UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

CARACTERIZACIÓN Y PREDICCIÓN DE CONDUCTA DE USUARIOS DE APLICACIÓN MÓVIL ENFOCADO A PROCESO 'ON BOARDING' UTILIZANDO HERRAMIENTAS DE MACHINE LEARNING

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

DIEGO NICOLÁS CASTILLO WARNKEN

PROFESOR GUÍA:
HÉCTOR ALVAREZ GÓMEZ

MIEMBROS DE LA COMISIÓN:
ERNESTO KRSULOVIC MORALES
ANDRÉS CABA RUTTE

Este trabajo ha sido parcialmente financiado por:
TIMEJOBS

SANTIAGO DE CHILE
2022

CARACTERIZACIÓN Y PREDICCIÓN DE CONDUCTA DE USUARIOS DE APLICACIÓN MÓVIL ENFOCADO A PROCESO 'ON BOARDING' UTILIZANDO HERRAMIENTAS DE MACHINE LEARNING

Timejobs es una *start up* que se especializa en el *staffing on demand*. Dentro de mayores dolores se encuentra la fidelización de sus usuarios al momento de registrarse en su aplicación de reclutamiento. Si bien existen grupo de operadores que ayudan a impulsar a los usuarios, el trabajo es muy extenso y poco eficiente. En este documento se presentan soluciones utilizando herramientas de *machine learning*. Se busca clasificar a los usuarios que acaban de terminar de registrarse si tomarán la inducción que les permite tomar tareas. En lo particular se probó y testeó los algoritmos Perceptrón Multicapa, Arbol de Decisión, Random Forest, Gradient Boosting y Naives Bayes. Se analiza que Random Forest y Gradient Boosting son los mejores modelos llegando a un *accuracy* de 75 % mientras que los otros modelos tienen un desempeño menor a 65 % en la misma métrica. De igual manera se entra en detalle en la recolección del dato, preprocesamiento e implementación del modelo para que sirva de guía para futuros trabajos en la empresa. El trabajo se asemeja a predicción de fuga que se encuentra bastante en la literatura pero agregando las condiciones propias del problema. Se concluye que el trabajo fue enriquecedor tanto para la empresa como para el departamento de ingeniería eléctrica.

Para keka y polo
los adoro

Agradecimientos

Quiero agradecer en una primera instancia a Andrés Caba y Marcelo Jimenez por escucharme y ofrecerme soluciones con mi problema de malla. Sin ellos no hubiera podido cumplir con los plazos correspondientes.

Por el otro lado, agradecer a Hector Alvarez por atreverse a recibir a su primer tesista y de otro departamento. A Ernesto, Oscar, Russell, Edson, Paty, Seba, Michael y a todo el equipo de digital en general por darme la confianza en Timejobs para explorar e innovar.

A mi familia; a mi papá por incentivarme a siempre ir más allá, a mi madre que siempre me ha impulsado en creer en mi. A mi hermana Vale por siempre ser mi mayor cómplice, a mi hermano Martín por darnos el ejemplo.

A somos12, mi mayor apoyo universitario. Mi familia en la universidad que siempre estuvieron en los momentos académicos más duros. Al grupo de difusión que me ayudaron a desarrollarme, al equipo de volley que siempre han dado el impulso a mejorar. A mis amigos del colegio (pokeface) que han sido y serán parte de mi.

Y bueno a todas las lindas personas con las que pude conectar en esta fase universitaria. Cada sonrisa e intercambio de palabras siempre me han llenado el alma. Y espero que siga siendo así.

Gracias totales.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Timejobs

1.1.1. Contexto

Time Group S.A es una *start up* nacida a principios de 2019 que se dedica principalmente a lo que se denomina como *Staffing on Demand*. La empresa ha crecido exponencialmente, abarcando cada vez más dominios y atrayendo más personal. A principios de 2020 contaban con 30 trabajadores y ahora a principios de 2022 ya son más de 200 empleados abarcando tareas tanto en Chile, Perú y Colombia.

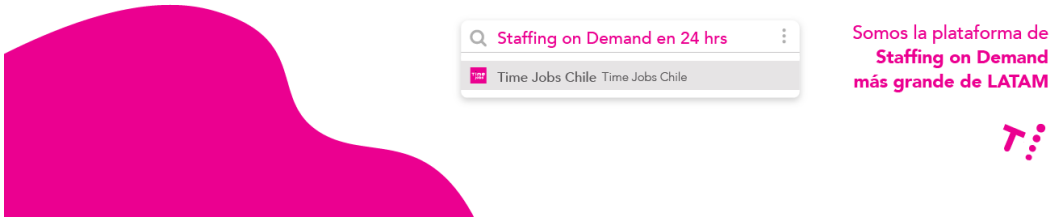

Figura 1.1: Banner de TimeJobs

Staffing on Demdand consiste en garantizar el cumplimiento de diversas actividades o trabajos de modalidad *part time*, mediante la publicación de las actividades en plataformas internas de Timejobs. Vale decir, Timejobs se asocia a empresas, estas empresas necesitan personal para una tarea específica. Timejobs ofrece estas tareas en sus plataformas digitales. El pozo de usuarios/jobbers que posee Timejobs identifica estas taras con la opción de tomarla para finalmente desarrollar la tarea. En toda esta secuencia existen tres actores:

- **Clientes:** son las tiendas que necesitan personal para diferentes tareas como podría ser Wallmart o Tottus.
- **Usuarios:** que se mencionarán como **jobbers** para respetar la nomenclatura interna son los *Free lancer* que buscan tareas diarias y dinámicas que ya se han registrado en las plataformas de Timejobs

- **Timejobs:** es el actor y empresa en la que se trabaja que logra juntar los dos actores a través de sus aplicaciones, tecnología y técnicas de operaciones.

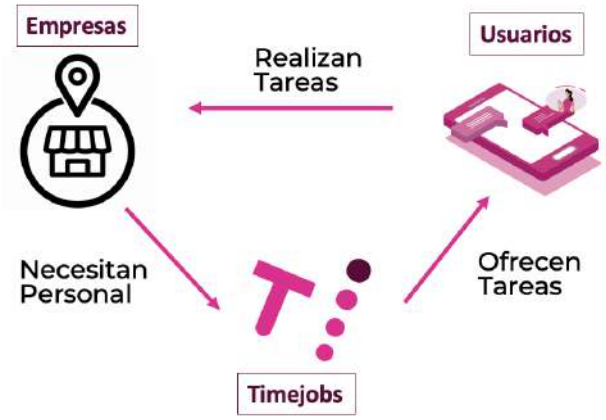

Figura 1.2: Flujo Staffing on Demand en Timejobs

Esta metodología permite a las empresas cliente cumplir con sus aumentos de demanda o tercerizar tareas, sin necesidad de aumentar el personal fijo. Por su parte, los usuarios de la aplicación, o Jobbers, cuentan con la flexibilidad de tomar tareas diversas y en horarios variados según estos lo estimen conveniente.

En particular, Timejobs cuenta con un flujo muy dinámico entre los jobbers y clientes. En efecto, Timejobs ha logrado alianzas con empresas de alta magnitud como lo son Jumbo, Rappi, Wallmart, Tottus, Unimarc entre otros. Y por el otro lado, ha logrado contar con más de 40 mil jobbers registrados, con más de mil activos cada mes.

1.1.2. Tipos de jobbers

Timejobs cuenta con 3 tipos de trabajos para jobbers. Estos se crean dependiendo del tipo de trabajo que requieran los clientes.

- **Operario:** Personal para trabajos de bodega.
- **Picker:** Personal con o sin auto que pueda ir a realizar una entrega sencilla.
- **Shopper:** Trabajador con auto para ir a hacer pedidos más complejos e ir a dejarlos a uno o más lugares. Esta tarea fue diseñada para adaptarse a partners que exigían una extensión a sus plataformas como Wallmart.

Estos tres tipos de jobbers existen para realizar diferentes tipos de tareas y todas viven en una de las tres verticales de Timejobs. Un usuario puede ser uno o más tipo de jobber. Entre más tipos de jobbers se puede postular a más tipos de tareas. Al ser independientes, cada vertical posee sus propias problemáticas como sus propias proyecciones. En lo general, las tres verticales siempre buscan aumentar la cantidad de jobber para suplir las nuevas tareas y nuevas sedes que se van abriendo.

1.1.3. Plataformas Digitales

Timejobs cuenta con múltiples plataformas internas como aplicaciones móviles para gestionar la gerencia operaciones. También se han levantado campañas y procesos para atraer jobbers al igual que estrategias para lograr una fidelización de los usuarios con la empresa. A mediados del 2021, Timejobs lanzó una nueva plataforma digital para estandarizar y potenciar el reclutamiento del jobber para incentivar el uso de la aplicación móvil dado el número de tareas que ha ido creciendo.

Desde un punto de vista técnico, todas las plataformas están creadas a través de NodeJS (basado en typescript) compilados en contenedores *dockers*. A lo largo de las diferentes versiones se ha guardado la data en base de datos relacionales (Aurora y RDS de Amazon Web Services (AWS)) y en base de datos no relacionales (Mongo DB). Además, el área de Data de Timejobs ha ido armando un *Data Lake* que tiene como intención centralizar la data no solo de las bases de datos, sino de otras fuentes de tecnológicas externas al área de Tecnología.

Para efectos de este trabajo, se quiere hacer un análisis en lo más nuevo, dado que es lo más rico en data y ordenado en comparación a las anteriores. Esto significa trabajar en lo que respecta Mongo DB. En Mongo DB existen dos bases de datos llamada ***user*** y ***applications***. El primero crea un usuario al momento que entra por primera vez a la plataforma y se ve rellenando a medida que va completando los pasos. Es la misma que almacena la información de todos los jobbers activos. El segundo va registrando como los jobbers van postulando a tareas.

En paralelo y fuera de la infraestructura interna, se utiliza una herramienta llamada *segment* que funciona como gatillador. El *framework* utilizado para el proceso se acopla con el *framework* NodeJS de desarrollo. A través de esta librería en el código, si el usuario pasa un hito particular, se crea un evento que pasa por *segment* y puede llegar a diversas plataformas como GCP, Mix Panel o un *bucket*. Esto permite ir teniendo más información de la interacción usuario-plataforma.

1.1.4. Reclutamiento

El flujo de *staffing on demand* explicado anteriormente funciona siempre y cuando existan suficientes jobbers para poder llenar los cupos de tareas. Es por eso que dentro de los objetivos de Timejobs, se encuentra tener la mayor cantidad de jobbers que se pueda.

Los jobbers pueden llegar a registrarse de diversas maneras. El área de marketing es la encargada de captar la atención de posibles jobbers. El jobber en ese proceso se llama *Applicant*. Posteriormente, al momento de interesarse y entrar al enlace de registro, existen diferentes pasos para completar su perfil como se muestra en Figura 1.4. Al finalizar todos los pasos, el jobber tiene que realizar una inducción antes de comenzar a tomar tareas. La inducción es la capacitación necesaria que definirá que trabajos podrá realizar.

En términos técnicos, el jobber a realizar una inducción predeterminada, le dará una etiqueta o *tag* dentro del sistema. El jobber a poseer un 'tag operador' por ejemplo, se le habilitan las tareas de operador dentro de la aplicación.

Figura 1.3: Flujo del reclutamiento de jobber

El nuevo proceso de reclutamiento, llamado *On boarding* se empezó a aplicar a mediados del 2021 y desde ese punto se ha empezado a recolectar la data con la nueva infraestructura web. Lo anterior a este proceso se considera como data sucia, dado que los atributos y la forma que se guarda el dato es diferente.

1.1.5. Problemática Actual en Reclutamiento

Utilizando los datos en mongo DB se procede a crear una visualización en torno a hitos y fuga dentro del proceso. Se escoge la separación de los siguientes bloques:

1. **Inicio Registro:** Nuevos usuarios que entran a la plataforma *on boarding*
2. **Datos Personales:** Completar el primer hito de datos, el más simple y el que desbloquea los otros hitos. Se separa del resto de los hitos, ya que se considera que si el *applicant* no completa este hito no estaba nada interesado en realizar el registro.
3. **Completado:** Se completa los 5 hitos del registro (Verificación Email, datos bancarios, antecedentes, otros datos e imagen de perfil)
4. **Adquisición skills:** haber tomado al menos una inducción
5. **Tomar Tareas Regular:** Haber postulado al menos una tarea en el mes en cuestión

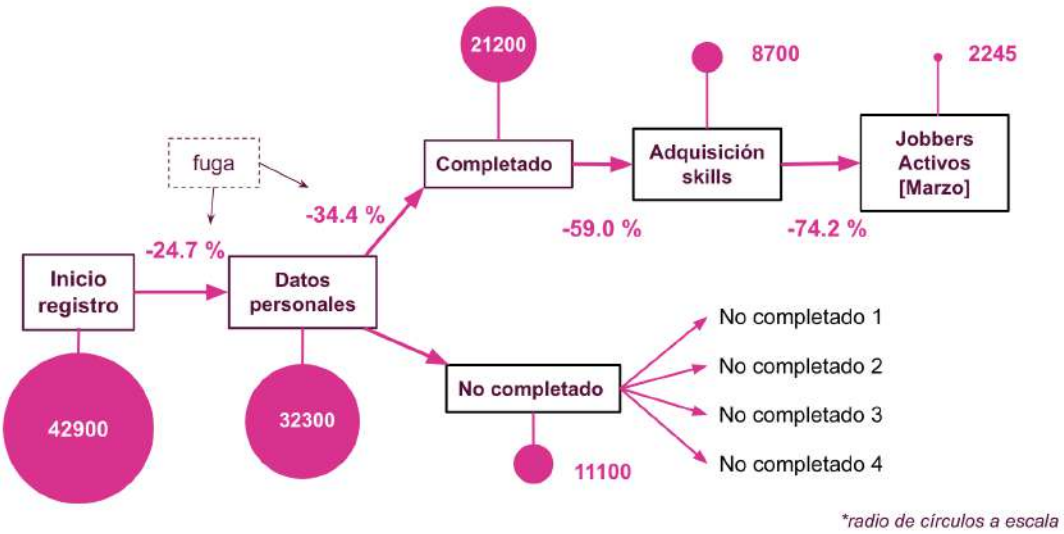

Figura 1.4: Número de jobbers y fuga de estos en porcentaje entre cada bloque del proceso *On Boarding*

Se observa que existe una gran fuga entre cada bloque que van desde los 25 % hasta un 74 %. Más aun si se toma el punto inicial Inicio Registro con más de 43 000 usuarios versus los jobber activos en marzo (2245) la fuga total es de un más de un 95 %. Es una fuga bastante importante considerando que existe un costo para traer usuarios a la plataforma a través de campañas de marketing.

A la fecha, existe un grupo de operadores que se encarga de llamar a jobbers para que terminen el registro, tomen inducciones y tomen tareas. Es ahí donde se quiere crear herramientas para enfocar las llamadas hacia las personas que sean más propensos a avanzar en el flujo y no estar llamando a aquellos que en definitivo no estén interesados en el servicio.

Se buscará aplicar el algoritmo entre los bloques Completados y Adquisición Skills dado que en el primer bloque ya se tiene suficiente información para predecir y por ser una solución eficiente al equipo operador de *on boarding*.

En la sección de No completados se realizó un *cluster* para agrupar las conductas de usuarios que no lograron completar el registro. El detalle de estos resultados se dejó en Anexo C.

1.2. Objetivos

1.2.1. Objetivo General

1. Diseñar e implementar un modelo de caracterización de usuario que logre clasificar la probabilidad de toma de inducciones utilizando herramientas de ciencia de datos

1.2.2. Objetivos Específicos:

- Comprender cualitativamente el proceso de registro e identificar los hitos de generación de data a través de herramientas de base de datos
- Caracterizar al usuario en los diferentes hitos del proceso de registro e identificar los hitos problemáticos utilizando herramientas de minería de datos
- Seleccionar y limpiar conjunto de datos apropiado para creación de modelos utilizando herramientas de pre procesamiento de datos
- Construir y testear algoritmos que logre predecir proyección del jobber en el flujo utilizando herramientas de *machine learning*
- Implementar prototipo funcional del modelo resultante que automatice la predicción de nuevos usuarios en tiempo real

1.3. Estructura del Documento

El documento comienza con esta introducción donde se intenta dar una contextualización de la empresa y las problemáticas actuales. En segundo lugar, se habla sobre el estado del arte que justifica el uso de las herramientas tanto de *on boarding* como de ciencia de datos. Posteriormente, se entra en Marco teórico, donde se abordan las herramientas y conceptos relevantes para el entendimiento particular del trabajo, en este caso *machine learning*. Se seguirá con la metodología que busca entender los pasos necesario de trabajo para abordar los problemas y solucionarlos. Se sigue con resultados y análisis donde se muestra todos los resultados relevantes de las experiencias y donde se analizan para su entendimiento. Luego, se habla sobre la implementación, donde solo se hace énfasis en la lógica del flujo. Para finalizar se presenta la conclusión donde se dice que se lograron los objetivos planteados y se da a conocer donde se recomienda proyectar los siguientes trabajos. Al final del documento se encuentran los Anexos que no son directamente relacionados con el enfoque del trabajo pero pueden aportar valor al lector.

Capítulo 2

Estado Del Arte

2.1. On Boarding

El concepto *On Boarding* es un término relativamente nuevo. Investigadores han estado investigando programas de orientación para nuevos empleados y técnicas de socialización por décadas. Los nuevos procesos parecen ser un proceso más centrado en la integración de nuevos integrante a nivel empresarial. El objetivo final de la incorporación es preparar a los integrantes para que tengan éxito en su trabajo lo más rápido posible. Existen dos indicadores claves de rendimiento en una incorporación exitosa:

1. Tiempo para la productividad
2. Compromiso y retención

On boarding puede considerarse dentro del contexto más amplio de socializar a los recién llegados a la organización. Históricamente, socialización es un término que se ha utilizado para describir el proceso en el que un individuo adquiere las aptitudes, comportamientos y conocimientos necesarios para participar con éxito como nuevo miembro de la organización.

La literatura sugiere que todo proceso *On Boarding* debe adaptarse considerando 6 puntos claves: Competencias de desempeño, personas, políticas de empresa, comunicación, visión y valores e historia de la empresa. [1]

En vista de ir mejorando el proceso se debe tener una buena percepción del tipo de usuario que va interactuando y al mismo tiempo cómo les está yendo en el registro. Vale decir, si existen hitos problemáticos o cuellos de botella que disminuyen el porcentaje de éxito. Dado que la mayoría de estos procesos son digitales, existen diversas formas de levantar herramientas de ayuda.

La recolección de datos del proceso se transforma en crucial para explorar y encontrar patrones en la información centralizada. No obstante, dar sentido a los datos está limitado por la memoria humana y la capacidad de atención del usuario. [2] . De aquí se origina la necesidad de adentrarse en formas de explotar los conceptos de *big data* utilizando data analítica.

2.2. Big Data

Llegando a 2022, el mundo se encuentra rodeado en la era de datos. En efecto, dentro de cualquier sector industrial, las herramientas de inteligencia de negocios impacta de manera directa en la toma de decisiones a cualquier escala. Sin ir muy lejos, la propia pandemia empujó de manera significativa la digitalización de empresas, obligándolas a recurrir a más herramientas de nube. Esto apunta a que el almacenamiento de información seguirá creciendo y que cada vez se genera más data recuperable en cualquier proceso. Se estima que menos del 3 % de la data almacenada es utilizada de manera efectiva.

Big data es un término que describe el gran volumen de datos (estructurados y no estructurados) que inundan una empresa en el día a día. Pero no es la cantidad de datos lo importante. Lo que importa es lo que las organizaciones hacen con los datos. El *big data* puede ser analizado para obtener *insights* que conlleven a mejores decisiones y acciones de negocios estratégicas. [3]

Big data empezó a ser utilizado en la década de 2000. Las primeras organizaciones en adoptar *Big Data* fueron en las empresas innovadoras de la época. El término se refiere a los datos que son tan grandes, rápidos o complejos que es difícil o imposible procesarlos con los métodos tradicionales. Según Davenport y Dyché , empresas como Google, eBay y Facebook fueron construido alrededor de *big data* desde el principio. Los macrodatos cambiaron la forma en que las empresas manipulaban los datos, proporcionando no solo nuevas oportunidades para manejar datos, pero también nuevas formas de usar y agregar valor a grandes cantidades de datos provenientes del Internet de las cosas (IoT), redes sociales, registros web y sensores.

Big Data también apoyan el suministro de datos como recurso que las organizaciones pueden utilizar. Los macrodatos también han llevado a la aparición de tecnologías modernas como los lagos de datos (*Data Lake*), que permiten a las empresas almacenar y manejar grandes volúmenes de datos estructurados y no estructurados en su formato nativo. Además de necesitar una base de datos consistente, un *Data Lake* es la herramienta definitiva para tener registro temporal de todos los datos a lo largo del tiempo. Si un dato es actualizado o sufre cambios, el *Data Lake* es el que lo va a poder percatar.

2.3. Inteligencia de Negocios

Inteligencia de Negocios o *Business Intelligence* (BI) es un enfoque moderno que combina metodologías, procesos, arquitecturas y tecnologías para transformar datos brutos en información relevante para cualquier toma de decisiones a nivel corporativo. BI puede desempeñar un papel vital en mejorar el desempeño organizacional, identificando nuevas oportunidades, revelando nuevos conocimientos empresariales y mejora de los procesos de toma de decisiones. Por lo tanto, BI se ha transformado en una prioridad máxima para organizaciones en la mayoría de las industrias modernas.

Tradicionalmente, BI se ha centrado principalmente en datos empresariales estructurados e internos, pasando por alto información potencialmente valiosa incrustada en datos no estructurados y externos. Esto podría resultar en una visión incompleta de la realidad y toma

de decisiones empresarial sesgada. El crecimiento acelerado y el desarrollo generalizado de las tecnologías de Internet, web y en la nube han proporcionado más énfasis en el significado de sobre información [4]

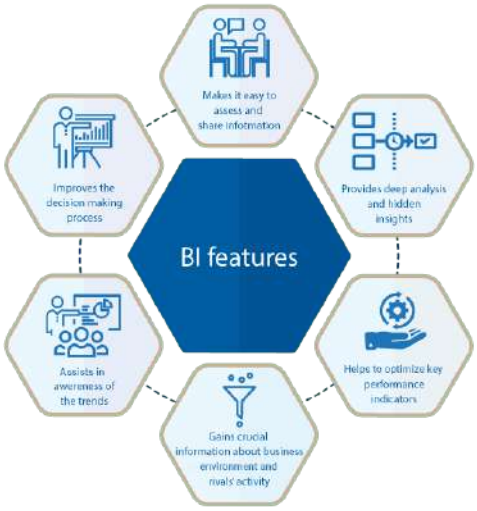

Figura 2.1: Herramientas de Inteligencia de Negocios

Estos avances tecnológicos han propiciado la generación de volúmenes y acumulaciones de datos sin precedentes. Los datos grandes y complejos a menudo se describen con el concepto de *Big data*. A medida que los macrodatos se vuelven cada vez más disponibles, el desafío de analizar conjuntos de datos grandes y crecientes es cada vez más urgente. Por lo tanto, BI enfrenta hoy nuevos desafíos, pero también oportunidades emocionantes.

2.4. Data Analítica

El análisis de datos o *Data Analytics* examina grandes cantidades de datos para descubrir patrones ocultos, correlaciones y otros conocimientos. Con la tecnología actual, es posible analizar sus datos y obtener respuestas de ellos casi de inmediato, un proceso que es más lento y menos eficiente con las soluciones de inteligencia de negocios más tradicionales. [5]

El análisis de data ayuda a las organizaciones a aprovechar sus datos y utilizarlos para identificar nuevas oportunidades. Eso, a su vez, conduce a movimientos comerciales más inteligentes, operaciones más eficientes, mayores ganancias y clientes más felices. Según estudios [5] mostraron que los trabajos de *Data Analytic* obtenían valor de las siguientes maneras:

1. Reducción de costo. La analítica basada en la nube brindan importantes ventajas de costos cuando se trata de almacenar grandes cantidades de datos, además de que pueden identificar formas más eficientes de hacer negocios.

2. Toma de decisiones mejor y más rápida. Con la velocidad de nube y los análisis en memoria, combinados con la capacidad de analizar nuevas fuentes de datos, las empresas pueden analizar la información de inmediato y tomar decisiones basadas en lo que han aprendido.

3. Nuevos productos y servicios. Con la capacidad de medir las necesidades y la satisfacción de los clientes a través de la analítica, surge el poder de darles a los clientes lo que quieren.

Figura 2.2: Estrategias Data Analítica

La Inteligencia de Negocios y la Analítica son semejantes, ya que ambas necesitan datos y presentan resultados en forma de reportes para apoyar a la toma de decisiones basadas en información. No obstante, son diferentes en el resultado y las decisiones que se pueden tomar con cada una: La Inteligencia de Negocios muestra solamente los datos del pasado y se puede reaccionar a lo que ya ocurrió, mientras que la Analítica usa los mismos datos del pasado para predecir lo que va a suceder, permitiendo decisiones que se anticipan a lo que va a pasar. Ambas son un complemento natural, ya que requieren datos y prepararlos para hacer sus análisis y presentar resultados. Los reportes pueden ser tanto de lo que ya ocurrió como de lo que se prevé que pasará.

Capítulo 3

Marco Teórico

3.1. Ciencia de Datos e Inteligencia Artificial

La Ciencia de Datos (o *Data Science*) es una ciencia que combina el método científico, matemáticas, programación, analítica avanzada para desenmascarar y explicar información oculta en data. [6]

La ciencia de datos alberga múltiples aspectos: abarca tanto la preparación de datos para su análisis y procesamiento, la realización de análisis de datos avanzados (data analítica por ejemplo) y la presentación de los resultados para revelar patrones y permitir que las partes interesadas saquen conclusiones respaldadas.

La preparación de datos puede implicar su limpieza, agregación y manipulación para que estén listos para tipos específicos de procesamiento. El análisis requiere el desarrollo y uso de algoritmos, análisis y modelos de inteligencia artificial. Está impulsado por un software que revisa los datos para encontrar patrones en su interior para transformar estos patrones en predicciones que respalden la toma de decisiones comerciales. La precisión de estas predicciones debe validarse mediante pruebas y experimentos diseñados científicamente. Y los resultados deben compartirse mediante el uso hábil de herramientas de visualización de datos que permitan que cualquiera pueda ver los patrones y comprender las tendencias.

La Inteligencia Artificial, por su parte, aprovecha las computadoras y las máquinas para imitar las capacidades de resolución de problemas y toma de decisiones de la mente humana. [7], la Inteligencia Artificial, a diferencia de la Ciencia de Datos es una área más genérica y abarca más disciplinas que las ingeniería y ciencias naturales. En efecto, profesionales de psicología, filosofía, antropología, entre otros, aportan de igual manera a la investigación de dicha disciplina.

3.1.1. Minería de Datos

Históricamente, el concepto de encontrar patrones útiles en datos ha recibido una variedad de nombres, incluida: extracción de conocimientos, descubrimiento de información, información recolección, arqueología de datos y patrón de datos procesados. El término minería de datos ha sido utilizado por estadísticos, analistas de datos y los sistemas de información de gestión (MIS) . También ha ganado popularidad en el campo de la base de datos en los últimos años [8].

La minería de datos es en esencia la ciencia de extraer conocimiento útil de grandes bases de datos utilizando todas las herramientas disponibles de *machine learning*. Su base interdisciplinaria es la informática. Las técnicas de minería de datos se han aplicado ampliamente a problemas en la industria, la ciencia, la ingeniería y el gobierno, y se cree ampliamente que la minería de datos tendrá un impacto profundo en nuestra sociedad. El creciente consenso de que la minería de datos puede aportar un valor real ha llevado a una explosión en la demanda de tecnologías novedosas de minería de datos y de estudiantes capacitados en minería de datos; los estudiantes que comprenden las técnicas de minería de datos pueden aplicarlas a problemas de la vida real. y están capacitados para la investigación y el desarrollo de nuevos métodos de minería de datos. [9]

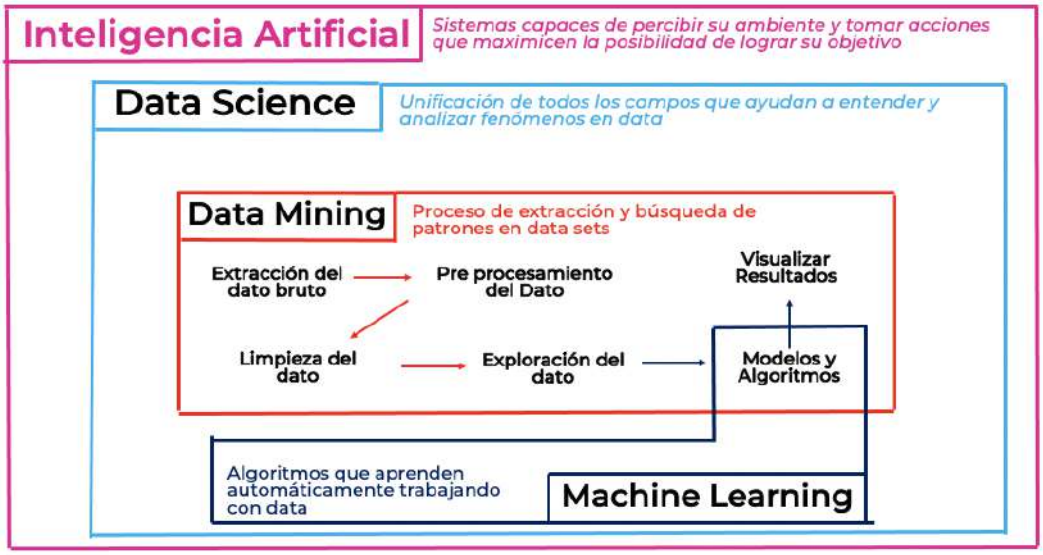

Figura 3.1: Diagrama Inteligencia Artificial - Ciencia de Datos

3.1.2. Aprendizaje de Maquina

El Aprendizaje de Máquinas, Aprendizaje Automatico o *Machine Learning* es el estudio de técnicas o algoritmos informáticos que mejoran con la experiencia. Ésta disciplina es un sub-campo de la Inteligencia Artificial, que busca definir modelos matemáticos que permitan resolver una tarea mediante la optimización de una función objetivo, por medio de un proceso de aprendizaje sobre un conjunto de datos de entrenamiento [10].

El fin de estos modelos es que, una vez entrenados, permitan resolver tareas sin haber programado explícitamente las reglas que le llevan a tomar sus decisiones. Las tareas a resolver por lo general resultan ser problemas sin solución algorítmica completamente satisfactoria. Algunas tareas de esta índole pueden ser, por ejemplo, construir un automóvil que se maneje completamente solo, clasificar entre imágenes de perros y gatos, o generar imágenes realistas de rostros humanos.

La diferencia de aprendizaje automático con programación tradicional es que para crear un modelo de *machine learning*, el programador tiene que crear un modelo que logre aprender por su propia cuenta los patrones que se están buscando. Por ejemplo, un programador puede entregar data etiquetada con tal que el programa saque sus propias conclusiones. En el modelo

tradicional, el programador solo daría reglas, para que el código identifique las condiciones ya predeterminadas.

3.1.3. Tipos de Aprendizaje

Dentro del campo del aprendizaje de máquina, existen dos tipos de aprendizajes: el supervisado y no supervisado. La principal diferencia entre los dos tipos es que el aprendizaje supervisado se estructura utilizando muestras previamente etiquetadas. El aprendizaje no supervisado, por su lado, no las tiene, por lo que su objetivo es inferir la estructura natural presente dentro de un conjunto de puntos de datos.

El aprendizaje supervisado se realiza normalmente en el contexto de la clasificación, cuando se quiere mapear la entrada a las etiquetas de salida, o la regresión, cuando se quiere mapear la entrada a una salida continua. Los algoritmos comunes en el aprendizaje supervisado incluyen regresión logística, *Naive Bayes*, *Support Vector Machine*, redes neuronales artificiales y *Random Forests*. Tanto en la regresión como en la clasificación, el objetivo es encontrar relaciones o estructuras específicas en los datos de entrada que permitan producir de forma eficaz los datos de salida correctos. [11]

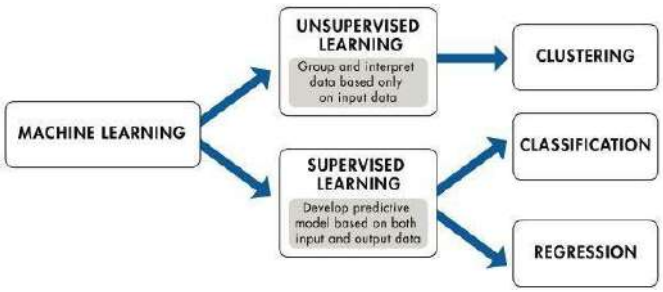

Figura 3.2: Tipos de Aprendizaje en *Machine Learning*

El aprendizaje no supervisado está enfocado en agrupación, aprendizaje de representación y/o estimación de densidad. En todos estos casos, se desea conocer la estructura inherente de los datos sin utilizar etiquetas proporcionadas explícitamente. Esto resulta muy útil en análisis exploratorio dado que se puede identificar la estructura interna de los datos. Por ejemplo, si un analista intentara segmentar a los consumidores, los métodos de agrupación no supervisados serían un excelente punto de partida para su análisis. En situaciones en las que es imposible o poco práctico para un ser humano proponer tendencias en los datos, el aprendizaje no supervisado puede proporcionar conocimientos iniciales que luego pueden usarse para probar hipótesis individuales.

3.2. Algoritmos de Aprendizaje de Maquina

El aprendizaje de máquinas recae en un computador que observa datos, construye un modelo basado en esos datos y utiliza ese modelo a la vez como una hipótesis acerca del mundo y una pieza de software que puede resolver problemas [12]. En esta sección se abordarán los algoritmos más útiles en el ámbito de aprendizaje de máquinas.

3.2.1. Red Neuronal

Una red neuronal es una estructura consistente en un conjunto de unidades de cómputo a las que llamamos neuronas artificiales o perceptrones.

3.2.1.1. Perceptrón

Un perceptrón intenta simular el comportamiento de una neurona biológica. Un perceptrón recibe como input un vector de dimensión finita. El resultado entregado consiste en la aplicación de una función a la suma ponderada de los valores del vector. A la función aplicada le llamamos función de activación. A los valores por los que ponderamos al vector de entrada les llamamos parámetros, y son modificados en la etapa de entrenamiento de la red.

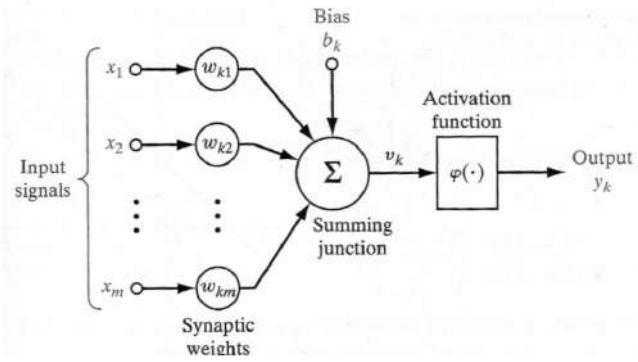

Figura 3.3: Neurona simplificada perceptron

En el flujo de imagen en Figura 3.3 los pesos del perceptrón se denotan $w_{k1}, w_{k2}, ...w_{km}$ m las entradas son los valores $x_1, x_2, ...x_m$. El *bias* (o parcialidad) externo es denotado b_k. La suma v_k llamada *hard limiter input* viene dada por

$$v_k = \sum_{i=1}^{m} w_{ki} x_i + b \tag{3.1}$$

El objetivo del perceptrón es clasificar las variables de entrada $\vec{x}$ en la clase que corresponda. Si se piensa en un problema binario de clasificación, el perceptrón tendría el objetivo de tener una salida +1 para clasificar la clase C_1 y un -1 para clasificar la clase C_2. Si se simplifica aun más el ejemplo, diciendo que hay $m = 2$ entradas, se busca el hiperplano separador:

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{3.2}$$

donde el umbral de decisión vendría dada por una linea recta. Cada valor del perceptrón ayuda a encontrar el mejor hiperplano separador de clases:

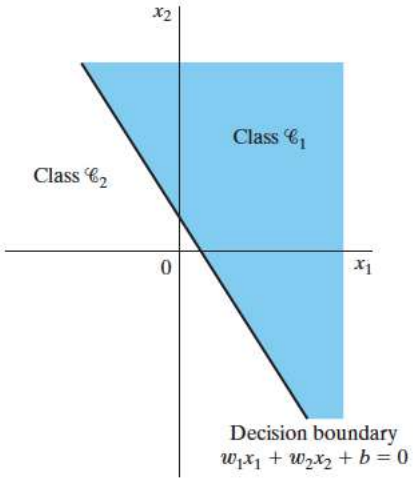

Figura 3.4: Ilustración de hiperplano como umbral de decisión para un problema bi-dimensional, con dos clases a clasificar

Para que el perceptrón funcione correctamente, las dos clases c1 y c2 deben ser linealmente separables. Esto, a su vez, significa que los patrones a clasificar deben estar suficientemente separados entre sí para asegurar que la superficie de decisión consiste en un hiperplano. Para simplificar los cálculos, se toma $w_0 = b$ y $x_0 = 1$ para agregar la parcialidad (*bias*) como entrada de $\vec{x}$.

Siguiendo el ejemplo,se supone que las variables de entrada del perceptrón se originan a partir de dos clases linealmente separables. Sea H_1 el subespacio de los vectores de entrenamiento $\vec{x_k}$ que pertenecen a la clase C_1, y sea H_2 el subespacio de los vectores de entrenamiento $\vec{x_k}$ que pertenecen a la clase C_2. La unión de H_1 y H_2 es el espacio completo denotado por H. dados los conjuntos de los vectores H_1 y H_2 para entrenar el clasificador, el proceso de entrenamiento involucra el ajuste del vector de peso w de tal manera que las dos clases C_1 y C_2 son linealmente separables. Es decir, existe un vector de pesos w tal que se puede afirma:

- $w^T x > 0$ para cada vector $\vec{x}$ perteneciente a la clase C_1
- $w^T x \leq 0$ para cada vector $\vec{x}$ perteneciente a la clase C_2

En la segunda línea del ítem anterior, se ha optado arbitrariamente por decir que el vector de entrada x pertenece a la clase $w^T x = 0$. Dados las características de lo planteado, el problema se resume en encontrar los pesos w que satisfagan la expresión anterior. El algoritmo para adaptar el vector de peso del perceptrón elemental puede formularse de la siguiente manera:

1. Si el n-ésimo miembro del conjunto de entrenamiento, $x(n)$, se clasifica correctamente por el peso vector $w(n)$ calculado en la n-ésima iteración del algoritmo, no se hace ninguna corrección al vector de peso del perceptrón de acuerdo con la regla:

$$\begin{aligned} w(n+1) &= w(n) \text{ si } w^T(n)x(n) > 0 \text{ y } x(n) \text{ pertenece a la clase } C_1 \\ w(n+1) &= w(n) \text{ si } w^T(n)x(n) \leq 0 \text{ y } x(n) \text{ pertenece a la clase } C_2 \end{aligned} \tag{3.3}$$

2. De lo contrario, el vector de peso del perceptrón se actualiza de acuerdo con la regla

$$\begin{aligned} w(n+1) &= w(n) - \mu x(n) \text{ si } w^T(n)x(n) > 0 \text{ y } x(n) \text{ pertenece a la clase } C_1 \\ w(n+1) &= w(n) + \mu x(n) \text{ si } w^T(n)x(n) \leq 0 \text{ y } x(n) \text{ pertenece a la clase } C_2 \end{aligned} \tag{3.4}$$

Donde $\mu(n)$ se denota como la tasa de aprendizaje o *learning rate* que controla el ajuste aplicado a los pesos en la iteración n. Se puede demostrar que tomando $\mu = 1$ el problema converge a una solución. Si se toma $w(0) = 0$ existe un n_{max} tal que n no puede ser más grande que él dado por:

$$n_{max} = \frac{\beta ||w_0||^2}{\alpha} \tag{3.5}$$

con

$$\begin{aligned} \alpha &= \min_{x(n) \in H_1} \; w_o^T x(n) \\ \beta &= \max_{x(k) \in H_1} \; ||x(k)||^2 \end{aligned} \tag{3.6}$$

Tomando $||.||$ como norma euclidiana y w_o como solución existente dado que las clases son linealmente separables. Para computar el problema se define la respuesta de salida $y(n)$:

$$y(n) = sgn[w^T(n)w(n)] \tag{3.7}$$

con

$$sgn(v) = \begin{cases} +1 \text{ si } v > 0 \\ -1 \text{ si } v < 0 \end{cases} \tag{3.8}$$

llamada función de activación. De la misma manera se introduce la respuesta deseada, es decir la clase que se busca predecir para la entrada expresada como:

$$d(n) = \begin{cases} +1 \text{ si } x(n) \text{ pertenece a la clase } C_1 \\ -1 \text{ si } x(n) \text{ pertenece a la clase } C_2 \end{cases} \tag{3.9}$$

De esta forma, al iterar el algoritmo, se calcula el *error-correction learning rule*

$$w(n+1) = w(n) + \mu[d(n) - y(n)]x(n) \tag{3.10}$$

que resulta más directo que hacerlo por casos como se mencionó anteriormente. La diferencia de $d(n)$ con $y(n)$ se define como la función de costo y en este caso solo se definió como una diferencia dado lo simple del ejemplo. No obstante, para casos más complejos es recomendable utilizar otro tipo de funciones.

La función de costo en sí es una función que permite la aplicación de un búsqueda de gradiente. Específicamente, se define la función de costo del perceptrón como:

$$J(w) = \sum_{x(n) \in H} (-w^T x(n) d(n)) \tag{3.11}$$

donde H es el conjunto de x mal clasificados. Si no existe mal clasificados, entonces $J(w) = 0$. Como la función de costo tiene a w como variable que se itera, se busca el vector gradiente:

$$\nabla J(w) = \sum_{x(n) \in H} (-x(n)d(n))) \tag{3.12}$$

donde el operador gradiente viene dado por:

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m} \right]^T \tag{3.13}$$

Finalmente, se introduce el método del *steepest descent*, donde el ajuste al vector de peso w en cada paso de tiempo del algoritmo se aplica en una dirección opuesta al vector gradiente. [13]

$$w(n+1) = w(n) - \mu \nabla J(w) \tag{3.14}$$

3.2.1.2. Perceptrón Multicapa

En la sección anterior, se definió el perceptrón de Rosenblatt, que es básicamente un sistema neuronal de una sola capa. No obstante, la red se limita a la clasificación de patrones linealmente separables. Para superar las limitaciones prácticas del perceptrón se busca crear la red conocida como perceptrón multicapa o *mutilayer perceptron*.

Las ventajas de un perceptrón multicapa son variadas: incluir funciones de activación no lineal diferenciables y poseer capas ocultas (*hidden layers*) con alto grado de conectividad para llegar a soluciones en problemas más complejos.

A continuación se muestra la red definida como *fully connected*. Esto significa que una neurona en cualquier capa de la red está conectado a todas las neuronas (nodos) en la capa anterior :

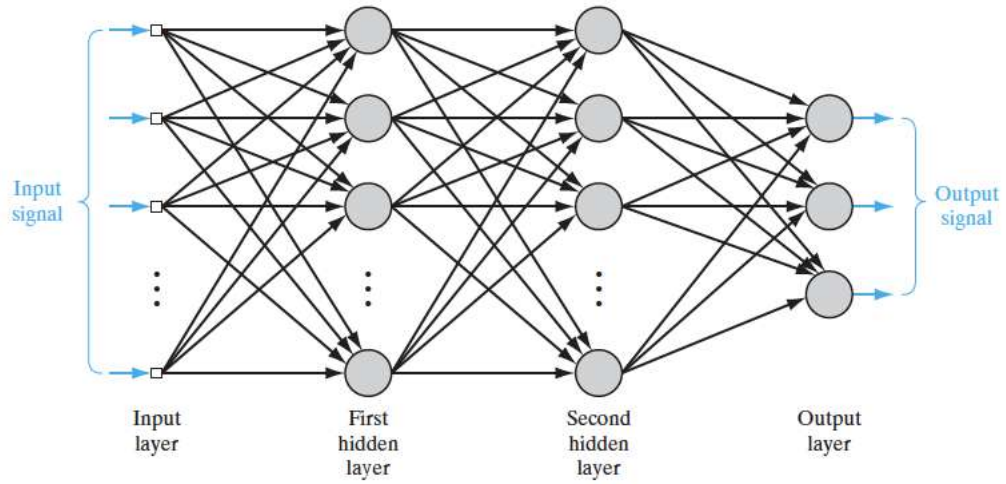

Figura 3.5: Arquitectura gráfica de un perceptrón multicapa con dos *hidden layers*

En la arquitectura de un perceptrón multicapa existen dos direcciones de flujo: la función señal, es la que viene desde la entrada de la red y se mueve en sentido *forward*. La señal error, por su parte, viene desde la final de la red, es la que calcula el error de la salida con lo que

la red espera. Esta señal va en sentido *backward*.

El flujo del algoritmo funciona de la siguiente manera. Se comienza inicializando los pesos y los *threshold* con una distribución uniforme en promedio 0. Luego, Dado un set de entrenamiento, se toma un época del set para comenzar a realizar una iteración.

Se realiza la *forward propagation* que busca calcular el output de la red. Considerando que al ser *multilayer* se tiene que hacer el cálculo en cada nodo hacia adelante. Para una neurona j en la capa l se tiene:

$$v_j^{(l)}(n) = \sum_{i=1} w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \tag{3.15}$$

Donde $y_i^{(l-1)}(n)$ es el output de la señal de la neurona i en la capa anterior $l-1$ en la iteración n. $w_{ji}^{(l)}$ por su parte es el peso asociado a la neurona j en la capa l alimentada por la neurona i de la capa $l-1$. Para $i=0$, se tiene $w_{j0}^{(l)}(n) = b_j^{(l)}(n)$ la parcialidad aplicada a la neurona j en la capa l. En cada red se tiene que elegir la función de activación a la salida de capa capa. De esta manera la salida de la neurona j en la capa l es :

$$y_j^l = \varphi_j(v_j(n)) \tag{3.16}$$

Con φ la función de activación. Si la nuerona j es el la capa de salida (*output layer*), es decir $l = L$ con L como la prodfunidad de la red se define:

$$y_j^L = o_j(n) \tag{3.17}$$

que se utiliza para computar el error de la señal:

$$e_j(n) = d_j(n) - o_j(n) \tag{3.18}$$

Con $d_j(n)$ el j-esimo elemento de la respuesta esperada $d(n)$. El error $e_j n$ depende del tipo de error que se escogerá. Existen diversos funciones de error para estas aplicaciones.

Posteriormente, se realiza el *Backward Computation* que busca actualizar los pesos para llegar a las salidas esperadas. Para esto, se calcula el gradiente local $\delta_j(n)$ definido por

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)} \varphi_j^{'}(v_j^{(L)}(n)) & \text{para la neurona } j \text{ en salida de la capa } L \\ \varphi^{'}(v_j^l(n)) \sum_k \delta_k^{(l+1)}(n) \omega_{kj}^{(l+1)}(n) & \text{para la neurona } j \text{ en la capa oculta } l \end{cases} \tag{3.19}$$

Siguiendo la misma lógica que el perceptrón, se debe actualizar los pesos pero considerando la conectividad de toda la red para orientar hacia los resultados que se espera. Siguiendo la álgebra se llega en definitiva a la siguiente expresión:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[\Delta w_{ji}^{(l)}(n-1)] + \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n) \tag{3.20}$$

con η el coeficiente de aprendizaje y α la constante de momentum. Se procede a iterar los pasos *forward* y *backward propagation* hasta alcanzar algún criterio de detención que puede ser de número de iteraciones máximas, como error mínimo esperado. [13]

3.2.2. Árboles y Bosques

3.2.2.1. Árboles de Decisión

El árbol de decisión es una herramienta poderosa y popular de clasificación y regresión en aprendizaje supervisado de *machine learning*. Un árbol de decisión es un diagrama de flujo como una estructura de árbol, donde cada nodo interno denota una prueba en un atributo, cada rama representa un resultado de la prueba y cada nodo hoja (nodo terminal) tiene una etiqueta de clase. [14]

La Figura 3.6 (a) ilustra un modelo de árbol de decisión simple que incluye una única variable de destino de 5 dimensiones ($R_1, ...R_5$) y dos variables continuas de entrada, x1 y x2, que van desde 0 a 1. Los componentes principales de un modelo de árbol de decisión son nodos y ramas y los pasos más importantes en la construcción de un modelo es dividir, detener y podar. [15]

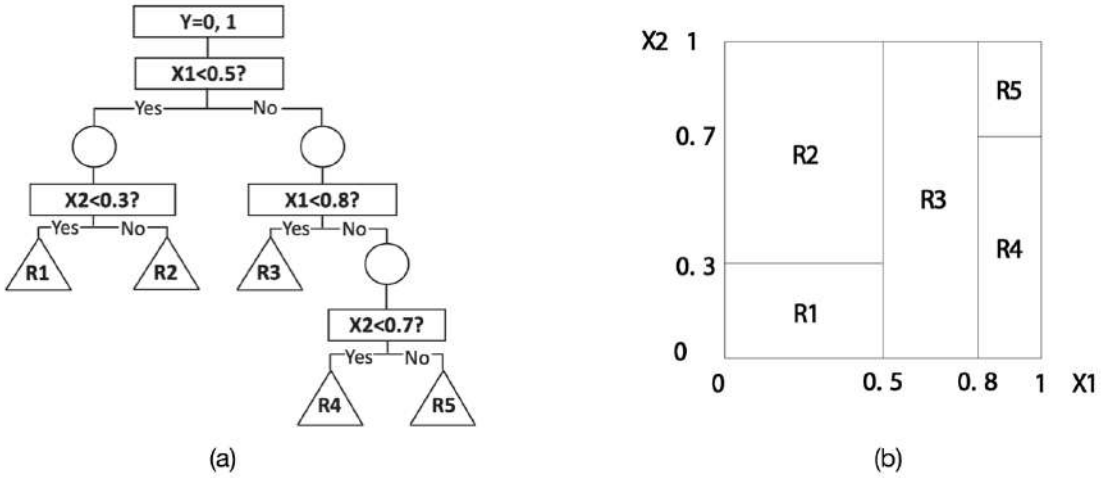

Figura 3.6: (a): Árbol de decisión simple basado en variable binaria, (b): Árbol de decisión ilustrado usando vista de espacios

Un árbol de decisión se sitúa en un espacio X de observaciones en la misma cantidad de regiones que hay hojas o nodos como se muestra en figura 3.6 (b). En una misma región, las observaciones reciben la misma etiqueta. Si se tienen n observaciones $\vec{x_1}, \vec{x_2}, ..., \vec{x_n}$ de X etiquetadas por por $\vec{y_1}, \vec{y_2}, ..., \vec{y_n}$ y R regiones $R_1, R_2, ...R_R$ el problema de dividir las regiones se define como:

$$f(\vec{x}) = \sum_{r_1}^{R} \delta_{\vec{x} \in R_r} \underset{c=1,...,C}{\arg\max} \sum_{i:\vec{x} \in R_r} \delta(y^i, c) \tag{3.21}$$

Con c como la clasificación del algoritmo y δ como la función Dirac definida por:

$$\delta \ : \ X \times X \to \{0,1\}$$
$$\delta(u,v) \to \begin{cases} 1 \text{ si } u = v \\ 0 \text{ si no} \end{cases} \tag{3.22}$$

Para ir armando los nodos de los arboles se tiene que levantar criterios de separación. Se dice que la data en el nodo m representado por R_m con n_m *samples*. Para cada candidato $\theta = (j, s_m)$ con j como *splitting variable* y s_m como *splitting point* o *threshold* define dos regiones regiones a las cuales se separa el conjunto de clasificación.

Si se tiene el caso de separación binaria, esta regiones se definen como:

$$\begin{aligned} R_m^{left}(\theta) &= \{\vec{x} : x_j = 0\} \\ R_m^{right}(\theta) &= \{\vec{x} : x_j = 1\} \end{aligned} \tag{3.23}$$

Para el caso de valores reales con clasificación en más dimensiones los nodos se definen como:

$$\begin{aligned} R_m^{left}(\theta) &= \{\vec{x} : x_j < s_m\} \\ R_m^{right}(\theta) &= \{\vec{x} : x_j \geq s_m\} \end{aligned} \tag{3.24}$$

La calidad del candidato del nodo m se encuentra computando alguna función de costo o impureza $Imp()$ dependiendo del usuario que lo configure. Para el nodo m se define:

$$G(R_m, \theta) = \frac{n_m^{left}}{n_m} Imp(R_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} Imp(R_m^{right}(\theta)) \tag{3.25}$$

y se busca seleccionar el parámetro que minimice la impureza:

$$\theta^* = \arg\min_{\theta} G(R_m, \theta) \tag{3.26}$$

Se itera recursivamente hasta que se encuentra el máxima profundidad posible, $n_m <$min$_{samples}$ o $n_m = 1$

La función de impureza, como se dijo anteriormente, puede tomar diferentes configuraciones. Para definiralas, se comienza por escribir la notación:

$$p_c(R_m) = \frac{1}{n_m} \sum_{i:\vec{x} \in R_m} \delta(y^i, c) \tag{3.27}$$

Que permite evaluar la clasificación. En el caso más simple, se puede definir la proporción de ejemplos que no pertenece a la clase mayoritaria:

$$Imp(R_m) = 1 - \arg\max_{c=1,\ldots,C} p_c(R_m) \tag{3.28}$$

Posteriormente, si se quiere tener un criterio de separación que maximice el *gain infor-*

mation, se utiliza la entropía cruzada. El objetivo es minimizar la cantidad de información adicional para etiquetar las clases.

$$Imp(R_m) = -\sum_{c=1}^{C} p_c(R_m)\log_2 p_c(R_m) \tag{3.29}$$

Finalmente si se busca cuantificar la probabilidad que un ejemplo del juego de entrenamiento sea mal etiquetado habiendo estado mal etiquetado anteriormente en función de la distribución se utiliza la impuridad de *Gini* [16] [17]:

$$Imp(R_m) = \sum_{c=1}^{C} p_c(R_m)(1 - p_c(R_m)) \tag{3.30}$$

Los árboles de decisión suelen ser útiles al momento de manejar datos de gran dimensión. Los puntos fuertes de los métodos de árbol de decisión son:

- Pueden generar reglas comprensibles.
- Realizan la clasificación con poco gasto computacional.
- Pueden manejar variables continuas y categóricas.
- Proporcionan una indicación clara de qué campos son más importantes para la predicción o la clasificación.

Por el otro lado, las debilidades son:

- Son menos apropiados para tareas de estimación donde el objetivo es predecir el valor de un atributo continuo.
- Son propensos a errores en los problemas de clasificación con muchas clases y un número relativamente. pequeño de ejemplos de entrenamiento
- Pueden ser computacionalmente costoso de entrenar.

3.2.2.2. Random Forest

El *bootstrap* es una herramienta general para evaluar la precisión estadística. Parecido a validación cruzada (método para estimar el error extra esperado), el *bootstrap* puede estimar tanto el error condicional Err_T como el error de predicción esperado Err.

Suponga que se tiene un modelo a ajustarse a un conjunto de datos de entrenamiento. Se dice que $Z = (z_1, z_2, ..., z_N)$ es el set de entrenamiento donde $z_i = (x_i, y_i)$. La idea es simplemente calcular los conjuntos de datos de testeo con los datos de entrenamiento, cada uno respetando las dimensiones del problema. Si se hace $B = 100$ veces, se dice que se tiene un *bootstrap* de dimensión B.

Se define S(Z) como el problema calculado a partir de los datos Z, por ejemplo, la predicción en algún punto de entrada. Del muestreo bootstrap se puede sacar su varianza:

$$\hat{\mathrm{Var}}[S(Z)] = \frac{1}{B-1}\sum_{b=1}^{B}(S(Z^{*b}) - S^*))^2 \tag{3.31}$$

Donde $\overline{S}^* = \sum_b \frac{S(Z^{*b})}{B}$. Luego se busca hacer promedio *bagging* del *bootrsap* para aumentar la precisión de la muestra disminuyendo su varianza. Si $\hat{f}^{*b}(x)$ es la predicción del modelo *bootstrap*, entonces el *bagging* estimado es: [18]

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{3.32}$$

La idea esencial en el *bagging* es promediar muchos modelos ruidosos pero de bajo sesgo y, por lo tanto, reducir la varianza. Los árboles de decisión son candidatos ideales para hacer *bagging*, ya que pueden clasificar estructuras complejas. Si crecen lo suficientemente, los arboles suelen tener baja parcialidad. Dado que estos son notoriamente ruidosos, se benefician enormemente del promedio. Además, dado que cada árbol generado en *bagging* se distribuye de forma idéntica, la expectativa de un promedio de B tales árboles es la misma que la expectativa de cualquiera de ellos.

De estos conceptos nacen los bosques aleatorios o *Random Forest*. Estos tienen la ventaja de no poseer tantos hiperparámetros como lo podría ser una red neuronal. Lo que se tiene que tener en cuenta son el número de variables consideradas en cada nodo, el número de observaciones utilizadas para cada árbol (n en el procedimiento que hemos descrito, pero que podría reducirse), el máximo número de observaciones en las hojas del árbol y el número de árboles, siempre que sea lo suficientemente grande. [17]

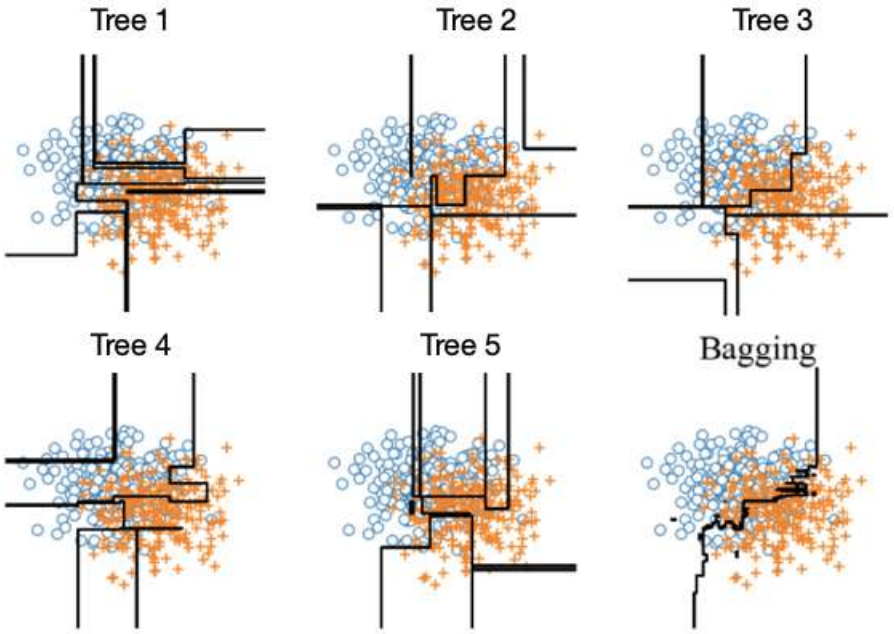

Figura 3.7: Desempeño en un conjunto de prueba de un clasificador entrenado por *bagging* y los 5 primeros árboles que la componen.

3.2.2.3. AdaBoost

AdaBoost, cuyo nombre proviene de *Adaptive Boosting*, es un algoritmo que permite construir un clasificador de forma iterativa, obligándolo a centrarse en los errores del modelo gracias a a un sistema de ponderación de ejemplo de entrenamiento.

Se dice que $D = \{(\vec{x}^{\,i}, y^i)\}_{i=1,..,n}$ es un conjunto de datos de clasificación binaria, M un número de iteraciones y un algoritmo de aprendizaje. Dado un conjunto de datos S, se denota f_S la función de decisión que entrega el algoritmo. Se define $Y = \{-1, 1\}$ y se dice que f_S pertenece a Y.

Luego, se define el conjunto de datos ponderados $D' = \{(w_i, \vec{x}^{\,i}, y^i)\}_{i=1,..,n} \in \mathbb{R}^n \times X^n \times Y^n$ de un conjunto de datos $\{(\vec{x}^{\,i}, y^i)\}_{i=1,..,n}$ con un peso w_i afectado por la i-esima observación. Se asume que el algoritmo de aprendizaje que se usa es capaz de integrar estos pesos. En el caso de los árboles de decisión, la ponderación de los ejemplos de aprendizaje se refleja en su ponderación en el criterio de impureza; la decisión también se toma por mayoría ponderada de votos.

El algoritmo AdaBoost funciona de la siguiente manera. Se inicializan todo los pesos $w_i^1 = 1/N$ para $i = 1, ..n$. Iterando desde $m = 1, .., M$, se calcula la función de decisión f_m dado el conjunto de datos $D'_m = \{(w_i^m, \vec{x}^{\,i}, y^i)\}_{i=1,..,n}$. Luego se calcula el error ponderado:

$$\epsilon_m = \sum_{i=1}^{m} w_i^m \delta(f_m(\vec{x}^{\,i}), y^i) \tag{3.33}$$

Con δ función Dirac definida en 3.22. A través del error, se puede sacar la confianza notada α_m expresada por:

$$\alpha_m = \frac{1}{2} log(\frac{1 - \epsilon_m}{\epsilon_m}) \tag{3.34}$$

Entre más pequeño es el error del modelo, más grande es la confianza α_m. El último paso iterativo es actualizar los pesos. El algoritmo sugiere hacerlo de la siguiente manera:

$$w_i^{m+1} = \frac{1}{Z_m} w_i^m exp(-\alpha_m y^i f_m(\vec{x}^{\,i})) \tag{3.35}$$

donde

$$Z_m = \sum_{l=1}^{n} w_l^m exp(-\alpha_m y^l f_m(\vec{x}^{\,l})) \tag{3.36}$$

Lo que busca Z_m es que la suma de los coeficientes $\sum_{i=1}^{n} w_i^{m+1} = 1$. Al terminar las M iteraciones, se retorna la función de decisión final [17] definida por

$$f \; : \; \vec{x} \rightarrow \sum_{m=1}^{M} \alpha_m f_m(\vec{x}) \tag{3.37}$$

3.2.2.4. Gradient Boosting

El algoritmo AdaBoost se originó originalmente desde un punto de vista muy diferente perspectiva que la presentada en la sección anterior. Unos 5 años después, se descubrió que era un caso particular de una técnica llamada *gradient boosting*. Esta generalizaba el concepto aplicando diferentes funciones de costo (donde en *AdaBoost* solo se usaba la función Dirac δ) [18].

Para entender *Gradient Boosting* se empieza por definir otra función de costo como lo podría ser el Error exponencial:

$$\begin{aligned} L &: \{-1,1\} \times \mathbb{R} \to \mathbb{R} \\ L(y, f(\vec{x})) &\to e^{-yf(\vec{x})} \end{aligned} \tag{3.38}$$

Si se dice que F_m es la función de decisión acumulada $F_m : \vec{x} \to \sum^m \alpha_k f_k(\vec{x})$. En la etapa m el error exponencial de F_m en el set de entrenamiento vale:

$$\begin{aligned} E_M &= \frac{1}{n}\sum_{i=1}^{n} exp\Big(- y^i \sum_{k=1}^{m-1} \alpha_k f_k(\vec{x}^{\,i})\Big) exp(\alpha_m y^i f_m(\vec{x}^{\,i})) \\ &= \frac{1}{n}\sum_{i=1}^{n} exp\Big(- y^i F_{m-1}(\vec{x}^{\,i})\Big) exp(\alpha_m y^i f_m(\vec{x}^{\,i})) \end{aligned} \tag{3.39}$$

Si se define ahora el peso $w_i^m = exp(-y^i F_{m-1}(\vec{x}^{\,i}))$, entonces la expresión anterior queda

$$E_m = \frac{1}{n}\sum_{i=1}^{n} w_i^m exp(\alpha_m y^i f_m(\vec{x}^{\,i})) \tag{3.40}$$

Este error es mínimo cuando α_m tiene la forma dada por la ecuación 3.34. Por lo tanto, *AdaBoost* combina la aprendices débiles para minimizar, en cada paso, el error exponencial del clasificador global. El error exponencial se puede reemplazar por otra función de costo, como la entropía cruzada o error al cuadrado. [17]

3.2.3. Naives Bayes

Naives Bayes es un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales. Naive Bayes es el clasificador de texto estándar por defecto. Es una herramienta que funciona bien en casos particulares, pero es importante poder identificar cuándo es efectiva y cuándo otras técnicas son más apropiadas [19].

3.2.3.1. Clasificador Bayesiano

Los clasificadores bayesianos asignan la clase más probable a un determinado ejemplo descrito por su vector de características. El aprendizaje de tales clasificadores se puede simplificar enormemente asumiendo que las características son clases independiente:

$$P(X|C) = \prod_{\text{í}=1}^{N} P(X_i|C) \tag{3.41}$$

donde $\vec{X} = (x_1, ..., x_n)$ es un vector de características y $\vec{C} = (c_1, ..., c_n)$ las clases asociadas. Todo vector de características puede tomar algunas de las k clases que existan, vale decir $c_i \in 1, ..., k$.

El clasificador de Bayes es un modelo de probabilidad de parámetro híbrido en esencia:

$$P(c_i|\vec{X}) = \frac{P(c_i)P(\vec{X}|c_i)}{P(\vec{X})} \tag{3.42}$$

Donde $P(c_i)$ es información previa de la probabilidad de aparición de clase c_i, $P(\vec{X})$ es la información de las observaciones, que es el conocimiento del propio vector a clasificar, y $P(\vec{X}|c_i)$ es el probabilidad de distribución del documento $\vec{X}$ en el espacio de clases. El Clasificador Bayesiano consiste en integrar esta información y computar por separado a posteriori del vector $\vec{X}$ correspondiente a cada clase, y asignar el documento a la clase con mayor probabilidad, es decir:

$$c^*(\vec{X}) = \arg\ \max_i P(\vec{X}|c_i) \tag{3.43}$$

Como se asume que las características son independientes y dado el hecho que $P(\vec{X})$ es igual a su clase c_i, la expresión queda de la siguiente forma:

$$c^*(\vec{X}) = \arg\ \max_i P(c_i) \prod_{\text{í}=1}^{N} P(X_i|C) \tag{3.44}$$

De donde nace el algoritmo naives Bayes [20]. Como la expresión queda basada en funciones de probabilidad, se puede variar tanto las funciones de probabilidad como Bernouilli o Gausiana entre otros en busca de diferentes desempeños.

3.2.4. Métricas de Desempeño

Una gran parte de los algoritmos de *machine learning* de clasificación siguen siendo cajas negras complejas para los usuarios finales, e incluso los mismos expertos a menudo no pueden comprender fundamentalmente sus decisiones. La falta de la transparencia de dichos sistemas puede tener graves consecuencias o un mal uso de recursos limitados y valiosos en diagnóstico médico, toma de decisiones financieras y en otros dominios de alto riesgo. Por lo tanto. la explicación de *machine learning* ha experimentado un aumento en el interés de la comunidad de investigación a la aplicación dominios. Existe la necesidad de evaluaciones para cuantificar la calidad de los métodos de explicación para determinar si y en qué medida los algoritmos propuestos logra el objetivo definido, y comparar los métodos de explicación disponibles y sugerir la mejor explicación de la comparación para una tarea específica. [21]

Dentro de un algoritmo predictivo, se pronostica una etiqueta para las variables de entrada. Esta etiqueta puede estar correcta o incorrecta. Si se aplica este concepto a clasificación binaria, la etiqueta es exactamente la esperada o la errónea. Para visualizar lo anterior, existe la matriz de contingencia que resume el éxito del procedimiento con las categorías. Se utiliza la nomenclatura positiva para referirse a correcta y negativa para errónea/incorrecta.

- Verdadero Positivo (*True Positive*)(TP): Clasificación positiva para clase positiva,
- Falso Positivo (FP): Clasificación positiva para clase negativa,
- Verdadero Negativo (*True Negative*) (TN): clasificación negativa para clase negativa, y
- Falso Negativo (FN): clasificación negativa para clase positiva.

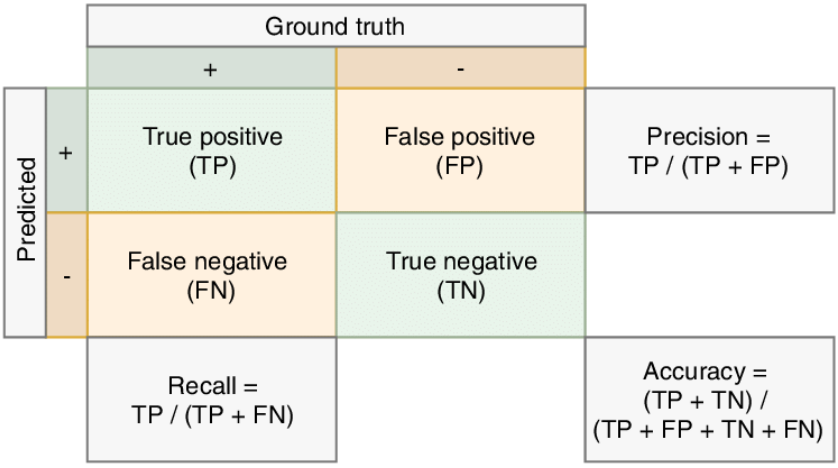

Figura 3.8: Matriz de Confusión y Métricas de evaluación [22]

Dado lo anteriormente expuesto, nacen las siguientes métricas:

- **Accuracy:** porcentaje de clasificación correcta $\frac{TP+TN}{TP+TN+FN+FP}$
- **Precision:** porcentaje de clasificación positiva como correcta $\frac{TP}{TP+FP}$
- **Recall:** porcentaje de clasificación correcta para la clase positiva $\frac{TP}{TP+FN}$
- **F1 Score:** es una ponderación entre *recall* y *precission* para manejar los dos parámetros simultáneamente. Se relacionan como dos resistencias en paralelo: $\frac{2PR}{R+P}$ [23]

Dependiendo del planteamiento del problema y sus objetivos se les da diferente importancia a las métricas mencionadas. Si se tiene un problema no binario, para el cálculo se tiene que escoger una clase en particular y el problema se puede abordar como binario.

3.3. Metodologias para proyectos de Ciencia de Datos

A principios de 1990 mientras que la minería de datos seguía evolucionando, los llamados *Data Scientist* tomaban muchísimo tiempo con tecnología y capacidad computacional acotada en intentar levantar inferencia de la información. Al final de cuentas se desenvolvían mucho más haciendo modelamiento predictivo mientras aún no existía tanta documentación al respecto.

A medida que se fue teniendo mejor tecnología, estos procesos se fueron agilizando, logrando que los *data scientits* podían iterar mucho más rápido con la data y los modelos. A raíz de esto se fueron desarrollando diversas metodologías logrando dinámicas diferentes. [24]

3.3.1. KDD

KDD (Knowledge Discovery in Databases) es el proceso no trivial de identificar patrones de datos válidos, novedosos, útiles y comprensibles de un set de datos definidos. El proceso KDD

puede verse como una actividad multidisciplinaria que abarca técnicas más allá del alcance de cualquier disciplina en particular de ciencia de datos, como aprendizaje automático [8]. KDD fue de las primeras metodologías inventadas para aplicar minería de datos.

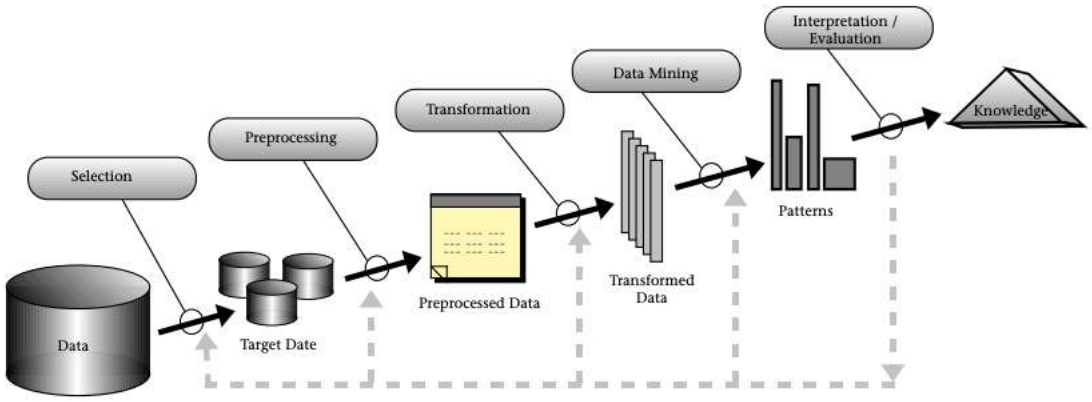

Figura 3.9: Metodologia KDD

El proceso KDD es interactivo e iterativo (con muchas decisiones tomadas por el usuario), involucrando numerosos pasos, resumidos de la siguiente forma:

1. Aprendizaje del dominio de la aplicación: incluye conocimientos previos relevantes y los objetivos de la aplicación.

2. Crear un conjunto de datos de destino: incluye seleccionar un conjunto de datos o centrarse en un subconjunto de variables o muestras de datos en las que se realizará el descubrimiento.

3. Limpieza y preprocesamiento de datos: incluye operaciones básicas, como eliminar el ruido o valores atípicos si corresponde, recopilar la información necesaria para modelar o contabilizar el ruido, decidir estrategias para manejar los campos de datos faltantes y contabilizar la información de secuencia de tiempo y los cambios conocidos. También decidir problemas de DBMS, como tipos de datos, esquema y mapeo de valores perdidos y desconocidos.

4. Reducción y proyección de datos: incluye encontrar características útiles para representar los datos, según el objetivo de la tarea, y usar métodos de reducción o transformación de dimensionalidad para reducir el número efectivo de variables bajo consideración o para encontrar representaciones invariantes para los datos.

5. Elegir la función de minería de datos: incluye decidir el propósito del modelo derivado del algoritmo de minería de datos (por ejemplo, resumen, clasificación, regresión y agrupamiento).

6. Elegir los algoritmos de minería de datos: incluye la selección de métodos que se utilizarán para buscar patrones en los datos, como decidir qué modelos y parámetros pueden ser apropiados (por ejemplo, los modelos para datos categóricos son diferentes de los modelos en vectores sobre reales) y hacer coincidir un método de minería de datos en

particular con los criterios generales del proceso KDD (por ejemplo, el usuario puede estar más interesado en comprender el modelo que en sus capacidades predictivas).

7. Minería de datos: incluye la búsqueda de patrones de interés en una forma de representación particular o un conjunto de tales representaciones, incluidas reglas o árboles de clasificación, regresión, agrupamiento, modelado de secuencias, dependencia y análisis de líneas.

8. Interpretación: incluye interpretar los patrones descubiertos y posiblemente volver a cualquiera de los pasos anteriores, así como la posible visualización de los patrones extraídos, eliminar patrones redundantes o irrelevantes y traducir los útiles en términos comprensibles para los usuarios.

9. Usar el conocimiento descubierto: incluye incorporar este conocimiento en el sistema de desempeño, tomar acciones basadas en el conocimiento o simplemente documentar y reportar a las partes interesadas, así como verificar y resolver posibles conflictos con conocimiento previamente creído (o extraído).

3.3.2. CRISP-DM

El proceso o metodología de CRISP-DM es otra metodología desarrollada también en los años 90 que tiene un proceso similar a KDD pero con enfoques diferentes. Estos enfoques se describe en estos seis pasos principales:

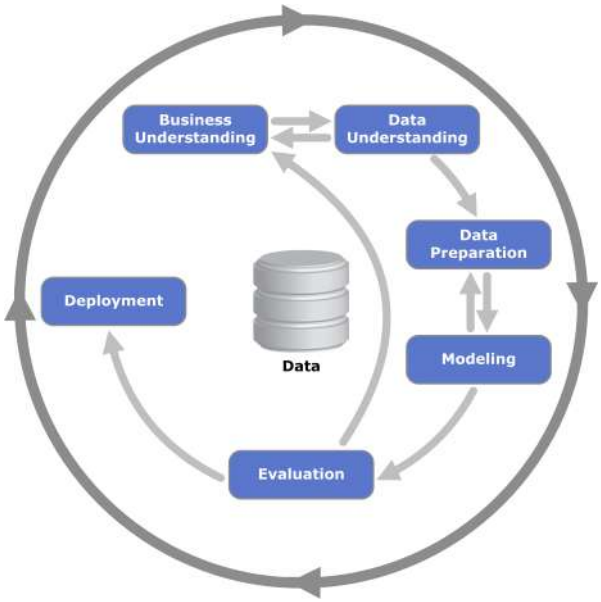

Figura 3.10: Metodologia CRISP-DM

1. Comprensión empresarial: Se enfoca en comprender los objetivos y requisitos del proyecto desde una perspectiva comercial y luego convertir este conocimiento en una definición de problema de minería de datos y un plan preliminar.

2. Comprensión de datos: Comienza con una recopilación de datos inicial y continúa con actividades para familiarizarse con los datos, identificar problemas de calidad de los

datos, descubrir los primeros conocimientos sobre los datos o detectar subconjuntos interesantes para formar hipótesis sobre información oculta.

3. Preparación de datos: La fase de preparación de datos cubre todas las actividades para construir el conjunto de datos final a partir de los datos en bruto iniciales.

4. Modelamiento: Se seleccionan y aplican técnicas de modelado. Dado que algunas técnicas como las redes neuronales tienen requisitos específicos con respecto a la forma de los datos, puede haber un bucle aquí para la preparación de datos.

5. Evaluación: Una vez que se han creado uno o más modelos que parecen tener alta calidad en función de las funciones de pérdida que se hayan seleccionado, estos deben probarse para garantizar que se generalicen frente a datos no vistos y que todos los problemas comerciales clave se hayan considerado suficientemente. El resultado final es la selección de los modelos campeones.

6. Despliegue: En general, esto significará implementar una representación de código del modelo en un sistema operativo para calificar o categorizar nuevos datos invisibles a medida que surgen y crear un mecanismo para el uso de esa nueva información en la solución del problema comercial original. Es importante destacar que la representación del código también debe incluir todos los pasos de preparación de datos que conducen al modelado para que el modelo trate los nuevos datos sin procesar de la misma manera que durante el desarrollo del modelo. [24]

Capítulo 4

Metodología

El trabajo por desarrollar consiste en el levantamiento de herramientas digitales con enfoque en los procesos de reclutamiento, registro y fidelización del jobber. Este tienen como objetivo diseñar e implementar un modelo de caracterización del jobber en el proceso de registro. No obstante, el trabajo completo, independiente del enfoque de la memoria, se basa tanto en la caracterización del usuario que se quiere predecir como elaborar un modelo y poder aplicarlo en las plataformas actuales.

Por lo anteriormente dicho, el trabajo completo se desarrolló en 3 partes fundamentales. caracterización, elaboración de modelo y despliegue del modelo. Si bien la segunda es el foco del documento, la primera parte de caracterización es fundamental para la segunda. Esto dado que para estructurar cualquier tipo de algoritmo, se tiene que alimentar y orientar dependiendo de las necesidades de la interacción empresa-usuario. La tercera, la parte de despliegue es la más importante para la empresa, pero se mencionará de manera sencilla dado que se escapa de los alcances de una memoria del Departamento de Ingeniería Eléctrica. A continuación se mencionará como se abordaron las tres partes.

4.1. Caracterización

Caracterización busca profundizar el tipo de usuario que se tiene, al igual que entender su comportamiento con la plataforma. Parte de esta primera parte es lograr conectarse a la base de datos, explorar las colecciones y entender las variables que maneja. Al hablar de caracterización del jobber, desde un punto de vista más cualitativo, se quiere abordar las siguientes preguntas que no poseían respuesta clara:

- ¿Quiénes son los usuarios que entran a registrarse?
- ¿Quiénes son los usuarios que logran completar el registro completo?
- ¿Por qué hay usuarios que no lo logran?
- ¿Cuáles son los puntos críticos en este proceso? ¿Como se puede mejorar?
- ¿Quiénes son los jobbers que estando registrado no toman tareas?
- ¿Quiénes son los jobbers que si se mantenien activos en la toma de tareas?

Al contestar estas preguntas se tiene una mejor idea del tipo de usuario con el que se está trabajando y se puede cuantificar el perfil de usuarios que tiene la empresa. Para abordar la caracterización se utilizó la metodología CRISP-DM dado que es iterativa y permite ir en cada paso ir un más lejos en el análisis. Se comienza solamente haciendo conexión y entendiendo que significa cada atributo hasta teniendo una imagen completa del flujo que se mostró en Introducción [1.1.5]. Dado que todo lo que se hace en esta sección es netamente teórica nunca se llega a la fase *'Deployment'* como se muestra en Figura 3.10 Lo bueno de cada iteración es que se iba planteando objetivos a cada uno. Se hicieron 4 iteraciones con las siguientes focos:

Iteraciones

1. Caracterización introductoria de usuarios dentro de proceso de registro utilizando visualizaciones
2. Agrupación de usuarios por conducta que no terminan el registro utilizando *k-means*
3. Agrupación de usuarios por sexo, edad y localización utilizando *k-means*
4. Creación de imagen completa del flujo registro con enfoque en postular tareas

Cabe destacar que en cada iteración, la sección *Evaluation* de la metodología escogida era realizada con la presentación de los resultados con los *head* de las otras áreas de la empresa para corroborar información y recibir comentarios y/o *feedbacks* de los resultados. De igual manera, ellos podían ir rescatando información valiosa para las proyecciones de estos procesos. Además, en ningún momento se hace un *Deployment* dado que solo se busca la información necesaria.

4.2. Modelos

Posterior a la caracterización, la segunda parte espera levantar nuevos modelos para tener predicciones en torno al punto de llegada del usuario dado sus características y conductas. El algoritmo tiene como intención que se gatille al momento de llegar a un cierto punto del proceso y predecir si llegará a otro punto más adelante. Esta configuración viene dada tanto por los resultados encontrados en la última iteración (4.) de caracterización [4.1] y dado las necesidades del equipo de operadores de *on boarding* mencionadas en Introducción [1].

Para esta sección se utiliza de la misma manera la metodología CRISP-DM, enfocando más en el bloque de modelamiento al ajustar los algoritmos. Esto salvo en la primera iteración que se concentra en preprocesamiento y por ende ser los pasos de *Data understanding* y *Data Preparation* en las siguientes 4 iteraciones. A diferencia de caracterización, las iteraciones no van retomando lo de la iteracion anterior ya que, lo que se busca en definitiva, es definir el mejor desempeño con el mismo set de entrenamiento.

Para esta sección se busca implementar y analizar las temáticas de Marco Teórico - Algoritmos de Aprendizaje Automatico [3.2]. Se buscará encontrar la mejor configuración posible para maximizar desempeño. Las iteraciones de esta parte quedaron de la siguiente manera:

Iteración

1. Pre-Procesamiento de todos las variables de entrada y creación de set de entrenamiento y testeo
2. Implementación de prueba MLP, Naives Bayes y Arboles
3. Desarrollo y mejoramiento MLP
4. Desarrollo y mejoramiento Árboles y Bosques
5. Desarrollo y mejoramiento Naives Bayes
6. Exportación de Modelo ganador

A diferencia de parte 1 y 3, esta sección fue la más autónoma dado que se debía trabajar en aspectos netamente técnicos de los algoritmos.

4.3. Implementación Modelo

La última sección es la más técnica en aspectos computacionales. En esta parte se busca automatizar el algoritmo escogido en la parte anterior. A diferencia de ambas secciones anteriores, aquí se utilizo una metodología ágil que busca ir conectando el trabajo con otros funcionarios de la empresa.

Se debió trabajar con personal UX/UI para diseñar visualización, con *Data Engineer* para conectar la ingesta de algoritmos con un *Data Lake* creado recientemente, con *Devops Engineer* para crear la máquina virtual donde se despliegue algoritmo, con *Back End Developper* para definir como se iba a guardar la información y con *Front End Developper* para implementarlo en la plataforma actual. Gracias a las metodologías ágiles que se utilizan en Timejobs, se pudo trabajar con la célula llamada *On Baarding* dirigida por un *Product Owner* que coordina las tareas y los tiempos.

Habrá una sección dedicada a la implementación del algoritmo, pero no se entrará mucho en detalle. El motivo es que muchos de los tópicos explicados aquí se alejan tanto de la visión de la memoria como del enfoque de la carrera ingeniería eléctrica.

Capítulo 5

Procesamiento y Modelamiento

En vista de alimentar los algoritmos predictivos, se desea habilitar la mayor cantidad de atributos que logren caracterizar de alguna manera a los usuarios. De la misma manera se plantea como se configuran los algoritmos escogidos para ir evaluando sus desempeños.

5.1. Caracterización del usuario

Para esta primera parte se conecta a la colección *users* de la base *account* dentro del dominio de Timejobs. A pesar de poseer varias bases de datos y colecciones (13 y 40 respectivamente), esta es la única que guarda información bruta del usuario. En *body* de los usuarios es bastante grande como se ve en Figura 5.1. En efecto, cada usuario puede poseer hasta 50 campos. Se estudió y analizó los atributos que sirvan para caracterizarlo.

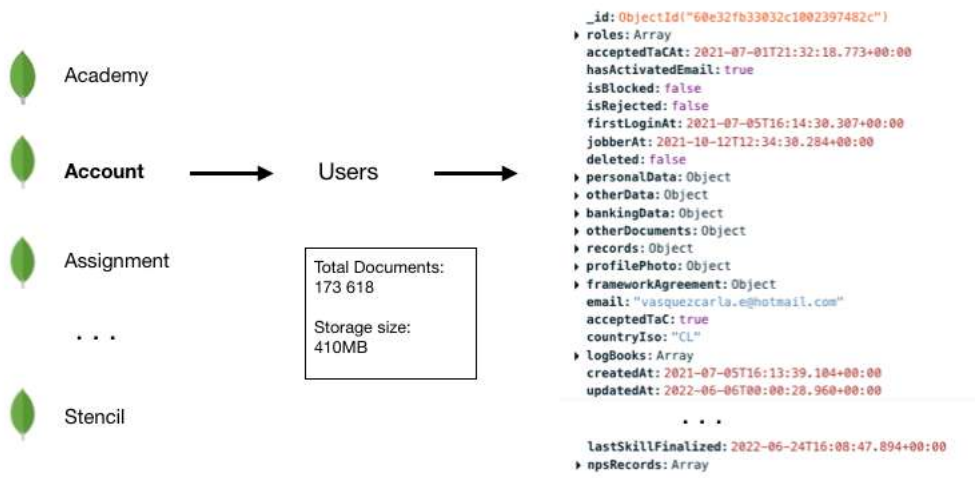

Figura 5.1: Ejemplo de un usuario en colección Account.Users del dominio Timejobs de Mongo DB

Se irán detallando todos los atributos encontrados en la base al mismo tiempo que su preprocesamiento para aplicarlos en los algoritmos. Un punto a destacar es que se tomaron solo los usuarios nuevos, es decir, todo aquel usuario que fue migrado de las base de datos anteriores no fue tomado en cuenta. De igual manera, solo se seleccionaron a los usuarios que viven en Santiago de la región Metropolitana como primer acercamiento del problema. Se

desea, en cualquier situación, que los valores queden estandarizados, vale decir, que queden entre 0 y 1.

- **Edad:** atributo calculado del *timestamp* [personalData.birthday]. Se calcula a entero y se guardaba como nuevo atributo.
- **Sexo:** [personalData.gender]. Opciones "Masculino", "Femenino". Se pasa a números 0 para masculino y 1 para femenino.
- **Dirección:** [personalData.address] Se separa por agrupaciones 7 ['Centro', 'Poniente', 'Norte', 'Centro Poniente', Sur', 'Sur Oriente', 'Sur Poniente'] inspirado del trabajo "La estructura de la densidad socio-residencial en el área metropolitana de santiago de la Facultad de arquitectura, diseño y estudios urbanos de la Pontificia Universidad Católica [25] . Se asigna de 1 a 7 las etiquetas y se normaliza para tenerlas entre 0 y 1.
- **País de Origen:** [personalData.originCountry] Se separa como nacionalidad chilena 1 y extranjeros 0.
- **Tipo de Celular:** [otherData.cellphoneOS] Opciones 'Android' y 'iOS' que ellos debían notificar. Se utiliza 0 para 'Android' y 1 para 'iOS'.
- **Vehículo:** [otherData.vehicle] Información opcional de sí poseía vehículo (1), motocicleta (2) o ninguna (0). Se normaliza.
- **Banco:** [bankingData.bank]. Se separan entre 'Banco Estado' (0) y otros (1).
- **Tipo de Cuenta Bancaria:** [bankingData.accountType]. Se separa entre cuenta vista (0) y cuantas corrientes (1).
- **Referencia:** [otherData.referral]. Respuesta de como había llegado a la plataforma. Se asigna número a cada etiqueta y se normaliza.
- **Campaña de marketing:** [utmSource]. Anexo de marketing que podía vincular si ha cliqueado por alguna campaña de marketing digital con su plataforma (Instagram, LinkedIn, Facebook, etc.). Se asigna número si existe su respectiva campaña-red social y 0 si no hay. Se normaliza.

5.2. Interacción con la plataforma

Anexo a los atributos de la base de datos, existe la herramienta segment que registra cierta información de la interacción del usuario online con los hitos proveniente del sistema explicado Plataformas Digitales [1.1.3]. Segment es gatillador y envía data a diferentes fuentes como se muestra en Figura 5.2. Para este trabajo se consumió el dato que se guarda en *Big Query*, plataforma de *datawarehouse* de *Google Cloud Platform*.

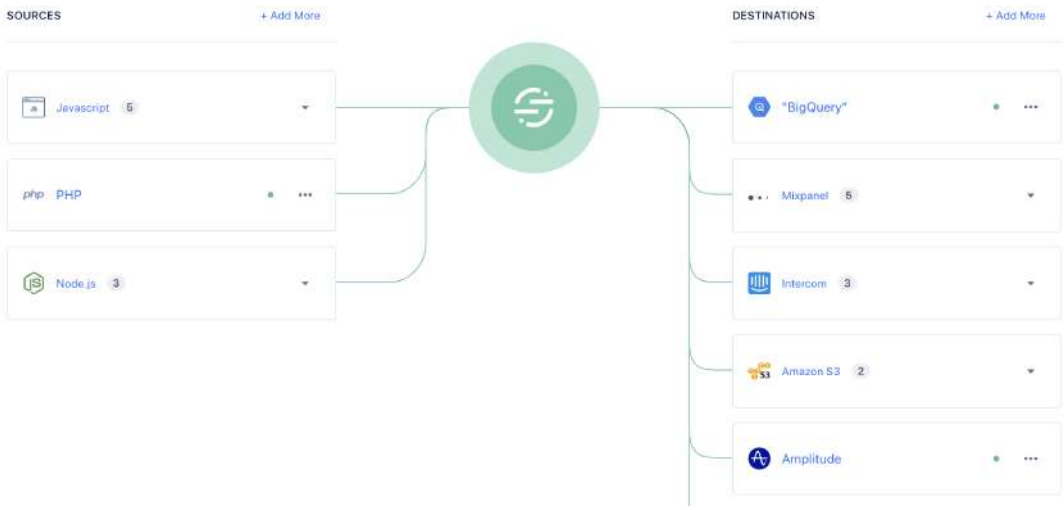

Figura 5.2: Fuentes y Destinos de la plataforma Segment

Para estos atributos se decide aplicar normalización logarítmica dado que hay tiempos muy largos de completar ciertos hitos. Si bien existen ciertos usuarios que completan los hitos en minutos, existen un numero no menor que terminan en semanas. Se tiene registro de los siguientes hitos:

- **Cuenta Creada:** Se posee información respecto del momento exacto (timestamp) que un usuario entra a la plataforma de registro. Esta se usará como referencia para calcular los siguientes atributos.
- **Datos Personales:** Timestamp de cuando el usuario completa el hito personales. Se crea [personal_data_time] como la diferencia en segundos con cuenta creada. Se aplica normalización logarítmica.
- **Datos Bancarios:** Timestamp de cuando el usuario completa el hito de datos bancarios. Se crea [banking_data_time] como la diferencia en segundos con cuenta creada. Se aplica normalización logarítmica.
- **Otros Datos:** Timestamp de cuando el usuario completa el hito de otros datos. Se crea [other_data_time] como la diferencia en segundos con cuenta creada. Se aplica normalización logarítmica.
- **Antecedentes Penales:** Timestamp de cuando el usuario completa el hito de subir sus antecedentes penales. Se crea [police_record_time] como la diferencia en segundos con cuenta creada. Se aplica normalización logarítmica.
- **Registro Completado:** Timestamp de cuando el usuario completa el registro completo. Se crea [register_fulfilled_time] como la diferencia en segundos con cuenta creada. Se aplica normalización logarítmica.
- **Navegador:** Llamado *contex user agent* logra describir el dispositivo (mac, windows, android, iOS) como version del sistema operativo y navegador.

Con estos atributos, se logra agrupar y normalizar todos los variables que podrían ser de relevancia para la predicción. A continuación se muestra la matriz de correlación entre los atributos:

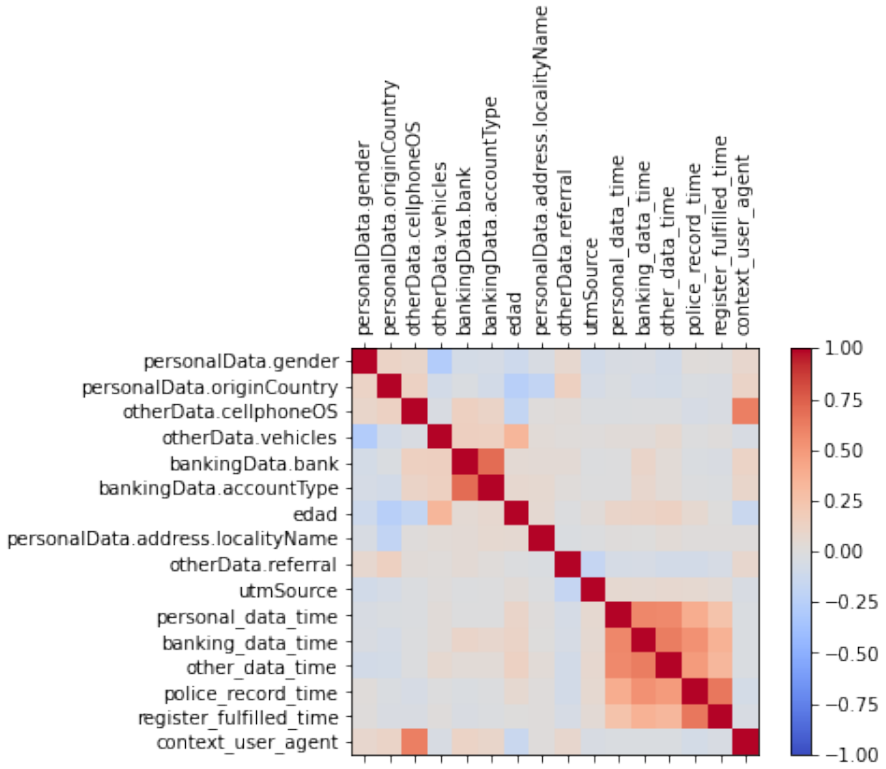

Figura 5.3: Matriz de correlación de las variables post procesamiento

Más detalles sobre las variables de entrada se puede encontrar en Anexo A.

5.3. Distribución de inducciones

Como se mencionó en introducción existen tres tipos de jobbers: operador, picker y shopper. Para obtener algunos de esos categorías se tiene que realizar una de las tres inducciones. El acto de tomar la inducción es lo que se buscará predecir.

En *user.acount* existe un atributo llamado *skill*. Este es un *array* de objetos que se relaciona con la otra base de datos llamada *academy.skils*. Esta tiene la información sobre el tipo de *skills*, de qué vertical pertenece, la fecha en la que lo realizó, descripción de la inducción entre otros. Para crear el dataset de entrenamiento se creó una función que iba jobber por jobber revisando si poseían inducciones. Se creó la variable inducción que era 1 si el jobber poseía una inducción y 0 si no.

Utilizando la función para pasar a nomenclatura 0 y 1 se logró ver como era la distribución de las inducciones:

$$\text{Numero total de inducciones: } 6549 \tag{5.1}$$

Tabla 5.1: Distribución de inducciones por vertical

Operario	Picker	Shopper
3854	498	2191

Tabla 5.2: Cantidad de jobbers que toman inducciones de misma vertical vs cantidad de jobber que toman inducciones de distintas verticales

Misma vertical	Distinta vertical
4619	563

5.4. Modelos

Para simplificar el modelo, se decide definir jobber con inducción como jobber que haya tomado inducción de cualquier vertical. Se buscará predecir si el jobber tomará cualquier inducción disponible. De la misma manera, para entrenar los modelos se decidió eliminar los atributos bankingData.accountType y register_fulfilled_time dado que tenían mucha correlación con los atributos bankingData.bank y police_record_time respectivamente.

Para entrenar el modelo se separa los sets solo en training y testing dejando de lado el set de validación dado la cantidad de datos. De la misma manera, se equilibraron los sets para que que tanto en training como testing tengan 60 % de jobbers sin inducción y con 40 % . Para esto, se eliminaron datos de usuarios sin inducción.

Adicionalmente se toman solo usuarios que se hallan registrado hace más de un mes y medio (a fecha de finales junio 2022), dado que hay jobber que se demoran más de un mes en finalmente tomar la inducción.

Para el modelo Red Neuronal, se implementa una perceptrón multicapa (MLP) utilizando la libreria *tensor flow* version 1.14.0. Para Àrbol de Decisión, Random Forest y Naive Bayes se utiliza la librería *scikit learn* 0.23.0.

Capítulo 6

Resultados y Análisis

6.1. Resultados

Para comenzar, algunos resultados interesantes del ciclo de caracterización que sirvieron para entender la data y orientas de mejor manera el algoritmo predictivo se dejarán en Anexo B. Estos resultados sirven para entender que tipo de jobbers existen en la plataforma.

Luego, se busca tener una imagen completa de la aplicación de los diferentes algoritmos, por lo que entrará en detalle en las experiencias de Redes Neuronales, Arboles-Bosques y Naives Bayes. Al tener tantas métricas de desempeño se decidió que los resultados mostrarán *Accuracy* y *F1 Score*. *Accuracy* es la métrica que generaliza que tan asertivo fue el algoritmo en términos de detectar los verdaderos negativos y positivos, mientras que *F1 Score* es calculado a partir de *precision* y *recall*. De esta manera, al visualizar *F1 Score* se podrá hacer alusión a los otros dos y por ende tener la imagen completa más compacta posible. De todas maneras se busca maximizar todas las métricas posibles, por lo que utilizar 2 facilita la lectura.

En cada modelo, se especificará los hiperparametros con los que se hacen las experiencias. Todos estos hiperparametros fueron explicados teóricamente en la sección de Marco Teórico 3.2.

Al final de resultados, se exhibirá el detalle de todos los algoritmos para hacer las conclusiones finales. Cabe destacar que los resultados de esta sección son probados con el set de *testing*, completamente anexo a los datos de *training* con los que se entrenaron.

6.1.1. Redes Neuronales

Se comienza creando una red estándar con 2 capas ocultas con un *dropout* antes de la salida. Se escoge 50 nodos de entrada para las 14 variables. Se escoge la función *ReLu* como función de activación en los nodos. En definitiva, la red queda de la siguiente manera:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 50)                750
_________________________________________________________________
dense_4 (Dense)              (None, 20)                1020
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 105
_________________________________________________________________
dropout_1 (Dropout)          (None, 5)                 0
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 12
=================================================================
Total params: 1,887
Trainable params: 1,887
Non-trainable params: 0
_________________________________________________________________
```

Figura 6.1: Arquitectura Red Neuronal Base

La función de costo escogida es la función entropía cruzada binaria (*Binary Crossentropy*) dado que es la más recomendada para problemas de clasificación binaria. Para experimentar con la red, se comienza variando la taza de aprendizaje utilizando tanto el optimizador Adam como el SGD:

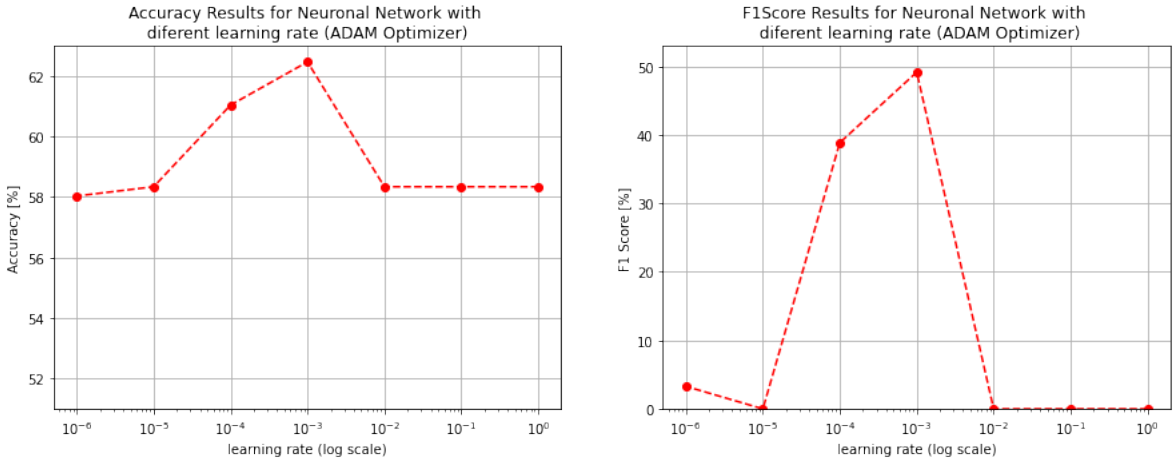

Figura 6.2: *Accuracy* y *F1 Score* para Red Neuronal variando la tasa de aprendizaje con optimizador Adam

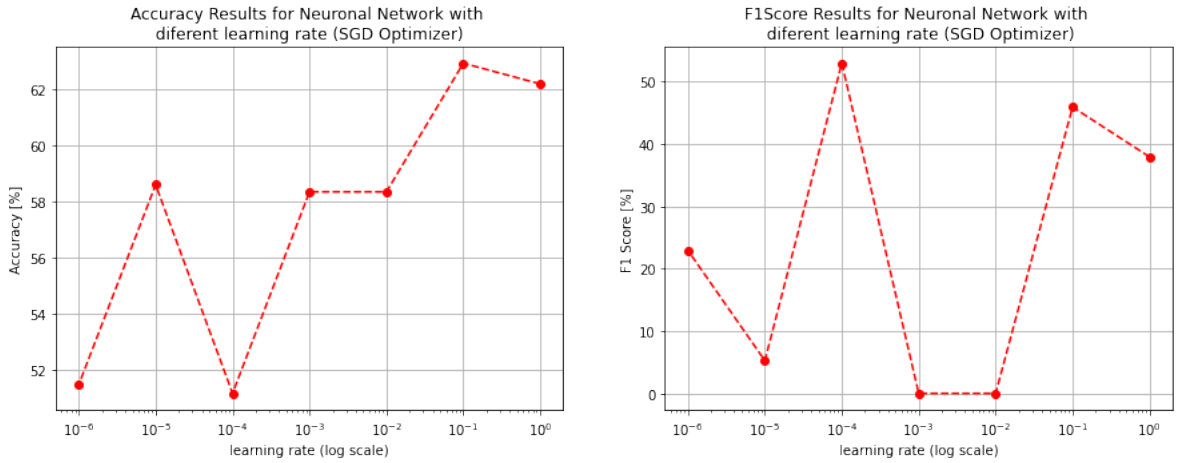

Figura 6.3: *Accuracy* y *F1 Score* para Red Neuronal variando la tasa de aprendizaje con optimizador SGD

Se escoge la mejor taza de aprendizaje para, posteriormente, modificar la configuración interna de la red haciéndola más profunda aumentando las capas ocultas. Se modifica de igual manera la función de activación entre cada capa. Se cambia la función $ReLU$ (*Rectified Linear Unit*) a *Leaky* $ReLU$ que no anula los valores negativos entre las capas. Es decir, en $ReLU$ si llega un valor negativo, se anula mientras que en *Leaky RelU* se multiplica por un coeficiente determinado. Esta técnica se utiliza para intentar mejorar desempeño en caso de estancarse en desempeño.

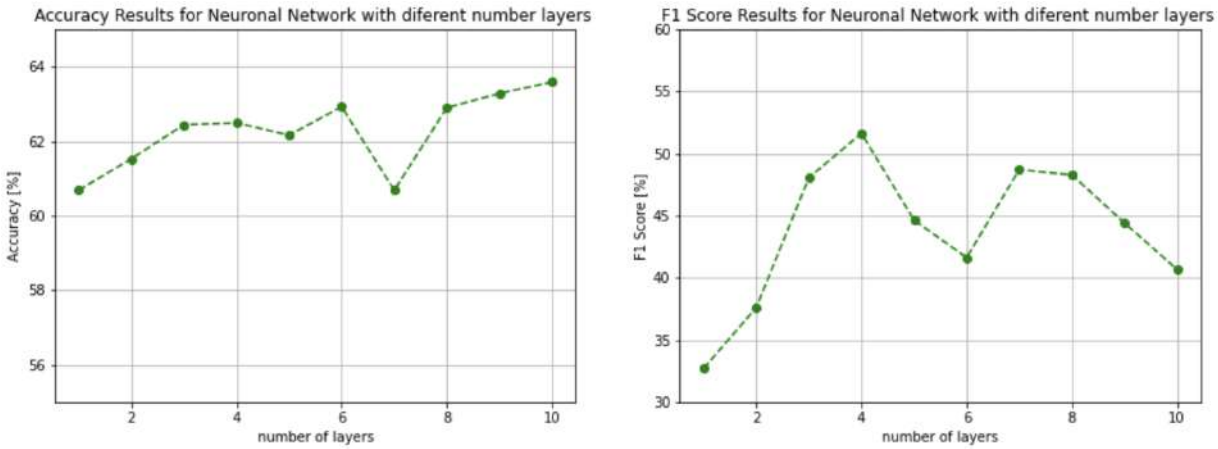

Figura 6.4: *Accuracy* y *F1 Score* para Red Neuronal aumentando el número de capas utilizando función de activación $ReLU$ entre capas

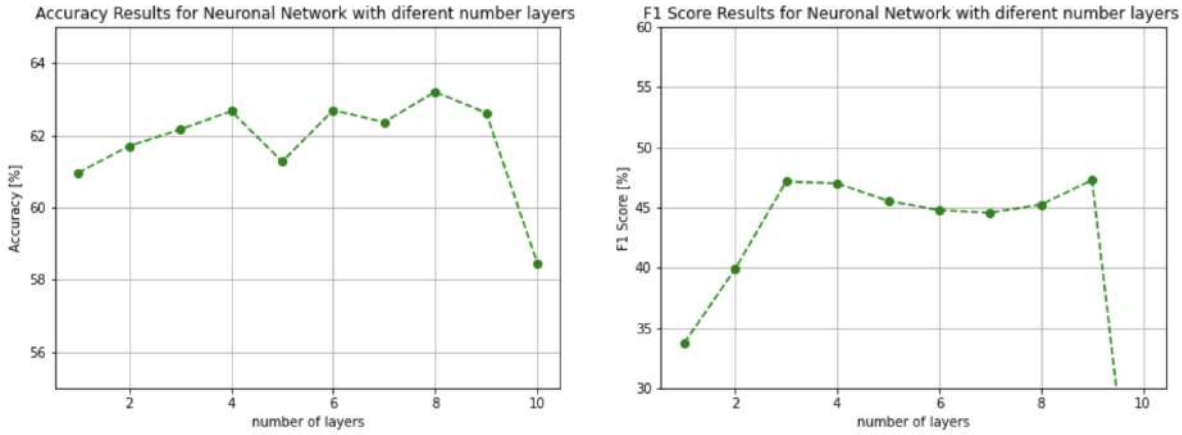

Figura 6.5: *Accuracy* y *F1 Score* para Red Neuronal aumentando el numero de capas utilizando utilizando función de activación *Leaky ReLU* entre capas

Tomando profundidad a 4 capas ocultas utilizando *ReLu*, se procede a observar como evoluciona la red a través de las épocas. Siempre utilizando el error como la Entropía Cruzada Binaria:

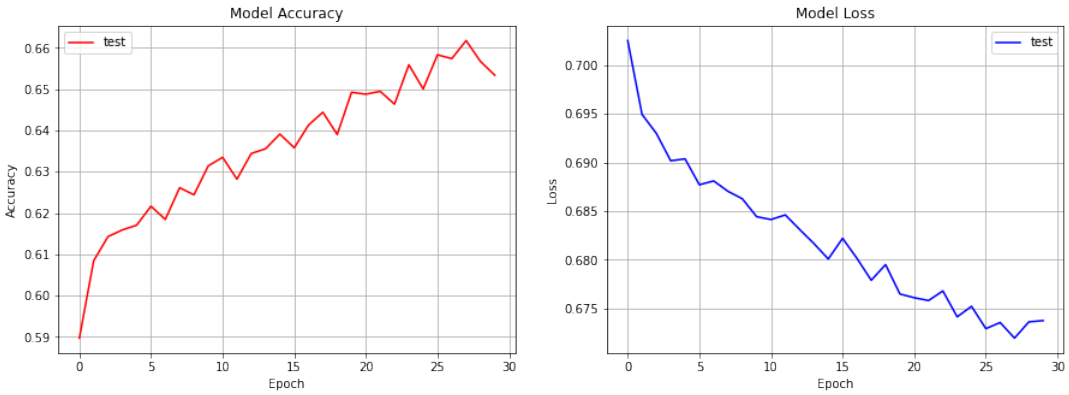

Figura 6.6: *Accuracy* y *Loss* para Red Neuronal durante proceso de entrenamiento

6.1.2. Arboles y Bosques

Se comienza por crear árboles cambiando respectivamente el largo máximo y mínimo de datos para separación de nodo.

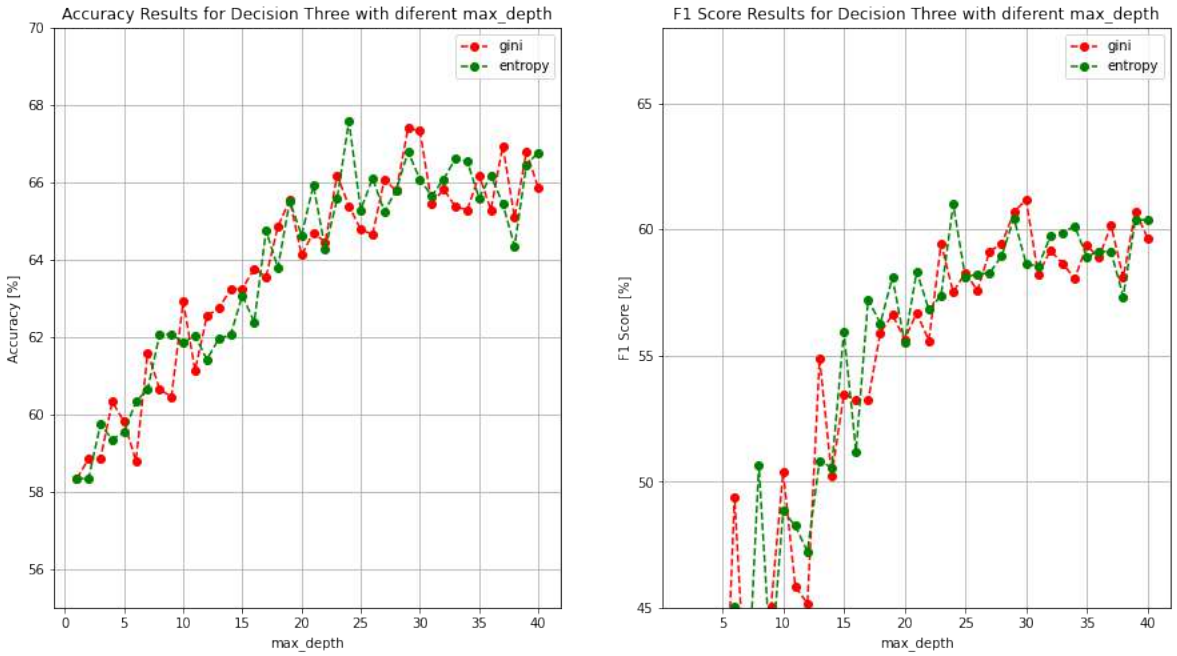

Figura 6.7: *Accuracy* y *F1 Score* para árbol de decisión cambiando max depth con utilizando impuereza Gini y Entropy

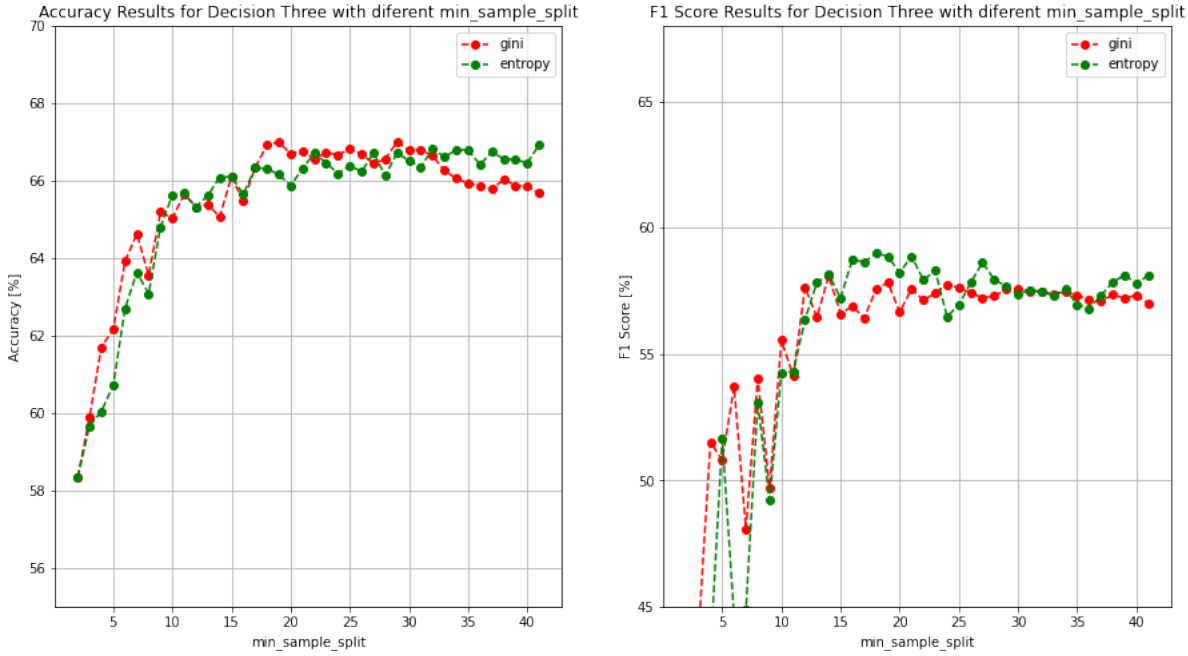

Figura 6.8: *Accuracy* y *F1 Score* para árbol de decisión cambiando min sample split utilizando impureza gini y entropia

Obteniendo los mejores resultados de las experiencias mostradas, se procede a cambiar la cantidad máxima de variables que se busca para hacer una separación de nodo.

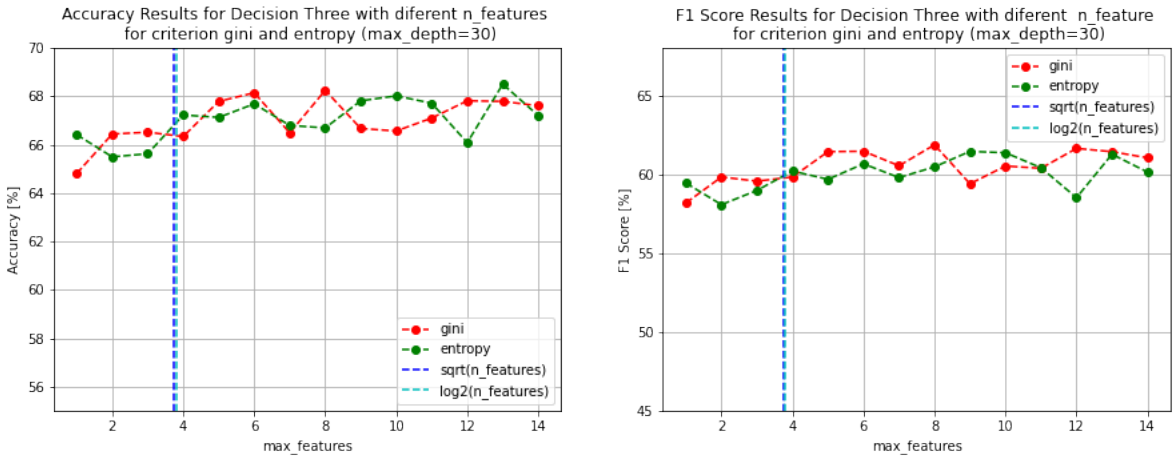

Figura 6.9: *Accuracy* y *F1 Score* para arbol de decesisión cambiando n features utilizando impuereza gini y entropia fijan do max depth en 30

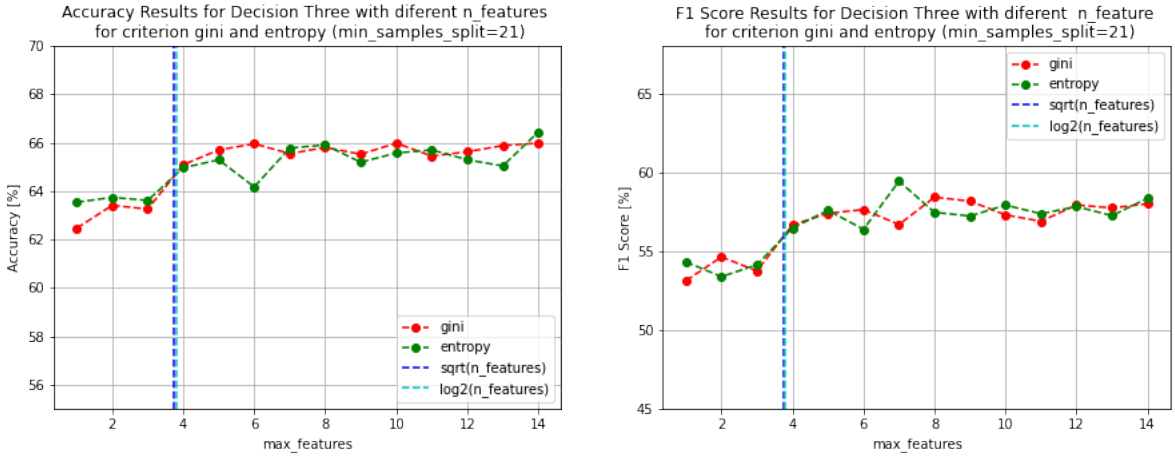

Figura 6.10: *Accuracy* y *F1 Score* para árbol de decisión cambiandio n feaures utilizando impureza gini y entropia fijando min samples split en 21

Obteniendo el mejor árbol, se busca analizarlo internamente. El árbol en sí es bastante grande y no se logra tener una imagen completa de los nodos. No obstante, se puede hacer un gráfico sobre la importancia de las variables para la estructura:

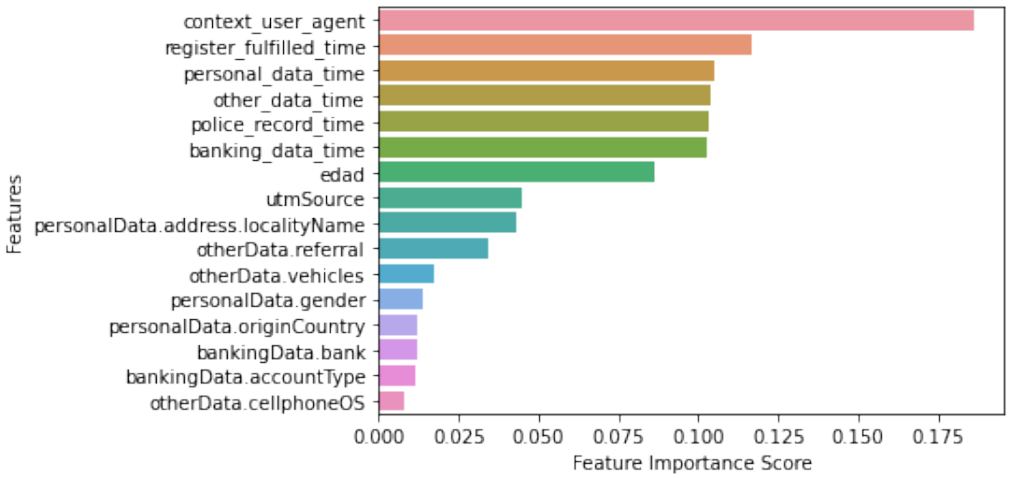

Figura 6.11: Variables relevantes (*Important Features*) del modelo Arból de Decisión

Utilizando la configuración del árbol más asertivo, se busca utilizarlo para experimentar con Random Forest y Gradient Boosting.

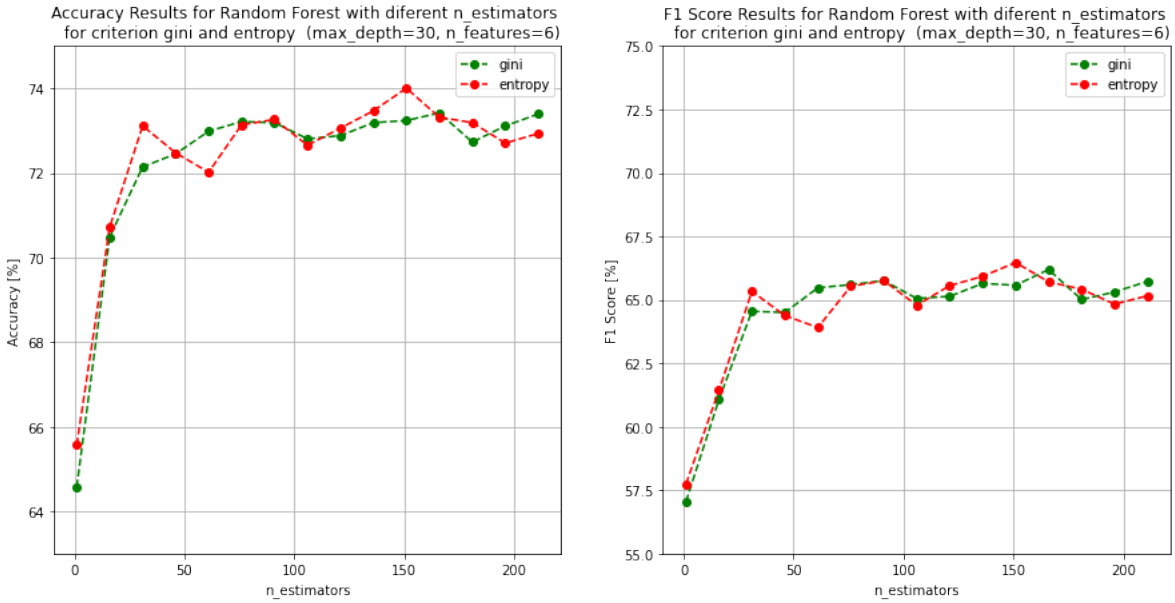

Figura 6.12: *Accuracy* y *F1 Score* para Random Forest cambiando n estimators utilizando impurezas gin y entropia fijando max depth en 30 y n features en 6

Como AdaBoost es un caso particular de Gradient Boosting, se mostrarán solo las experiencias del segundo. De todas maneras, AdaBoost tenia un accuracy que no subia más de un 66 %.

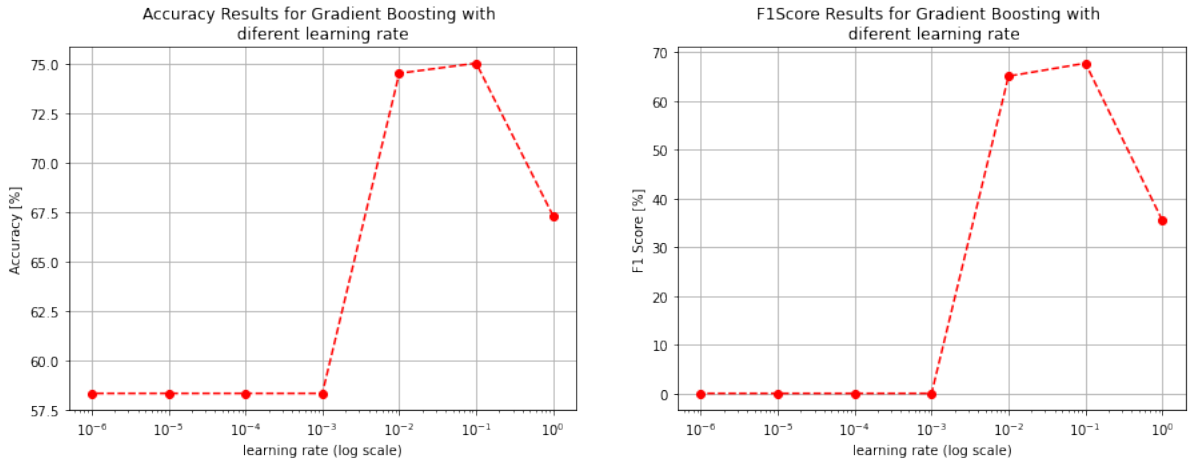

Figura 6.13: *Accuracy* y *F1 Score* de Gradient Boosting cambiando taza de aprendizaje

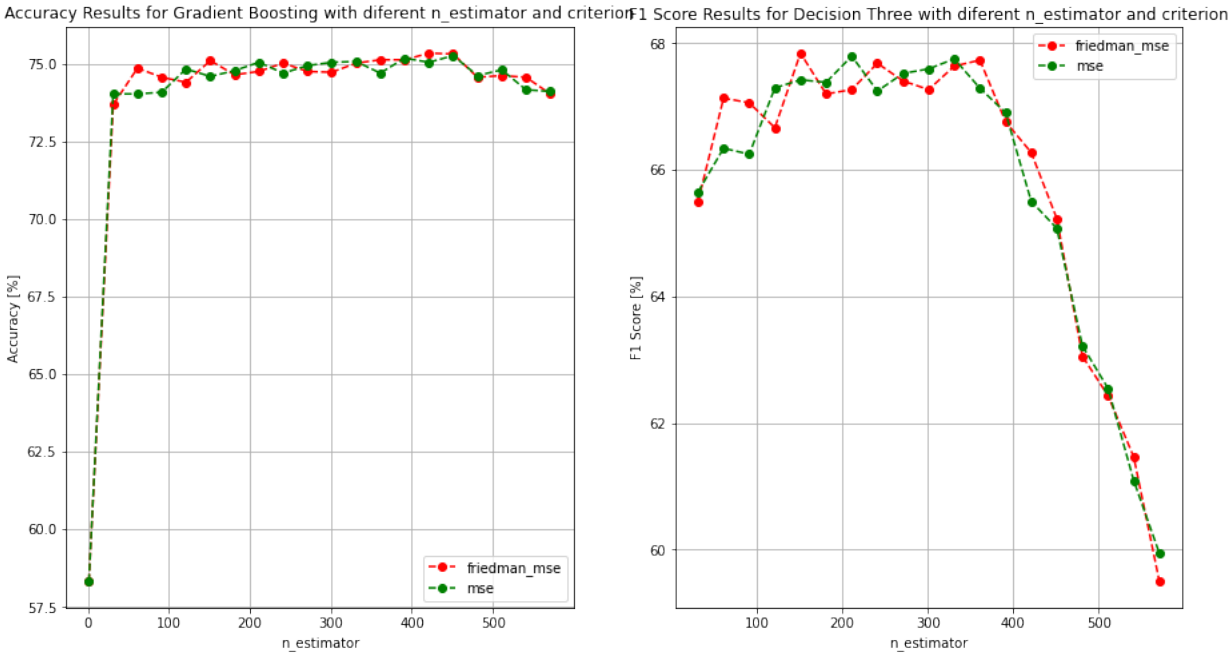

Figura 6.14: *Accuracy* y *F1 Score* de Gradient Bosting cambiando n estimator utilizando ciretior friedman y mse

6.1.3. Naives Bayes

Naives Bayes es el algoritmo menos personalizable dado que funciona solo con una propiedad estadística. Por lo anterior solo se puede cambiar la función de probabilidad. Las posibilidades que existen en la librería son función Gausiana, Bernouilli, Categórica y Complementaria.

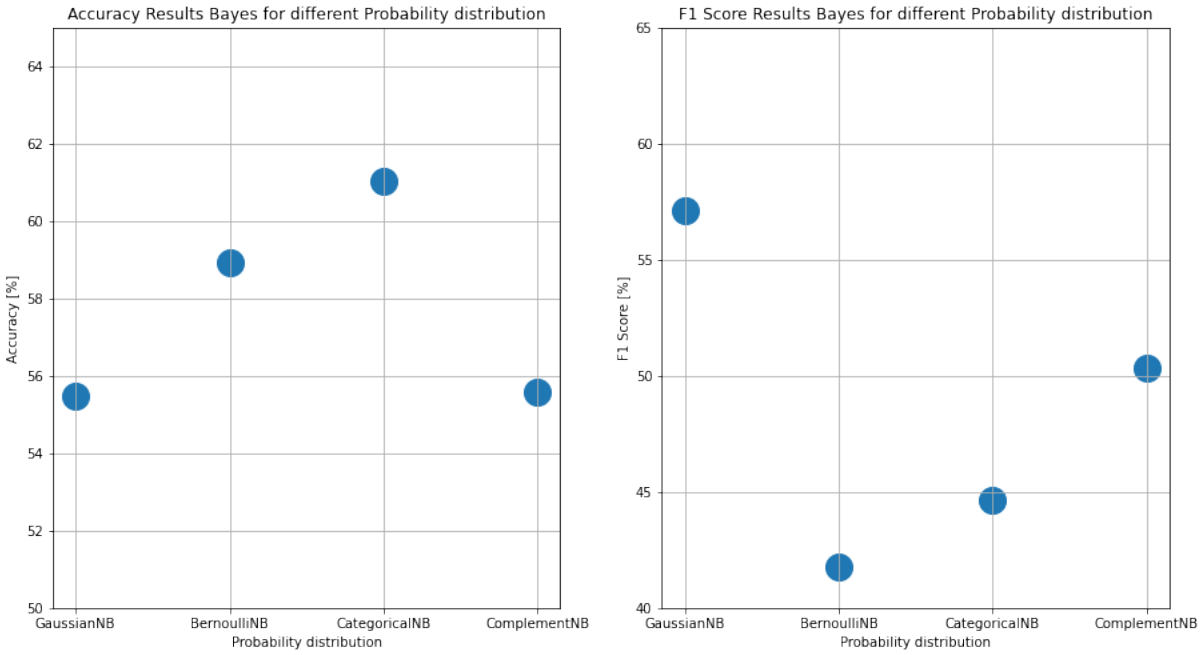

Figura 6.15: *Accuracy* y *F1 Score* para naives bayes utilizando función de distribución Gausiana, Bernoulli, Categorica y Complemnt

6.1.4. Resultados completos modelos

Se muestran los mejores resultados de cada modelo para compararlos dentro de una misma tabla. De igual forma se muestran las matrices de confusión de los mejores dos modelos.

Tabla 6.1: Mejor desempeño por modelo

	Red Neuronal	A. Decision	R. Forest	G. Boosting	N. Bayes
Accuracy	62 %	67 %	74 %	75 %	61 %
F1 Score	40 %	64 %	67 %	68 %	45 %
Precision	32 %	65 %	64 %	62 %	37 %
Recall	58 %	64 %	70 %	73 %	54 %

Las matrices de confusión están al revés que como se mostró en Marco Teórico 3.2.4 dado que el caso positivo es la etiqueta 1 que se encuentra en la posición inferior derecha al contario de la posición superior izquierda de la imagen de referencia. De la misma manera está invertido *precision* y *recall*.

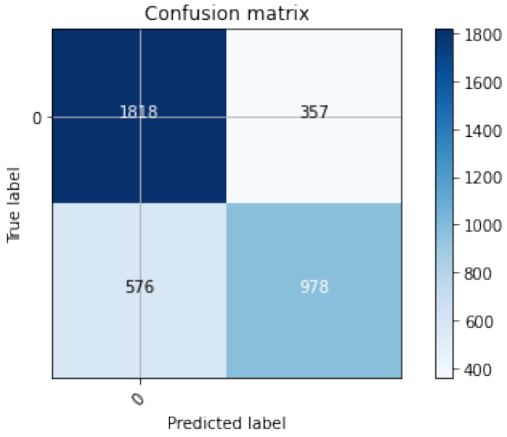

Figura 6.16: Matriz de Confusión de Random Forest

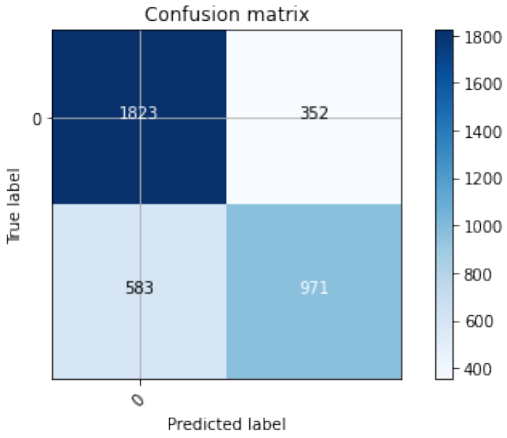

Figura 6.17: Matriz de Confusión de Gradient Boosting

6.2. Análisis

Esta sección entra a anlizar, comentar e interpretar resultados y desempeños de los algoritmos mostrados en la sección anterior.

6.2.1. Redes Neuronales

Se comienza visualizando Figura 6.2, optimizador ADAM tiene mejor desempeño para taza de aprendizaje = 10^{-3} llegando a un 63 % de *accuracy*. Al observar *F1 score*, se observa que solo para taza de aprendizaje = 10^{-3}, 10^{-4} no se indefine la variable, dado que para las otras entradas el modelo solamente predice una sola clase. Esto hace que *F1 Score* baje a 0 dado que *recall* resulta ser de igual manera 0. La taza de aprendizaje óptima calza con el orden de magnitud esperado dado que si es muy grande no tiene la sensibilidad necesaria para converger y si es más pequeño no logra llegar al mínimo global del problema. Comparando con Figura 6.3 optimizador SGD, logra el máximo de desempeño en taza de aprendizaje = 10^{-1} con *accuracy* 63 % y *F1 score* por 47 % por debajo del mejor desempeño de ADAM. Si bien ADAM muestra mejor resultado, su desempeño sigue quedando bastante bajo, haciendo que su predicción quede bajo de un 80 % como minimo aceptable esperado en este tipo. Una predicción aleatoria debería hacer que se tenga una *accuracy* de 50 % y 62 % queda cerca de aquello.

Posteriormente se interesa en observar como cambia la red cambiando la cantidad de capas ocultas llamadas *layers*. Se busca hacer más compleja la red tomando el mejor resultado de la experiencia anterior. Aumentando de 2 capas a 10 como se muestra en Figura 6.4 el *accurary* logra aumentar un 3 % aproximadamente. *F1 Score* encuentra su mejor rendimiento en 4 capas ocultas solo superando por un leve margen el 50 %. En Figura 6.5, se cambió a la función de activación *Leaky ReLU* dado que para *ReLU*, si el resultado es negativo, la función retorna 0. Se utilizó un $\alpha = 0.1$, pero los resultados no fueron muy diferentes a utilizar *ReLU*. Al contrario, se observa un *accurary* levemente inferior a a Figura 6.4 (entre 61 y 63 %) con un *F1 Score* que no logra superar los 47 %.

Figura 6.6 muestra que la red si tuvo un entrenamiento apropiado. A través de las épocas se fue disminuyendo su error al mismo tiempo que iba aumentando el *accuracy*. Esto dice que la red si estaba bien configurado pero es el concepto teórico de la red que no permite aumentar de alguna manera su desempeño. A estas alturas lo mejor sería complementar las redes con otro modelo o aplicar técnicas en el proceso de preprocesamiento. En definitiva, la red neuronal queda con un desempeño por abajo de lo deseable para el problema.

6.2.2. Arboles de Decisión

Dentro de la configuración de un árbol de decisión, como se menciono en 3.2.2.1, existen varios parámetros interesantes al estructurar su base. Uno de ellos es el máximo largo del árbol. Es decir, cuantos nodos hacia abajo permite el árbol llegar para que se detenga. De igual manera, el algoritmo necesita una función de impureza. En estos casos se utilizó las función gini y entropía dado que son las que se pueden utilizar para este tipo de problemas. Como se espera, se observa en Figura 6.7 un aumento del *accuracy* desde un 58 % a un 66 % para un *max depth*=30. Lo mismo se refleja para *F1 Score* que tiene un comportamiento

similar. En lo que respecta la función de impureza se observa que son similares en términos de desempeño.

A continuación se varía el mínimo *samples split* que expresa cuantos datos mínimos tienen que quedar para que el árbol de decisión siga creciendo. Es opositora al largo del árbol dado que el algoritmo respeta las reglas que se le entregan para la detención del mismo. En Figura 6.8 se quiso experimentar solo con el mínimo de *samples* por nodo. Ser observa que de manera similar a Figura 6.7 se tiene un aumento de *accuracy* hasta un 67 % sin hacer diferencia en las funciones de impureza. En Figura 6.8 se tiene un estancamiento en *min smaples split*=21 con un *F1 score* que alcanza a llegar hasta 57 %.

Para la creación de cada nodo, el algoritmo tiene que buscar que variables o *features* son más convenientes para hacer la partición o *split*. Por defecto, este número es igual a $\sqrt{n_{\text{feautures}}}$ No obstante, esto puede cambiar en vista de mejorar los desempeños. Esto es lo que se hace en Figuras 6.9 y 6.10 donde se escogió los mejores parámetros de las figuras pasadas para intentar encontrar la mejor configuración. En ambas figuras se muestra el valor que viene por defecto (por la cantidad de atributos) en azul oscuro y una alternativa que también se utiliza ($log_2(n_{\text{feautres}})$) en azul claro.

En Figura 6.9 se tiene un leve aumento de *accuracy* hasta 68 % para valores arriba de 6, alejados de la raíz cuadrada en azul que se aproximaría a 4. *F1 Score* también experimenta un leve aumento a llegar a 61 % sin todavía experimentar una diferencia notable entre las funciones de impureza. Para figura 6.10 comienzan en un 2-3 % más bajo en *accuracy* y un 4-5 % en *F1 Score*. El gráfico muestra un aumento parecido en Figura 6.10, pero llegando a valores más bajo (66 % y 57 %) para *accuracy* y *F1 Score* respectivamente dado por su partida más baja.

El árbol de decisión tiene la ventaja que por su configuración interna se puede obtener la relevancia de ciertos atributos a la hora de crear los nodos de separación. Para este caso se muestra en Figura 6.11. Los primeros cinco atributos son las variables sacadas de segment. Muestran que lo que más influye es el navegador que utilizan, al igual que los tiempos entre cada paso del registro. La edad es el primer atributo de caracterización que tiene algún peso. De esta manera queda en evidencia que es más importante la conducta del usuario más que atributos que lo caractericen.

6.2.3. Random Forest y Gradient Boosting

De la sección anterior, se puede decir entonces que para Árbol de Decisión se tiene un mejor desempeño utilizando *max depth*=30, *max features*=6 llegando casi a un 68 % de *accuracy* por arriba que la red neuronal. Esta configuración se utiliza para los próximos dos algoritmos Random Forest y Gradient Bossting que itera sobre arboles de decisión.

Para experimentar con Random Forest, lo que se hace es ir aumentando la cantidad de muestras (*n estimators*) para sacar el cálculo estadístico correspondiente. Se observa que llegando a $n_{\text{estimator}} = 100$ el *accuracy* ya converge a un 73 % tanto para gini y entropía. Los resultados suben considerablemente comparando al mejor caso de Árbol de Decisión y

esto ocurre por el hecho de ir haciendo un promedio estilo *bagging*. De esta manera, se logra generalizar de mejor manera la muestra completa.

La misma metodología se aplica para Gradient Boosting. Primero se busca el mejor coeficiente de aprendizaje que ayuda para encontrar la función decisión definitiva. En Figura 6.13 se cambia el orden de magnitud. Tanto *accuracy* como *F1 Score* tienen exactamente la misma forma, encontrando los mayores resultados en 10^{-2}, 10^{-1}. Tomando el segundo coeficiente mencionado se procede a aplicar Gradient Boosting cambiando el *n estimator* como experiencia análoga de Random Forest en Figura 6.14. Se observa que el desempeño es el mismo para estimadores de 10 a 350 para ambos criterios. Posteriormente, se observa una caída brusca en *F1 Score* haciendo que el algoritmo se sobre aprenda los valores de entrenamiento. Se observa que tanto Gradient Boosring y Random Forest tienen desempeños bastantes similares a pesar de utilizar técnicas diferentes sobre los Árboles de Decisión.

6.2.4. Naives Bayes

El último algoritmo que se experimentó fue Naives Bayes. Este algoritmo no posee tantas variables para calibrar como se mostró en sección 3.2.3. Lo único que relevante en la configuración es la función de distribución que puede implementar. Para esto se implementó las funciones Gausiana, Bernouilli, Categórica y Complementaria.

El mejor *accuracy* lo tiene la función Categórica llegando a 61 %, pero posee un *F1 Score* de 45 %. La única función que tiene arriba de un 50 % de *F1 Score* es la función Gaussiana, pero posee el peor *accuracy* en menos de 56 %.

Al observar solo estos dos casos, se identifica que Naives Bayes no posee un desempeño competitivo en comparación a los otros algoritmos y esto se puede explicar en los ámbitos son utilizados . Todos los algoritmos son herramientas para encontrar solución a un mismo problema pero con enfoques diferentes. Naives Bayes es bastante utilizado para detectar correos de spam por ejemplo, pero no logra tener un desempeño aceptable en predicción de conducta.

6.2.5. Desempeños Generales

Finalmente se tienen todos los desempeños de tabla 6.1. Como se estuvo analizando, Random Forest y Gradient Boosting son los que tienen mejor rendimiento y Naives Bayes con MLP los peores.

Curiosamente, ningún modelo tuvo arriba de 65 % en *presision*. Al contrario, parece siempre ser la peor métrica. Recordar que *precision* muestra el número de Verdaderos Positivos que hay en función del mismo número de Verdaderos Positivos sumado a los Falsos Positivos. Una baja *precision* dice que existen muchos Falsos Positivos. En la aplicación, todos los algoritmos tienen más problemas al detectar los casos positivos que negativo. La interpretación de esto es que existen usuarios que tienen conductas muy similares a alguien que si consiguió la inducción, pero que no llegan a aquello por alguna razón que se desconoce.

Haciendo foco en los dos mejores desempeños, se tiene que Random Forest tienen mejor *precision* que es lo que se busca (64 vs 62 %). No obstante, *recall* es más importante porque es peor que se pierda un potencial jobber que iba a tomar inducción a que se incentive a alguien que no tenía la intención de tomarla. En este ámbito, Gradient Boosting presenta mejor porcentaje (73 vs 70 %).

No obstante, ambos resultados son semejantes. También es importante recalcar que si bien se obtuvo un 75 % de *accuracy* [6.1], esto sigue siendo bajo en el mundo de algoritmos predictivos. Por lo general se espera tener al menos un 80 % de *accuracy* siempre y cuando sea posible. El tema aquí es que se tiene poca información para predecir conductas humanas. Los atributos de mayor influencia son los tiempos de cada hito y se nota que no son suficientes para mejorar la predicción. Por lo general se espera tener al rededor de un 75 % de *accuracy* en algoritmos de fuga de clientes por lo que Random Forest y Gradient Boosting quedan a la altura de lo que se puede llegar.

Finalmente entre elegir Random Forest o Gradient Boosting se considera que ambos son aptos para el problema. Al mirar las matrices de confusiones 6.16 y 6.17 se observa que la diferencia son entre 10 usuarios más o menos en cada casilla. Se sugiere elegir el algoritmo que requiere menos capacidad computacional que vendría siendo Random Forest. El resto de las matrices de confusión se pueden encontrar en Anexo D.

Capítulo 7

Implementación

En esta sección se mostrará de manera breve las herramientas y la lógica utilizada para llevar a cabo la aplicación del algoritmo. Se quiere mostrar de manera sintetizada como se logra el despliegue del modelo. Cabe mencionar que este trabajo pudo ser realizado dado que el *Data Lake* ya estaba funcionando. El *Data Lake* va guardando en un *bucket* en *Cloud Storage* todos los usuarios actualizados del día anterior.

En una primera instancia se entrenó el modelo y se guardó la mejor configuración en un archivo *joblib*. Se crearon tres scripts *gcp.py*, *prediction_alg.py* y *main.py* como se ve en Figura 7.1. *Gcp.py* lee los datos necesarios entre GCP (*Big Query* y *Data Lake*) y Mongo DB. Para cada uno tiene que ser capaz de autentificar y dejar los datos en formato adecuado para el resto del código. *Prediction_alg* preprocesa los datos y calcula la probabilidad dado el modelo guardado en *joblib*. *Main.py* es el orquestador y está diseñado para auto-ejecutarse. Al ejectuarlo, verifica que día es, va a buscar los usuarios actualizados en *Data Lake*, hace el cruce con los datos de mongo como de *Big Query*(segment) y los deja listo para el resto del código.

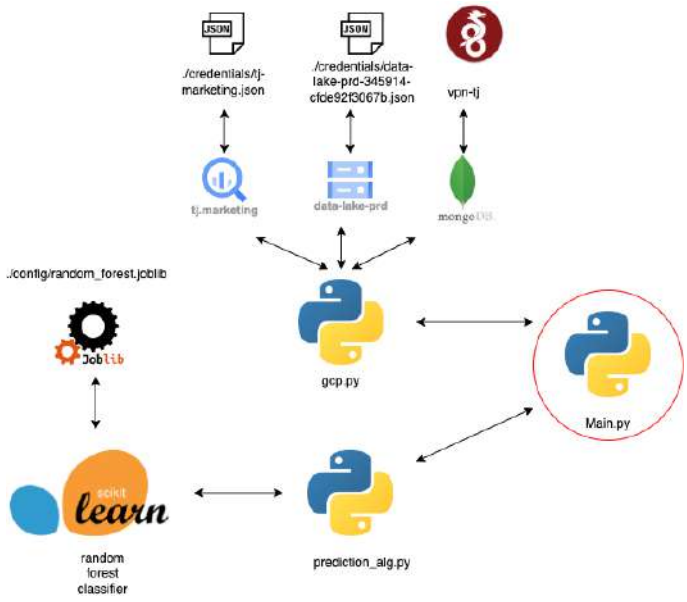

Figura 7.1: Lógica de código del modelo en python

Posteriormente se tuvo que configurar máquina virtual de *Compute Engine* de GCP para que pueda albergar la estructura de código mostrada en Figura 7.1. Esta se configuró para que se ejecute en la madrugada.

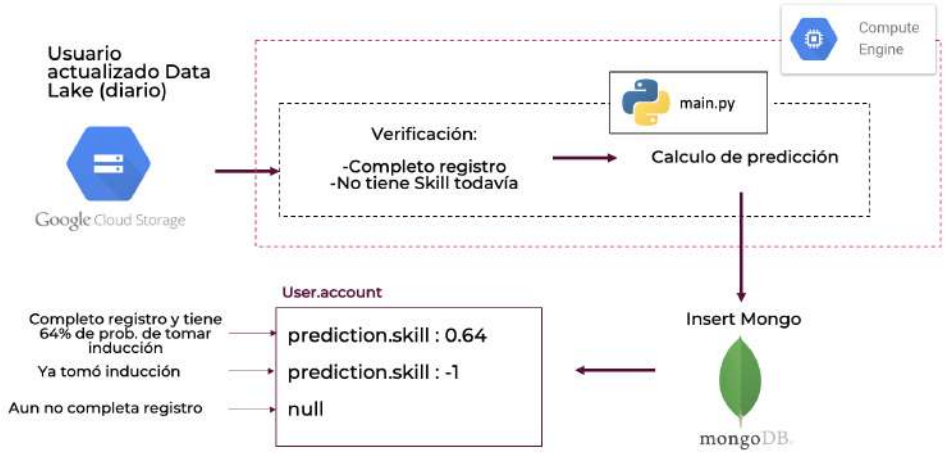

Figura 7.2: Lógica del flujo completo

Al terminar de ejecutar el código, se hace el *insert* a la base de datos de mongo como se muestra en figura 7.2. Son tres las opciones que se puede tener en el usuario posterior a que corra el algoritmo.

Si el usuario cumple con los requisitos, se calcula la probabilidad entre 0 y 1. Si el usuario ya tomó inducción, no se calcula nada y se inserta un -1. Si el usuario aun no completa el registro, no se inserta nada y el campo queda vació hasta que lo complete y logre ser calculado.

Finalmente, desde el FrontEnd (lo visual), se lee el dato y se muestra en la tabla de usuarios que aún no están listos para tomar tareas. En la plataforma de Timejobs existen dos secciones para visualizar los jobbers. La primera son los pendientes *applicants* que les falta o tomar inducción o terminar el proceso de registro. El segundo son los jobbers que ya están disponibles para tomar tareas. Esta visualización se implementa en la primera.

Si bien el algoritmo retorna un número, se pensó en una escala de colores estilo semáforo para hacer más intuitivo el dato al operador. Se utiliza rojo-amarillo-verde para los rangos [0,0.33], [0.33,0.66], [0.66,1] respectivamente. En definitiva, la vista queda como se ve en Figura 7.3 donde se agrega la columna Posibilidad inducción. Si se lee un número se agrega el logotipo mostrado con el color asociado. Se muestra 'sin información' si es que el campo es vació (no termina el registro aun) y si ya tiene inducción no aparece en esa tabla por defecto.

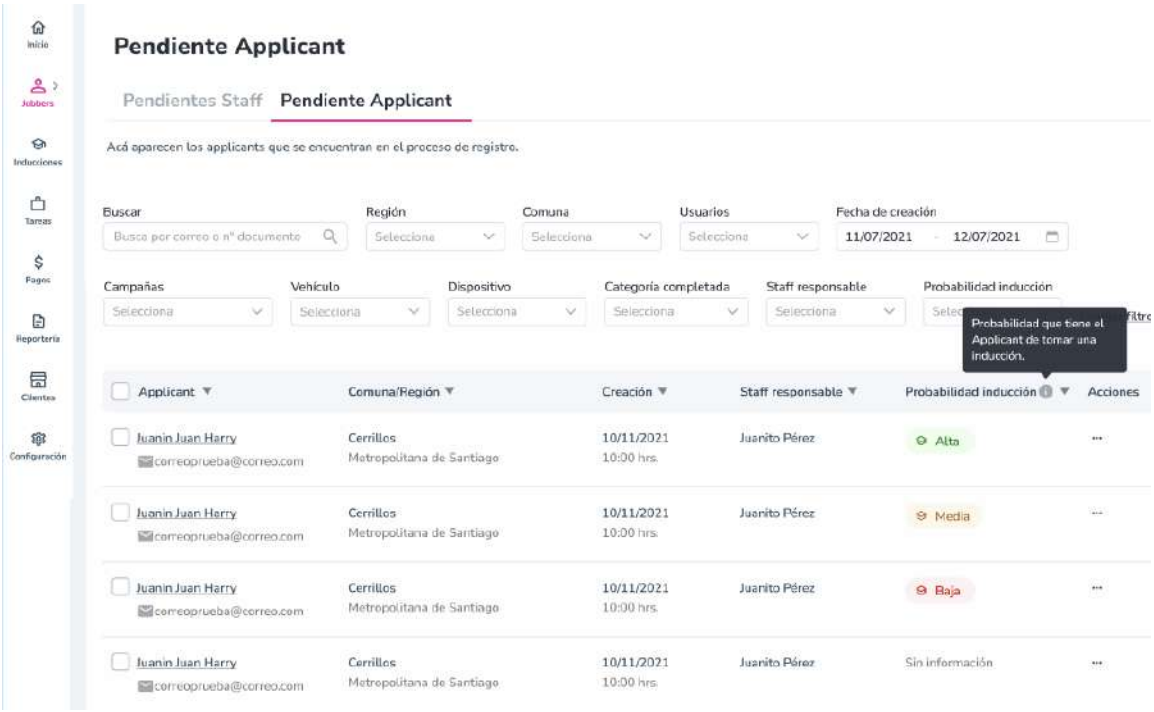

Figura 7.3: Visualización de la tabla de jobbers que usan los operadores

Capítulo 8

Conclusiones

En una primera instancia es importante destacar que se cumplieron los objetivos propuestos. Se logró comprender cualitativamente el proceso, caracterizar al usuario, seleccionar la data pertinente, construir el algoritmo e implementarlo. Esto hace enriquecedor el trabajo en torno a la profundidad y alcance que se le dio al principio del proceso.

Por el otro lado, se logró aplicar los conocimientos de ingeniería eléctrica hacia un problema real y doloroso para una empresa. Del mismo modo la solución dio valor a la empresa, entregándole nuevas herramientas tecnológicas. Esto es fundamental para memorias aplicadas a la industria. No obstante, no fue directo poder aplicar los conocimientos de la carrera. Un trabajo previo de formación del área de Datos en Timejobs al igual que la incorporación de un *Data Engineer* fue necesaria para la culminación del proyecto, sin contar con la ayuda de otras personas del área de Tecnología.

Asimismo, se encontró que algoritmos de Redes Neuronales y Naives Bayes muestran bajos desempeños para predecir conductas humanas. Random Forest y Gradient Boosting, por el otro lado, si se muestran competentes para para implementarse versus algoritmos de carácter tradicional dado sus mejores metricas. Su mejor desempeño en relación a los otros algoritmos propuestos radica en su naturaleza estadística que poseen los últimos dos que ayuda a no ser tan estricto con las especificaciones de entrada.

La mayor dificultad del trabajo fue poder entrelazar los ritmos universitarios con los ritmos de la vida laboral. Muchas veces la universidad tienen un enfoque teórico mientras que la empresa busca tener resultados rápidos para los nuevos problemas que van surgiendo.

Se recomienda, como trabajo futuro, implementar otros modelos como *Support Vector Machine* que la literatura recomienda. Pero más aún, se recomienda a Timejobs seguir utilizando herramientas de *machine learning* para resolver problemas tangibles como la asignación de jobber tarea o algoritmos de bonificación de jobber para aumentar su actividad. Se cree que esta es la vía para automatizar dolores poco abordados en la empresa. Este trabajo abarcó todo lo necesario para diseñar, entrenar e implementar modelos de aprendizaje automático, por lo que puede servir de guía para un trabajo de la misma índole en Timejobs.

Bibliografía

[1] Guangrong Dai, K. P. D. M., "A review of onboarding literature," Lominger Limited, Inc., a subsidiary of Korn/Ferry International, 2007.

[2] Christina Stoiber, Davide Ceneda, M. W. V. S. T. G. M. S. S. M. W. A., "Perspectives of visualization onboarding and guidance in va," Visual Informatics, 2022, https://reader.elsevier.com/reader/sd/pii/S2468502X22000134?token=6F035BF9B9B6C863D660E01A35492A30299164F92368DA3C46C31B9F5594B9014EB5A98D0CDA52BB2E4718075A026A79&originRegion=us-east-1&originCreation=20220511224617.

[3] Institute, S. A. S., "Big data qué es y por qué es importante," 2016, https://www.sas.com/es_mx/insights/big-data/what-is-big-data.html.

[4] Corporation, I. B. M., "¿qué es business intelligence?," 2022, https://www.ibm.com/cl-es/topics/business-intelligence.

[5] Institute, S. A. S., "Big data analytics what it is and why it matters," 2016, https://www.sas.com/en_us/insights/analytics/big-data-analytics.html.

[6] Corporation, I. B. M., "Data science," 2020, https://www.ibm.com/cloud/learn/data-science-introduction.

[7] Corporation, I. B. M., "Artificial intelligence (ai)," 2020, https://www.ibm.com/cloud/learn/what-is-artificial-intelligence.

[8] Fayyad, U., "From data mining to knowledge discovery in databases," 1996, https://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1996-Fayyad.pdf.

[9] KDD, "Data mining curriculum: A proposal," 2021, http://www.kdd.org/curriculum/index.html.

[10] Zhang, X.-D., "Machine learning," 2020, https://link.springer.com/chapter/10.1007/978-981-15-2770-8_6.

[11] Soni, D., "Supervised vs. unsupervised learning," 2018, https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d.

[12] Flach, P., "Machine learning: The art and science of algorithms that make sense of data," 2012, https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d.

[13] Haykin, S. O., "Neural networks and learning machines," McMaster University, 1999.

[14] Tom Mitchell, M. H., "Machine learning," 1997.

[15] Lu, Y., "Decision tree methods: applications for classification and prediction," 2015.

[16] L Breiman, J Friedman, R. O. C. S., "Classification and regression trees," Wadsworth, Belmont, 1984.

[17] Azencott, C.-A., "Introduction au machine learning," Dunod, 2019.

[18] Trevor Hastie, Robert Tibshirani, J. F., "The elements of statistical learning," Springer, 2009.

[19] Rennie, J. D., "Improving multi-class text classification with naive bayes," massachusetts institute of technology — artificial intelligence laboratory, 2001.

[20] Wei Zhang, F. G., "An improvement to naive bayes for text classification," Advanced in Control Engineeringand Information Science, 2011, https://reader.elsevier.com/reader/sd/pii/S1877705811019059?token=761C5F1C962D4E95FC8F1CD396602AF95DACE24AFC293FCC41E6B2E1D48CD8E23332B49657F550F6D36C6EF5EA66B994&originRegion=us-east-1&originCreation=20220512203553.

[21] Jianlong Zhou, Amir H. Gandomi, F. C. y Holzinger, A., "Evaluating the quality of machine learning explanations: A survey on methods and metrics," electronics, 2021, https://pubs.acs.org/doi/abs/10.1021/acs.molpharmaceut.8b00546?casa_token=4kjSwoMSKMMAAAAA:g8eu0lamIUObxuGmUdlG6spsRhhzkdPy2N0HKvKnXZQ7Ruy09sZm_30K2LNL5B6_WewTZoiWkGsGJVbE.

[22] Jacob Høxbroe Jeppesena, Rune Hylsberg Jacobsena, F. I. T. S. T., "A cloud detection algorithm for satellite imagery based on deep learning," Remote Sensing of Environment, 2019, https://www.researchgate.net/publication/334840641_A_cloud_detection_algorithm_for_satellite_imagery_based_on_deep_learning.

[23] Daniel P. Russo, Kimberley M. Zorn, A. M. C. H. Z. y Ekins, S., "Comparing multiple machine learning algorithms and metrics for estrogen receptor binding prediction," Molecular Pharmaceutics, 2019, https://pubs.acs.org/doi/pdf/10.1021/acs.molpharmaceut.8b00546.

[24] Vorhies, W., "Crisp-dm – a standard methodology to ensure a good outcome," 2016, https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome.

[25] Valenzuela, F. L. F., "La estructura de la densidad socio–residencial en el Área metropolitana de santiago," Proyecto Fondecyt Nº 1161550, 2018.

Anexos

Anexo A. Distribución variables de entrada

Se presenta cierta estadística con respecto a las variables de entrada luego del preprocesamiento

	personalData.gender	personalData.originCountry	otherData.cellphoneOS	otherData.vehicles	bankingData.bank	bankingData.accountType
count	21825.000000	21825.000000	21825.000000	21825.000000	21825.000000	21825.000000
mean	0.553906	0.669278	0.290584	0.281232	0.374296	0.246140
std	0.497097	0.470484	0.454042	0.433769	0.483952	0.430771
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Figura A.1: Estadística de variables de entrada post preprocesar parte 1

	edad	personalData.address.localityName	otherData.referral	utmSource	personal_data_time	banking_data_time
count	21825.000000	21825.000000	21825.000000	21825.000000	21825.000000	21825.000000
mean	0.144842	0.417637	0.222671	0.887892	0.164518	0.162508
std	0.119332	0.330662	0.260757	0.296175	0.174434	0.157712
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.050633	0.160000	0.000000	1.000000	0.071679	0.075901
50%	0.113924	0.330000	0.200000	1.000000	0.093224	0.099858
75%	0.202532	0.500000	0.400000	1.000000	0.142691	0.153861
max	0.772152	1.000000	1.000000	1.000000	0.777181	0.763405

Figura A.2: Estadística de variables de entrada post preprocesar parte 2

	other_data_time	police_record_time	register_fulfilled_time	context_user_agent
count	21825.000000	21825.000000	21825.000000	21825.000000
mean	0.169391	0.223933	0.318149	0.608902
std	0.171270	0.169589	0.123446	0.308662
min	0.000000	0.000000	0.000000	0.000000
25%	0.069004	0.103232	0.231289	0.358978
50%	0.094533	0.144261	0.319631	0.643808
75%	0.168772	0.330752	0.384221	0.899226
max	0.748504	0.736385	0.674490	1.000000

Figura A.3: Estadística de variables de entrada post preprocesar parte 3

Anexo B. Caracterización

A continuación se muestran los resultados más relevantes dentro del primer ciclo Caracterización que le es de utilidad a la empresa. Estos resultados logran detallar a los usuarios internos.

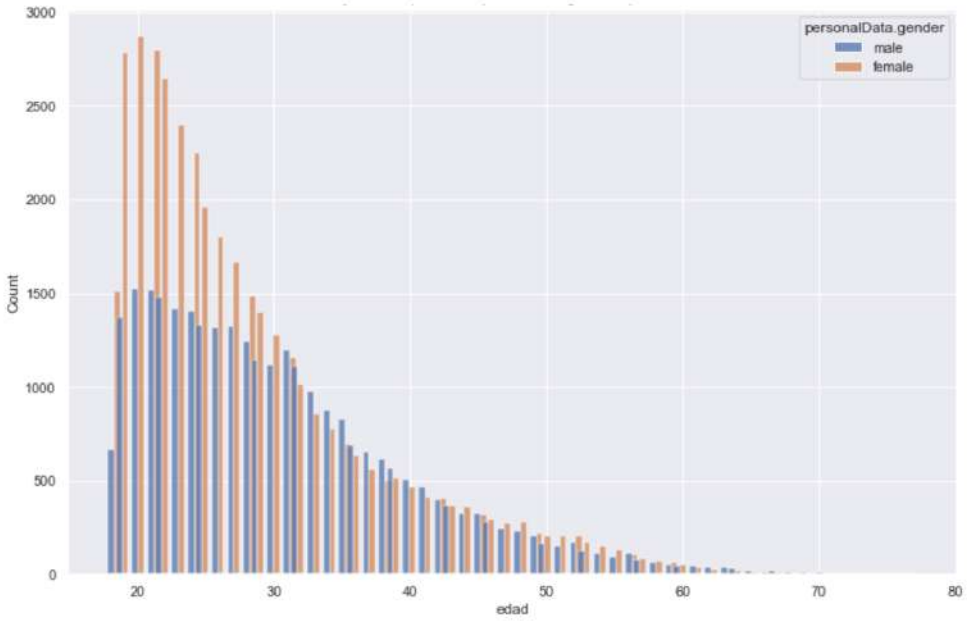

Figura B.1: Edad Y Sexo de los usuarios que completaron registro

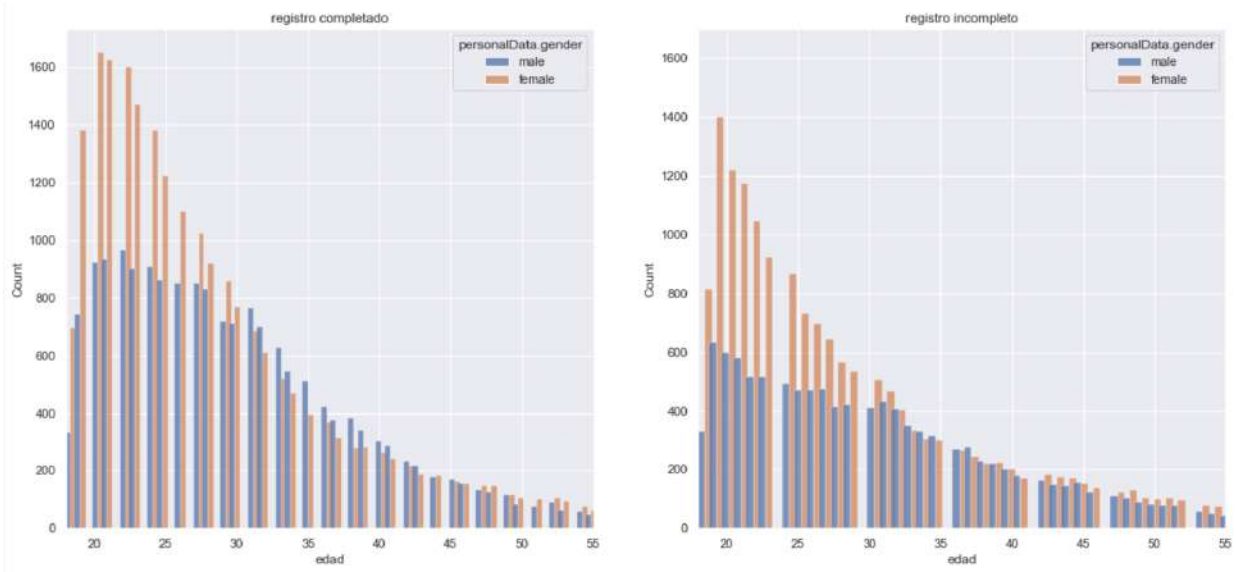

Figura B.2: Edad Y Sexo de los usuarios que comienzan el registro vs los que completaron el registro

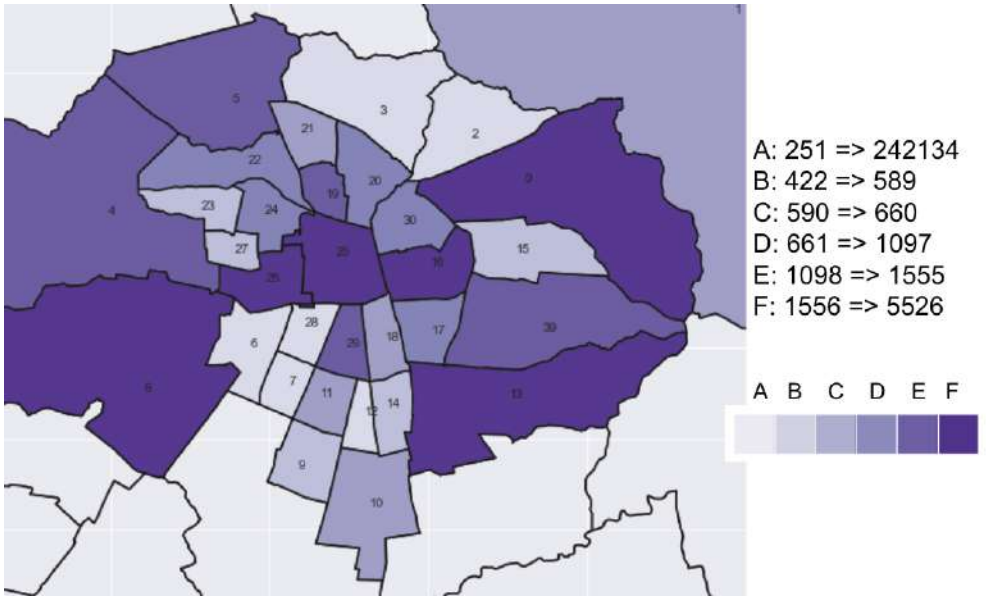

Figura B.3: Mapa de Calor de comunas donde los usuarios comienzan a registrarse en aplicación Timejobs

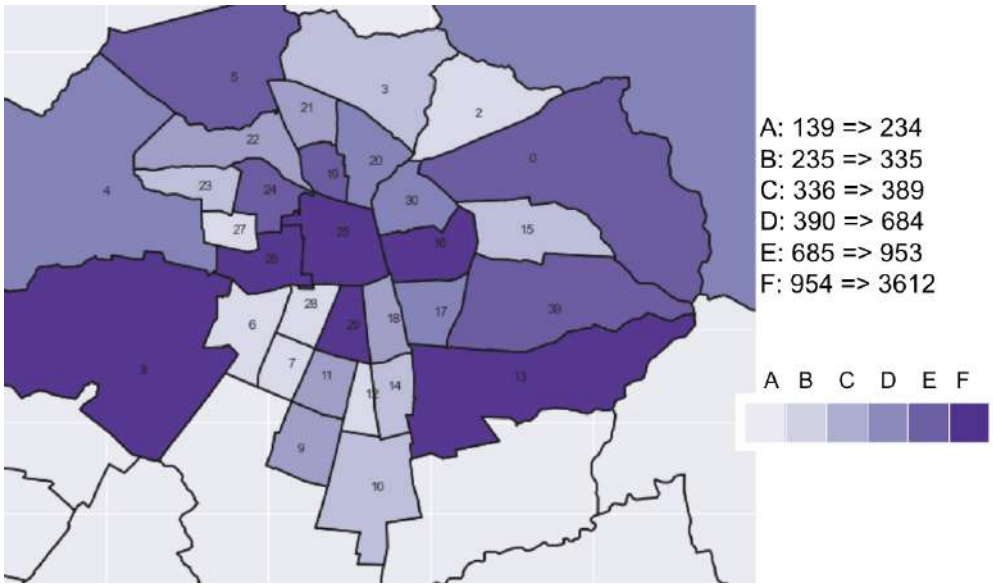

Figura B.4: Mapa de Calor de comunas donde los usuarios terminan de registrarse en aplicación Timejobs

Anexo C. Agrupamiento de Usuarios que no terminan el registro

A continuación se muestra el agrupamiento o *cluster* de los usuarios que no terminaron el registro presentado en Introducción - Problemáticas en Reclutamiento 1.1.5. Se presenta primero la conversión de estados a numéricos y posteriormente los agrupamientos.

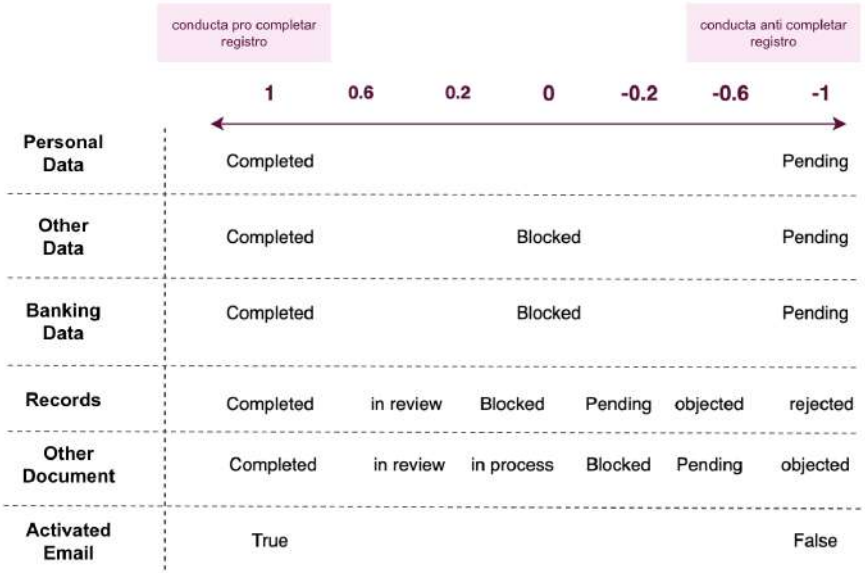

Figura C.1: Preprocesamiento de string a numérico de los diferentes estados al registrarse en aplicación

El método del codo dio dos posibilidades viables. Agrupamiento de 4 y de 6. Por interpretabilidad se escogió mostrar solamente el agrupamiento de a 4.

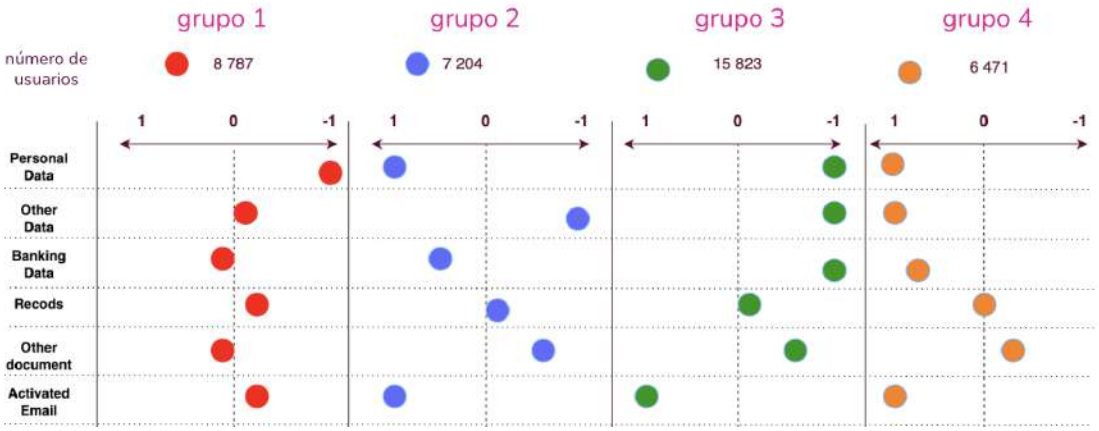

Figura C.2: *Cluster* para 4 grupos de usuarios que no terminan registro a través de estado en los pasos de la aplicación

Anexo D. Resultados Modelos con bajo desempeño

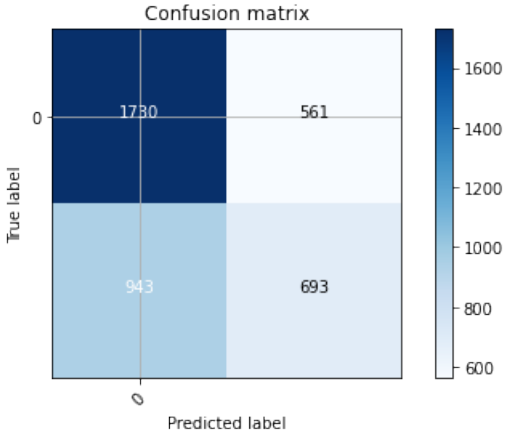

Figura D.1: Matriz de Confusión de MLP

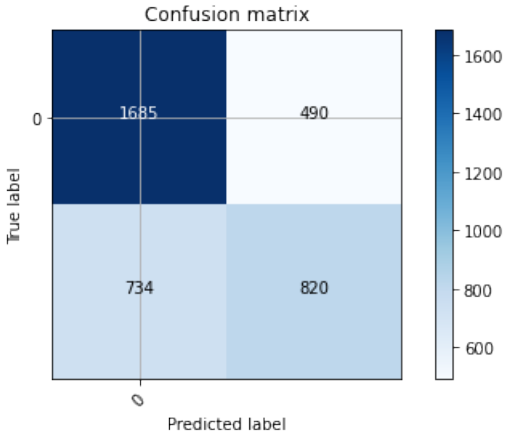

Figura D.2: Matriz de Confusión de Arból de Decisión

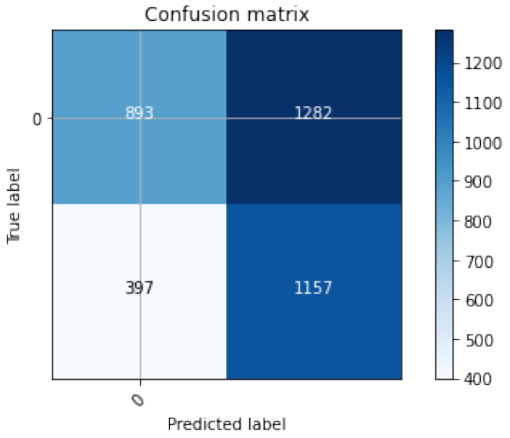

Figura D.3: Matriz de Confusión de Naive Bayes (Gaussiana)

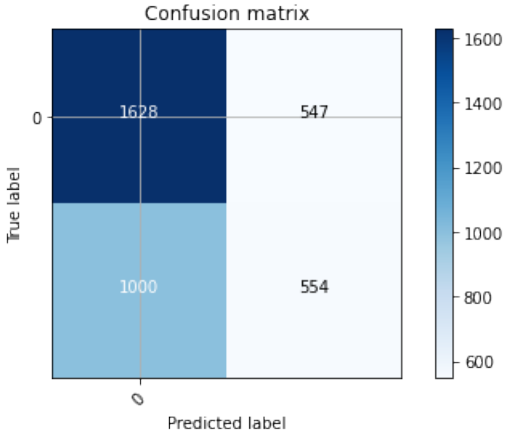

Figura D.4: Matriz de Confusión de Naive Bayes (Bernouilli)

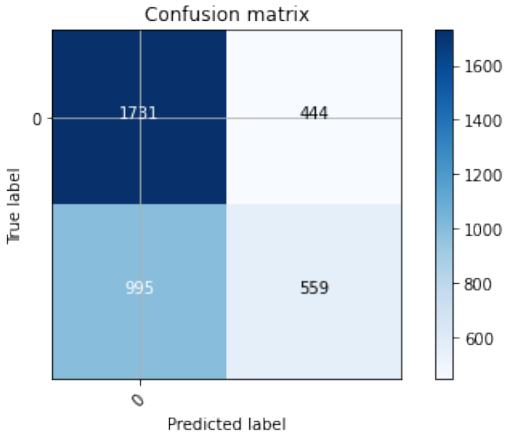

Figura D.5: Matriz de Confusión de Naive Bayes (*Categorical*)

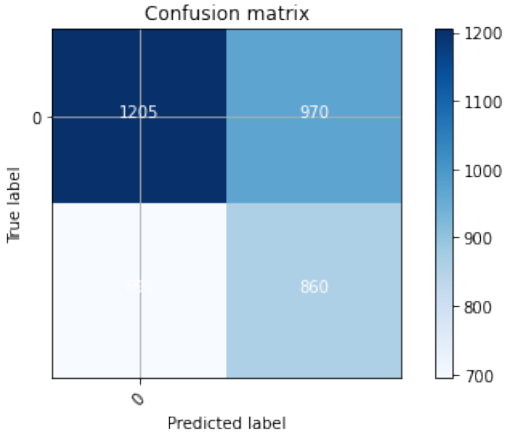

Figura D.6: Matriz de Confusión de Naive Bayes (*Complement*)