos conceptos principales. Por un lado, un objeto de tipo Database que ya se explicó anteriormente. Por el otro, un objeto

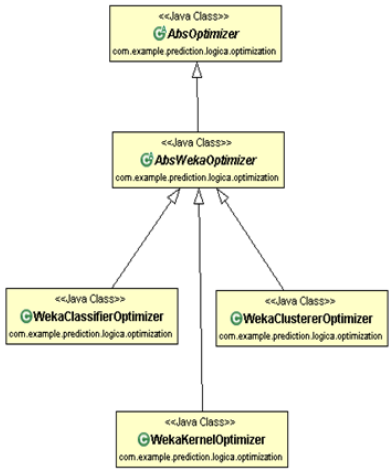


Figura 4.15: Diagrama de clases de los optimizadores implementados.

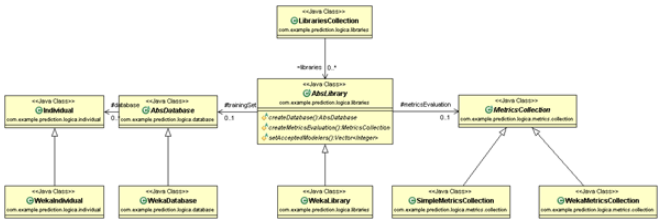


Figura 4.16: Diagrama de clases de las librerias y las relaciones con los otros objetos.

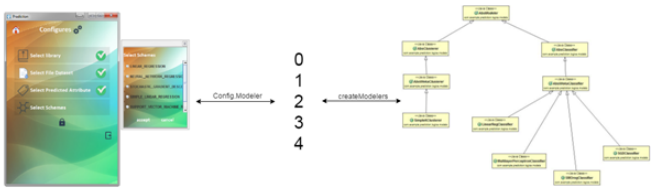


Figura 4.17: Diagrama conceptual de la configuración que presenta la herramienta

MetricsCollection. Este último representa el conjunto de métricas que sirven para valorizar los modelos.

Ya implementado se provee el conjunto *SimpleMetricsCollection* que permiten valorizar cualquier modelo, ademas del *WekaMetricsCollection* que sólo contiene la metricas basadas en el *WekaEvaluation*.

La última función que se debe implementar de forma ineludible retorna un vector de constantes. Estas constantes deben estar declaradas en la clase *Config.Modeler* con el formato:

Nombre a mostrar en la aplicación: constante

Las mismas declaran los modelos que son aceptados por la liberia. Esto se hace a modo de índice para los modelos y, al momento de utilizar la herramienta, las clases de los modelos sean creadas al momento de ser seleccionadas. Esto se permite implementando la última función:

```
public abstract Vector<AbsModeler> createModelers(Vector<Integer> selectedModels, int index);
```

Esta última función crea la relación entre la constante con las clases que se deben crear para representar a los modelos.

Capítulo 5

Evaluación

En este capítulo se describe el procedimiento de evaluación que se lleva a cabo y una descripción completa sobre los escenarios considerados y cada uno de los dataset involucrados. También, se presentan los resultados obtenidos junto con las inferencias y relaciones que se han determinado entre las técnicas y el contexto en el que se aplican.

El capítulo se organiza de la siguiente manera: en la sección 5.1 se detalla la metodología de evaluación propuesta para analizar la eficacia de los modelos de predicción, en esta se incluye información complementaria y particular que fue tomada en cuenta para llevar a cabo la evaluación mediante sub-secciones individuales, de esta manera se especifican, las métricas usadas, el formato de los resultados obtenidos, la determinación empleada para la configuración de los parámetros más fundamentales de cada técnica y las características de los dispositivos móviles usados para las pruebas.

En la sección 5.2 se presentan los escenarios de estudio considerados, brindando una descripción detallada de los dataset involucrados junto a gráficas y análisis sobre la distribución de los datos, con el fin de comprender mejor el desempeño de las tecnicas de regresion analizadas. Los modelos de predicción obtenidos son expuestos en la sección 5.3 desglosados por escenario para analizar cómo impacta en el desempeño de las técnicas, las particularidades de cada dominio. Finalmente en la sección 5.4 se obtienen las conclusiones del capítulo.

5.1 Metodología de Evaluación

Existen diversas formas que permiten evaluar la eficacia del clasificador. La calidad de los modelos será evaluada a través de seis métricas de regresión que representan de forma distinta el error de predicción. Así mismo, para un análisis más profundo sobre las distintas técnicas y su desempeño se usaron cuatro escenarios o contextos dispares entre sí para determinar, de ser posible,

las técnicas que mejor se adecuan a un escenario (por su naturaleza), o por el contrario, no aplican adecuadamente ante un escenario particular.

Cada modelo es definido por una técnica *X* (regresión lineal, red neuronal, etc.) que se aplica para predecir una propiedad *Y*, (tiempo de respuesta y precisión) de un componente *Z*. Cada uno de los contextos empleados incluyen tres o cuatro datasets, en algunos casos para individualizar el análisis hacia los componentes específicos, en otros, para separar las pruebas realizadas en móviles distintos.

En todo el conjunto de dataset, se añadió el tiempo de respuesta como propiedad a predecir, y sólo en aquellos donde el contexto lo permitía se incorporó como propiedad la optimalidad o precisión en el resultado arrojado por el componente.

5.1.1 Métricas de evaluación

Tras construir una serie de modelos de regresión diferentes, existe una gran cantidad de criterios por los cuales pueden ser evaluados y comparados. Todos los indicadores comparan los valores reales con sus estimaciones, pero lo hacen de una manera ligeramente diferente.

Mean Absolute Error (MAE) representa 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)). Mediante la cuadratura de los errores en lugar de la media y luego tomar la raíz cuadrada de la media, se llega a una medida del tamaño del error que da más peso a los errores grandes e infrecuentes que la media.

La comparación entre las métricas *RMSE* y *MAE* puede ser útil para determinar si un dataset contiene errores significativos y poco frecuentes, cuanto mayor sea la diferencia entre ambos indicadores, más inconsistente es el tamaño del error.

El coeficiente de correlación (*CC*) es analizado en conjunto con el indicador *RMSE* ya que existe una estrecha relación entre ambos. Por ejemplo, si *Correlation Coefficient (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*.

Por otro lado, complementariamente se utilizan indicadores relativos en lugar de absolutos como los descriptos anteriormente, ya que ponderan el error de predicción respecto a la variación estándar de las observaciones. De esta manera se obtienen indicadores *Relative Absolute Error (RAE)* y *Root Relative Squared Error (RRSE)* de valores entre 0 y 1 para obtener una visión de las observaciones respecto a la media.

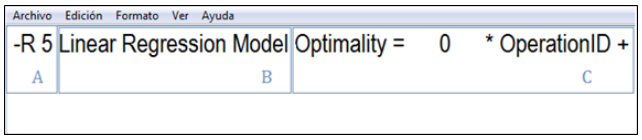


Figura 5.1: Ejemplo de resultado del modelo 'Linear Regression'.

5.1.2 Formato de los resultados

La herramienta devuelve como resultado un archivo en formato TXT almacenando, por un lado, los valores resultantes de los parámetros fundamentales de cada técnica y, por otro lado, la descripción del modelo construido. La herramienta genera un archivo por cada técnica elegida por el usuario para optimizar, en la dirección especificada en la configuración bajo el nombre de la técnica. La figura 5.1 permite una descripción gráfica de lo expresado anteriormente, donde detalla A) los parámetros, B) Nombre del modelo y C) la función definida para el atributo de predicción.

La manera en que los datos son expuestos al usuario no representa un formato compatible para cualquier herramienta sobre aprendizaje de máquina, esto se debe a que no existe actualmente, un formato estándar para almacenar los modelos, y esta discrepancia entre las herramientas disponibles exige un seteo manual de las técnicas por cada evaluación que se realice

5.1.3 Parámetros de configuraciónde las técnicas

La optimización y posterior obtención de un modelo de calidad tiene lugar a partir del establecimiento de diversos parámetros de configuración.

Estos parámetros pueden requerir distintos valores adaptándose a condiciones particulares de los datos que optimicen el resultado de la predicción. Para brindar más detalle de estos parámetros, el Cuadro 5.1 presenta la lista completa de parámetros propios de cada técnica, el valor por defecto adoptado para cada una y el rango de valores establecido con el cual se determinará el mejor valor resultante.

El proceso de optimización de las técnicas utiliza el rango de configuración establecido para iterar en pasos tomando valores intermedios y evaluando la calidad de la técnica con tales configuraciones, finalmente se contrastan entre sí y se determina la mejor configuración para un dataset y atributo a predecir particular.

Los rangos para cada técnica se establecieron tras un análisis exhaustivo de la influencia de estos valores sobre los datos. Los rangos de prueba se definieron de manera aleatoria en conjunto de fundamentos teóricos asociados a cada técnica, al mismo tiempo que el análisis de un rango ya propuesto servía como base para definir uno nuevo. Los análisis se llevaron a cabo sobre 20 archivos dataset dispuestos para esta tesis.

Por otro lado, los valores elegidos por defecto fueron tomados de los valores propuestos por la herramienta Weka, a excepción del parámetro *Training Time*

Técnica	Parámetro		Valor por Defecto	Ran Confi
Linear Regression	Ridge	R	1	-
MultiLayer Perceptron	Learning Rate	LR	0.8	0.0
	Momentum	M	0.3	0.0
	Training Time	TT	200	
	Validation setSize	VS	0	
Stochastic Gradient Descent	Learning	L	0.01	
	Lambda	R	0.05	
Simple k-cluster	Cluster Numbers	N	30	
Support Vector Machine	Complexy	C	1.0	
	Exponent	E	3.0	
	Kernel	K	Normalized PolyKernel	

Kernel	Parámetro		Valor por defecto	Ran confi
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 5.1: Parámetros de configuración de las técnicas de regresión

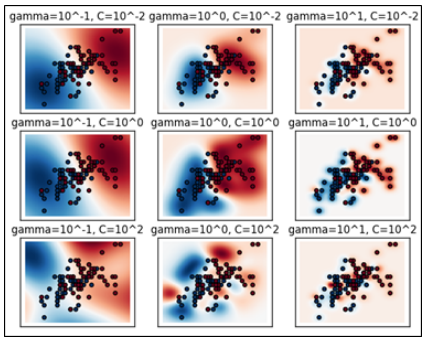


Figura 5.2: Efecto de los parámetros de la técnica SVM.

de la técnica *MultiLayer Perceptron* que fue reducido de 500 a 200 para mejorar la performance de tiempo del algoritmo.

Para el caso de la técnica *Support Vector Machine* la configuración es más compleja ya que involucra parámetros propios simples y un kernel con parámetros específicos. Para la técnica *Sequential Minimal Optimization (SMO)*, el parámetro *gamma* y *complexy* fueron analizados en conjunto con el apoyo de fundamentos teoría y empíricas. La figura 5.2 refleja el impacto que tienen los parámetros sobre la clasificación de los datos.

Las mejores opciones para configurar los parámetros se obtienen observando la diagonal en donde los valores se incrementan a la par, la primer imagen de la cuadrícula aplica una clasificación muy simple sobre los datos, dejando en evidencia un alto grado de error en las predicciones. La última imagen, en cambio, muestra un claro ejemplo de *overfit*, por lo que no resulta ser una buena opción a la hora de buscar generalización. La imagen central resulta entonces, ser la más apropiada para definir los valores por defecto para estos parámetros. Por otro lado, la optimización para el parámetro *complexy* se determinó bajo el rango 0 - 25 debido a pruebas empíricas realizadas sobre el conjunto base de dataset.

Respecto a la optimización del kernel, los cuatro algoritmos considerados fueron analizados mediante la técnica prueba y error y de esa forma, se determinaron los valores por defecto para los parámetros propios de cada uno. Del análisis se desprenden algunas conclusiones consecuentes:

Normalized Polynomial El valor por defecto para el parámetro *exponent* es 2.0 y hasta valores de 5.0 los resultados esperados son buenos.

Poly Kernel El valor por defecto para el parámetro *exponent* es 1.0. Si el valor se incrementa se obtienen mejores resultados, sin embargo para valores mayores a 5.0 el error de predicción comienza a aumentar gradualmente.

Puk El valor por defecto para el parámetro *omega* es 1.0, la modificación de este valor es indiferente a los resultados por lo que no aplica a la optimización.

	CPU Procesador / Núcleos	GPU tarjeta gráfica	Memoria RAM	Memoria interna	Memoria expansible
Lenovo K3	1.7Ghz Octa-Core ARM Cortex-A53	ARM Mali-T760 MP2	2GB LPDDR3	16GB (12GB accesible al usuario)	Hasta 32 GB microSD, microSDHC
Samsung Galaxy S3	Quad-core 1.4 GHz Cortex-A9	Mali-400MP	1 GB	16/32/64 GB	Hasta 64 GB microSD
Samsung Galaxy S4	Qualcomm APQ8064T Snapdragon 600 1.9Ghz / 4	PowerVR SGX 544MP / Adreno 320	2 GB LPDDR3	16/32/64 GB	Hasta 64 GB microSD
Samsung Galaxy S5	Qualcomm MSM8974AC Snapdragon 801 2.5Ghz / 4	Adreno 330	2 GB	32 GB	Hasta 128 GB microSD
Samsung Galaxy S6	2.1Ghz Samsung Exynos /ocho núcleos	Mali-T760	3 GB LPDDR4	32/64/128 GB	Ninguna

Cuadro 5.2: Especificaciones de los dispositivos móviles utilizados.

Respecto a *sigma* se adopta el valor 0.01 ya que el incremento en este valor no produce buenos resultados.

RBF Partiendo de un valor *gamma* de 0.01 se obtienen buenos resultados, e incluso incrementando este valor a razón de 0.01 los resultados tienden a mejorar.

5.1.4 Características de los dispositivos móviles usados

En la mayoría de las pruebas se han utilizado dos dispositivos móviles fundamentales, el modelo K3 Note de la marca Lenovo y el modelo Galaxy S3 de la marca Samsung principalmente por las características del procesador y memoria interna. El móvil lenovo es superior debido a sus 8 núcleos de procesamiento frente a los 4 núcleos del Samsung y a una memoria RAM con el doble de capacidad.

También se utilizaron tres modelos más sólo para el escenario del problema del viajante. Todas las especificaciones son expuestas en el cuadro 5.2.

5.2 Escenarios

Definidas las métricas, la metodología de evaluación y los parámetros de configuración, en la presente sección se detallarán los escenarios o dominios estudiados a partir de una descripción detallada de los dataset formados, cada uno de los componentes involucrados y los atributos de entrada considerados

formando un contexto particular para la predicción. Por otro lado, se analizará el comportamiento o formato que adquieren los datos sobre los atributos predichos para comprender el desempeño de las diferentes técnicas sobre los mismos y la manera en que éstos adecuan sus predicciones.

5.2.1 Escenario 1: Algoritmos para el problema del viajante

En el campo de la teoría de complejidad computacional, el problema del viajante o TSP por sus siglas en inglés, es tratado como 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.

Con el fin de garantizar instancias variadas del problema, (diferentes cantidades de ciudades, distintas representaciones y pesos), se incorporó a la herramienta una librería llamada TSPLib con múltiples archivos de instancias del problema. La librería cuenta con 111 archivos de formato TSP, y 32 archivos de formato OPT.TOUR que contienen el recorrido óptimo del archivo que referencian. Los ejemplos oscilan desde 14 y hasta 18512 ciudades con un sólo ejemplo extremo de 85900, y utilizan cuatro funciones de cálculo de la distancia entre ciudades: distancia euclidiana, distancia geométrica, distancia pseudo euclidiana y función techo. Para la obtención de las mediciones correspondientes, se realizó una preselección 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.

Los dataset para este dominio han sido formados tras la ejecución de seis algoritmos de resolución sobre 32 entradas del problema pertenecientes a la librería TSPLIB dando lugar a un total de 192 instancias por dataset; las pruebas fueron llevadas a cabo sobre el dispositivo móvil Samsung Galaxy S4, y repetidas en los dispositivos S5 y S6, obteniendo así, tres dataset análogos que incluyen como propiedades 7 atributos, Id de operación, nombre de componente (algoritmo), entrada del problema (nombre de archivo), exactitud y precisión en el resultado, tiempo de respuesta y valor solución. Todos los dataset han sido almacenados en archivos con formato Comma-Separated Values (CSV) bajo el nombre ''Sx-results' según el modelo de dispositivo utilizado. Los componentes involucrados incluyen el algoritmo del vecino más cercano, Programación Lineal, Mejor ajuste, Kruskal, Prim, y el algoritmo de Transformación Local. El algoritmo de Backtracking fue descartado de la lista debido al gran consumo de memoria en los dispositivos imposibilitando el desarrollo de las pruebas.

En cuanto a la precisión y exactitud de las respuestas se consideraron tres indicadores que toman en cuenta las repeticiones en la respuesta y la totalidad o falta de los vértices incluidos. El indicador TP (*true positive*), para los vértices

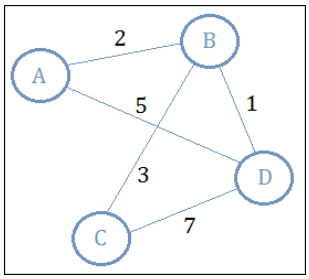


Figura 5.3: Ejemplo de grafo para el problema del viajante.

incluidos en la solución, FP(*false positive*), para las repeticiones de vértices y FN (*False Negative*) para los vértices no incorporados en la solución. Por lo tanto, se consideró:

$$Precisión = \frac{TP}{FP + TP}$$

$$Exactitud = \frac{TP}{FP + TP + FN}$$

A modo de ejemplo, se presenta en la figura 5.3 un grafo con comienzo en la ciudad A. Suponiendo que un algoritmo de resolución arroja el siguiente camino solución: {A, B, D, A}, se obtiene una precisión del 100 % al tener un valor de True Positive (TP) = 3 y un valor de False Positive (FP) = 0 ya que no hay vértices repetidos en la solución, sin embargo, la solución no es correcta ya que no incluye la totalidad de vértices del conjunto (excluye al vértice C). En cambio, al calcular la exactitud se obtiene un valor del 66 % al tener un valor de TP = 4, FP = 1 y False Negative (FN) = 1 brindando un concepto más realista.

El uso del indicador de precisión o el de exactitud dependerá del contexto en el que se aplique.

El objetivo de este escenario es la creación de modelos capaces de predecir el tiempo de latencia de cada algoritmo de resolución frente a distintas características de las entradas, y modelos para predecir la precisión en los caminos retornados. En la figura 5.4 expuesta a continuación se muestra el comportamiento del conjunto de datos de entrenamiento ejecutado en el dispositivo Samsung Galaxy S4; ambos dataset restantes no son expuestos ya que presentan mínimas diferencias.

A primera vista la figura 5.4 permite observar una gran concentración de datos sobre los extremos para los valores de precisión y tiempo de respuesta. En el primer caso, aproximadamente un 94 % de los datos toman el valor 1, y en el segundo caso, un 88 % de los datos toman valores entre 0 y 6.77 segundos. Este comportamiento uniforme y poco distribuido de los datos obstaculiza el desarrollo de un buen modelo predictivo.

Respecto al atributo *accuracy*, si bien presenta más dispersión entre los datos, aproximadamente un 82 % circunda alrededor del valor 1 y un 15 % sobre valores cercanos a 0.008.

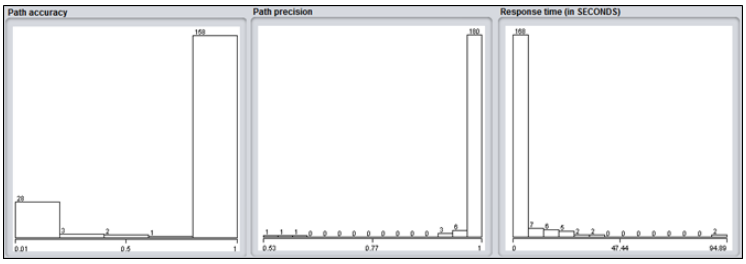


Figura 5.4: Comportamiento del dataset para el escenario del problema del viajante.

5.2.2 Escenario 2: Servicios para detección de rostros

Hoy en día muchas soluciones para detección de rostro se proporcionan como servicios Web, que aunque no son tan rápidos y usables sin conexión a Internet como las bibliotecas de software, resultan generalmente más precisos y proporcionan características muy peculiares del rostro. 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 la Central Processing Unit (CPU), conexión de red). Por lo tanto el objetivo de este escenario es crear modelos para predecir el tiempo de respuesta a partir de características variables de la imagen de entrada y de los parámetros de configuración del componente en cuestión.

Los dataset para este dominio han sido formados tras la ejecución de diferentes componentes para detección de rostros en imágenes en un único dispositivo móvil (Lenovo K3 Note), por tal motivo no se incluyen en los archivos características propias del móvil ya que significan valores constantes en la predicción. Los componentes involucrados incluyen un servicio o proceso Android (Google Play Services (GMS) face detector) y tres servicios Web (FaceRect API, Sky Biometry API y Microsoft Face API). Estos cuatro componentes serán evaluados a partir de subconjunto de 290 imágenes pertenecientes al dataset llamado FDDB [8] con características muy variadas. Cada servicio es llamado especificando una imagen y un conjunto de parámetros booleanos (a través del valor 0 y 1) que determinan las funciones requeridas del algoritmo determinado. En los cuatro servicios, el tiempo de respuesta es una variable continua y registrada en milisegundos.

A continuación se detallan las principales características de los servicios y los respectivos dataset creados para cada uno:

GMS La compañía Google ha implementado una colección extensa y variada de aplicaciones para Android. Entre ellas, ofrece el paquete 'com.google.android.gms.vision' el cual proporciona una funcionalidad común para trabajar con detectores de objetos visuales. En la figura 5.5se muestra la forma en que el objeto *GoogleAPIClient* proporciona una interfaz para conectar y hacer llamadas a cualquiera de los servicios de Google Play disponibles tales como Google Play Games,

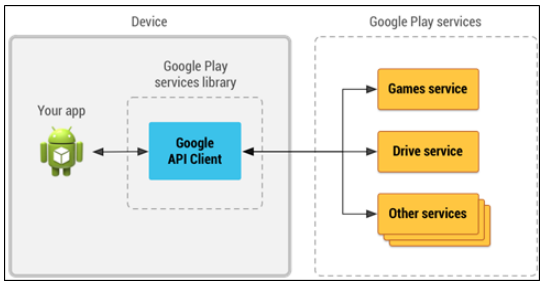


Figura 5.5: Esquema conceptual para el uso de servicios de Google Play Services.

Google Drive, Google Maps, etc

El algoritmo permite setear tres parámetros diferentes (*allLandmarks* , *allClassifications* , *accurateMode*) dando lugar a 8 configuraciones posibles por imagen. Para el servicio GMS se utilizaron 290 imágenes originando un dataset de 2320 instancias.

Archivo Dataset: *ResponseTime - Google Play Service face detector.csv*.

FaceRect API y Sky Biometry API Los servicios web *SkyBiometry* y *FaceRect* son consumidos directamente desde la aplicación online *Mashape*, la cual ofrece una gran variedad de aplicaciones, incluida una colección de Aplicaciones sobre detección y reconocimiento de rostros. SkyBiometry y FaceRect son aplicaciones que utilizan interfaz REST, es decir, los métodos son llamados a través de internet usando los métodos Hypertext Transfer Protocol (HTTP) standard como GET y POST a las direcciones correspondientes. Dependiendo de los parámetros especificados en el request, el servidor puede generar la respuesta tanto en formato JSON como eXtensible Markup Language (XML). Adicionalmente, se utilizó el servicio GMS Visión de Google a partir de la librería correspondiente. Los tres servicios son descritos en detalle en el apéndice B.

El algoritmo de la aplicación FaceRect permite setear sólo una opción (*features*) por imagen, de modo que se construyó un dataset con sólo dos propiedades, incluyendo el tiempo de respuesta y un total de 54 instancias.

Archivo Dataset: *ResponseTime - FaceRect API.csv*.

Por otro lado, el algoritmo de Sky Biometry permite setear siete parámetros diferentes dando lugar a 128 configuraciones posibles por imagen (*aggressive*, *gender*, *glasses*, *smiling*, *mood*, *age*, *eyes*). Para el servicio Sky Biometry se utilizaron 30 imágenes originando un dataset de 3840 instancias y se incluyen 8 propiedades, además de los parámetros configurables del tipo binario se añade el tiempo de respuesta.

Archivo Dataset: *ResponseTime - SkyBiometry API.csv*.

Microsoft Face API Face API es un servicio web en la nube que provee los algoritmos de detección y reconocimiento de rostros más avanzados. El servicio es consumido a través del sitio de Microsoft y las respuestas son retornadas

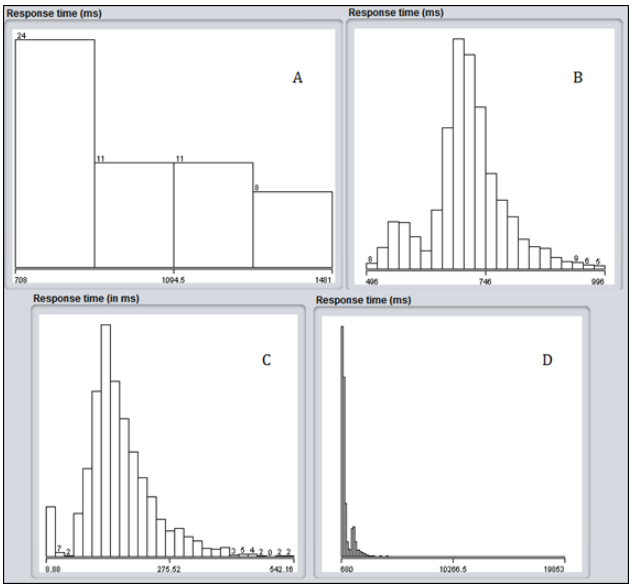


Figura 5.6: Comportamiento del atributo 'Tiempo de respuesta' de los servicios de detección de rostros.

en formato JSON. El algoritmo permite setear seis propiedades (*landmarks*, *age*, *gender*, *facialHair*, *smile* y *headPos*). El dataset está formado por 1826 instancias y siete propiedades, además de los parámetros configurables como atributos binarios se incluye el tiempo de respuesta.

Archivo Dataset: *ResponseTime - Microsoft Face API.csv*.

En la figura 5.6 que se muestra a continuación se expone el comportamiento de los datos del atributo *Tiempo de respuesta* para cada uno de los servicios, en referencia con la imagen A) FaceRect API, B) Microsoft Face API, C) Google Play Service y D) Sky Biometry API.

El servicio Sky Biometry presenta entre sus datos un 5% de valores infrecuentes y en esos casos valores extremos, siendo el único servicio en presentar estas características.

El servicio de Microsoft Face arrojó valores de tiempo de respuesta uniformemente distribuidos en el intervalo tomando la forma de campana de Gauss, los datos se distribuyen simétricamente sobre el intervalo, es decir, no hay presencia de sesgos hacia la izquierda o hacia la derecha. Google Play también tiene una distribución de Gauss con sesgo hacia la izquierda. Puede observarse que presenta un intervalo muy amplio, de modo que las propiedades configurables del algoritmo incrementan considerablemente el tiempo de operación. Sin embargo, al contrastarlo con el servicio de FaceRect, el mínimo de tiempo requerido por este servicio es mucho mayor que el tiempo máximo arrojado por el de Google Play, aún así cualquier análisis sobre el servicio FaceRect puede ser precipitado, ya que se evaluó con un bajo número de instancias.

5.2.3 Escenario 3: Problema de la mochila

Al igual que el problema del agente viajero, el problema de la mochila o 'Knapsack problem' por su nombre en inglés, es un problema NP-Completo de optimización combinatoria, es decir, se busca la mejor solución entre un conjunto finito de posibles soluciones al problema. El problema simula la colocación de ítems u objetos en una mochila de tal forma que se maximice el valor de los ítems que contiene con la restricción de no superar el peso (o volumen) máximo que puede soportar la mochila.

La evaluación de este dominio se llevó a cabo en dos modelos distintos de dispositivos móviles (Samsung Galaxy S3 y Lenovo K3 Note) cuyo objetivo es crear modelos de predicción del tiempo de respuesta y la optimalidad del componente, es decir, la relación entre la solución encontrada y la solución óptima retornando un valor entre cero y uno. Para este dominio se implementaron dos algoritmos de resolución al problema como componentes, el algoritmo greedy de complejidad O(n2 log(n)) y el algoritmo backtracking que encuentra la solución óptima pero en tiempo exponencial O(2n). Ambos componentes fueron implementados puramente en Java. Las instancias del problema se obtuvieron aleatoriamente de las cuales se consideraron los siguientes atributos:

Primero ID de Operación.

'KnapsackWeightRatio': teniendo en cuenta W como el peso límite de la mochila y bi los pesos individuales de cada ítem, se calcula bajo la fórmula $KWR = W/b_1 + b_2 + ... + b_n$ arrojando valores entre 0 y 1

Número de ítems en la mochila denotado como N.

Tiempo de respuesta registrado en segundos.

Segundo Nombre de Componente.

Optimalidad o precisión de los componentes siempre medida respecto a la solución brindada por el algoritmo Backtracking.

Valor solución al problema.

Respecto a las características propias del dispositivo de ejecución, sólo se tomó en cuenta la frecuencia de CPU ya que ambos algoritmos son single threads y corren en CPU, con la suposición que la frecuencia de CPU incide sobre la predicción.

Se formaron cuatro dataset para este escenario, dos por cada uno de los dispositivos móviles evaluados. Bajo el nombre *Optimality and Response time - dispositivo XXX* se crearon datasets de 720 instancias (360 para cada uno de los componentes) y siete propiedades como se describieron anteriormente (Grupo A y B). Las entradas del problema fueron consideradas desde un mínimo de 5 hasta un tamaño máximo de 24 ítems límite impuesto debido a la complejidad

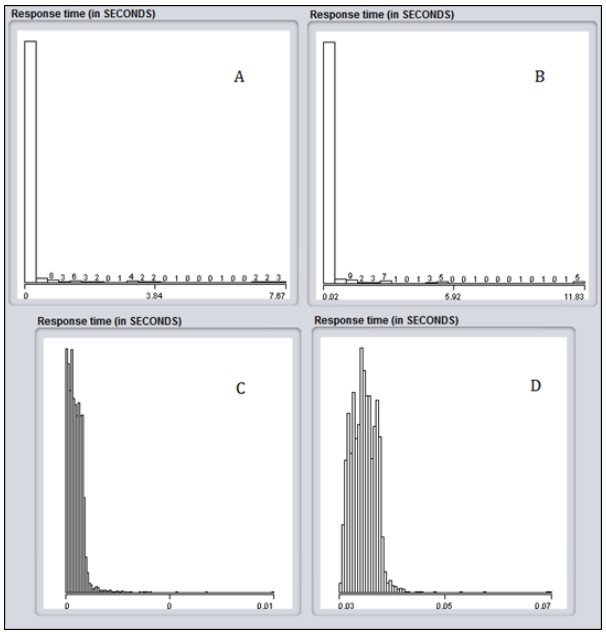


Figura 5.7: Comportamiento del atributo 'Tiempo de respuesta' para el problema de la mochila.

del algoritmo backtracking y su correspondiente latencia exponencial. Por cada ítem se evaluaron 18 entradas al problema generadas aleatoriamente a partir de cuatro valores: cantidad total de ítems, máximo valor y peso de un ítem, y finalmente el peso o capacidad de la mochila.

Por otro lado, bajo el nombre *Response time - Greedy algorithm - dispositivo XXX*, se crearon datasets de 4704 instancias y cuatro propiedades (Grupo A) con entradas del problema de tamaño mínimo de 5 ítems, e incrementalmente hasta un máximo de 200, con 24 entradas dedicadas a cada número de ítem generadas también aleatoriamente. A continuación en lafigura 5.7 se muestra el comportamiento de los datos del atributo *Tiempo de respuesta*, en referencia a la imagen, A) 'Opt & Resp Lenovo K3', B) 'Opt & Resp Samsung Galaxy S3', C) Greedy Lenovo K3 y D) Greedy Samsung S3.

En la ejecución de los algoritmos de Backtracking y Greedy en ambos dispositivos (Imagen A y B) se observa una concentración aproximada del 93% de los datos en un intervalo muy pequeño y un 5% de los datos se encuentran en el intervalo mayor a 1 segundo, por lo cual podría concluirse que son valores atípicos o poco frecuentes. Este comportamiento muy homogéneo de los datos, dificultará la construcción de un modelo predictivo de alta calidad. Por otro lado, los dataset que incluyen al algoritmo Greedy, presentan más dispersión de los datos cuya distribución toma forma exponencial (Imagen C) y forma de campana de Gauss con sesgo hacia la izquierda (Imagen D). La heterogeneidad en los datos puede posibilitar la construcción de un buen modelo. Respecto al

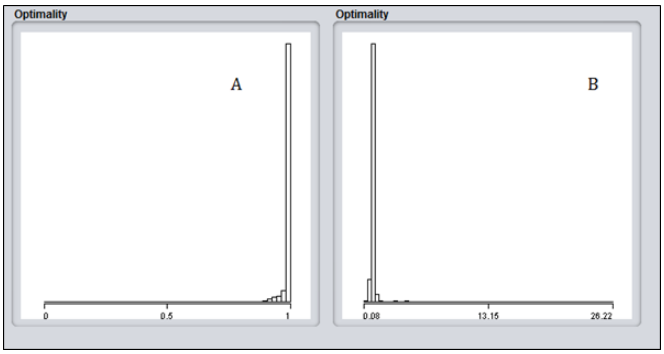


Figura 5.8: Comportamiento del atributo 'Optimalidad' para el problema de la mochila

análisis de los valores registrados, se puede observar que las mismas pruebas ejecutadas en el dispositivo Samsung Galaxy S3 consumen notoriamente más tiempo de operación.

Por otro lado, se expone también en la figura 5.8 el comportamiento para el atributo optimalidad de los dataset que incluyen ambos algoritmos. (Backtracking y Greedy).

Las pruebas ejecutadas en el modelo K3 de Lenovo arrojaron valores de optimalidad muy altos, un 80% de los datos con optimalidad del 100%, y menos del 5% del total adquiere valores por debajo de 0.95. Esta cifra incluye valores extremos que representan el 1% de la totalidad de observaciones. Por lo tanto, la solución brindada por el algoritmo Greedy es generalmente óptima. Por otro lado, las pruebas ejecutadas en el dispositivo Samsung Galaxy arrojaron un grado de optimalidad del 69%, y sólo un 12% de los datos corresponden a valores entre 0.9 y 1.0, sin incluir este último. Se presume que el dataset presenta algunas anomalías ya que un 10% de los datos tienen valores superiores a 1.0, contraponiendo el concepto de optimalidad.

5.2.4 Escenario 4: Multiplicación de matrices

El producto de matrices corresponde a problemas de clase P ya que a diferencia de los problemas NP, su complejidad es polinómica. El producto entre matrices no es conmutativo, depende del orden de las matrices intervinientes y su multiplicación sólo es posible si el número de filas de la primera matriz es igual al número de columnas de la segunda.

Los dataset para este dominio han sido formados tras la ejecución de diferentes algoritmos de multiplicación de matrices, con tamaños variados de matrices sobre 2 modelos de dispositivos móviles, Lenovo K3 y Samsung Galaxy S3. Lo interesante de este dataset, es que los algoritmos fueron implementados con diferentes librerías y diseñados para ser ejecutados por diferente elementos de hardware (CPU, Graphics Processor Unit (GPU), etc).

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

Matrix Multiplication implementación simple desarrollada puramente en lenguaje Java.

Matrix Multiplication Multi Thread corresponde a la misma versión del componente anterior paralelizada en ocho threads de ejecución.

Matrix Multiplication Render Script es una versión implementada con RenderScript , de modo que si el dispositivo tiene un GPU compatible con RenderScript, éste lo ejecutará. Por tal razón el tiempo de operación es mucho más rápida.

Native Matrix Multiplication es una versión que usa Java Native Interface (JNI) para que la multiplicación se realice sobre código nativo desarrollado en lenguaje C++, el cual se compila específicamente según cada arquitectura de CPU (Armeabi, x86, etc).

Matrix Multiplication with Eigen versión que usa JNI para ejecutar la multiplicación de matrices provista por la librería nativa Eigen.

Matrix Multiplication with OpenCV versión que usa JNI para ejecutar la multiplicación a través de la librería OpenCV .

Las entradas del problema utilizadas para la evaluación fueron generadas aleatoriamente a partir de las dimensiones (tamaño) de las matrices involucradas a partir del número de filas y columnas de la primer matriz denominada matriz A y el número de columnas de la segunda matriz denominada matriz B, esta distinción es importante teniendo en cuenta la propiedad de no conmutatividad del producto de matrices. Adicionalmente, ambas matrices pueden ser configuradas externamente. De cada problema, las propiedades que se tomaron en cuenta fueron: el número de columnas de la primer matriz (*AColumn*), el número de filas y el de columnas de la segunda matriz (*ARow* y *BColumn*), la complejidad temporal de la operación definida como O(AColumn x ARow x BColumn), el nombre del componente y el tiempo de respuesta de la operación registrada en segundos. No fueron incluidos atributos relativos al dispositivo de ejecución, sin embargo podrían considerarse propiedades como número de cores de CPU, cores de GPU, frecuencia, etc. ya que se piensa son buenos predictores de tiempo para algunos algoritmos. Los dataset bajo el nombre *Response time NxNxN - Samsung Galaxy S3* y *Response time NxNxN - Lenovo K3 Note* se componen de 894 instancias lo cual comprende un total de 149 entradas por cada uno de los seis componentes. Por otro lado, los dataset bajo el nombre de *Response time NxMxL - Sin RenderScript component - Samsung Galaxy S3* y *Response time NxMxL - Sin RenderScript component - Lenovo K3 Note* excluyen, la versión implementada con RenderScript y comprenden 745 instancias.

A continuación en la figura 5.9 se muestra el comportamiento del atributo *Tiempo de respuesta* de los dataset formados para este dominio, en referencia a

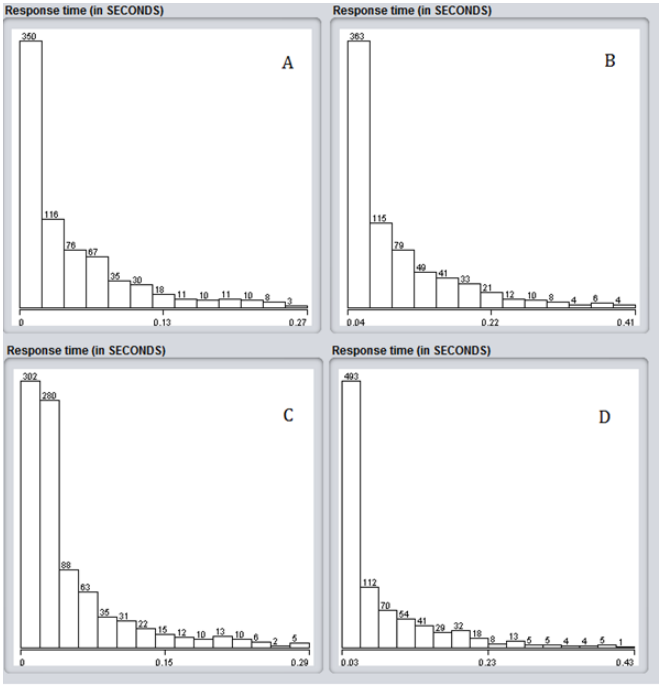


Figura 5.9: Comportamiento del atributo 'Tiempo de respuesta' para el problema de la multiplicación de matrices.

la imagen, A) Sin componente RenderScript en Lenovo K3, B) Sin componente RenderScript en Samsung Galaxy S3, C) Todos los componentes en Lenovo K3 y D) Todos los componentes en Samsung Galaxy S3.

En todos los casos puede observarse una distribución exponencial de los datos incluyendo valores extremos o infrecuentes. Respecto a los valores de tiempo registrados, puede notarse a simple vista que las pruebas ejecutadas en el dispositivo Lenovo K3 consumen menor tiempo en contraste con el dispositivo Samsung Galaxy S3.

Lo interesante de este escenario es que permite la creación de modelos simples, por ejemplo, a partir de la técnica Linear Regression que arroje los mismos o mejores resultados que otras técnicas más complejas ya que entre los términos de la función de predicción se expresa la complejidad del algoritmo de matrices. Incluso, se podrían usar técnicas específicas para cada componentes evaluando y analizando el impacto sobre la predicción. Por ejemplo, para el componente *Multiplicación MultiThread* en la técnica Linear Regression puede considerarse el número de threads; para el componente con RenderScript puede considerarse alguna característica propia del GPU donde se ejecuta si implica alguna mejora en los resultados predictivos.

5.3 Resultados y discusión

En esta sección final de la evaluación se agrupan los resultados para contrastar los escenarios entre sí con el fin de determinar relaciones entre las técnicas, el contexto o dominio en el que se aplican y el desempeño sobre los atributos que predicen, esto es, si la técnica se ajusta más adecuadamente a un tipo de atributo o a otro, a modo de inferir las conclusiones pertinentes. Además, se analizan detalles sobre la performance del proceso de optimización de las técnicas respecto al tiempo de cómputo que insumen y su relación con el tamaño del dataset que procesa.

5.3.1 Resultados para el problema del viajante.

A partir de la figura 5.3 se desprenden las conclusiones con respecto al primer escenario. Como primer punto tanto el modelo de *MultiLayer Perceptron* como el de Support Vector Machine (SVM) son, claramente, los mejores en cuanto a adaptación al modelo. Ambos presentan una correlación con los datos de 1 (*CC*), lo que representa la exactitud de los datos calculado con respecto a los reales. Como ya se explicó en la sección2.4.5, esto puede parecernos lo más óptimo, pero observando las métricas y las curvas de errores, ambos modelos presentan un problema de overfitting.

Teniendo esto en cuenta, se descartan ambos como los mejores modelos para este tipo de problema. A su vez, se descarta el *K-mean Clusterer* ya que los niveles de correlación son los mas bajos y los errores los más altos.

En todos los escenarios presentados, se optó por el caso intermedio de adaptación, considerando que el mismo es el mejor obtenido evitando casos de overfitting o underfitting. Este se escogió teniendo en cuenta los valores de CC intermedios, con un error MAE aceptable considerando los valores propios y errores promedios bajos. Así, en este caso, el más eficiente es el *Linear Regression.* A continuación, se presenta el comportamiento de este último con respecto a los valores de referencia.

Con el mismo método de analisis se analizó el dataset TSPlib.csv considerando la precisión del algoritmo.

Así, se puede apreciar que los que presentan una mejor adaptación en este caso siguen siendo el MultiLayer Perceptron (MLP) y el SMO. Aun así, considerando el analisis antes hecho, esta vez el mejor aloritmo es el Stochastic Gradient Descendent (SGD).

Si tenemos en cuenta lo visto en la sección 2.4.3.1, el mejor algoritmo teórico que mejor se adapta a el problem del viajante es el *LinearRegression* es sus dos implementaciones (Ridge y SGD).

5.3.1.1 Resultados para los servicios de detección de rostros.

Durante el proceso de formación de los dataset, se consideró en primer medida, la incorporación de variables sobre la imagen, como el tamaño, la intensidad de color, entre otras. Sin embargo, fueron descartadas en la versión final ya que

	Modelo de Tiempo de respuesta - TSPLIB S4					
	CC	MAE	RAE	RRSE	RMSE	COMB
Linear Regression	0.72	4.35	0.8	0.69	7.72	1.77
MultiLayer Perceptron	1	0.62	0.11	0.09	1.02	0.21
Stochastic gradient descent	0.65	3.39	0.62	0.85	9.48	1.83
K-means Clusterer	0.63	4.24	0.78	0.78	8.67	1.93
Support Vector Machine	1	0.09	0.02	0.01	0.1	0.02

	Modelo de Tiempo de respuesta - TSPLIB S5					
Linear Regression	0.72	2.12	0.81	0.69	3.69	1.77
MultiLayer Perceptron	1	0.15	0.06	0.04	0.2	0.09
Stochastic gradient descent	0.65	1.6	0.61	0.85	4.54	1.81
K-means Clusterer	0.58	2.135	0.81	0.82	4.36	2.05
Support Vector Machine	1	0.05	0.02	0.01	0.05	0.03

	Modelo de Tiempo de respuesta - TSPLIB S6					
Linear Regression	0.71	1.22	0.85	0.71	2.06	1.86
MultiLayer Perceptron	1	0.1	0.07	0.04	0.13	0.11
Stochastic gradient descent	0.62	0.86	0.6	0.89	2.58	1.87
K-means Clusterer	0.65	1.01	0.71	0.76	2.2	1.81
Support Vector Machine	1	0.02	0.02	0.01	0.02	0.02

Cuadro 5.3: Resultados del atributo Tiempo de respuesta para el escenario 'problema del viajante'.

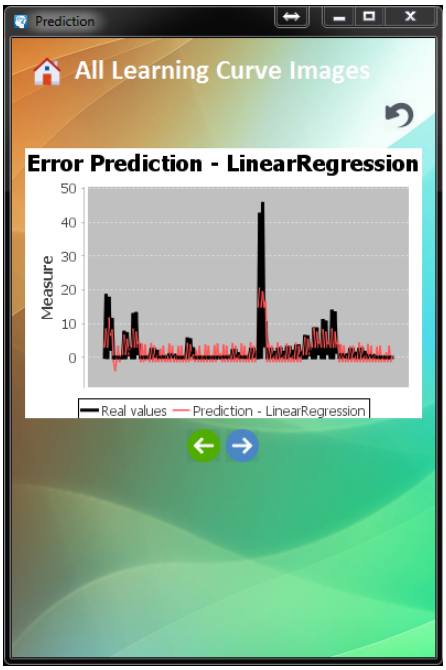


Figura 5.10: Lineas de predicción en la herramienta Nekonata

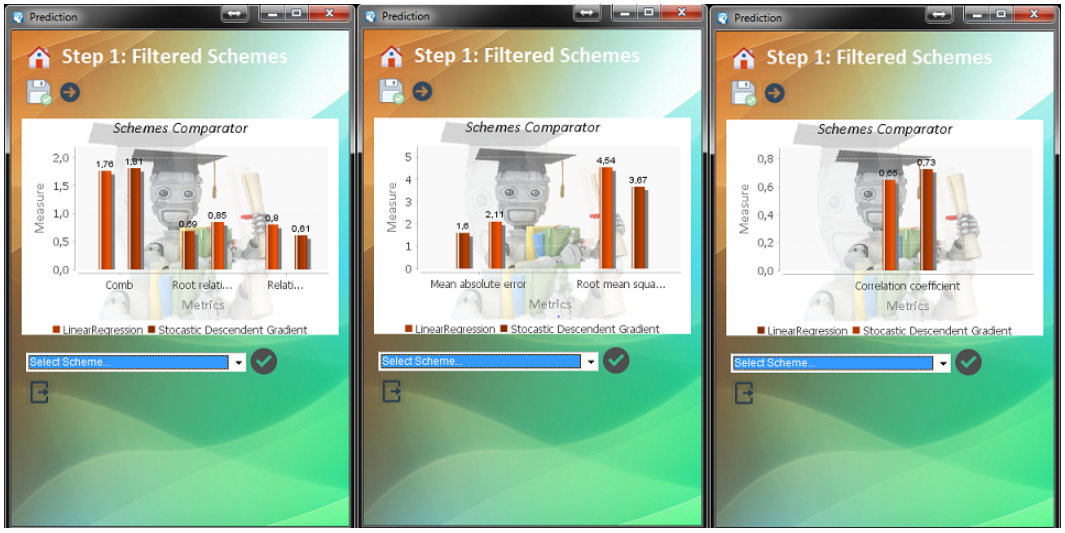


Figura 5.11: Figura comparativa de las dos implementaciones de regresión lineal para el problema del vaiajante

	Modelo de precisión - TSPLIB S4					
	CC	MAE	RAE	RRSE	RMSE	COMB
Linear Regression	0.71	0.02	0.93	0.7	0.04	1.92
MultiLayer Perceptron	1	0	0.03	0.01	0	0.04
Stochastic gradient descent	0.75	0.02	0.97	0.79	0.04	2.01
K-means Clusterer	0.21	0.0155	0.91	0.98	0.5	2.68
Support Vector Machine	1	0	0.03	0.01	0	0.04
	Modelo de precisión - TSPLIB S5					
Linear Regression	0.71	0.02	0.93	0.7	0.04	1.92
MultiLayer Perceptron	1	0	0.03	0.01	0	0.05
Stochastic gradient descent	0.74	0.02	1.06	0.83	0.04	2.15
K-means Clusterer	0.21	0.0157	0.91	0.98	0.05	2.68
Support Vector Machine	1	0	0.03	0.01	0	0.04
	Modelo de precisión - TSPLIB S6					
Linear Regression	0.77	0.01	0.87	0.64	0.03	1.75
MultiLayer Perceptron	1	0	0.02	0.01	0	0.03
Stochastic gradient descent	0.75	0.02	0.97	0.79	0.04	2
K-means Clusterer	0.18	0.0158	0.93	0.98	0.05	2.73
Support Vector Machine	1	0	0.03	0.01	0	0.04

Cuadro 5.4: Resultados del atributo precisión para el escenario 'problema del viajante'.

no demostraron influir sobre la precisión y el tiempo de respuesta al alcanzar un grado de correlación por debajo de 0.05 entre las variables de la imagen y las variables a predecir. Sólo los parámetros de configuración con los que se invocan a la función de detección de cada componente afectaron directamente al tiempo de respuesta requerido por el servicio. Los resultados de los modelos obtenidos para la predicción del atributo *Tiempo de respuesta* se exponen en el cuadro 5.5

	Modelo de tiempo de respuesta - Servicio FaceRect					
	CC	MAE	RAE	RRSE	RMSE	COMB
Linear Regression	0.7	118.12	0.64	0.72	153.6	1.66
MultiLayer Perceptron	0.7	119.46	0.64	0.72	153.52	1.66
Stochastic gradient descent	0.7	681.67	3.67	3.29	704.11	7.26
K-means Clusterer	0.99	19.125	0.1	0.13	27.92	0.24
Support Vector Machine	0.7	111.27	0.6	0.74	159.34	1.64
	Modelo de tiempo de respuesta - Servicio SkyBiometry					
Linear Regression	0.08	372.88	0.99	1	637.54	2.91
MultiLayer Perceptron	0.09	358.52	0.95	1	637.83	2.86
Stochastic gradient descent	0.06	302.27	0.8	1.6	676.21	2.8
K-means Clusterer	0.13	372.636	0.99	0.99	633.84	2.85
Support Vector Machine						
	Modelo de tiempo de respuesta - Servicio Microsoft					
Linear Regression	0.41	53.26	0.93	0.91	71.78	2.44
MultiLayer Perceptron	0.58	44.53	0.78	0.81	64.04	2.01
Stochastic gradient descent	0.41	56.29	0.99	0.93	73.3	2.51
K-means Clusterer	0.56	46.998	0.82	0.83	65.28	2.09
Support Vector Machine						
	Modelo de tiempo de respuesta - Servicio GooglePlay Vision					
Linear Regression	0.6	41.39	0.76	0.8	59.17	1.97
MultiLayer Perceptron	0.66	37.34	0.69	0.75	55.54	1.78
Stochastic gradient descent	0.59	41.24	0.76	0.8	59.31	1.97
K-means Clusterer	0.78	26.249	0.48	0.62	45.83	1.32
Support Vector Machine						

Cuadro 5.5: Resultados del atributo 'Tiempo de respuesta' para los servicios de detección de rostros.

	Modelo de tiempo de respuesta - Lenovo K3					
	CC	MAE	RAE	RRSE	RMSE	COMB
Linear Regression	0.46	0.39	1.19	0.89	0.77	2.62
MultiLayer Perceptron	0.97	0.08	0.24	0.28	0.25	0.55
Stochastic gradient descent	0.45	0.22	0.67	0.99	0.87	2.22
K-means Clusterer	0.97	0.05	0.16	0.23	0.2	0.41
Support Vector Machine						

	Modelo de tiempo de respuesta - Samsung Galaxy S3					
Linear Regression	0.44	0.58	1.17	0.9	1.21	2.63
MultiLayer Perceptron	0.97	0.14	0.28	0.28	0.38	0.59
Stochastic gradient descent	0.44	0.31	0.62	1.01	1.35	2.19
K-means Clusterer	0.97	0.085	0.13	0.19	0.33	0.34
Support Vector Machine						

	Modelo de tiempo de respuesta - Greedy Lenovo K3					
Linear Regression	0.68	0	0.36	0.73	0	1.4
MultiLayer Perceptron	0.69	0	0.42	0.73	0	1.47
Stochastic gradient descent	0.68	0	1.75	1.4	0	3.47
K-means Clusterer	0.69	0	0.36	0.72	0	1.39
Support Vector Machine						

	Modelo de tiempo de respuesta - Greedy Samsung Galaxy S3					
Linear Regression	0.84	0	0.3	0.54	0	1
MultiLayer Perceptron	0.84	0	0.6	0.64	0	1.39
Stochastic gradient descent	0.84	0	0.29	0.54	0	0.99
K-means Clusterer	0.85	0	0.32	0.54	0	0.99
Support Vector Machine						

Cuadro 5.6: Resultados del atributo Tiempo de respuesta para el escenario 'problema de la mochila'.

5.3.2 Resultados para los servicios de detección de rostros.

5.3.3 Resultados para el problema de la mochila.

5.3.4 Resultados para el problema de la multiplicación de matrices.

5.4 Conclusiones

	Modelo de precisión - Lenovo K3					
	CC	MAE	RAE	RRSE	RMSE	COMB
Linear Regression	0.26	0.02	1.05	0.97	0.05	2.76
MultiLayer Perceptron	0.41	0.01	0.94	0.91	0.04	2.44
Stochastic gradient descent	0	0.01	0.91	1	0.05	2.89
K-means Clusterer	0.82	0.008	0.59	0.57	0.03	1.34
Support Vector Machine						

	Modelo de precisión - Samsung Galaxy S3					
Linear Regression	0.14	0.2	1.28	0.99	0.99	3.14
MultiLayer Perceptron	0.18	0.2	1.29	0.99	0.99	3.1
Stochastic gradient descent	0.04	0.12	0.75	1	1.01	2.71
K-means Clusterer	0.95	0.09	0.58	0.32	0.33	0.96
Support Vector Machine						

Cuadro 5.7: Resultados del atributo precisión para el escenario 'problema de la mochila'.

	Modelo de tiempo de respuesta - NNN Sin RenderScript Lenovo K3					
	CC	MAE	RAE	RRSE	RMSE	COMB
Linear Regression	0.91	0.02	0.4	0.4	0.02	0.89
MultiLayer Perceptron	0.99	0	0.09	0.12	0.01	0.22
Stochastic gradient descent	0.89	0.02	0.38	0.47	0.03	0.96
K-means Clusterer	0.98	0.007	0.18	0.22	0.01	0.42
Support Vector Machine						

	Modelo de tiempo de respuesta - NNN Sin RenderScript S3					
Linear Regression	0.92	0.02	0.36	0.38	0.03	0.82
MultiLayer Perceptron	1	0	0.06	0.07	0.01	0.14
Stochastic gradient descent	0.91	0.02	0.33	0.44	0.03	0.86
K-means Clusterer	0.98	0.008	0.16	0.19	0.01	0.36
Support Vector Machine						

	Modelo de tiempo de respuesta - NNN Lenovo K3					
Linear Regression	0.87	0.02	0.49	0.49	0.03	1.11
MultiLayer Perceptron	0.99	0	0.07	0.12	0.01	0.2
Stochastic gradient descent	0.87	0.02	0.48	0.51	0.03	1.12
K-means Clusterer	0.99	0.004	0.12	0.15	0.01	0.28
Support Vector Machine						

	Modelo de tiempo de respuesta - NNN S3					
Linear Regression	0.87	0.02	0.45	0.49	0.03	1.06
MultiLayer Perceptron	1	0	0.08	0.09	0.01	0.18
Stochastic gradient descent	0.87	0.02	0.45	0.49	0.03	1.07
K-means Clusterer	0.99	0.005	0.11	0.14	0.01	0.26
Support Vector Machine						

Cuadro 5.8: Resultados del escenario 'Multiplicación de matrices'

Capítulo 6

Conclusiones

6.1 Limitaciones

6.2 Trabajos Futuros

Bibliografía

[1] 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.

[2] Larsson M Crnkovic I and Preiss O. Concerning predictability in dependable component-based systems: Classification of quality attributes. *IEEE Transactions on Software Engineering*, 37:593 – 615, 2011.

[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] Zoubin Ghahramani. Unsupervised learning. 2004.

[6] 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.

[7] 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.

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

[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] Florian Rosenberg and Predrag Celikovic. An end-to-end approach for qos-aware service composition. In *17th IEEE International Enterprise Distributed Object Computing Conference*, 2009.

[16] A. Nath K. Subbiah S. Kumar, M. K. Pandey and M. K. Singh. 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.

[17] Diaz-Pace J. A. Sanchez L. E. and A. Zunino. An approach based on feature models and quality criteria for adapting component-based systems. *Software Eng. R and D*, 3:10.

[18] B.W. Silverman. *Density Estimation for statistics and data analysis*. 1986. ISBN 0412246201.

[19] 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.

Apéndice A

Implementación de los algoritmos de resolución de TSP

El Problema del Agente Viajero (*TSP -Travelling salesman Problem*) corresponde a uno de los problemas de optimización más estudiados de la clase NP-Completos; es un problema de minimización que comienza y termina en un vértice específico y se visita el resto de los vértices exactamente una vez. El Travelling Salesman Problem (TSP) puede ser modelado como un grafo ponderado no dirigido, de manera que las ciudades sean los vértices del grafo, los caminos son las aristas y las distancias de los caminos son los pesos de las aristas.

Algoritmos de resolución

1. Best Fit - Bin Packing -

Best fit es uno de los algoritmos heurísticos más simples,brinda soluciones óptimas (aproximadas) aunque el algoritmo no asegura el retorno de la mejor solución. Para dominios de grafos no completos incluso, el camino solución puede contener arcos con costo infinito (ausencia de arista). Partiendo del punto *A* se analiza el trayecto más corto a la siguiente ciudad, sea ésta por ejemplo la ciudad *C*, ambas ciudades se añaden al conjunto Solución: {A, C, A}

En cada paso del algoritmo las ciudades son seleccionadas de acuerdo al costo de sus trayectos y son incorporadas al conjunto Solución en base al cálculo de la distancia marginal de las intersecciones.

La distancia marginal representa la variación en el costo total teniendo en cuenta el costo directo entre *A* y *C* y el costo del camino entre *A* y *C* pasando

por B. (A,B,C). El par de ciudades A y C son seleccionadas de forma tal que hagan mínimo el costo del paso por la ciudad B.-

Ejemplo

	B	C	D	E
A	2	1	10	25
B		18	5	5
C			20	2
D				8

S: { A, C, A}

Selección ciudad E:

Trayectos: AC - CA

En tal caso, los trayectos son análogos como así también la distancia marginal: E se añade entre las ciudades C y A.

S: { A,C,E,A}

Costo: 28

En el siguiente paso la ciudad B es seleccionada:

Trayectos:

$$AC : D = costo(A,B) + costo(B,C) - costo(A,C) = 19$$

Resultando S: {A, B, C, E, A} Costo ahora:28+19=47

$$CE : D = costo(C,B) + costo(B,E) - costo(C,E) = 21$$

Resultando S: {A, C, B, E, A} Costo ahora:28+21=49

$$EA : D = costo(E,B) + costo(B,A) - costo(E,A) = -18$$

Resultando S: {A, C, E, B, A} Costo ahora:28-18=10

B es añadida entre las ciudades del trayecto cuya distancia es menor, en el ejemplo ilustrado, la opción del tercer trayecto.

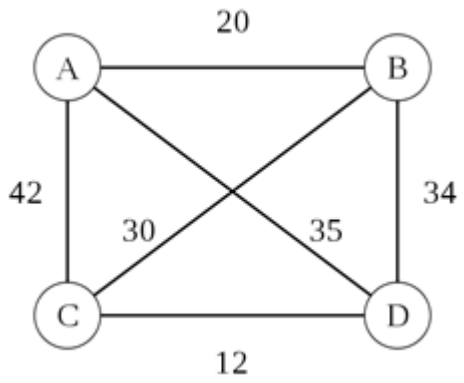
2. Vecino más cercano

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. Tal análisis prosigue a partir del último vértice añadido hasta haber visitado todas las ciudades.

A menudo, el TSP sigue un modelo de grafo completo donde cada par de vértices es conectado por una arista, en tal caso el algoritmo brinda soluciones óptimas partiendo desde y hacia la ciudad inicial pasando exactamente una vez por cada ciudad restante.

Por otro lado,en modelos donde no existe camino entre algunos pares de ciudades, la elección iterativa de la arista de mínimo costo sin una vista global del modelo, puede apartar vértices de la solución retornando soluciones parciales.

Ejemplo



Solución: A -> B -> C -> D -> A
Costo: 97

3. Programación Lineal

El TSP puede ser formulado de manera teórica mediante una (o varias) formulaciones lineales enteras usando programación lineal en enteros. Se plantean las variables del problema:

vértices : 0.. n Xij = 1 | Xij = 0

4. Backtracking

Backtracking es un algoritmo general para encontrar todas o algunas soluciones para problemas computacionales con notables restricciones de satisfacción para dicho problema.El método de búsqueda es incremental de los candidatos a la solución, abandonando aquellos (*backtrackeando*) tan pronto como se determina que dicha solución no es o va a ser válida.

En el caso del TSP, los posibles candidatos a la solución se determinan mediante el grado de incidencia de todos los nodos que componen el grafo. Ya que la ciudad sólo debe ser visitada una vez, el grado de incidencia de cada ciudad debe ser igual a 2, por lo que aquellas ciudades que no han sido visitadas (grado de incidencia 0) o aquellas que sólo han sido destino (grado de incidencia 1) son candidatas a componer la solución.

Pseudo-código:

backtracking(costoAct, minCost, lastV, currPath, bestPath, costoAct) si costoAct<minCost

TSP simétrico

El TSP simétrico es una variedad del problema TSP que para todas las aristas se cumple la desigualdad triangular o de Minkowski:

$$(a,c) < (a,b) + (b,c)$$

Considerando que no todos los grafos creados cumplen con dicha condición, y que aún así hay que devolver un resultado óptimo, se considera prioridad la distancia entre los nodos y no si los mismos se repiten o no.

4. Árbol de recubrimiento

Para resolver el *Metric TSP*, se puede utilizar el método de árbol de recubrimiento mínimo. El mismo se puede obtener mediante Prim o Kruskal para, luego de eso, nos aseguramos que el grafo será simétrico mediante la duplicación de cada arista. Gracias a esto, podemos recorrerlo mediante un algoritmo recursivo y obtenemos el ciclo euleriano del mismo. Para finalizar la resolución y mediante la desigualdad antes planteada, se omiten los nodos repetidos, obteniendo el ciclo hamiltoniano.

4.1 Prim

El algoritmo Prim toma un tomar un vértice arbitrario de los ya visitados y lo une con el vértice no visitado mediante el menor arco que salga de él.

Pseudocódigo:

```
Prim(firstCity ){ visited={ firstCity }; while (visited!= allCities ){
```

4.2 Kruskal

Kruskal mientras tanto tiene ordenados las aristas por costo y va agregando a la solución aquellos que ya han sido visitados con los que no.

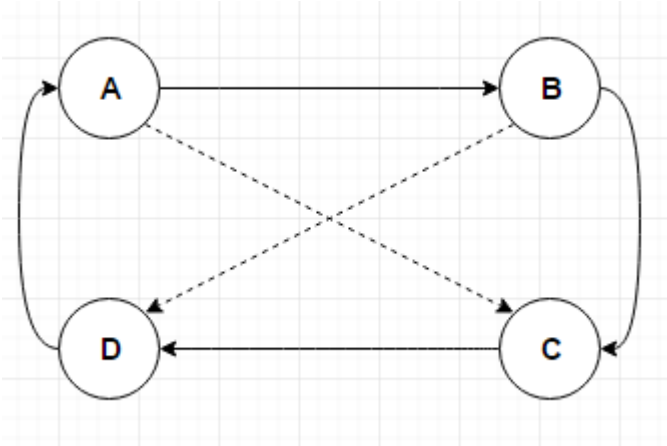
Pseudocódigo:

```
Kruskal(firstCity ){ visited={ firstCity }; edges=getSortedEdges();
```

La única diferencia entre ambos algoritmos de recubrimiento es la forma de trabajo de los mismos y en consecuencia su complejidad.

4.3 Transformaciones Lineales

Las transformaciones locales se utilizan para mejorar el hamiltoniano obtenido por los algoritmos antes presentados. Para introducir el algoritmo, consideremos la siguiente situación:

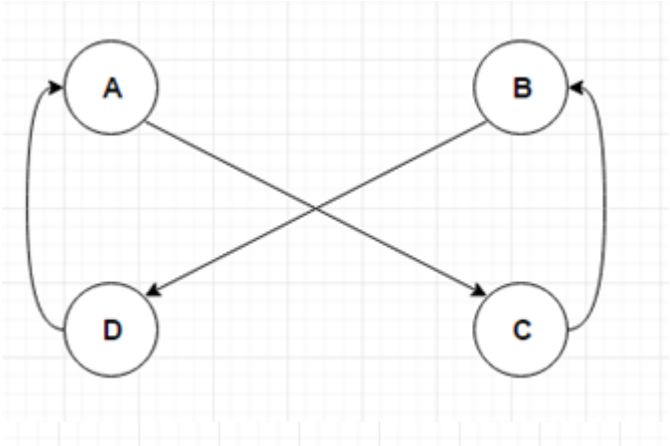


Los nodos unidos en el ciclo hamiltoniano son (A,B) ... (C,D) - línea no punteada en el grafo.

Ahora bien, para aplicar transformaciones locales se debe verificar:

$$(A,C) + (B,D) < (A,B) + (C,D)$$

De suceder esto, se pueden modificar los arcos y obtener:



El resultado es un grafo con menor costo que el anterior presentado.

Apéndice B

Descripción de los servicios de detección facial

Servicio SkyBiometry

En referencia al uso de la API – Autenticación

Cada llamada a la API debe ser autorizada mediante el uso de dos claves: api_key y api_secret. Ambas claves son obtenidas mediante una registración en el sitio oficial de skybiometry (*www.skybiometry.com*) a través de una cuenta de email como usuario y una contraseña.

Límites de uso

El consumo de este servicio presenta algunas limitaciones que varían en base a la suscripción particular del usuario. En general, las suscripciones gratuitas restringen un límite de 100 llamadas a métodos por hora y de 5000 en el mes. Para el caso del método POST, la limitación se presenta tanto en 100 request a la API para el procesamiento de imágenes individuales como para 1 request con 100 imágenes a procesar.

Factores ofrecidos

Respecto al cuadrado del rostro detectado:

center OBJECT { x :Float, y : Float }
width. FLOAT. 0-100 % del ancho del rostro respecto al ancho de la imagen.

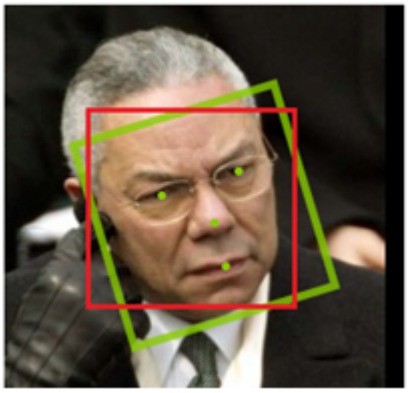
height. FLOAT. 0-100 % del largo del rostro respecto al ancho de la imagen.

Otros puntos detectados:

mouth_center	OBJECT { x :Float, y : Float }
mouth_left	OBJECT { x :Float, y : Float }
mouth_right	OBJECT { x :Float, y : Float }
eye_left	OBJECT { x :Float, y : Float }
eye_right	OBJECT { x :Float, y : Float }
nose	OBJECT { x :Float, y : Float }
ear_left	OBJECT { x :Float, y : Float }
ear_right	OBJECT { x :Float, y : Float }
chin	OBJECT { x :Float, y : Float }
yaw.	FLOAT.Perfil. Value -90° a 90°
roll.	FLOAT. Ángulo de rotación del rostro. Value -90° a 90°
pitch.	FLOAT. Value -90° a 90°

yaw: ángulo positivo en rostros donde predomina el perfil derecho (respecto al sujeto, no a la imagen). Caso contrario, ángulo negativo al predominar el perfil izquierdo.

Roll: Ejemplo ilustrativo. El rectángulo verde representa al rostro detectado por la API cuyo valor roll es igual a -17°, mientras que el rectángulo rojo fue añadido para ilustrar el caso donde el ángulo roll es igual a 0°. Una rotación inversa significa un valor roll positivo.



Todos los puntos detectados descritos no especificados corresponden a un formato JSON, cuya clave es el nombre del mismo, y el valor es también un objeto del tipo JSON que contiene los siguientes valores: id:INTEGER, confidence:INTEGER, y:FLOAT, x:FLOAT, a excepción del valor *center* cuyo objeto JSON contiene sólo los valores: x, y.

Nota1: en caso de valor de atributo no se puede determinar que no se devuelve. Si se determina el valor que se devuelve junto con el valor de confianza como porcentaje de 0 a 100.

Nota2: La API sólo devuelve información necesaria para representar en forma de rectángulo los rostros detectados. Para el resto de opciones, la API sólo devuelve información del punto central en cuestión.

Nota3: Ya que la API automáticamente re-escala las imágenes a 1024 pixeles para su procesamiento interno, todas las coordenadas son provistas de forma porcentual respecto al ancho y largo de la imagen (abstracción).

Nota4: Nota: el atributo face es el valor por defecto y siempre es retornado, independientemente de los atributos especificados. Si el atributo "glasses" fue requerido adicionalmente se retorna el atributo 'dark_glasses' para diferenciar entre gafas oscuras y claras. El atributo mood (estado de ánimo) se devuelve junto con 7 más atributos: confianza para cada una de las emociones básicas, además de la confianza neutral_mood.

Atributos faciales

The result of the API call is a JSON or XML object containing the requested information. Cada uno de los atributos son objetos del tipo JSON cuya clave es el nombre del mismo y el valor es también un objeto JSON con dos valores específicos: el primero con clave *value* y el segundo con clave *confidence*. El valor de confidence es del tipo Integer para representar el porcentaje de probabilidad del valor detectado en el campo *value*.

Mensajes de error

ERROR CODE	ERROR MESSAGE
20	IMG_DECODE_ERROR
21	IMG_RESIZE_ERROR
30	DOWNLOAD_ERROR
31	DOWNLOAD_ERROR_FILE_NOT_FOUND
32	DOWNLOAD_ERROR_SERVER_TIMEOUT
33	DOWNLOAD_ERROR_FILE_TOO_LARGE
34	DOWNLOAD_ERROR_MALFORMED_URL
35	DOWNLOAD_ERROR_UNKNOWN_HOST
36	DOWNLOAD_ERROR_CONNECTION_REFUSED
104	INTERNAL_ERROR
105	SERVICE_TEMPORARILY_UNAVAILABLE
107	UNKNOWN_ERROR
201	API_KEY_DOES_NOT_EXIST
202	API_KEY_USAGE_PASSED_QUOTA
203	API_KEY_CONCURRENT_USAGE_PASSED_QUOTA
204	API_KEY_NOT_AUTHENTICATED
205	API_PASSWORD_NOT_CORRECT
206	MAX_NUMBERS_OF_UIDS_TRAINED_IN_NAMESPACE _EXCEEDED
207	TOO_MANY_ERRORS
301	TAG_NOT_FOUND
303	FILTER_SYNTAX_ERROR
304	AUTHORIZATION_ERROR
306	TAG_ALREADY_EXIST
307	ACTION_NOT_PERMITTED
401	UNKNWOWN_REST_METHOD
402	MISSING_ARGUMENTS
403	MISSING_USER_NAMESPACE
404	UNAUTHORIZED_USER_NAMESPACE
405	UNAUTHORIZED_UID
406	INVALID_ARGUMENTS_VALUE
407	ARGUMENT_LIST_TOO_LONG
408	UNAUTHORIZED_CALLBACK_URL_DOMAIN
409	UID_TOO_LONG
410	SYNCHRONOUS_REQUEST_TOO_BIG

Post (Request en lenguaje JAVA)

```
HttpResponse<JsonNode> response = Unirest.post(https://
    face.p.mashape.com/faces/detect?api_key=[api_key]&
    api_secret=[api_secret])
.header(X–Mashape–Key, 7
    rS5YDw5YHmshtdgMHP2ZYBLAljfp1OxKNzjsn1GJxNBgad6C9)
.header(Accept, application/json)
.field(attributes, all)
.field(detector, Aggressive)  [ver opciones]
.asJson();
```

Nota: En los espacios [api_key] y [api_secret] deben colocarse las claves correspondientes disponibles con la registración de una cuenta en el sitio www.skybiometry.com

Opciones:

.field("files", Vector<File> imagenes)

.field("files", File imagen)

.field("urls", ""url_1", "url_2", ...)

Nota: Los formatos de imagen aceptados por la API son los siguientes: PNG, JPEG, BMP, JPEG2000.

La respuesta recibida por el servicio del tipo HttpResponse<JsonNode> puede ser manipulada como un objeto del tipo JSONObject a través de la siguiente codificación:

```
JSONObject obj = response.getBody().getObject();
```

Parse - Overview

La respuesta en formato JSONObject contiene en primera parte un objeto del tipo JSONObject con clave *photos* y cuyo valor, representado en formato JSONArray, contiene toda la información detectada desde las imágenes. Cada elemento del arreglo representa un archivo de imagen; a su vez, estos elementos contienen un objeto del tipo JSONObject cuya clave es *tags* y su valor es un JSONArray para representar cada una de los rostros detectados en la imagen particular; a su vez contiene la información de ancho y largo asociado.

En segunda parte, seguido de *photos* se anexan en la respuesta la información relacionada a la operación http.

Ejemplo codificación básica:

```
JSONArray imágenes = obj.get(photos);
for(int p=0; p<imágenes.length(); p++) {
        //Obtención de cada una de las imágenes
        JSONObject imagen=imagenes.getJSONObject(p);
        //Ejemplo obtención del ancho de la imagen
        int imgWidth = (int) imagen.get(width);
        //Obtención de los rostros detectados en la
            imagen
        JSONArray rostros = imagen.getJSONArray(tags);
        for(int r=0; r<rostros.lenght(); r++) {
                //Obtención de cada uno de los rostros
                    detectados
                rostro=rostros.getJSONObject(r);
                //Ejemplo obtención de los puntos
                    detectados JSONObject
```

```
                    mouth = rostro.getJSONObject(mouth_center
                        );
                    int coordX= (int) mouth.get(x);
                    int coordY= (int) mouth.get(y);
                    //Ejemplo obtención de los atributos
                        faciales JSONObject
                    atributos=imagen.getJSONObject(attributes
                        );
                    String valor = atributos.getJSONObject(
                        mood).getString(value);
            }
}
```

Servicio FaceRect

En referencia al uso de la API

El consumo de esta API requiere la registración de una cuenta en el sitio Mashape. La registración se realiza con email y contraseña y se realiza de manera gratuita e instantánea. El servicio no restringe el número de request a la API pero permite el procesamiento de sólo una imagen por request realizado.

Factores ofrecidos

Rostro detectado:

Orientation:	tipo String. Valores posibles:<frontal, profile-right, profile-left>
x:	tipo Integer
y:	tipo Integer
width:	tipo Integer
height:	tipo Integer
Features:	(opcional)
eyes	
nose	
mouth	

Los *features* son analizados por la API sólo en rostros cuya orientación sea frontal. El formato de respuesta de los mismos son del tipo JSONObject en el caso de *nose* y *mouth*. Para el caso de *eyes* se devuelve un objeto del tipo JSONArray, cuyos elementos (1 o 2) son del tipo JSONObject. La API representa el rostro y los features detectados en forma de rectángulos, por lo que incorpora en sus respuestas las coordenadas x, y (esquina superior izquierda respecto a la imagen) y los tamaños *width*, *height* del rectángulo en cuestión.

Post (Request en lenguaje JAVA)

HttpResponse<JsonNode> response = Unirest.post(ENDPOINT).header(X−Mashape −Key, 7rS5YDw5YHmshtdgMHP2ZYBLAljfp1OxKNzjsn1GJxNBgad6C9)[ver opción correspondiente] .field(features, **true**).asJson();

Actualmente la API soporta dos endpoints difiriendo entre sí en el modo en que las imágenes son especificadas en el request. Mediante el método http *GET* se añade el URL de la imagen (Endpoint: http://apicloud-facerect.p.mashape.com/process-url.json); por otra parte con el método *POST* se añade el archivo correspondiente (Endpoint: http://apicloud-facerect.p.mashape.com/process-file.json) De acuerdo con el endpoint especificado se añade la línea correspondiente:

Opciones

.field(“url”, “url_image”)

.field(image, new File(<path_image>))

Nota: Para ambos casos, la API restringe el procesamiento de algunas imágenes, permitiendo procesar aquellas cuyo formato sea JPEG, PNG y GIF, la resolución de la misma no supere los 4096 píxeles (4096×4096) y el tamaño sea inferior a 10 MBytes.

La respuesta recibida por el servicio del tipo HttpResponse<JsonNode> puede ser manipulada como un objeto del tipo JSONObject a través de la siguiente codificación:

```
JSONObject obj = response.getBody().getObject();
```

Parse - overview

La respuesta en formato JSONObject contiene dos objetos JSON. El primero, cuya clave es *faces* contiene toda la información detectada desde la imagen, el segundo objeto cuya clave es *image* contiene la información acerca del ancho (*width*) y largo (*height*) de la misma. La información detectada se presenta en un objeto del tipo JSONArray donde cada elemento del arreglo constituye cada uno de los rostros que ha detectada la API; Cada uno de los elementos son JSONObject que contienen la información del rostro (*orientation*, *x*, *y*, *height*, *width*) y los *features*.

Ejemplo codificación básica

```
//Ejemplo obtención del ancho de la imagen
int imgWidth = (int) obj.getJSONObject(image).get(width);
//Obtención de los rostros detectados en la imagen
JSONArray rostros = obj.getJSONArray(faces);
for(int r=0; r<rostros.lenght(); r++) {
        //Obtención de cada uno de los rostros detectados
        rostro=rostros.getJSONObject(r);
```

```
        //Ejemplo obtención información del rostro:
        String orientation =(String) rostro.get(
            orientation);
        int coordX=(int) rostro.get(x);
        //Ejemplo obtención de los features
        JSONObject features = rostro.getJSONObject(
            features);
        JSONArray eyes =features.getJSONArray(eyes);
        JSONObject eye_left=eyes.getJSONObject(0);
}
```

Servicio GMS Vision

Factores ofrecidos

El detector puede computar los siguientes atributos (accesibles a través de los distintos métodos de la clase).

Profile: rotación del rostro. FLOAT

Clasificadores:<característica facial presente en el rostro>

Open	eyes: (probabilidad) Constant value: 1 FLOAT (*)
Smiling:	(probabilidad) Constant value: 1 FLOAT (*)
Height,	width: medidos en píxeles. FLOAT
Position:	Punto superior izquierdo del rostro. FLOAT

Landmarks:<Puntos de interés en el rostro detectado> LIST<Landmark>

BOTTOM_MOUTH Constant Value: 0 CENTER
LEFT_CHEEK Constant Value: 1 CENTER
LEFT_EAR_TIP Constant Value: 2
LEFT_EAR Constant Value: 3
MIDPOINT LEFT_EYE Constant Value: 4
CENTER LEFT_MOUTH Constant Value: 5
NOSE_BASE Constant Value: 6
MIDPOINT RIGHT_CHEEK Constant Value: 7
CENTER RIGHT_EAR_TIP Constant Value: 8
RIGHT_EAR Constant Value: 9
MIDPOINT RIGHT_EYE Constant Value: 10
CENTER RIGHT_MOUTH Constant Value: 11

Angles: <Face Orientation>

EulerY [valor 'y' en la imagen]

EulerZ [valor 'r' en la imagen]

(*) En el trabajo desarrollado se consideró el uso de valores booleanos, determinando a los clasificadores como verdaderos para valores de probabilidad mayor al 50%, en casos contrario se determinaron los mismos como falsos.

Nota: El ángulo EulerZ se encuentra disponible siempre en la detección, mientras que el ángulo EulerY sólo si en el detector se predetermina el modo *accurate*. El concepto de *left* y *right* son relativos al sujeto no a la posición cuando se observa la imagen. Las posiciones son accedidas a través del método getPosition() de la clase Landmark; el retorno es del tipo PointF.

Construcción del detector

```
FaceDetector detector = new FaceDetector.Builder(context)
.setTrackingEnabled(false)
.setMode(int Mode)
.setProminentFaceOnly(true)
.setClassificationType(int classificationType)
.setLandmarkType(int landmarkType)
.build();
```

Es necesario especificar en el detector el entorno de nuestra aplicación (mediante el uso de la clase android.utils.Context)

Es posible obtener el entorno de la app a través del método *getApplicationContext()* de la clase Activity.

El contexto es sólo una interfaz cuya implementación está prevista por el sistema android y permite el acceso a los recursos y clases específicas de la aplicación, como así también realizar operaciones tales como el despacho de actividades, la difusión y recibimiento de objetos del tipo Intent, etc.

El servicio dispone de la posibilidad de elegir la preferencia de procesamiento en la detección, *FAST_MODE* prioriza la rapidez en el análisis por sobre la cantidad de rostros detectados y su precisión, contrariamente el servicio dispone del modo *ACCURATE_MODE*

Habilitar la opción *tracking enabled* es recomendable para la detección de imágenes individuales no relacionadas (a diferencia de un vídeo o una serie de imágenes fijas capturadas consecutivamente).

Tanto los clasificadores y landmarks pueden especificarse uno a uno añadiendo las líneas correspondientes en el constructor; adicionalmente la clase FaceDetector proporciona las siguientes constantes:
ALL_CLASSIFICATIONS

ALL_LANDMARKS

NO_CLASSIFICATIONS

NO_LANDMARKS

El servicio no ofrece la capacidad de setar los landmark y clasificadores bajo demanda, sino que son definidos estáticamente al momento de crear el detector; esta restricción condiciona el tiempo de respuesta del servicio ya que el requerimiento de al menos un landmark, por ejemplo, significa el uso de la variable *FaceDetector.ALL_LANDMARKS*, incrementando así el tiempo de procesamiento de la imagen.

Uso del detector

El detector se puede llamar de forma sincronizada con un objeto del tipo Frame para detectar las caras.

```
Frame frame = new Frame.Builder().setBitmap(<file_image>)
    .build();
SparseArray<Face> faces = detector.detect(frame);
```

Ejemplo codificación básica

```
for(int i=0;i<faces.size();i++){
        //Obtención de cada uno de los rostros detectados
            en la imagen
        Face rostro=faces.valueAt(i);
        //Obtención del rectángulo del rostro
        Float coordX=rostro.getPosition().x;
        Float coordY=rostro.getPosition().y;
        Float width=rostro.getWidth();
        Float height=rostro.getHeight();
        //Obtención del ángulo EulerZ
        Float angleZ=rostro.getEulerZ();
        //Obtención del clasificador eyes open
        Float probEyeRight=rostro.
           getIsRightEyeOpenProbability();
        Float probEyeLeft=rostro.
           getIsLeftEyeOpenProbability();
        //Obtención de los Landmarks detectados:
        List<Landmark> landmarks=rostro.getLandmarks();
        //Obtención de cada uno de los landmark
        for(int j=0;j<landmarks.size();j++){
                int LandmarkType=landmarks.get(j).getType
                    ();
                PointF Landmarkposition=landmarks.get(j).
                    getPosition();
        }
}
```